,

Programmation *Web* en *PHP*, Conception, Architectures et Développement de *Web Services*

Coté serveur : *PDO, DAL, MVC, Front Controller, API Restful*

Rémy Malgouyres
LIMOS UMR 6158, IUT, département info
Université Clermont Auvergne
B.P. 86
63172 AUBIERE cedex
https://malgouyres.org/

Tous mes cours sur le *Web* sont sur le *Web* :

Cours de programmation *WEB* sur les documents hypertexte *HTML/CSS* :

https://malgouyres.org/programmation-html-css

Tutoriel sur le *CMS Drupal* :

https://malgouyres.org/tutoriel-drupal

Cours de programmation *WEB* côté serveur en *PHP* :

https://malgouyres.org/programmation-php

Cours de programmation *WEB* côté client en *JavaScript* :

https://malgouyres.org/programmation-javascript

Cours sur l'administration de serveurs (Serveurs *WEB* avec *apache*, *SSL*, *LDAP*...) :

https://malgouyres.org/administration-reseau

Table des matières

Première partie

Bases du langage *PHP*

Table of Contents

Chapitre 1

PHP procédural

1.1 Notion de *CGI*

On appelle *Common Gateway Interface*, ou en abrégé *CGI*, une interface, utilisée par les serveurs *HTTP*, qui permet de générer la réponse du serveur par un programme, qui s'exécute sur le serveur. Le programme pourra, assez typiquement, générer du code *HTML* qui sera affiché par un navigateur côté client. L'interface *CGI* est indépendante du langage de programmation utilisée par le serveur, et n'utilise que les flux standards et les variables d'environnement.

Voici un exemple de *CGI* programmé en *C* :

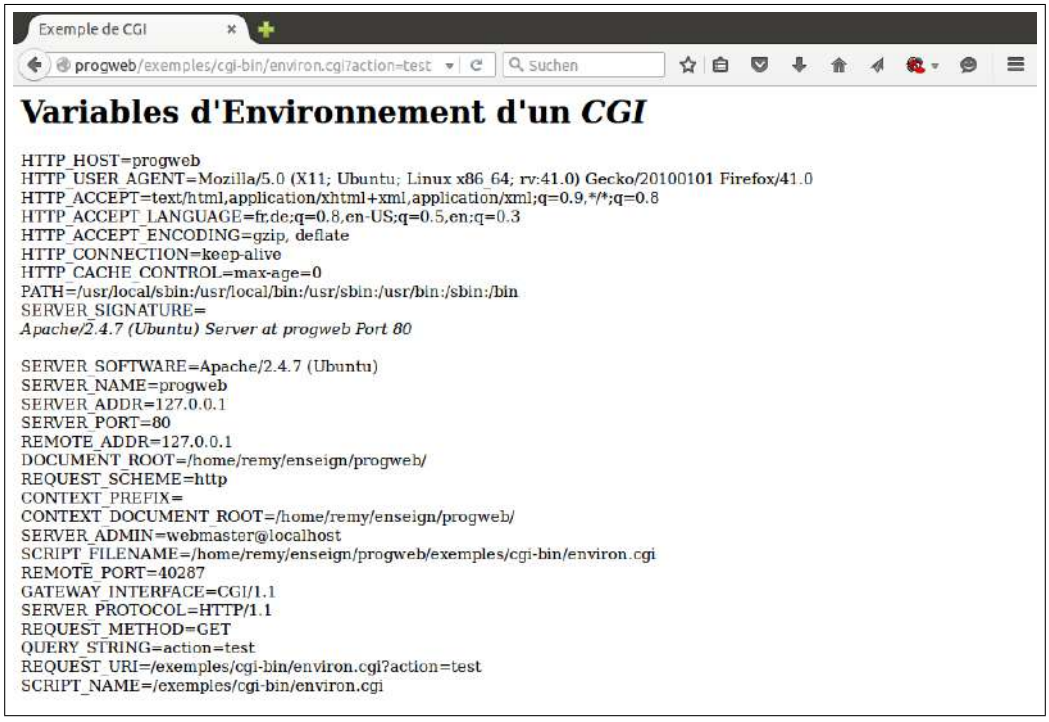


FIGURE 1.1 : Illustration du code source 1.1

Code Source 1.1 : /cgi-bin/environ.c (cf. Fig 1.1)

```
1 #include <stdio.h>
2
3 extern char **environ;
```

```
4
5  int main(void)
6  {
7      int i;
8      printf("%s%c%c\n","Content-Type:text/html;charset=iso-8859-1",13,10);
9
10     printf("<html>");
11     printf("<head>");
12     printf("<title>Exemple de CGI</title>");
13     printf("</head>");
14     printf("<body>");
15     printf("<h1>Variables d'Environnement d'un <i>CGI</i></h1>");
16
17     for (i=0 ; environ[i]!=NULL ; i++){
18       printf("%s<br/>\n", environ[i]);
19     }
20
21     printf("</body>");
22     printf("</html>");
23     return 0;
24 }
```

1.2 Générer du code *HTML* avec un *CGI* en *PHP*

Le PHP est un langage de programmation (ou langage de scripts) qui permet de générer et d'afficher des pages webs dynamiques, c'est à dire des pages dont le contenu dépend des actions de l'utilisateur ou de l'état, par exemple, d'une base de données. En fin de compte, le code affiché est toujours du code HTML. Ce code HTML est généré par le programme PHP via la commande `echo`. La protection des caractères spéciaux du HTML (comme les guillemets) et le mélange du code PHP et du code HTML rend souvent le code d'un script PHP. Nous verrons plus loin comment atténuer ce problème par une approche modulaire fondée sur la programmation objet.

Le script PHP est inséré à l'intérieur d'une balise `<?php ?>` qui peut s'insérer au sein du code HTML.

FIGURE 1.2 : Illustration du code source 1.2

Code Source 1.2 : /php1/ex01-helloWorld.php (cf. Fig 1.2)

```
1 <!doctype html>
2 <html lang="fr">
3   <head>
4     <meta charset="utf-8"/>
5     <title>Hello World en PHP</title>
6   </head>
```

```
7   <body>
8     <p>
9       <?php // début du script PHP
10      echo "Hello World !";
11      // On affiche du code HTML si la sortie
12      ?> <!-- fin du script PHP -->
13    </p>
14  </body>
15 </html>
```

1.3 Exemple de fonction en PHP

Ici, nous voyons une fonction PHP qui génère l'en-tête XHTML du document et son header. Cette fonction prend en paramètre le titre, le charset et l'url d'une feuille de style CSS à appliquer dans le header HTML. Le résultat est que lors de l'utilisation de la fonction, presque tout le code HTML disparait pour être remplacé par une seule ligne de code, ce qui en fin de compte allégera de beaucoup le code source PHP.

Code Source 1.3 : /php1/ex02-function.php

```
1  <?php // début d'un script PHP
2  function outputEnTeteHTML5($title, $charset, $css_sheet){
3    // sortie du doctype. Les guillemets HTML sont protégés par \
4    echo "<!doctype html>\n";
5    echo "<html lang=\"fr\">\n";
6    echo "<head>\n";
7    echo "<meta charset=\"";
8    echo $charset;
9    echo "\"/>\n";
10   echo "<link rel=\"stylesheet\" href=\"";
11   echo $css_sheet;
12   echo "\" />\n";
13   // concaténation de chaînes de caractères.
14   echo "<title>".$title."</title>\n";
15   echo "</head>\n<body>\n";
16 }
17
18 function outputFinFichierHTML5(){
19   echo "</body>\n</html>\n";
20 }
21 ?>
22
23 <?php
24   outputEnTeteHTML5('Hello world version 2', 'UTF-8', 'myStyle.css');
25   echo "<p>Hello World !</p>"; //
26   outputFinFichierHTML5();
27 ?>
```

1.4 Inclure un fichier PHP dans un autre

Évidemment, si le but des fonctions PHP est de cacher et de réutiliser une partie du code, il est commode de pouvoir écrire une fois pour toutes la fonction dans un seul fichier, puis

d'utiliser la fonction dans tous nos scripts par la suite. Ici les fonctions `outputEnTeteXHTML` et `outputFinFichierXHTML` sont utilisées dans tous les scripts qui affichent du code HTML. (en effet, nous verrons plus loin que certains fichiers PHP sont de la pure programmation et n'affichent rien.)

Code Source 1.4 : /php1/commonFunctions.php

```php
<?php // début d'un script PHP
  function outputEnTeteHTML5($title, $charset, $css_sheet){
    // sortie du doctype. Les guillemets HTML sont protégés par \
    echo "<!doctype html>\n";
    echo "<html lang=\"fr\">\n";
    echo "<head>\n";
    echo "<meta charset=\"";
    echo $charset;
    echo "\"/>\n";
    echo "<link rel=\"stylesheet\" href=\"";
    echo $css_sheet;
    echo "\" />\n";
    // concaténation de chaînes de caractères.
    echo "<title>".$title."</title>\n";
    echo "</head>\n<body>\n";
  }

  function outputFinFichierHTML5(){
    echo "</body>\n</html>\n";
  }
?>
```

Code Source 1.5 : /php1/ex03-include.php

```php
<?php require('./commonFunctions.php');

  outputEnTeteHTML5('Hello world version 3', 'UTF-8', 'myStyle.css');
?>
<p>
<?php // début du script PHP
  echo "Hello World !"; // On affiche du code HTML si la sortie
  // fin du script PHP
?>
</p>
<?php
  outputFinFichierHTML5();
?>
```

1.5 Arithmétique : types `int` et `float`

En PHP, on ne déclare pas les types des variables ou des paramètres de fonctions. Celui-ci est défini lors de l'initialisation de la fonction. Des fonctions permettent cependant de tester le type ou d'accéder au nom du type d'une variable. Nous en verrons par la suite.

Code Source 1.6 : /php1/ex04-arithmetique-types.php (cf. Fig 1.3)

```php
<?php require_once './commonFunctions.php'; ?>
<?php
```

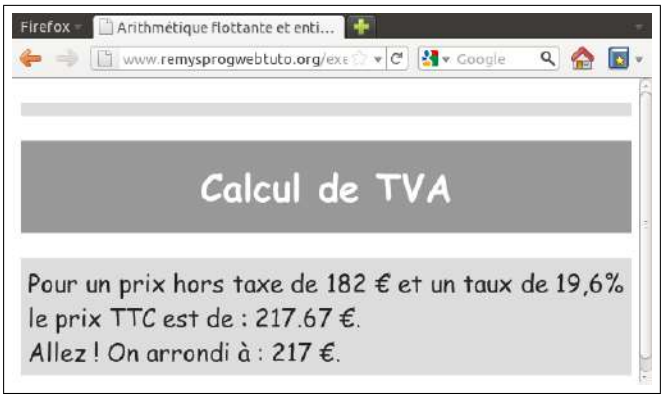


FIGURE 1.3 : Illustration du code source 1.6

```
3     outputEnTeteHTML5('Arithmétique flottante et entière', 'UTF-8', 'myStyle.css')
          ;
4   ?>
5   <p>
6   <?php // début du script PHP
7     function appliqueTVA($prixHT, $taux) {
8       $prixTTC = $prixHT*(1.0+$taux/100.0);
9       return $prixTTC;
10    }
11  ?>
12  <h1>Calcul de TVA</h1>
13  <p>
14  <?php
15    $prix = 182.0;
16    echo "Pour un prix hors taxe de ".$prix." &euro; et un taux de 19,6%\n";
17    echo "le prix TTC est de : "
18    .round(appliqueTVA($prix, 19.6),2)." &euro;.\n";
19    echo "<br/>\nAllez ! On arrondi à : ".intval(appliqueTVA($prix, 19.6))." &euro
          ;.\n";
20  ?>
21  </p>
22  <?php
23    outputFinFichierHTML5();
24  ?>
```

1.6 Tableaux indexés : avec une clé de type `int`

On crée un tableau avec la fonction `array`. On accéde à ses éléments (ici indexés par un `int`) en utilisant des crochets [ ]. La taille des tableaux peut être obtenue via la fonction `sizeof`.

Code Source 1.7 : /php1/ex05-tableaux-keyInt.php (cf. Fig 1.4)

```
1   <?php require_once './commonFunctions.php'; ?>
2
3   <?php
4     outputEnTeteHTML5('Tableaux 1', 'UTF-8', 'myStyle.css');
5   ?>
6   <p>
7   <h1>Tableaux avec clé entières</h1>
```

FIGURE 1.4 : Illustration du code source 1.7

```
8  <p>
9  <?php
10   $tableau = array(23, 45, 41, 6, 04);
11   echo "(";
12   for ($i=0 ; $i < count($tableau); $i++) {
13     echo $tableau[$i];
14     if ($i + 1 < count($tableau))
15       echo ", ";
16   }
17   echo ")\n";
18 ?>
19 </p>
20 <?php
21   outputFinFichierHTML5();
22 ?>
```

1.7 Tableaux associatifs : avec une clé de type `String`

Il existe en PHP une deuxième sorte de tableaux : les tableaux associatifs, ainsi nommés car ils associent une valeur à une clef qui est une chaîne de caractères. On peut tout de même parcourir l'ensemble du tableau en utilisant une boucle `foreach`.

FIGURE 1.5 : Illustration du code source 1.8

Code Source 1.8 : /php1/ex06-tableaux-keyString.php (cf. Fig 1.5)

```php
<?php require_once './commonFunctions.php'; ?>

<?php
  outputEnTeteHTML5('Tableaux 2', 'UTF-8', 'myStyle.css');
?>
<p>
<h1>Tableau avec clé de type String</h1>
<p>
<?php
  $tableau = array('nom' => 'Caesar', 'prénom' => 'Jules');
  echo "<ul>\n";
  // accès aux éléments :
  echo "<li>Accès aux éléments du tableau :<br/>";
  echo "Nom : ".$tableau['nom']."<br/>\n";
  echo "Prénom : ".$tableau['prénom']."</li>\n";

  // affichage de l'ensemble des valeurs du tableau par foreach :
  echo "<li>Les valeurs du tableau sont :</li>\n";
  foreach ($tableau as $chaine) {
    echo $chaine." ";
  }
  echo "<br/>\n";

  // affichage des clés et des valeurs du tableau
  echo "<li>Les données du tableau sont :<br>";
  foreach ($tableau as $cle => $chaine) {
    echo $cle." : ".$chaine."</li>\n";
  }
  echo "</ul>\n";
?>
</p>
<?php
  outputFinFichierHTML5();
?>
```

1.8 Passage de paramètre à un script PHP

Dans l'exemple suivant, le premier script passe deux paramètes au second : le titre de la page et le texte à afficher.

Nous transmettons ici les paramètres par la méthode GET, la méthode POST, qui a l'avantage de ne pas faire apparaître les paramètres dans l'URL, est similaire au niveau programmation et sera vue plus loin.

L'url du second script dans le navigateur est ici :

```
http://www.remysprogwebtuto.org/exemples/php1/\
              ex08-passages-parametres2.php?texte=Bonjour&titre=monTitre
```

Code Source 1.9 : /php1/ex07-passages-parametres1.php (cf. Fig 1.6)

```php
<?php require_once './commonFunctions.php'; ?>

<?php
```

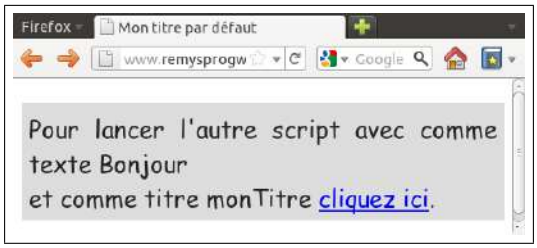


FIGURE 1.6 : Illustration du code source 1.9

```
4    $titre = 'Mon titre par défaut';
5    if (isset($_GET['titre'])){
6      $titre = $_GET['titre'];
7    }
8    outputEnTeteHTML5($titre, 'UTF-8', 'myStyle.css');
9  ?>
10 <p>
11 Pour lancer l'autre script avec comme texte
12 <?php
13    $texte ="Bonjour";
14    echo $texte;
15    echo "<br/> et comme titre ";
16    $titre = "monTitre";
17      echo $titre." ";
18    echo "<a href= \"";
19    echo './ex08-passages-parametres2.php?texte='
20         .$texte
21         ."&titre="
22         .$titre
23         .'">cliquez ici</a>';
24 ?>.
25 </p>
26 <?php
27    outputFinFichierHTML5();
28 ?>
```

Le second script peut alors récupérer les paramètres `texte` et `titre` dans un tableau associatif `$GET`. On peut vérifier que les variables `texte` et `titre` ont bien été utilisées via la fonction `isset`.

FIGURE 1.7 : Illustration du code source 1.10

Code Source 1.10 : /php1/ex08-passages-parametres2.php (cf. Fig 1.7)

```
1 <?php require_once './commonFunctions.php'; ?>
2 <?php
3    $titre = 'Mon titre par défaut';
4    if (isset($_GET['titre'])){
```

```
5       $titre = $_GET['titre'];
6     }
7     outputEnTeteHTML5($titre, 'UTF-8', 'myStyle.css');
8   ?>
9   <p>
10  <?php // début du script PHP
11    if (isset($_GET['texte'])){
12      echo $_GET['texte'];
13    }else{
14      echo "Hello World !"; // On affiche du code HTML si la sortie
15    }
16    // fin du script PHP
17  ?>
18  </p>
19  <?php
20    outputFinFichierHTML5();
21  ?>
```

Certains navigateurs ne supportant pas les URL avec des caractères comme des accents ou autres caractères UTF-8 quelconques (notamment le & !!!), si on veut passer une chaîne un peu générale en paramètre, on la codera en une string simple via la fonction `htmlentities`.

Dans l'exemple suivant, l'URL du second script est :

```
http://www.remysprogwebtuto.org/exemples/php1/ex10-passages-parametres4.php?\
        texte=L%27%C3%A9t%C3%A9%20va%20%C3%AAtre%20chaud%20cette%20ann%C3%A9e\
        &titre=Passage%20de%20param%C3%A8tres%20avec%20accents%20et%20espaces
```

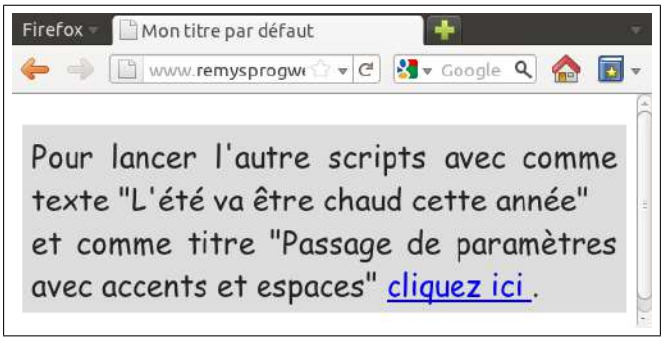


FIGURE 1.8 : Illustration du code source 1.11

Code Source 1.11 : /php1/ex09-passages-parametres3.php (cf. Fig 1.8)

```
1   <?php require_once './commonFunctions.php'; ?>
2   <?php
3     $titre = 'Mon titre par défaut';
4     if (isset($_GET['titre'])){
5       $titre = $_GET['titre'];
6     }
7     outputEnTeteHTML5($titre, 'UTF-8', 'myStyle.css');
8   ?>
9   <p>
10  Pour lancer l'autre scripts avec comme texte
11  <?php
12    $texte ="L'été va être chaud cette année";
```

```
13    echo '"'.$texte.'"';
14    echo "<br/> et comme titre ";
15    $titre = "Passage de paramètres avec accents et espaces";
16    echo '"'.$titre.'" ';
17    echo "\n<a href= \"";
18    echo './ex10-passages-parametres4.php?texte='
19         .htmlentities($texte, ENT_COMPAT, "UTF-8")
20         ."&titre="
21         .htmlentities($titre, ENT_COMPAT, "UTF-8")
22         ."\">\n\tcliquez ici\n</a>";
23  ?>.
24  </p>
25  <?php
26    outputFinFichierHTML5();
27  ?>
```

FIGURE 1.9 : Illustration du code source 1.12

Code Source 1.12 : /php1/ex10-passages-parametres4.php (cf. Fig 1.9)

```
1  <?php require_once './commonFunctions.php'; ?>
2  <?php
3    $titre = 'Mon titre par défaut';
4    if (isset($_GET['titre'])){
5      $titre = html_entity_decode($_GET['titre']);
6    }
7    outputEnTeteHTML5($titre, 'UTF-8', 'myStyle.css');
8  ?>
9  <p>
10 <?php // début du script PHP
11   if (isset($_GET['texte'])){
12     echo html_entity_decode($_GET['texte']);
13   }else{
14     echo "Hello World !"; // On affiche du code HTML si la sortie
15   }
16   // fin du script PHP
17 ?>
18 </p>
19 <?php
20   outputFinFichierHTML5();
21 ?>
```

1.9 Variables Locales ou Globales, Références

FIGURE 1.10 : Illustration du code source 1.13

Code Source 1.13 : /php1/ex11-porteeVariables.php (cf. Fig 1.10)

```
1  <?php require_once './commonFunctions.php';
2
3  outputEnTeteHTML5("Portée des Variables", 'UTF-8', 'myStyle.css');
4  ?>
5  <h1>Variables locales et globales</h1>
6  <?php
7    // Déclaration d'une variable globale
8    $a = "Contenu initial de la variable globale";
9
10   // Fonction avec une variable locale "homonyme"
11   function myFunctionWithLocalVariable(){
12     $a = "Contenu de la variable affecté dans la fonction"; // variable locale
           $a
13   }
14
15   // Fonction accédant à une variable globale.
16   function myFunctionWithGlobalVariableAccess(){
17     global $a;  // accès à la variable globale $a
18     $a = "Contenu de la variable affecté dans la fonction";
19   }
20
21   myFunctionWithLocalVariable();
22   echo "Contenu de la variable <code>a</code> après la fonction <code>
         myFunctionWithLocalVariable</code> :<br/>".$a."<br/>";
23   myFunctionWithGlobalVariableAccess();
24   echo "Contenu de la variable <code>a</code> après la fonction <code>
         myFunctionWithGlobalVariableAccess</code> :<br/>".$a."<br/>";
25
26   outputFinFichierHTML5();
27 ?>
```

Code Source 1.14 : /php1/ex12-passageParReference.php (cf. Fig 1.11)

```
1  <?php require_once './commonFunctions.php';
2
3  outputEnTeteHTML5("Portée des Variables", 'UTF-8', 'myStyle.css');
4  ?>
5  <h1>Passage par Référence et par Valeur</h1>
```

FIGURE 1.11 : Illustration du code source 1.14

```
6  <?php
7    // Déclaration d'une variable globale
8    $a = "Contenu initial de la variable globale";
9
10   // Fonction avec passage par valeur (paramètre "homonyme")
11   function myFunctionWithLocalVariable($myParam){
12     $myParam = "Contenu de la variable affacté dans la fonction";
13   }
14
15   // Fonction avec passage par référence (paramètre modifiable)
16   function myFunctionWithGlobalVariableAccess(&$myParam){
17     $myParam = "Contenu de la variable affecté dans la fonction";
18   }
19
20   myFunctionWithLocalVariable($a);
21   echo "Contenu de la variable <code>a</code> après la fonction "
22        ."<code>myFunctionWithLocalVariable</code> :<br/>".$a."<br/>";
23   myFunctionWithGlobalVariableAccess($a);
24   echo "Contenu de la variable <code>a</code> après la fonction "
25        ."<code>myFunctionWithGlobalVariableAccess</code> :<br/>".$a."<br/>";
26
27   outputFinFichierHTML5();
28 ?>
```

Chapitre 2

Les classes en *PHP*

2.1 Conception Objet, Modularité et Interopérabilité

Les exemples de classes présentées dans ce chapitre visent à expliquer les mécanismes de base de la programmation objet en *PHP*. Ils ne sont pas forcément réalistes ou adaptés pour l'implémentation d'une application *Web* bien organisée. Nous invitons pour cela le lecteur à consulter les exemples présentés à partir du chapitre 4, partie 5.2, dans lequel nous présentons le *Design Pattern POPO*

php

2.1.1 Notion de Programmation Objet

La programmation objet permet, en développant une bonne fois pour toutes un ensemble de classes appelé *framework*, de simplifier grandement le travail de développement et de maintenance de logiciels complexes, de manière que ces logiciels soient facilement adaptables. Ainsi, une entreprise telle qu'une société de services, d'un client à l'autre, reprendra tel quel une grande partie de son code, sans même le retoucher. Ce code doit avoir une *interface de développement*, c'est à dire qu'il doit mettre à disposition des développeurs un ensemble de *méthodes* qui permettent de réaliser toutes les tâches de base dont le programmeur peut avoir besoin pour développer chaque application particulière.

Les caractéristiques d'un *framework* doivent être :

1. Robustesse : les classes de base du *framework* doivent être testées et doivent être conçus pour réduire le risque de bugs lorsqu'un développer utilise les classes du *framework*, ou d'attaques lorsqu'un utilisateur malveillant utilise un site construit à partir du *framework*.

2. Généricité et versatilité : Le code doit pouvoir s'adapter, sans le retoucher, au plus grand nombre d'applications possibles.

3. Facilité de maintenance du *framework* lui-même, avec une modularité interne. Les grand outils (librairies, *frameworks* externes, etc.) utilisés par le *framework* doivent etre circonscrits à des sous-modules avec des *wrappers* ou *helpers* de manière à pouvoir changer l'un de ces outils sans revoir l'ensemble du code.

4. Une bonne lisibilité et une bonne documentation, notamment parce que les développeurs qui utilisent le *framework* ne sont pas nécessairement les mêmes que les développeurs du *framework* lui-même.

Par rapport à ces quatre objectifs, des outils sont à disposition des développeurs du *framework* :

1. Robustesse : la visibilité des variables et des méthodes (variables et méthodes privées, protected ou publiques) permet au développeur du *framework* de garantir que l'utilisateur du *framework* n'ira pas faire des bêtises en rentrant dans le code du *framework*, ce qui pourrait amener les instances de classes du *framework* dans un état incohérent. Des techniques de *tests unitaires* permettent de valider systématiquement les méthodes des classes, pour bâtir sur du solide.

2. Généricité : les patrons de conception (ou *design patterns*) permettent de développer des interfaces pour le *framework* qui rendent les code similaire d'une application à l'autre, suivant des principes d'organisation éprouvés, et qui permet de séparer différents aspects du développement d'une application.

3. Facilité de maintenance du *framework* : la conception *UML* permet d'avoir une vision schématique du *framework*, qui peut souvent contenir des centaines de classes. Chaque classe est si possible très simple et la complexité se situe dans la communication entre classes. La réécriture ou la modification d'une classe demande alors une intervention limitée, et ne doit pas affecter les autres classes, pourvu que l'interface entre les classes reste la même.

4. Lisibilité : une forme stéréotypée pour l'interface des classes, les identificateurs, etc. rend le code plus lisible. De plus, des outils permettent de générer automatiquement une documentation (*HTML*, *LATEX*, *PDF*, etc.) des classes à partir de commentaires dans le code. C'est le cas par exemple de *Doxygen* pour le *PHP*.

Enfin, la conception objet permet de concevoir la structure (ou l'*architecture*) du logiciel indépendamment du langage de programmation, par représentation en *Unified Modeling Language* (*UML*). Nous utiliserons dans ce cours des *diagrammes de classes*, des *diagrammes de séquence*, et des *diagrammes de cas d'utilisation*.

2.1.2 Standard de Codage pour l'Interopérabilité (*PSR*)

S'agissant du code source *PHP*, des standards concernant l'organisation du code ont été définis, qui visent à garantir l'interopérabilité des *frameworks* et de leurs *plugins* et, en général, des applications écrites en *PHP*.

L'organisme *PHP-FIG* (*Framework Interoperability Group*) définit de tels standards, appelés *PSR*, pour *PHP Standard Recommendations*. L'un des objectifs de ce cours est de présenter, dans la partie IV, les principes d'organisation d'une application suivant les recommandations du standards *PSR-1 : Basic Coding Standard*.

Ce standard impose de suivre une organisation d'auto-chargement des classes qui repose sur une organisation où les répertoires contenant du code source correspondent à des *namespaces PHP*, ou autrement dit, des modules, qui représentent des *packages* au niveau de la conception

et représentation *UML* du logiciel. Pour cette raison, nous présentons dès les premiers chapitres une conception objet qui inclut un découpage en modules explicité par des *namespaces*.

Disons enfin que l'organisation des modules suit elle-même certains *Design Patterns*, telle que l'architecture trois tiers *MVC* (voir le chapitre 12) ou la couche d'accès aux données *DAL* (voir le chapitre 9). Ces patrons de conception visent à garantir la modularité par le *découplage* des différentes parties d'une application, permettant de faciliter les évolutions (par exemple un changement de technologie pour l'interface homme-machine *IHM*), du fait de l'indépendance logique des parties.

2.2 Exemples de classes PHP

2.2.1 Classes de Base

Un classe doit permettre de manipuler un certain type d'objets. La classe doit permettre de représenter les caractéristiques des objets, à travers un certain nombre d'attributs, qui sont les variables communes à chacun des objets de ce type. La classe doit aussi permettre à un développeur qui l'utilise de réaliser toutes les opération nécessaires sur ces objets, à traves des méthodes. Les méthodes d'une classe sont les fonctions qui opèrent en interne sur la classe. La manipulation des attributs se fait presque systématiquement à travers des méthodes, cet qui évite que l'utilisateur de la classe ne mette les attributs dans un état incohérent (exemple : variables `NULL` alors qu'elle n'est pas censée l'être, ce qui génère un bug). Pour celà, on met les attributs privés, c'est à dire que seules les méthodes de la classe peuvent accéder à ces attributs. Pour les autres classes, ces attributs ne sont pas visibles : elle ne peuvent pas y accéder directement mais uniquement à travers des méthodes.

Voici un premier exemple d'une classe appelée `VueHtmlUtils` qui définit deux méthodes statiques générant respectivement l'en-tête d'une fichier *HTML5* et la fin d'un fichier *HTML* (fermeture des balises). Cette classe utilitaire sera utilisée dans la génération du code *HTML* dans les *vues*.

Code Source 2.1 : /php2/classes/VueHtmlUtils.php

```
1  <?php
2  namespace CoursPHP\Vue;
3  /** @brief Utilitaire de génération de code HTML
4   * Définit des méthodes de génération de Header HTML et de fin de fichier */
5  class VueHtmlUtils {
6    /** Génère le code d'un header HTML5 à partir du titre de la page,
7     * du charset, et de la feuille de style CSS */
8    public static function enTeteHTML5($title, $charset, $css_sheet){
9      // sortie du doctype. Les guillemets HTML sont protégés par \
10     $htmlCode = "<!doctype html>\n<html lang=\"fr\">\n<head>\n";
11     $htmlCode .= "<meta charset=\"".$charset."\"/>\n";
12     $htmlCode .= "<link rel=\"stylesheet\" href=\"".$css_sheet."\" />\n";
13     $htmlCode .= "<title>".$title."</title>\n";
14     $htmlCode .= "</head>\n<body>\n";
15     return $htmlCode;
16   }
17
18   /** Genère le code HTML de cloture du BODY et du code HTML */
19   public static function finFichierHTML5(){
20     return "</body>\n</html>\n";
```

```
21    }
22  }
23  ?>
```

Voici maintenant un exemple avec une classe contenant le numéro de téléphone d'une personne. Les commentaires ont une forme spéciale pour pouvoir générer la documentation du code avec l'outil `Doxygen`. Les attributs sont privés et sont toujours initialisés via les *setters*, qui sont des méthodes spacialement conçues qui testent les condition que doivent satisfaire les attributs (ici être non `null`) avant des les initialiser. Le constructeur utilise les setters ce qui a l'avantage de *factoriser* le code, c'est à dire que les tests sur les valeurs des attributs ne sont réalisés qu'une seule fois.

Code Source 2.2 : /php2/classes/Telephone.php

```
1  <?php
2  namespace CoursPHP\Metier;
3
4  class Telephone {
5    /** Numéro de téléphone, ne doit pas être null mais peut être vide */
6    private $numero;
7
8    /** Libellé du nuéro de téléphone (domicile, travail, mobile, etc).
9     * Ne doit pas être null mais peut être vide */
10   private $libelle;
11
12   /** @brief Accesseur : permet d'obtenir le numéro de téléphone. */
13   public function getNumero(){
14     return $this->numero;
15   }
16
17   /** @brief Accesseur : permet d'obtenir le libellé du téléphone */
18   public function getLibelle(){
19     return $this->libelle;
20   }
21
22  /** @brief Setter : Initialiser ou de modifie le numéro de téléphone
23    * @param $numero le numéro de téléphone à utiliser. peut être null. */
24   public function setNumero($numero){
25   if (empty($numero))
26     $this->numero = "";
27   else
28     $this->numero = $numero;
29   }
30
31  /** @brief Setter : Initialiser ou de modifie le libellé de téléphone
32    * @param $numero le libellé de téléphone à utiliser. peut être null. */
33   public function setLibelle($libelle){
34   if (empty($libelle))
35     $this->libelle = "";
36   else
37     $this->libelle = $libelle;
38   }
39
40  /** @brief Constructeur : Construire et initialiser un Objet Telephone
41    * Appelle systématiquement les setters. */
42   public function __construct($libelle, $numero){
```

```
43      $this->setLibelle($libelle);
44      $this->setNumero($numero);
45    }
46
47   /** @brief Méthode de génération d'HTML. Permet d'afficher un téléphone.
48     * Les attributs doivent être non null. (mais normalement ça ne risque pas
49     * d'arriver car les attributs sont privés donc l'utilisateur de la classe
50     * n'a pas pu les mettre à null. Les setters et le constructeur est ainsi
51     * conçu que les attributs ne peuvent pas être null.)
52     * @return le code HTML du téléphone */
53    public function toHTML(){
54      return $this->libelle." : ".$this->numero;
55    }
56  }
57  ?>
```

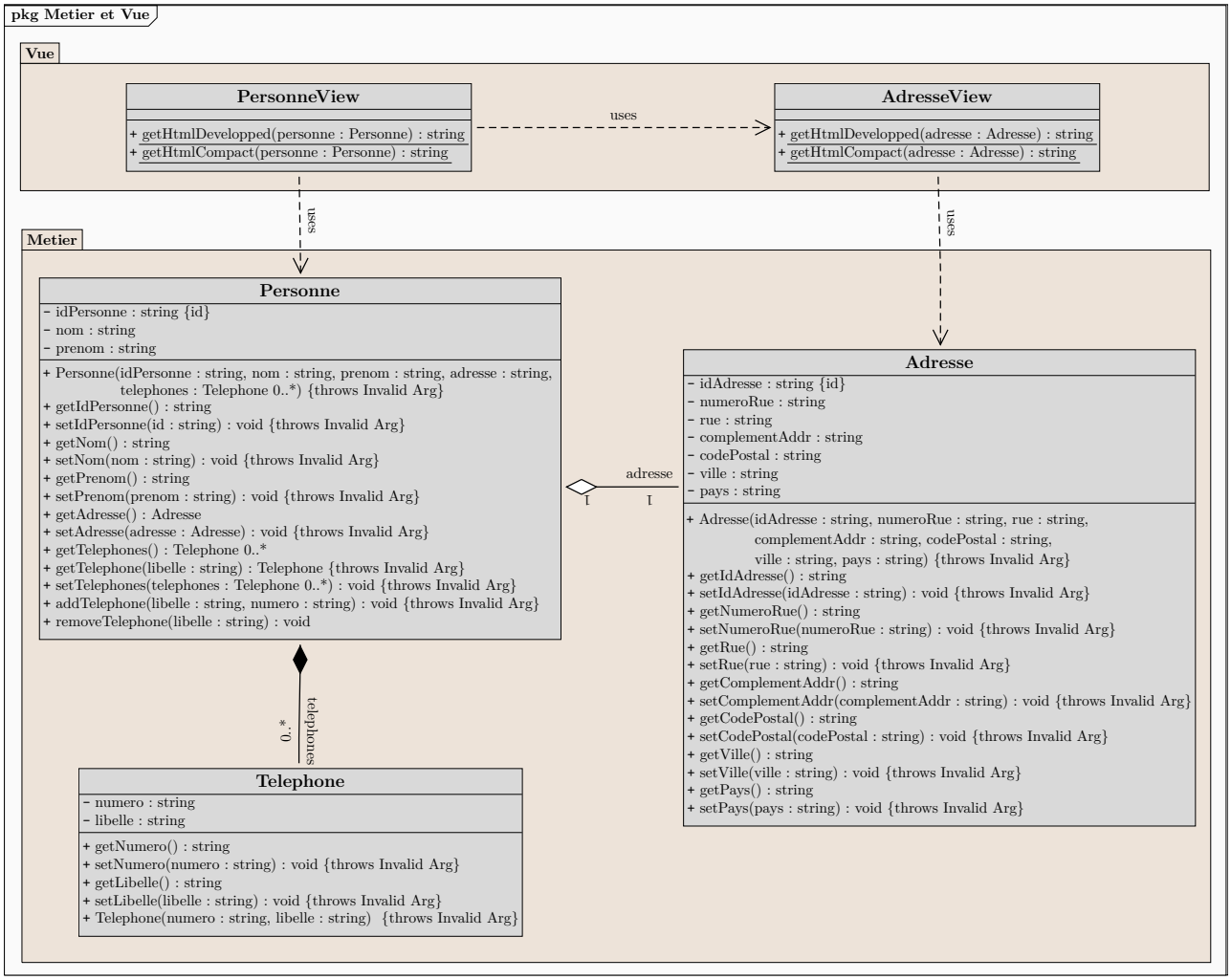
Comme on le voit, la classe `Telephone` fait partie d'un sous-*namespace* `People\Contact` du *namespace* `People`. Les *namespace* sou un bon mayen en *PHP* de réaliser un *package*, au sens de la conception objet.

2.2.2 Structuration des Objets, Vues

Nous allons maintenant voir une classe un peu plus complexe, au moins en ce sens qu'elle possède plus d'attributs. Nous allons voir comment nous pouvons, dès ce stade de la conception, respecter un certain nombre de bonnes pratiques, à la fois dans l'organisation du code et pour sa division en fichiers, mais aussi, au niveau de la *Conception Objet* dans la séparation du modèle de données (objets*Métier*) et de la mise en forme de ces données pour l'affichage vers l'utilisateur (vues).

La classe `Adresse` représentera le modèle de données pour une adresse postale et la classe `AdresseView` implémentera (en l'occurrence) deux vues *HTML* d'une `Adresse`, l'une développée et l'autre compacte. Comme toujours, notre modélisation n'est pas cannonique et plusieurs choix seraient possibles.

Par ailleurs, pour limiter la longueur des fichiers sources, nous utilisons un `trait`. Un `trait` permet de regrouper dans un fichiers séparé un ensemble de méthodes qui font partie d'une classe. Un trait peut meme définir une parte de plusieurs classes, mais les méthodes de ces classes doivent avoir exactement le meme code. (c'est une manière un peu "bricole" de faire de la programmation générique en *PHP*. Dans notre exemple, le `trait AdresseProperties` contient tous les *getters* et *setters* de la classe `Adresse`. Le `trait` et ses méthodes sont insérés dans la classe `Adresse` avec le mot clé `use`.



Diag 1. Diagramme de Classes des Package `Metier` et `Vue`

Nous développons maintenant le code *PHP* de la classe `Adresse`.

Code Source 2.3 : /php2/classes/Adresse.php

```php
<?php
namespace CoursPHP\Metier;

require_once(dirname(__FILE__).'/AdressePropertiesTrait.php');

/** @brief La classe adresse contient l'adresse d'une personne
  (qui peut être un client, un employé, un fournisseur, etc...) */
class Adresse {
  /** Identifiant unique de l'adresse */
  private $idAdresse;
  /** Numéro dans la rue, ne doit pas être null mais peut être vide */
  private $numeroRue;
  /** Nom de la rue, ne doit pas être null mais peut être vide */
  private $rue;
  /** Complément (lieu dit, etc. ne doit pas être null mais peut être vide */
  private $complementAddr;
  /** code postal */
  private $codePostal;
  /** nom de la ville. ne doit pas être null mais peut être vide */
```

```
20    private $ville;
21    /** nom du pays. ne doit pas être null mais peut être vide */
22    private $pays;
23
24    // Inclusion du trait AdresseProperties définissant les accesseurs et setters
25    use AdresseProperties;
26
27   /** @brief Constructeur : initialise les attributs à partir des paramètres.
28     * Les paramètres correspondent aux valeurs à mettre dans les attributs.
29     * Tout objet doit être initialisé avec le constructeur (appel à new).
30     * Ici, les paramètres peuvent être null. Les attributs sont alors initialisés
31     * à une chaîne vide, permettant la cohérence de la classe. */
32    public function __construct($idAdresse, $numeroRue, $rue, $complementAddr,
33                  $codePostal, $ville, $pays) {
34      $this->setIdAdresse($idAdresse);
35      $this->setNumeroRue($numeroRue);
36      $this->setRue($rue);
37      $this->setComplementAddr($complementAddr);
38      $this->setCodePostal($codePostal);
39      $this->setVille($ville);
40      $this->setPays($pays);
41    }
42  } // end of class Adresse
43  ?>
```

Voici maintenant le code *PHP* du `trait AdresseProperties`.

Code Source 2.4 : /php2/classes/AdressePropertiesTrait.php

```
1   <?php
2   namespace CoursPHP\Metier;
3
4   /** @brief La classe adresse contient l'adresse d'une personne
5     *  (qui peut être un client, un employé, un fournisseur, etc...) */
6   trait AdresseProperties {
7
8     /** @brief Accesseur : permet d'obtenir l'identifiant de l'instance. */
9     public function getIdAdresse() {
10      return $this->idAdresse;
11    }
12
13    /** @brief Accesseur : permet d'obtenir le numéro dans la rue. */
14    public function getNumeroRue() {
15      return $this->numeroRue;
16    }
17
18    /** @brief Accesseur : permet d'obtenir le nom la rue. */
19    public function getRue() {
20      return $this->rue;
21    }
22
23    /** @brief Accesseur : permet d'obtenir le nom le complément d'adresse. */
24    public function getComplementAddr() {
25      return $this->complementAddr;
26    }
27
28    /** @brief Accesseur : permet d'obtenir le nom le code postal. */
```

```
29    public function getCodePostal() {
30      return $this->codePostal;
31    }
32
33    /** @brief Accesseur : permet d'obtenir le nom la ville. */
34    public function getVille() {
35      return $this->ville;
36    }
37
38    /** @brief Accesseur : permet d'obtenir le pays. */
39    public function getPays() {
40      return $this->pays;
41    }
42
43    /** @brief setter : permet d'initialiser ou de modifier le nom de la rue.
44      * @param $NumeroRue le numéro à utiliser. peut être null. */
45    public function setIdAdresse($idAdresse) {
46      $this->idAdresse = empty($idAdresse) ? "" : $idAdresse;
47    }
48
49    /** @brief setter : permet d'initialiser ou de modifier le nom de la rue.
50      * @param $NumeroRue le numéro à utiliser. peut être null. */
51    public function setNumeroRue($numeroRue) {
52      $this->numeroRue = ($numeroRue == null) ? "" : $numeroRue;
53    }
54
55    /** @brief setter : permet d'initialiser ou de modifier le numéro dans la rue.
56      * @param $Rue le nom de la rue ou de la place à utiliser. peut être null. */
57    public function setRue($rue) {
58      $this->rue = ($rue == null) ? "" : $rue;
59    }
60
61    /** @brief setter : permet d'initialiser/modifier le complément d'adresse.
62      * @param $ComplementAddr le complément d'adresse à utiliser. */
63    public function setComplementAddr($complementAddr) {
64      $this->complementAddr = ($complementAddr == null) ? "" : $complementAddr;
65    }
66
67    /** @brief setter : permet d'initialiser ou de modifier le code postal.
68      * @param $CodePostal le numéro à utiliser. peut être null */
69    public function setCodePostal($codePostal) {
70      $this->codePostal = ($codePostal == null) ? "" : $codePostal;
71    }
72
73    /** @brief setter : permet d'initialiser ou de modifier le nom de la ville.
74      * @param $Ville le nom de la ville à utiliser. peut être null */
75    public function setVille($ville) {
76      $this->ville = ($ville == null) ? "" : $ville;
77    }
78
79    /** @brief setter : permet d'initialiser ou de modifier le nom du Pays
80      * @param $pays le nom du Pays à utiliser. peut être null */
81    public function setPays($pays) {
82      $this->pays = ($pays == null) ? "" : $pays;
83    }
84  }
```

```
85 ?>
```

Voici maintenant le code *PHP* de la classe `AdresseView`.

Code Source 2.5 : /php2/classes/AdresseView.php

```
1  <?php
2  namespace CoursPHP\Vue;
3  /** @brief La classe AdresseView implémente la génération d'HTML pour afficher
4    * une adresse dans une vue dans un navigareur.
5    * Implémente aussi des utilistaires de conversion à partir d'une Adresse
6    * pour obtenir facilement le code HTML pour afficher une Adresse. */
7  class AdresseView {
8    /** @brief Méthode de génération de code HTML. Permet d'afficher une adresse.
9     * Les attributs doivent être non null.
10    * Normalement ça ne risque pas d'arriver car les attributs sont privés
11    * donc l'utilisateur de la classe n'a pas pu les mettre à null.
12    * Les setters et constructeur est ainsi conçu que les attributs
13    * ne peuvent pas être null.) */
14   public static function getHtmlDevelopped($adresse){
15     $htmlCode = "";
16     $htmlCode .= "<strong>Adresse : </strong><br/>\n";
17     $htmlCode .= $adresse->getNumeroRue();
18     if (!empty($adresse->getNumeroRue()))
19       $htmlCode .= ", ";
20     $htmlCode .= $adresse->getRue();
21     if (!empty($adresse->getRue()))
22       $htmlCode .= "<br/>";
23     $htmlCode .= $adresse->getComplementAddr();
24     if (!empty($adresse->getComplementAddr()))
25       $htmlCode .= "<br/>";
26     $htmlCode .= $adresse->getCodePostal()." ";
27     $htmlCode .=  $adresse->getVille();
28     if (!empty($adresse->getVille()))
29       $htmlCode .= "<br/>";
30     $htmlCode .= $adresse->getPays()."<br/>";
31
32     return $htmlCode;
33   }
34
35   /** @brief Méthode de génération d'HTML. Permet d'afficher une adresse.
36    *  Les attributs doivent être non null.
37    *  car les attributs sont privés, donc l'utilisateur de la classe n'a pas pu
38    *  les mettre à null. Les setters et le constructeur est ainsi conçu que les
39    * attributs ne peuvent pas être incohérents
40    *  La méthode retourne le code HTML pour un affichage compact sur 1 ligne */
41   public static function getHtmlCompact($adresse){
42     $htmlCode = "";
43     $htmlCode .= $adresse->getNumeroRue();
44     if (!empty($adresse->getNumeroRue()))
45       $htmlCode .= ", ";
46     $htmlCode .= $adresse->getRue();
47     if (!empty($adresse->getRue()))
48       $htmlCode .= ", ";
49     $htmlCode .= $adresse->getComplementAddr();
50     if (!empty($adresse->getComplementAddr()))
51       $htmlCode .= ", ";
```

```
52      $htmlCode .= $adresse->getCodePostal() . " ";
53      $htmlCode .=  $adresse->getVille();
54      if (!empty($adresse->getVille()))
55        $htmlCode .= ", ";
56      $htmlCode .= $adresse->getPays();
57
58      return $htmlCode;
59    }
60  } // end of class AdresseView
61  ?>
```

2.2.3 Utilisation des Classes et Vue *HTML*

Voyons maintenant un petit script de test qui crée des adresses et les affiche en générant une vue *HTML*. Seul le script de test génère du code *HTML* et comporte un en-tête HTML (même si ce code *HTML* est en fait généré dans une méthode statique de la classe `AdresseView`).

De plus, on préfèrera une structure dans laquelle lé génration du code *HTML* se trouve dans une scirpt séparé, appelé *vue*. Pour celà, le script de test prépar les données et les mémorise dans des instances de classes (adresses, téléphones, etc.), puis appelle la vue par un `require`. Enfin, la vue accède aux variables et instances de classes prépérées par le script de test pour les afficher.

FIGURE 2.1 : Illustration du code source 2.6

Code Source 2.6 : /php2/ex05-testExampleImportNamespace.php (cf. Fig 2.1)

```
1  <?php
2    require_once(dirname(__FILE__) . '/classes/Telephone.php');
3    require_once(dirname(__FILE__) . '/classes/Adresse.php');
4
5    $telephone = new CoursPHP\Metier\Telephone("Travail", "01 23 45 67 89");
6    // Adresse Complète :
7    $adresse1 = new CoursPHP\Metier\Adresse("0af46d3bd9", '10', 'allée du net',
8              'Quartier de l\'avenir', '63000', 'Clermont-Ferrand', 'France');
9    // Adresse sans code postal ni complément d'adresse
10   $adresse2 = new CoursPHP\Metier\Adresse("2bf46d3ba32", '10', 'Downing Street',
11                                        null, null, 'London', 'United Kingdom');
```

```
12    // Appel de la vue (Génération du code HTML)
13    require('ex05-vueExampleImportNamespace.php');
14  ?>
```

Voici maintenant le code de la vue :

Code Source 2.7 : /php2/ex05-vueExampleImportNamespace.php

```
1  <?php
2    require_once(dirname(__FILE__).'/classes/VueHtmlUtils.php');
3    require_once(dirname(__FILE__).'/classes/AdresseView.php');
4
5    echo CoursPHP\Vue\VueHtmlUtils::enTeteHTML5('Ma première classe PHP',
6                                                'UTF-8', 'myStyle.css');
7
8    echo "<h1>Test de Classe</h1>";
9    echo "<p>";
10   echo "<strong>Téléphone </strong>".$telephone->toHTML()."<br/>";
11
12   echo "<strong>Adresse au format compact :</strong><br/>".
13           CoursPHP\Vue\AdresseView::getHtmlCompact($adresse1)."<br/>";
14   echo CoursPHP\Vue\AdresseView::getHtmlDevelopped($adresse2)."<br/>";
15   echo "</p>";
16   echo CoursPHP\Vue\VueHtmlUtils::finFichierHTML5();
17 ?>
```

Notons que l'on peut aussi importer une classe par la directive `use`, et pas seulement un *namespace*.

2.3 Validation en entrée et gestion d'une exception

2.3.1 Qu'est-ce que le filtrage ?

Les *setters* de la classe vont jouer un rôle important de *filtrage* des données. Le filtrage consiste à réaliser des tests sur les données entrées (généralement des données issues d'un utilisateur final), et à générer des erreurs en cas de données incorrectes, ou encore en remplaçant automatiquement des données incorrecte par des données, sinon correctes, au moins inoffensives.

En particulier, lorsque les données viendront de la saisie d'un formulaire, ces données devront être systématiquement filtrées car l'utilisateur, qui n'est pas toujours bienveillant, et peut mettre n'importe quoi dans les champs d'un formulaire. Le filtrage jouera donc un rôle très important pour la sécurité. Par exemple, on prendra soin de limiter la longueur des attributs de type `String` à la fois au niveau du filtrage, puis au niveau de la base de données (voir chapitres ultérieurs). On pourra aussi utiliser des expressions régulières lors du filtrage grâce aux fonctions `preg_match_all` ou `preg_match` (voir man regex(7) pour la formation des expressions régulières). Le gros avantage du PHP par rapport à d'autres langages comme javascript, est que PHP s'exécute côté serveur donc un pirate n'aura pas la possibilité d'analyser précisément ce que fait le filtrage.

Si une valeur invalide est détectée au niveau du filtrage, on générera une exception avec un message d'erreur. Cette exception pourra être gérée à un autre niveau dans l'application, ici au niveau du script de test qui affiche quelques employés. Certaines parties ultérieures de ce cours sont dédiées au filtrage précis des données et à garantir la sécurité du code grâce au

filtrage. Dans cette partie, nous réalisons un filtrage sommaire, pour illustrer le mecanisme de gestion des erreurs par *exceptions*.

2.3.2 Classe `Personne` avec filtrage dans les *setters*

Nous voyons ici une classe *Personne*, suivant un peu le meme schéma de conception que la classe *Adresse* de la partie précédente. Cependant, au niiveau des *setters*, nous implémenterons un filtrage (minimal et peu réaliste pour le moment), rejetant une exception en cas de données incorrectes.

Code Source 2.8 : /php2/classes/Personne.php

```php
<?php
namespace CoursPHP\Metier;
require_once(dirname(__FILE__).'/PersonnePropertiesTrait.php');
/** @brief Représente une personne (client, employé, contact...)
   Elle contient l'identité (nom prénom), l'adresse, le noméro de téléphone
   et le salaire mensuel de l'employé. */
class Personne {
  /** Identifiant unique de la personne */
  protected $idPersonne;
  /** nom de l'employé : obligatoire. Le nom de l'employé ne peut être vide. */
  protected $nom;
  /** prénom de l'employé */
  protected $prenom;
  /** adresse de l'employé (instance d'Adresse) */
  protected $adresse;
  /** Tableau des numéros de téléphone */
  protected $telephones;

  // Inclusion du trait avec les accesseurs/setters des propriétés
  use PersonneProperties;

 /** @brief Constructeur : initialise les attributs à partir des paramètres.
   * Les paramètres correspondent aux valeurs à mettre dans les attributs.
   * Tout objet doit être initialisé avec le constructeur (appel à new).
   * Des exceptions sont rejetées en cas de paramètres invalide. */
  public function __construct($idPersonne, $nom, $prenom, $adresse, $telephones)
      {
       $this->setIdPersonne($idPersonne);
       $this->setNom($nom);
       $this->setPrenom($prenom);
       $this->setAdresse($adresse);
       $this->setTelephones($telephones);
  }
}
?>
```

Code Source 2.9 : /php2/classes/PersonnePropertiesTrait.php

```php
<?php
namespace CoursPHP\Metier;

trait PersonneProperties {
  /** @brief accesseur : permet d'obtenir le nom de l'employé */
  public function getidPersonne() {
```

```
7       return $this->idPersonne;
8     }
9
10    /** @brief accesseur : permet d'obtenir le nom de l'employé */
11    public function getNom() {
12      return $this->nom;
13    }
14
15    /** @brief accesseur : permet d'obtenir le prénom de l'employé */
16    public function getPrenom() {
17      return $this->prenom;
18    }
19
20    /** @brief accesseur : permet d'obtenir l'adresse de l'employé */
21    public function getAdresse() {
22      return $this->adresse;
23    }
24
25    /** @brief accesseur : permet d'obtenir le tableau des téléphones */
26    public function getTelephones() {
27      return $this->telephones;
28    }
29
30    /** @brief accesseur : permet d'obtenir un numéro de téléphone de l'employé
31     * @param libelle Le libellé du numéro souhaité */
32    public function getTelephone($libelle) {
33      if (empty($this->telephones[$libelle])){
34        throw new \Exception('Désolé, Le téléphone "'.$libelle.'" n\'existe pas.
              Have a try in the phonebook...');
35      }
36      return $this->telephones[$libelle];
37    }
38
39    /** setter : permet d'initialiser ou de modifier l'identifiant de la personne
40      * @param $idPersonne l'identifiant de la personne. Doit être non vide */
41    public function setIdPersonne($idPersonne) {
42    if (empty($idPersonne) || strlen($idPersonne) != 10){
43        throw new \Exception('Désolé, toute personne doit avoir un identifiant de
              10 caractères !');
44      }else{
45        $this->idPersonne= $idPersonne;
46      }
47    }
48
49    /** setter : permet d'initialiser ou de modifier le nom de la personne
50      * @param $Nom le nom de la personne. Doit comporter au moins 1 caractère */
51    public function setNom($nom) {
52    if (empty($nom) || strlen($nom) > 100){
53        throw new \Exception('Désolé, toute personne doit avoir un nom et le nom a
              au plus 100 caractères !');
54      }else{
55        $this->nom = $nom;
56      }
57    }
58
59    /** setter : permet d'initialiser ou de modifier le nom de la personne */
```

```
60    public function setPrenom($prenom) {
61      if (empty($prenom) || strlen($prenom) > 50){
62        throw new \Exception('Désolé, toute personne doit avoir un prenom et le
              prenom a au plus 50 caractères !');
63      }else{
64        $this->prenom = $prenom;
65      }
66    }
67
68    /** setter : permet d'initialiser ou de modifier l'adresse de la personne */
69    public function setAdresse($adresse) {
70      if ($adresse == null || get_class($adresse) != 'CoursPHP\Metier\Adresse'){
71        throw new \Exception('Erreur : Adresse Invalide');
72      }else{
73        $this->adresse = $adresse;
74      }
75    }
76
77    /** setter : permet d'initialiser ou de modifier l'adresse de la personne */
78    public function setTelephones($telephones) {
79      if (!is_array($telephones)){
80        throw new \Exception('Erreur : Téléphones Invalide');
81      }else{
82        $this->telephones = $telephones;
83      }
84    }
85
86    /** setter : permet d'ajouter un numéro de téléphone de la personne */
87    public function addTelephone($libelle, $numero) {
88      if (!empty($numero) && strlen($numero) <= 15) {
89        if (!is_array($this->telephones)){
90          $this->telephones = array();
91        }
92        $this->telephones[$libelle] = new Telephone($libelle, $numero);
93      }else{
94        throw new \Exception('Erreur : Téléphone Invalide');
95      }
96    }
97
98    /** setter : permet d'ajouter un numéro de téléphone de la personne */
99    public function removeTelephone($libelle) {
100     if (!empty($this->telephone[$libelle])){
101       unset($this->telephone[$libelle]);
102     }
103   }
104 }
105 ?>
```

Code Source 2.10 : /php2/classes/PersonneView.php

```
1  <?php
2  namespace CoursPHP\Vue;
3  /** Utilitaire de génération de code HTML pour la mise en forme
4   * des attributs d'une Personne */
5  class PersonneView {
6    /** @brief Méthode de génération de code HTML. Permet d'afficher une personne.
```

```
7     * Les attributs doivent être non null.
8     * (mais normalement ça ne risque pas d'arriver car les attributs sont privés
9     * donc l'utilisateur de la classe n'a pas pu les mettre à null.
10    * Les setters et le constructeur est ainsi conçu que les attributs
11    * ne peuvent pas être null.) */
12    public static function getHtmlDevelopped($personne){
13      $htmlCode = "";
14      $htmlCode .= "nom : ".$personne->getNom()."<br/>\n";
15      if (strlen($personne->getPrenom())>=1)
16        $htmlCode .= "Prénom : ".$personne->getPrenom()."<br/>\n";
17      $htmlCode .= AdresseView::getHtmlDevelopped($personne->getAdresse());
18      $count = 0;
19      foreach ($personne->getTelephones() as $telephone) {
20        if ($count != 0){
21          $htmlCode .= "<br/>";
22        }
23        $count++;
24        $htmlCode .= $telephone->toHTML();
25      }
26      $htmlCode .= "<br/>\n";
27      return $htmlCode;
28    }
29
30    /** @brief Méthode de génération d'une ligne de tableHTML.
31     * Permet d'afficher des Personnes  dans une table HTML. */
32    public static function getHtmlTableRow($personne){
33      $htmlCode = "<tr>";
34      $htmlCode .= "<td>".$personne->getNom()."</td>";
35      $htmlCode .= "<td>".$personne->getPrenom()."</td>";
36      $htmlCode .= "<td>".AdresseView::getHtmlCompact($personne->getAdresse())."</
            td>";
37      $htmlCode .= "<td>";
38      $count = 0;
39      foreach ($personne->getTelephones() as $telephone) {
40        if ($count != 0){
41          $htmlCode .= "<br/>";
42        }
43        $count++;
44        $htmlCode .= $telephone->toHTML();
45      }
46      $htmlCode .= "</td>";
47      $htmlCode .= "</tr>";
48
49      return $htmlCode;
50    }
51
52    /** Permet d'obtenir une ligne de table HTML avec les en-têtes de colonnes
53     * pour affichage d'une table de Personnes. */
54    public static function getHtmlTableHeads(){
55      $htmlCode = "<tr>";
56      $htmlCode .= "<th>Nom</th>";
57      $htmlCode .= "<th>Prénom</th>";
58      $htmlCode .= "<th>Adresse</th>";
59      $htmlCode .= "<th>Téléphone(s)</th>";
60      $htmlCode .= "</tr>";
61      return $htmlCode;
```

```
62     }
63 } // end of class PersonneView
64 ?>
```

2.3.3 Test de construction de Personnes et récupération des exceptions

Voyons tout d'abord la construction normale et l'affichage d'une personne.

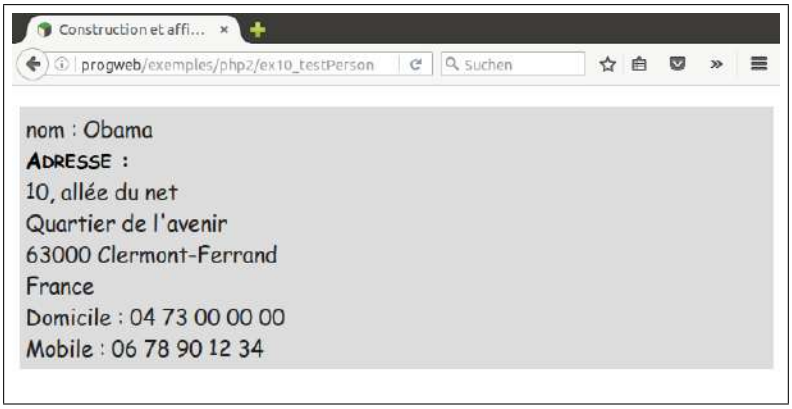


FIGURE 2.2 : Illustration du code source 2.11

Code Source 2.11 : /php2/ex10-testPersonnes.php (cf. Fig 2.2)

```
1  <?php
2    require_once(dirname(__FILE__).'/classes/Telephone.php');
3    require_once(dirname(__FILE__).'/classes/Adresse.php');
4    require_once(dirname(__FILE__).'/classes/Personne.php');
5
6    use CoursPHP\Metier\Adresse;
7    use CoursPHP\Metier\Telephone;
8    use CoursPHP\Metier\Personne;
9
10   try {
11     $adresse = new Adresse("0af46d3bd9", '10', 'allée du net',
12                 'Quartier de l\'avenir', '63000', 'Clermont-Ferrand', 'France');
13     $telephones = array(new Telephone("Domicile", "04 73 00 00 00"),
14                         new Telephone("Mobile", "06 78 90 12 34"));
15     $personne = new Personne("043f46d3a3", "Obama", "Barack", $adresse,
16                              $telephones);
17
18     // La personne a bien été construite, on affiche
19     require ("ex10-vueNormale.php");
20   } catch (Exception $e) {
21     // Une erreur s'est produite, on la gère
22     require ("ex10-vueErreur.php");
23   }
24 ?>
```

Code Source 2.12 : /php2/ex10-vueNormale.php

```php
<?php
    require_once(dirname(__FILE__) . '/classes/VueHtmlUtils.php');
    require_once(dirname(__FILE__) . '/classes/AdresseView.php');
    require_once(dirname(__FILE__) . '/classes/PersonneView.php');

    use \CoursPHP\Vue\PersonneView;

    echo \CoursPHP\Vue\VueHtmlUtils::enTeteHTML5(
            'Construction et affichage d\'une Personne', 'UTF-8', 'myStyle.css');
    echo "<p>";
    echo PersonneView::getHtmlDevelopped($personne);
    echo "</p>";

    echo \CoursPHP\Vue\VueHtmlUtils::finFichierHTML5();
?>
```

Code Source 2.13 : /php2/ex10-vueErreur.php

```php
<?php
    require_once('classes/VueHtmlUtils.php');

    echo \CoursPHP\Vue\VueHtmlUtils::enTeteHTML5('Gestion d\'une exception',
                                                 'UTF-8', 'myStyle.css');

    echo "<h1>Une Erreur c'est produite</h1>";
    echo "<p>Exception reçue : ".$e->getMessage()."</p>";

    echo \CoursPHP\Vue\VueHtmlUtils::finFichierHTML5();
?>
```

Voyons maintenant le test d'une vue affichant plusiers personnes dans une table *HTML*.

FIGURE 2.3 : Illustration du code source 2.14

Code Source 2.14 : /php2/ex11-testTableViewPersonnes.php (cf. Fig 2.3)

```php
<?php
    require_once(dirname(__FILE__) . '/classes/Telephone.php');
```

```
3   require_once(dirname(__FILE__).'/classes/Adresse.php');
4   require_once(dirname(__FILE__).'/classes/Personne.php');
5
6   use CoursPHP\Metier\Adresse;
7   use CoursPHP\Metier\Telephone;
8   use CoursPHP\Metier\Personne;
9
10  echo "<p>";
11  try {
12    $adresse1 = new Adresse("0af46d3bd9", '10', 'allée du net', 'Quartier de l\'
        avenir',
13                  '63000', 'Clermont-Ferrand', 'France');
14    $telephones1 = array(new Telephone("Domicile", "04 73 00 00 00"));
15    $personne1 = new Personne("043f46d3a3", "Obama", "Barack", $adresse1,
        $telephones1);
16    $adresse2 = new Adresse("0af46d3be0", '12', 'Georgy', null, '63000', 'Trench
        Town', 'Jamaica');
17    $telephones2 = array(new Telephone("Emergency", "911"));
18    $personne2 = new Personne("af3f46d27f", "Modèle", "Jean", $adresse2,
        $telephones2);
19    $adresse3 = new Adresse("0af46d3be1", '10', 'Rock\'n Roll Street', 'Bronx',
        '63000', 'Rackamadour', 'France');
20    $personne3 = new Personne("3aef46d7fe", "Génération", "iGrec", $adresse3,
21                  array(new Telephone("Travail", "01 23 45 67 89")));
22    $personnes = array(1 => $personne1, 2 => $personne2, 3 => $personne3);
23
24    require("ex11-vueNormale.php");
25  } catch (Exception $e) {
26    require("ex10-vueErreur.php");
27  }
28 ?>
```

Code Source 2.15 : /php2/ex11-vueNormale.php

```
1  <?php
2    require_once(dirname(__FILE__).'/classes/VueHtmlUtils.php');
3    require_once(dirname(__FILE__).'/classes/AdresseView.php');
4    require_once(dirname(__FILE__).'/classes/PersonneView.php');
5
6    use CoursPHP\Vue\PersonneView;
7
8    echo CoursPHP\Vue\VueHtmlUtils::enTeteHTML5('Génération de Table',
9                                                'UTF-8', 'myStyle.css');
10   echo "<p><strong>Affichage d'une table de Personnes :</strong></p>";
11   echo "<table>";
12   echo "<thead>".PersonneView::getHtmlTableHeads()."</thead>";
13   echo "<tbody>";
14   foreach ($personnes as $personne){
15     echo PersonneView::getHtmlTableRow($personne);
16   }
17   echo "</tbody>";
18   echo "</table>";
19   echo CoursPHP\Vue\VueHtmlUtils::finFichierHTML5();
20 ?>
```

Voyons enfin ce qui se passe lorsqu'une erreur dans les données déclenche une exception au

niveau du *setter*, alors que notre récupération de l'exception nous permet d'afficher un message d'erreur intelligible, évitant le *crash* complet du site.

Dans l'exemple suivant, nous créons deux personnes en récupérant, pour chacune, une exception. Nous accumulons les éventuels messages exceptions dans un tableau associatifs. Dans la vue qui suit, nous testons la présence d'erreurs dans ce tableau associatif avant d'afficher soit la personne, soit le message d'erreur concernant cette instance (le cas échéant).

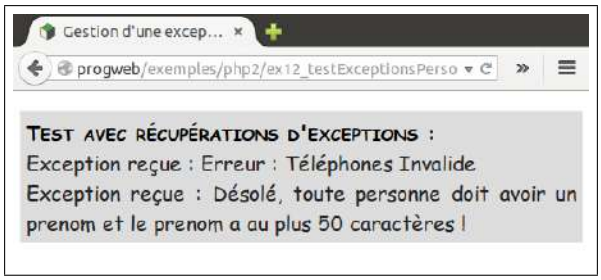


FIGURE 2.4 : Illustration du code source 2.16

Code Source 2.16 : /php2/ex12-testExceptionsPersonnes.php (cf. Fig 2.4)

```php
<?php
  require_once(dirname(__FILE__).'/classes/Telephone.php');
  require_once(dirname(__FILE__).'/classes/Adresse.php');
  require_once(dirname(__FILE__).'/classes/Personne.php');

  use CoursPHP\Metier\Adresse;
  use CoursPHP\Metier\Telephone;
  use CoursPHP\Metier\Personne;

  $dataError = array();

  try {
    $adresse1 = new Adresse("0af46d3bd9", '10', 'Downing Street', null,
                            null, 'London', 'United Kingdom');
    $personne1 = new Personne("e2f46d3ba6", "Thatcher", "Marggy", $adresse1,
                  "01 23 45 67 89"); // Le téléphone devrait être une instance
  } catch (Exception $e) {
    $dataError["personne1"] = $e->getMessage();
  }

  try {
    $adresse2 = new Adresse("b3f46d3a5d", '10', 'allée du net',
            'Quartier de l\'avenir', '63000', 'Clermont-Ferrand', 'Technique');
    $personne2 = new Personne("a4b46d3a5c", "Urluberlu", null, $adresse2, null);
        // Prénom ??
  }catch (Exception $e) {
    $dataError["personne2"] = $e->getMessage();
  }

  // Appel de la vue :
  require('ex12-vueExceptionsPersonnes.php');
?>
```

Code Source 2.17 : /php2/ex12-vueExceptionsPersonnes.php

```php
<?php
  require_once(dirname(__FILE__).'/classes/VueHtmlUtils.php');
  require_once(dirname(__FILE__).'/classes/AdresseView.php');
  require_once(dirname(__FILE__).'/classes/PersonneView.php');

  use CoursPHP\Vue\AdresseView;
  use CoursPHP\Vue\PersonneView;

  echo CoursPHP\Vue\VueHtmlUtils::enTeteHTML5('Gestion d\'une exception',
                                              'UTF-8', 'myStyle.css');

  echo "<p>";
  echo "<strong>Test avec récupérations d'exceptions :</strong><br/>";
  if (empty($dataError["personne1"])){
    echo PersonneView::getHtmlDevelopped($personne1);
  }else{
    echo $dataError["personne1"];
  }
  echo "</p>";
  echo "<p>";
  if (empty($dataError["personne2"])){
    echo PersonneView::getHtmlDevelopped($personne2);
  }else{
    echo $dataError["personne2"];
  }
  echo "</p>";

  echo CoursPHP\Vue\VueHtmlUtils::finFichierHTML5();
?>
```

2.4 Classe `Employe` héritant de la classe `Personne`

Notons l'attribut `categoriesEmployes`, qui répertorie dans un tableau toutes les catégories d'employés possibles, et la méthode `validCategorie`, qui détermine si une chaine de catactères correspond à une catégorie d'employés. Cette donnée et cette méthode sont déclarées *statiques*. Il s'agit donc d'une variable de classe et d'une méthodes de classe.

Voici le script de test qui crée quelques employés. Lorsque'une exception est reçue, au lieu d'afficher l'employé, on affiche le message d'erreur. Nous verrons plus loin comment ce mécanisme de gestion des exceptions permet de renvoyer à l'utilisateur des informations sur les attributs invalides qu'il a saisi dans le formulaire.

Code Source 2.18 : /php2/classes/Employe.php

```php
<?php
namespace CoursPHP\Metier;

require_once(dirname(__FILE__).'/EmployeProperties.php');

class Employe extends Personne {
  /** salaire mensuel de la personne en euros/mois */
  protected $salaireMensuel;

  /** catégorie d'employé : "secrétaire", "commercial", "technique" ou "pdg" */
```

```
11    protected $categorie;
12
13    /** @brief tableau de toutes les catégories d'employés possibles.
14     * un attribut statique est un attribut qui existe en un seul exemplaire
15     * commun à tous  les objets de la classe.
16     * Cela évite d'avoir autant de copies du tableau  $categoriesEmployes
17     * qu'il y a d'instance de la classe Employe en mémoire. */
18    private static $categoriesEmployes = array("secrétaire", "commercial",
19                                               "technique", "boss");
20
21    // Inclusion du trait avec les accesseurs/setters
22    use EmployeProperties;
23
24    /** Constructeur avec des paramètres correspondant aux attributs */
25    public function __construct($idPersonne, $nom, $prenom, $adresse,
26                                $telephones, $salaire, $categorie) {
27      // Appel du constructeur de la classe mère :
28      parent::__construct($idPersonne, $nom, $prenom, $adresse, $telephones);
29      // Initialisation des attributs de la classe self
30      $this->setSalaireMensuel($salaire);
31      $this->setCategorie($categorie);
32    }
33  }
34  ?>
```

Code Source 2.19 : /php2/classes/EmployeProperties.php

```
1   <?php
2   namespace CoursPHP\Metier;
3
4   trait EmployeProperties {
5    /** Méthode statique de validation d'un paramètre de catégorie.
6     * la valeur doit se trouver dans les tableau $categoriesEmployes.
7     * Une méthode statique ne s'applique pas à un objet particulier.
8     * On l'utilise avec self:: ou à l'extérieur de la classe avec Employe:: */
9     public static function isValidCategorie($categorie) {
10      if ($categorie == null || !is_string($categorie) ||
11          !in_array($categorie, self::$categoriesEmployes)){
12        return false;
13      }
14      return true;
15    }
16
17    /** @brief accesseur : permet d'obtenir la catégorie de l'employé */
18    public function getCategorie() {
19      return $this->categorie;
20    }
21
22    /** @brief accesseur : permet d'obtenir le téléphone 1 de l'employé */
23    public function getSalaireMensuel() {
24      return $this->salaireMensuel;
25    }
26
27    /** setter : permet d'initialiser ou de modifier la catégorie de la personne
28      * @param $categorie doit être une catégorie d'employé répertoriée */
29    public function setCategorie($categorie) {
```

```php
30      if (!self::isValidCategorie($categorie)){
31        throw new \Exception("Erreur, catégorie d'employé \"".$categorie
32                             ."\" invalide.");
33      }else{
34        $this->categorie = $categorie;
35      }
36    }
37
38    /** setter : permet d'initialiser ou de modifier le salaire de la personne
39      * @param $salaire salaire mensuel en euros/mois */
40    public function setSalaireMensuel($salaire) {
41      if ($salaire == null || !is_numeric($salaire)){
42        $this->salaireMensuel = 0.0;
43      }else {
44        $this->salaireMensuel = $salaire;
45      }
46    }
47  }
48  ?>
```

Code Source 2.20 : /php2/classes/EmployeView.php

```php
1  <?php
2  namespace CoursPHP\Vue;
3
4  class EmployeView {
5    /** @brief Méthode de génération de code HTML. Permet d'afficher un Employe */
6    public static function getHtmlDevelopped($employe){
7      $htmlCode = PersonneView::getHtmlDevelopped($employe);
8      $htmlCode .= "Salaire mensuel : ".$employe->getSalaireMensuel()." &euro; par
           mois<br/>\n";
9      $htmlCode .= "Catégorie : ".$employe->getCategorie()."<br/>\n";
10     return $htmlCode;
11   }
12 } // end of class EmployeView
13 ?>
```

FIGURE 2.5 : Illustration du code source 2.21

Code Source 2.21 : /php2/ex16-testEmploye.php (cf. Fig 2.5)

```php
<?php
  require_once(dirname(__FILE__).'/classes/Telephone.php');
  require_once(dirname(__FILE__).'/classes/Adresse.php');
  require_once(dirname(__FILE__).'/classes/Personne.php');
  require_once(dirname(__FILE__).'/classes/Employe.php');

  use CoursPHP\Metier\Adresse;
  use CoursPHP\Metier\Telephone;
  use CoursPHP\Metier\Employe;

  $dataError = array();

  try {
    $adresse1 = new Adresse("0af46d3bd9", '10', 'allée du net', 'Quartier de l\'avenir',
                  '63000', 'Clermont-Ferrand', 'France');
    $telephones1 = array(new Telephone("Domicile", "04 73 00 00 00"));
    $employe1 = new Employe("0af46d3bd9", "Obama", "Barack", $adresse1, $telephones1, 300000.0, 'boss');
  } catch (Exception $e) {
    $dataError["employe1"] = $e->getMessage();
  }

  try {
    $adresse2 = new Adresse("5b246d3da2", '10', 'Downing Street', null,
                  null, 'London', 'United Kingdom');
    $employe2 = new Employe("d7c46d3a3b", "Thatcher", "Margaret", $adresse2,
                  array(new Telephone("Emergency", "911")), null, 'badCategory');
  } catch (Exception $e) {
    $dataError["employe2"] = $e->getMessage();
  }

  // Appel de la vue :
  require('ex16-vueEmploye.php');
?>
```

Code Source 2.22 : /php2/ex16-vueEmploye.php

```php
<?php
  require_once(dirname(__FILE__).'/classes/VueHtmlUtils.php');
  require_once(dirname(__FILE__).'/classes/AdresseView.php');
  require_once(dirname(__FILE__).'/classes/PersonneView.php');
  require_once(dirname(__FILE__).'/classes/EmployeView.php');

  use CoursPHP\Vue\EmployeView as EmployeView;

  echo CoursPHP\Vue\VueHtmlUtils::enTeteHTML5('Gestion d\'une exception', 'UTF-8', 'myStyle.css');

  echo "<h2>Construction et Affichage d'un Employé :</h2>";
  echo "<p>";
  echo "<strong>Test avec récupérations d'exceptions :</strong><br/>";
  echo "</p>";

  echo "<p>";
```

```
17    if (empty($dataError["employe1"])){
18      echo EmployeView::getHtmlDevelopped($employe1);
19    }else{
20      echo $dataError["employe1"];
21    }
22    echo "</p>";
23
24    echo "<p>";
25    if (empty($dataError["employe2"])){
26      echo EmployeView::getHtmlDevelopped($employe2);
27    }else{
28      echo $dataError["employe2"];
29    }
30    echo "</p>";
31
32    echo CoursPHP\Vue\VueHtmlUtils::finFichierHTML5();
33  ?>
```

Deuxième partie

Formulaires et Filtrage des Données Utilisateur

Table of Contents

Chapitre 3

Formulaires *HTML/PHP*

Les formulaires HTML permettent de faire saisir des données par l'utilisateur via son navigateur. Ces données sont saisies dans des champs appelés *inputs*, qui sont définis avec la balise `<input>`. Les données sont ensuite récupérées dans un script, ici un script *PHP*. Ces données doivent impérativement être testées et filtrées pour des raisons de sécurité.

3.1 Formulaires *HTML*

Un formulaire est créé par une balise `<form>` qui contient la méthode de transmission des données (*GET* ou *POST*) et l'action, qui est l'*URL* du script (ici un script *PHP*) qui va récupérer les données du formulaire. Chaque input a son label, qui explique à l'utilisateur ce qu'il doit saisir dans ce champ. La correspondance entre les inputs et le labels se fait via l'attribut `for` du label qui doit correspondre à l'attribut `id` de l'input. L'attribut `name` de l'input servira lors de la récupération des données. Les attributs `id` et `name` de l'input peuvent être égaux si on veut simplifier.

3.1.1 Premier Formulaire *HTML*

Code Source 3.1 : /forms1/ex01-form-html.html (cf. Fig 3.1)

```
1  <!doctype html>
2  <html lang="fr">
3  <head>
4    <meta charset="UTF-8" />
5    <title>Mon premier formulaire HTML</title>
6  </head>
7  <body>
8    <h1>Saisie d'un employé</h1>
9    <form method="post" action="ex02-reception.php">
10     <p>
11       <label for="nomEmploye">Nom</label>
12       <input type="text" name="nom"  id="nomEmploye" size="30"/>
13     </p>
14     <p>
15       <label for="prenomEmploye">Prénom</label>
16       <input type="text" name="prenom"  id="prenomEmploye" size="30"/><br/>
17     </p>
18     <p>
```

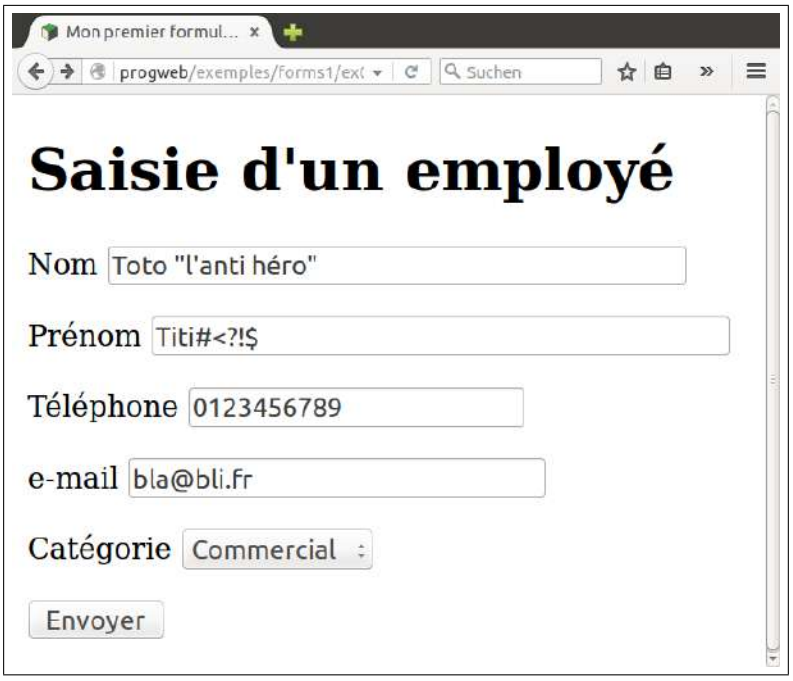


FIGURE 3.1 : Illustration du code source 3.1

```
19        <label for="telephone">Téléphone</label>
20        <input type="text" name="telephone" id="telephone" size="15"/><br/>
21      </p>
22      <p>
23        <label for="email">e-mail</label>
24        <input type="text" name="email" id="email" size="20"/><br/>
25      </p>
26      <p>
27        <label for="categorie">Catégorie</label>
28        <select name="categorie">
29          <option value="secretaire" selected="selected">Secrétaire</option>
30          <option value="commercial">Commercial</option>
31          <option value="technique">Technique</option>
32          <option value="boss"/>The Big Boss</option>
33        </select>
34      </p>
35      <p>
36        <input type="submit" value="Envoyer"></input>
37      </p>
38    </form>
39  </body>
40  </html>
```

3.1.2 Exemple de style *CSS* pour formulaire

Code Source 3.2 : /forms1/myStyle.css (cf. Fig 3.2)

```
1  /* style par défaut du texte */
2  body {
3    font-family : "Comic Sans MS";
```

FIGURE 3.2 : Illustration du code source 3.2

```
4     font-size : 18pt;
5     background-color : #fff;
6     color : #222;
7   }
8
9   /* style du titre */
10  h1 {
11    font-weight : bold;
12    font-size : 150%;
13    color : white;
14    text-align : center;
15    background-color : #999;
16    padding : 15px;
17  }
18
19  /*****************************************
20  | mise en forme du formulaire             |
21  \*****************************************/
22
23  /* Largeur minimale pour que la mise en page "ne casse pas" */
24  form {
25    width : 800px;
26  }
27
28  form span.formField{
29    width  : inherit;
30    display : inline-block;
31    margin  : 5px 0;
32  }
33
34  /* tous les labels ont la même largeur pour aligner les inputs */
35  form label {
36    float : left;
37    width : 400px;
38    text-align : right;
39    padding : 6px;
40    font-weight : bold;
41  }
42
43  form input {
44    padding : 6px;
```

```
45    margin-left : 20px;
46    background-color : #ddd;
47    border-style : groove;
48    border-width : 5px;
49    border-color : #444;
50    border-radius :10px;
51  }
52
53  form select {
54    padding : 1px 6px;
55    margin-left : 20px;
56    background-color : #ddd;
57    border-style : groove;
58    border-width : 5px;
59    border-color : #444;
60    border-radius :10px;
61    font-size : 110%;
62  }
63
64  /* input spécial pour le boutton submit */
65  form input.sansLabel {
66    margin-left : 432px; /* 400+6+6+20 : aligné sur les autres inputs */
67    font-size : 130%;
68    font-weight : bolder;
69    background-color :#333333;
70    color :white;
71  }
72
73  form p {
74    background-color : #fff;
75    padding : 0px;
76  }
77
78  form span.errorMsg {
79    font-size : 90%;
80    font-style : italic;
81    color :red;
82  }
83
84  /*****************************************/
85
86
87  /* style par défaut des liens */
88  a:link {
89    text-decoration : underline; /* souligné */
90    color : #00e;
91  }
92
93  /* style des liens visités */
94  a:visited {
95    text-decoration : underline; /* souligné */
96    color : #00c;/* bleu clair */
97  }
98
99  /* style des liens visités */
100 a:hover {
```

```
101   text-decoration : underline ; /* souligné */
102   color : #e40 ;/* rouge vif */
103 }
104
105 /* style des éléments importants */
106 strong {
107   font-variant : small-caps ;
108   font-weight : bolder ;
109   color : black ;
110 }
111
112 /* style des éléments mis en évidence */
113 em {
114   font-style : italic ;
115   color : black ;
116 }
117
118 p {
119   background-color : #ddd ;
120   text-align : justify ;
121   padding : 5pt ;
122 }
```

3.1.3 Réception des données en *PHP*

La réception des données se fait ici par la méthode `POST` (comme indiqué dans la balise `<form>`). Les données sont récupérées dans un tableau associatif `$_POST`, dont les clefs sont les attributs `name` des inputs du formulaire précédent. On teste si ces attributs existent bien via la fonction `isset`.

Code Source 3.3 : /forms1/ex02-reception.php

```
1  <?php
2    $nom = isset($_POST['nom']) ? $_POST['nom'] : "";
3    $prenom = isset($_POST['prenom']) ? $_POST['prenom'] : "";
4    $telephone = isset($_POST['telephone']) ? $_POST['telephone'] : "";
5    $email = isset($_POST['email']) ? $_POST['email'] : "";
6    $categorie = isset($_POST['categorie']) ? $_POST['categorie'] : "";
7
8    require('ex04-validation.php');
9
10   if (empty($dataErrors)){
11     require('ex05-vueSuccess.php');
12   }else{
13     require('ex06-vueError.php');
14   }
15 ?>
```

3.2 Validation pour la sécurité : Appel de `filter_var`

Pour des raisons de sécurité (voir le chpitre 4), un filtrage systématique doit être effectué sur les données reçus dans les tableaux `$_GET`, `$_POST`, `$_COOCKIE`, etc.

Pour cela, on fait généralement un script de validation qui valide ou nettoie les données, par exemple en utilisant la fonction `filter_var`.

Code Source 3.4 : /forms1/ex04-validation.php

```php
<?php
  require_once ("ex03-validUtils.php");

  $dataErrors = array();

    // validation du nom
  $nom = filter_var($nom, getSanitizeFilter('string'));

    // validation du prénom
  $prenom = filter_var($prenom, getSanitizeFilter('string'));

    // validation du téléphone
  $telephone = filter_var($telephone, getSanitizeFilter('string'));

  // validation de l'adresse e-mail

  if (filter_var($email, getValidateFilter('email'))===false){
    $dataErrors['email'] = "Erreur : l'adresse e-mail est invalide.";
  }

    // validation de la catégorie
  $categorie = filter_var($categorie, getSanitizeFilter('string'));
?>
```

Code Source 3.5 : /forms1/ex03-validUtils.php

```php
<?php
// Méthode retournant le filtre de validation à utiliser
// dans la fonction filter_var
function getValidateFilter($type)
{
  switch($type){
    case "email":
      $filter = FILTER_VALIDATE_EMAIL;
      break;
    case "int":
      $filter = FILTER_VALIDATE_INT;
      break;
      case "boolean":
      $filter = FILTER_VALIDATE_BOOLEAN;
      break;
    case "ip":
      $filter = FILTER_VALIDATE_IP;
      break;
    case "url":
      $filter = FILTER_VALIDATE_URL;
      break;
    default: // important !!!
      $filter = false; // Si type est faux, la valid. échoue.
  }
  return $filter;
}
```

```
27
28 // Méthode retournant le filtre de nettoyage à utiliser
29 // dans la fonction filter_var
30 function getSanitizeFilter($type)
31 {
32   switch($type){
33     case "string":
34             $filter = FILTER_SANITIZE_STRING;
35         break;
36     case "text":
37             $filter = FILTER_SANITIZE_FULL_SPECIAL_CHARS;
38         break;
39     case "url":
40             $filter = FILTER_SANITIZE_URL;
41         break;
42     default: // important !!!
43       $filter = false; // Si type est faux, la valid. échoue.
44   }
45   return $filter;
46 }
47 ?>
```

3.3 Appel des vues

Comme nous l'avons vu dans la partie 3.1.3, un test permet, à la suite dee la validation, de savoir si une erreur s'est produite. Suivant le cas,

- La vue normale affiche les données saisies ;
- Une vue d'erreurs affiche les messages d'erreur.

Code Source 3.6 : /forms1/ex05-vueSuccess.php (cf. Fig 3.3)

```
1  <?php
2    require_once(dirname(__FILE__).'/commonFunctions.php');
3
4    outputEnTeteHTML5('Affichage des données saisies', 'UTF-8', 'myStyle.css');
5
6    echo "<h1>Données reçues</h1>\n";
7    echo "<p>\n";
8    echo "nom : ".$nom."<br/>\n";
9    echo "prenom : ".$prenom."<br/>\n";
10   echo "Téléphone : ".$telephone."<br/>\n";
11   echo "E-mail : ".$email."<br/>\n";
12   echo "Catégorie : ".$categorie."<br/>\n";
13   echo "</p>\n";
14
15   outputFinFichierHTML5();
16
17 ?>
```

FIGURE 3.3 : Illustration du code source 3.6

Code Source 3.7 : /forms1/ex06-vueError.php (cf. Fig 3.4)

```php
<?php
  require_once(dirname(__FILE__).'/commonFunctions.php');
  outputEnTeteHTML5('Erreurs données saisies', 'UTF-8', 'myStyle.css');

  echo "<h1>Données reçues incorrectes</h1>\n";

  echo "<ul>";
  foreach ($dataErrors as $field => $message){
    echo '<li>Problème avec l\'attribut <code>'.$field
          .'</code>. <span style="color : red;">'.$message.'</span></li>';
  }
  echo "</ul>";

  echo '<p>Merci de bien vouloir <a href="ex01-form-html.html">Essayer à nouveau
      </a></p>';

  outputFinFichierHTML5();
?>
```

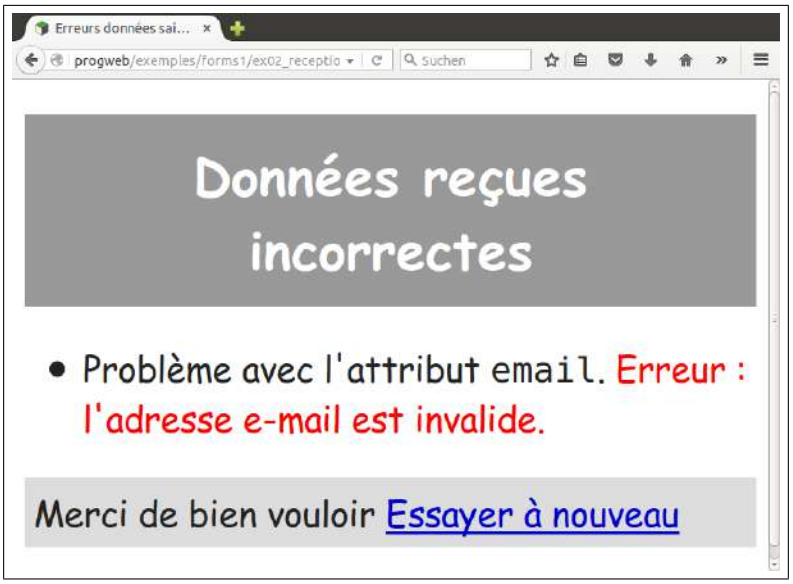


FIGURE 3.4 : Illustration du code source 3.7

Dans tous les cas, **seuls les scripts implémentant des vues envoie du code *HTML* sur la sortie standard**.

3.4 Tableaux `$_POST` `$_GET` `$_REQUEST`

Nous avons vu, pour le moment, deux méthodes pour transmettre des données d'un script *PHP* à l'autre : la méthode *GET* et la méthode *POST*. On réceptionne alors les données (respectivement) dans des tableaux associatifs `$_POST` `$_GET`. On peut aussi utiliser un tableau associatif `$_REQUEST`, qui contient à la fois les éléments du tableau `$_POST` et les éléments du tableau `$_GET`. Remarque : le tableau `$_REQUEST` contient aussi les éléments du tableau `$_COOKIE`.

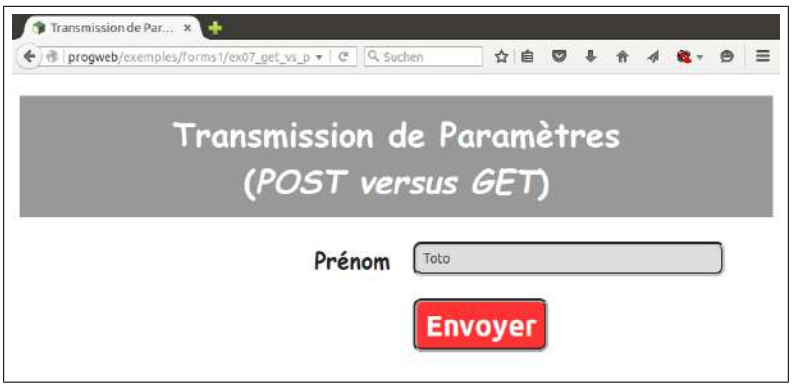


FIGURE 3.5 : Illustration du code source 3.8

Code Source 3.8 : /forms1/ex07-get-vs-postHidden.php (cf. Fig 3.5)

```
1 <!doctype html>
2 <html lang="fr">
3 <head>
```

```
4   <meta charset="UTF-8" />
5 <link rel="stylesheet" href="./myStyle.css" />
6   <title>Transmission de Paramètres</title>
7 </head>
8 <body>
9   <h1>Transmission de Paramètres<br/>(<i>POST versus GET</i>)</h1>
10  <!-- Ce formulaire transmet trois valeurs non saisies par l'utilisateur -->
11  <form method="post" action="ex08-get-post-request-param.php?language=fr&region
        =eu">
12    <input type="hidden" name="referredFrom" value="searchEngine">
13    <p>
14      <label for="prenomEmploye">Prénom</label>
15      <input type="text" name="prenom"     id="prenomEmploye"   size="30"/><br/>
16    </p>
17    <p>
18      <input type="submit" value="Envoyer" class="sansLabel"></input>
19    </p>
20  </form>
21 </body>
22 </html>
```

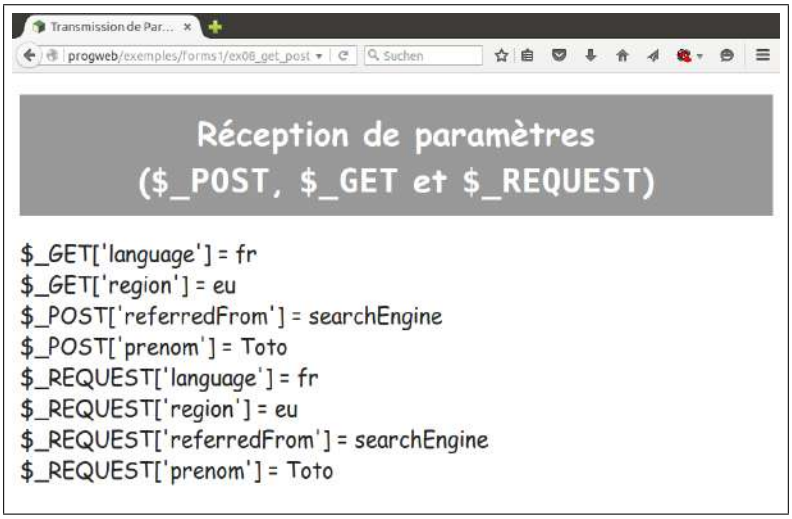


FIGURE 3.6 : Illustration du code source 3.9

Code Source 3.9 : /forms1/ex08-get-post-request-param.php (cf. Fig 3.6)

```
1 <!doctype html>
2 <html lang="fr">
3 <head>
4   <meta charset="UTF-8" />
5 <link rel="stylesheet" href="./myStyle.css" />
6   <title>Transmission de Paramètres</title>
7 </head>
8 <body>
9   <h1>Réception de paramètres<br/>(<code>$_POST</code>, <code>$_GET</code> et <
        code>$_REQUEST</code>)</h1>
10    <?php
11    foreach ($_GET as $key => $val){
12      echo htmlentities("\$_GET['".$key."'] = ".$val, ENT_COMPAT, "UTF-8").
            "<br />";
```

```
13      }
14      foreach ($_POST as $key => $val){
15        echo htmlentities("\$_POST['".$key."'] = ".$val, ENT_COMPAT, "UTF-8").
              "<br/>";;
16      }
17      foreach ($_REQUEST as $key => $val){
18        echo htmlentities("\$_REQUEST['".$key."'] = ".$val, ENT_COMPAT, "UTF-8").
              "<br/>";;
19      }
20       ?>
21  </body>
22  </html>
```

3.5 Formulaires dynamiques an javascript

Nous voyons ici un exemple d'utilisation du *Javascript* pour créer un formulaire dont les attributs dépendent de la valeur d'un premier champ. Lorsqu'on sélectionne "deuxième année", un nouveau champ apparaît. Pour celà, on utilise l'événement `onchange` sur l'`input` de l'année, qui est géré par la fonction `anneeChange`. On teste alors la valeur de l'attribut, puis le cas échéant on génère un nouveau champ dans un `div` d'`id` `attributSupplementaire`. Pour plus d'information sur les pages *web* dynamiques en *Javascript*, voir le cours correspondant sur `www.malgouyres.org`.

Code Source 3.10 : /javascript/formulaire-dynamique.html

```
1   <!doctype html>
2   <html lang="fr">
3     <head>
4       <meta charset="UTF-8"/>
5       <title>Formulaire dynamique</title>
6     </head>
7     <body>
8       <form method="post" action="reception.php">
9         <p>
10          <label for="nom">Nom</label><input name="nom" id="nom"/>
11        </p>
12        <p>
13          <select name="annee" id="annee" pattern="(premiere) | (deuxieme)"
                onchange='anneeChange();'>
14      <option value="choisissez" selected disabled>-- choisissez --</option>
15      <option value="premiere">Première année</option>
16      <option value="deuxième">Deuxième année</option>
```

```
17          </select>
18        </p>
19        <div id="attributSupplementaire">
20
21        </div>
22        <p>
23          <input type="submit" value="-- OK --"/>
24        </p>
25      </form>
26 <script>
27    function anneeChange(){
28       var paragraphe = document.getElementById("attributSupplementaire");
29       paragraphe.innerHTML=document.getElementById("annee").value+" année.";
30       if (document.getElementById("annee").value == "deuxième"){
31          paragraphe.innerHTML+="<label>Orientation prévue pour l'année prochaine
                :</label>"
32          +'<select name="orientation" id="orientation">'
33          +'<option value="LP">LP</option>'
34          +'<option value="master">master</option>'
35          +"<option value=\"inge\">Ecole d'ingé</option>"
36          +'<option value="boulot">Boulot</option>'
37          +'<option value="autre">Autre</option>'
38          +'</select>';
39
40       }
41    }
42    anneeChange();
43 </script>
44    </body>
45 </html>
```

Code Source 3.11 : /javascript/reception.php

```
1  <!doctype html>
2  <html lang="fr">
3    <head>
4      <meta charset="UTF-8"/>
5      <title>Formulaire dynamique</title>
6    </head>
7    <body>
8  <?php
9    $nom= (isset($_POST["nom"])) ? $_POST["nom"] :"nom indéterminé";
10   $annee = (isset($_POST["annee"])) ? $_POST["annee"] :"année indéteminée";
11            echo "Nom : ".$nom."<br/>";
12            echo "Année : ".$annee."<br/>";
13   if ($annee=="deuxième")
14        echo " Orientation : ".$_POST["orientation"];
15
16
17 ?>
18   </body>
19 </html>
```

Chapitre 4

Injections *XSS*, Filtrage, Expressions Régulières

4.1 Injections *HTML* et échappement

4.1.1 Injections *HTML*

Les *injections XSS* sont un moyen pour un pirate d'exécuter du code non prévu en exploitant les interfaces entre PHP et d'autres langages (HTML, Javascript, SQL, etc...). Pour celà, le pirate rentre dans un input du code, qui n'est pas détecté par PHP (on a simplement une chaîne de caractères au niveau de PHP), mais qui est interprété par un autre langage interfacé avec PHP.

Voyons un exemple d'*injection HTML*. L'utilisateur malveillant va entrer dans un `textarea`, de nom "desctiption", du code *HTML* pour introduire dans le site victime un lien vers un site pirate. Si l'utilisateur inaverti ou distrait clique sur ce lien, le pirate peut alors demander à l'utilisateur de rentrer ses identifiants (usurpation d'identité) ou ses données de carte bancaire (escroquerie), etc.

Voyons tout d'abord de formulaire et sa réception dans le cadre de son utilisation normale.

FIGURE 4.1 : Un gentil formulaire

FIGURE 4.2 : Le site affiche en *HTML* les données saisies dans le gentil formulaire

Le code source du formulaire et de sa réception est le suivant :

Code Source 4.1 : /filtrage/ex00-1-postParamForHTML-inject.php

```
1  <!doctype html>
2  <html lang="fr">
3  <head>
4    <meta charset="UTF-8" />
5  <link rel="stylesheet" href="./myStyle.css" />
6    <title>Post un Nom</title>
7  </head>
8  <body>
9    <h1>Post d'une chaîne</h1>
10   <form method="post" action="ex00-2-receptParamForHTML-inject.php">
11     <label for="description" style="margin-right : 10px; vertical-align :top;">
           Description :</label>
12     <textarea name="description" id="description" cols="50" row="6" style="
           vertical-align :top;">
13     </textarea>
14     <input type="submit" value="Envoyer"/>
15   </form>
16 </body>
17 </html>
```

Code Source 4.2 : /filtrage/ex00-2-receptParamForHTML-inject.php

```
1  <!doctype html>
2  <html lang="fr">
3  <head>
4    <meta charset="UTF-8" /><link rel="stylesheet" href="./myStyle.css" />
5    <title>Réception Vulnérable à Injections XSS</title>
6  </head>
7  <body>
8    <h1>Réception Vulnérable à Injections <i>XSS</i></h1>
9    <p>
10     <strong>La Description du client est&amp;nbsp;:</strong><br/>
11     <?php
12       if (isset($_POST['description'])){
13         echo $_POST['description'];
14       }
15     ?>
16     <br/>Ceci est la suite du document.
17   </p>
18 </body>
19 </html>
```

Le pirate entre alors dans un input du code *HTML* :

FIGURE 4.3 : Injection HTML ajoutant un lien au site

Le résultat est l'apparition d'un lien non prévu sur le site :

FIGURE 4.4 : L'affichage des données du formulaire sort le code *HTML* entré par le pirate

4.1.2 Prévention des injections *HTML* par échappement

4.1.2.a Échappement par `htmlentities`

Différents outils de filtrage sont disponibles en PHP. Le plus simple pour la sécurité consiste à utiliser la méthode `htmlentities` qui tranforme dans une chaîne tous les caractères spéciaux en leurs entités HTML (code spécial pour afficher un caractère en HTML).

Il faut cependant prendre garde que si l'utilisateur ne rentre pas les caratères en entrée avec le même encodage que celui utilisé par PHP en sortie, les caractères spéciaux n'ont pas le même code et la fonction `htmlentities` ne fonctionnera pas bien, laissant la porte ouverte à des attaques. On peut spécifier l'encoding de sortie de `htmlentities` dans son troisième paramètre.

Dans l'exemple d'injection HTML ajoutant un lien ci-dessus, on obtiendrait lors de l'affichage :

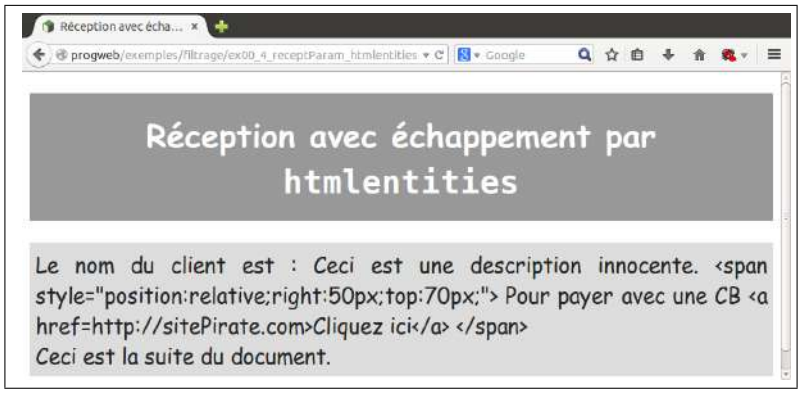


FIGURE 4.5 : L'injection a été évitée par échappement.

Le code *HTML* produit par le *CGI* est le suivant :

Code Source 4.3 : /filtrage/ex00-5-htmlentitiesHTML-Output.html

```
1  <!doctype html>
2  <html lang="fr">
3  <head>
4    <meta charset="UTF-8" /><link rel="stylesheet" href="./myStyle.css" />
5    <title>Réception avec échappement</title>
6  </head>
7  <body>
8    <h1>Réception avec échappement par <code>htmlentities</code></h1>
9    <p>
10     Le nom du client est :
11     Ceci est une description innocente.
12 &lt;span style=&quot;position:relative;right:50px;top:70px;&quot;&gt;
13     Pour payer avec une CB
14     &lt;a href=http://sitePirate.com&gt;Cliquez ici&lt;/a&gt;
15 &lt;/span&gt;   <br/>Ceci est la suite du document.
16   </p>
17 </body>
18 </html>
```

Ça n'est pas très joli mais c'est inoffensif sauf si l'utilisateur fait vraiment exprès de copier l'adresse du lien dans sa barre d'adresse. Pour éviter complètement l'apparition de code, *HTML* ou autre, ou plus généralement de données non conforme à un format attendu, nous veroons plus loin comment utiliser des expressions régulières.

4.1.2.b Options d'échappement

Nous voyons ici trois exemples d'échappement qui traitent différemment les guillemets et les apostrophes (doubles et simples *quotes*). Les chaines positées sont les suivantes :

Code Source 4.4 : /filtrage/ex02-postParam.php

```
1  <!doctype html>
2  <html lang="fr">
3  <head>
4    <meta charset="UTF-8" />
5  <link rel="stylesheet" href="./myStyle.css" />
6    <title>Post de deux chaînes</title>
```

```
7  </head>
8  <body>
9    <h1>Post de deux chaînes</h1>
10   <form method="post" action="ex03-escapeTestRequest.php">
11     <input type="hidden" name="chaine1" value="Ceci est l'exemple de chaîne avec
         apostrophe"/>
12     <input type="hidden" name="chaine2" value="Ceci est soit disant un &quot;
         autre&quot; exemple."/>
13     <input type="submit" value="Envoyer" class="sansLabel"/>
14   </form>
15 </body>
16 </html>
```

À la réception, on observe la chose suivante suivant les options données en paramètre de `htmlentities` :

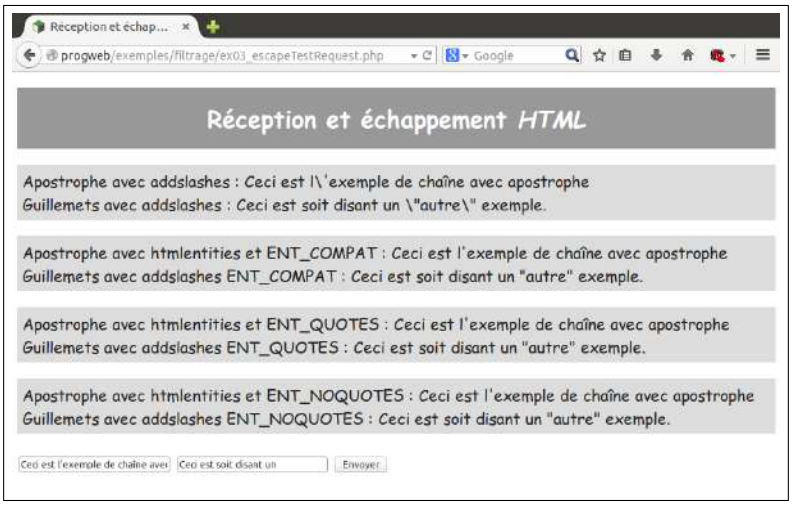


FIGURE 4.6 : Illustration du code source 4.5

Code Source 4.5 : /filtrage/ex03-escapeTestRequest.php (cf. Fig 4.6)

```
1  <?php
2    require('./commonFunctions.php');
3
4    outputEnTeteHTML5('Réception et échappement HTML', 'UTF-8', 'myStyle.css');
5
6    echo "<h1>Réception et échappement <i>HTML</i></h1>";
7
8    $chaine1 = $_REQUEST['chaine1'];
9    $chaine2 = $_REQUEST['chaine2'];
10
11   echo "<p>\n";
12   echo "Apostrophe avec addslashes : ".addslashes($chaine1)."<br>\n";
13   echo "Guillemets avec addslashes : ".addslashes($chaine2)."<br>\n";
14   echo "</p>\n";
15
16   echo "<p>\n";
17   echo "Apostrophe avec htmlentities et ENT_COMPAT : ".htmlentities($chaine1,
       ENT_COMPAT, 'UTF-8', false)."<br>\n";
18   echo "Guillemets avec addslashes ENT_COMPAT : ".htmlentities($chaine2,
       ENT_COMPAT, 'UTF-8', false)."<br>\n";
```

```
19    echo "</p>\n";
20
21    echo "<p>\n";
22    echo "Apostrophe avec htmlentities et ENT_QUOTES : ".htmlentities($chaine1,
          ENT_QUOTES, 'UTF-8', false)."<br>\n";
23    echo "Guillemets avec addslashes ENT_QUOTES : ".htmlentities($chaine2,
          ENT_QUOTES, 'UTF-8', false)."<br>\n";
24    echo "</p>\n";
25
26    echo "<p>\n";
27    echo "Apostrophe avec htmlentities et ENT_NOQUOTES : ".htmlentities($chaine1,
          ENT_NOQUOTES, 'UTF-8', false)."<br>\n";
28    echo "Guillemets avec addslashes ENT_NOQUOTES : ".htmlentities($chaine2,
          ENT_NOQUOTES, 'UTF-8', false)."<br>\n";
29    echo "</p>";
30
31 ?>
32      <form method="post" action="ex03-escapeTestRequest.php">
33      <input type="text" name="chaine1" value="<?php echo $chaine1;?>">
34      <input type="text" name="chaine2" value="<?php echo $chaine2;?>">
35      <input type="submit" value="Envoyer" class="sansLabel"></input>
36    </form>
37 <?php
38    outputFinFichierHTML5();
39 ?>
```

Le code source *HTML* généré par le *CGI* est le suivant :

Code Source 4.6 : /filtrage/ex03-escapeTestRequest-php-htmlOutput.html

```
1  <!doctype html>
2  <html lang="fr">
3  <head>
4  <meta charset="UTF-8"/>
5  <link rel="stylesheet" href="myStyle.css" />
6  <title>Réception et échappement HTML</title>
7  </head>
8  <body>
9  <h1>Réception et échappement <i>HTML</i></h1><p>
10 Apostrophe avec addslashes : Ceci est l\'exemple de chaîne avec apostrophe<br>
11 Guillemets avec addslashes : Ceci est soit disant un \"autre\" exemple.<br>
12 </p>
13 <p>
14 Apostrophe avec htmlentities et ENT_COMPAT : Ceci est l'exemple de cha&icirc;ne
       avec apostrophe<br>
15 Guillemets avec addslashes ENT_COMPAT : Ceci est soit disant un &quot;autre&quot
       ; exemple.<br>
16 </p>
17 <p>
18 Apostrophe avec htmlentities et ENT_QUOTES : Ceci est l&#039;exemple de cha&
       icirc;ne avec apostrophe<br>
19 Guillemets avec addslashes ENT_QUOTES : Ceci est soit disant un &quot;autre&quot
       ; exemple.<br>
20 </p>
21 <p>
22 Apostrophe avec htmlentities et ENT_NOQUOTES : Ceci est l'exemple de cha&icirc;
       ne avec apostrophe<br>
```

```
23 Guillemets avec addslashes ENT_NOQUOTES : Ceci est soit disant un "autre"
       exemple.<br>
24 </p>     <form method="post" action="ex03-escapeTestRequest.php">
25     <input type="text" name="chaine1" value="Ceci est l'exemple de chaîne avec
           apostrophe">
26     <input type="text" name="chaine2" value="Ceci est soit disant un "autre"
           exemple.">
27     <input type="submit" value="Envoyer" class="sansLabel"></input>
28   </form>
29 </body>
30 </html>
```

4.1.2.c Inverser l'échappement

Après un échappement à réception des données, on peut inverser cet échappement pour restaurer les données bruttes d'origine, par exemple pour renvoyer ces données dans les inputs d'un formulaire :

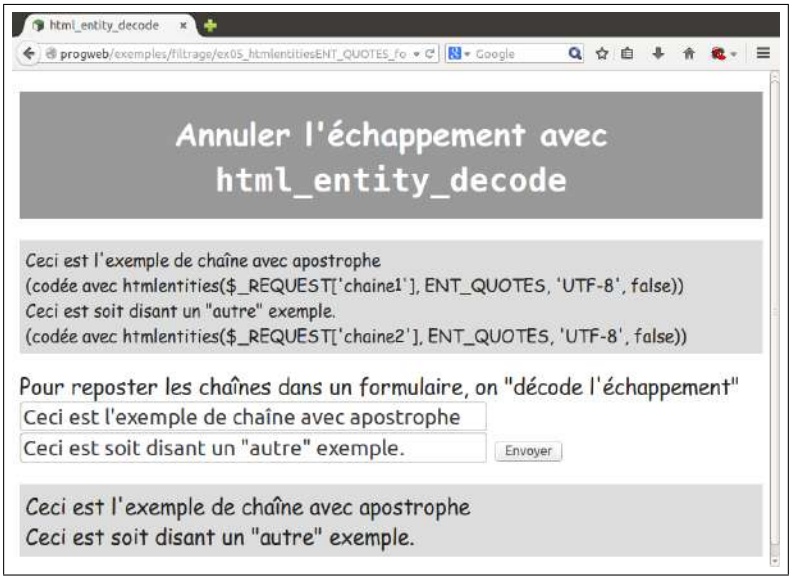


FIGURE 4.7 : Illustration du code source 4.7

Code Source 4.7 : /filtrage/ex05-htmlentitiesENT-QUOTES-formInput.php (cf. Fig 4.7)

```
1  <?php
2    require('./commonFunctions.php');
3
4    outputEnTeteHTML5('html_entity_decode', 'UTF-8', 'myStyle.css');
5
6    echo "<h1>Annuler l'échappement avec <code>html_entity_decode</code></h1>";
7
8    $chaine1 = htmlentities($_REQUEST['chaine1'], ENT_QUOTES, 'UTF-8', false);
9    $chaine2 = htmlentities($_REQUEST['chaine2'], ENT_QUOTES, 'UTF-8', false);
10
11   echo "<p style=\"font-size:80%;\">\n";
12   echo $chaine1."<br/>(codée avec ".htmlentities("htmlentities(\$_REQUEST['
         chaine1'], ENT_QUOTES, 'UTF-8', false)", ENT_QUOTES, 'UTF-8', false).")<br
         />\n";
```

```
13    echo $chaine2."<br/>(codée avec ".htmlentities("htmlentities(\$_REQUEST['
          chaine2'], ENT_QUOTES, 'UTF-8', false)", ENT_QUOTES, 'UTF-8', false).")<br
          />\n";
14    echo "</p>Pour reposter les chaînes dans un formulaire, on \"décode l'é
          chappement\"<br/>\n";
15  ?>
16      <form method="post" action="ex05-htmlentitiesENT-QUOTES-formInput.php">
17      <input style="font-size :100%;" type="text" name="chaine1" value="<?php echo
            $chaine1;?>" size="30"/>
18      <input style="font-size :100%;" type="text" name="chaine2" value="<?php echo
            $chaine2;?>" size="30"/>
19      <input type="submit" value="Envoyer" class="sansLabel"/>
20    </form>
21  <?php
22    echo "<p>\n";
23    echo html_entity_decode($chaine1, ENT_QUOTES, 'UTF-8')." <br>\n";
24    echo html_entity_decode($chaine2, ENT_QUOTES, 'UTF-8')." <br>\n";
25    echo "</p>\n";
26
27    outputFinFichierHTML5();
28  ?>
```

4.2 Injections *SQL*

Une *injection SQL* consiste à entre dans les inputs utilisateur du code *SQL*. Ces attaques sont particulièrement dansgereuses car elles peuvent etre exploitées par un pirtae pour :

- Accéder à des données confidentielles, par exemple à toutes les données de la base ;
- Détruire ou altérer des données, par exemple supprimer la totalité d'une table.

Voici un exemple de code qui insère une donnée (colonne `chaine`) de type chaine (`varchar`) dans une table `Table1`.

Code Source 4.8 : /filtrage/ex06-0-postParamForDB.php

```
1  <!doctype html>
2  <html lang="fr">
3  <head>
4    <meta charset="UTF-8" />
5  <link rel="stylesheet" href="./myStyle.css" />
6    <title>Formilaire HTML</title>
7  </head>
8  <body>
9    <h1>Saisie d'une Chaîne</h1>
10   <!-- Ce formulaire transmet trois valeurs non saisies par l'utilisateur -->
11   <form method="post" action="ex07-exInjectMySql.php">
12     <label for="chaine1">Entrez une chaîne</label>
13     <input type="texte" id="chaine1" name="chaine1" size="45"/><br/>
14
15     <input type="submit" value="Envoyer" class="sansLabel"></input>
16   </form>
17 </body>
18 </html>
```

Code Source 4.9 : /filtrage/ex07-exInjectMySql.php

```
1
2  <?php
3    include('./commonFunctions.php');
4    outputEnTeteHTML5('Exemple d\'injection SQL', 'UTF-8', 'myStyle.css');
5
6    echo "<h1>Réception vulnérable à une injection</h1>";
7
8    // On se connecte au serveur de bases de données.
9    // Adapter le nom d'hote où se trouve le serveur mysql
10   // mysqli($sql_server_url, $sql_user, $sql_user_password, $database_name)
11   $mysqli = new mysqli('progweb', "testUser", "motdepasse", 'baseTest');
12
13   // Vérification de la connexion :
14   if (mysqli_connect_errno()) {
15     printf("Échec de la connexion : %s\n", mysqli_connect_error());
16     exit();
17   }
18   echo 'Connecté au serveur de bases de données mysql.<br/>';
19
20   $chaine1="";
21   if (isset($_POST['chaine1'])){
22     $chaine1 = $_POST['chaine1'];
23   }
24   echo "Chaîne entrée par l'utilisateur : <code>".$chaine1."</code><br/>";
25
26   // Insertion de la chaîne dans la table Table1 (requête SQL) :
27   $requete = 'INSERT INTO Table1(chaine) VALUES ("'.$chaine1.'")';
28
29   echo "Requête exécutée :<br/><code>".$requete."</code><br/>";
30
31   $result = $mysqli->multi_query($requete) or die('Query failed : ' . mysql_error
         ()."<br/>");
32
33   // On ferme la connection
34   $mysqli->close();
35
36   outputFinFichierHTML5();
37 ?>
```

Voici deux exemple d'injections exploitant l'absence de filtrage dans ce code. Le premier exemple, gentillet, consiste juste à insérer deux données au lieu d'une dans la table :
Le deuxième exemple, plus méchant, consiste pour le pirate à supprimer toutes les données contenues dans la table :

Code Source 4.10 : /filtrage/ex09-mySqlEscapeString.php (cf. Fig 4.15)

```
1  <?php
2    include('./commonFunctions.php');
3    outputEnTeteHTML5('Echappement mysqli::real_escape_string', 'UTF-8', 'myStyle.
         css');
4
5    echo "<h1>Réception avec échappement <code>mysqli::real_escape_string</code></
         h1>";
6
7    // On se connecte au serveur de bases de données.
```

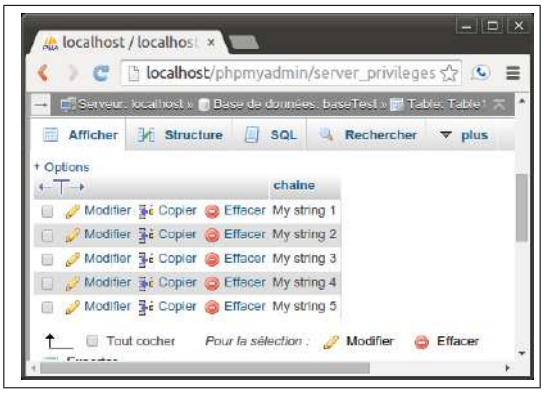


FIGURE 4.8 : L'état des données avant l'injection *SQL* par le pirate

FIGURE 4.9 : Données saisies par le pirate pour l'injection *SQL*

FIGURE 4.10 : Requête exécutée lors de l'injection *SQL*

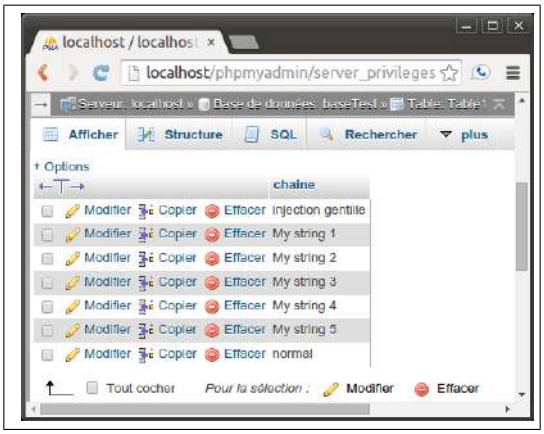


FIGURE 4.11 : L'état des données après l'injection *SQL* par le pirate

FIGURE 4.12 : Données saisies par le pirate pour l'injection *SQL*

FIGURE 4.13 : Requête exécutée lors de l'injection *SQL*

FIGURE 4.14 : L'état des données après l'injection *SQL* par le pirate

FIGURE 4.15 : Illustration du code source 4.10

```php
8   // Adapter le nom d'hote où se trouve le serveur mysql
9   // mysqli($sql_server_url, $sql_user, $sql_user_password, $database_name)
10  $mysqli = new mysqli('progweb', "testUser", "motdepasse", 'baseTest');
11
12  // Vérification de la connexion :
13  if (mysqli_connect_errno()) {
14    printf("Échec de la connexion : %s\n", mysqli_connect_error());
15    exit();
16  }
17  echo 'Connecté au serveur de bases de données mysql.<br/>';
18
19  $chaine1="";
20  if (isset($_POST['chaine1'])){
21    $chaine1 = $mysqli->real_escape_string($_POST['chaine1']);
22  }
23  echo "Chaîne entrée par l'utilisateur : <code>".$_POST['chaine1']."</code><br
      />";
24  echo "Chaîne entrée par l'utilisateur échapée : <code>".$chaine1."</code><br/>
      ";
25
26  // Insertion de la chaîne dans la table Table1 (requête SQL) :
27  $requete = 'INSERT INTO Table1(chaine) VALUES ("'.$chaine1.'")';
28
29  echo "Requête exécutée :<br/><code>".$requete."</code><br/>";
30
31  $result = $mysqli->multi_query($requete) or die('Query failed : ' . mysql_error
      ()."<br/>");
32
33  // On ferme la connection
34  $mysqli->close();
35
36  outputFinFichierHTML5();
37 ?>
```

Code Source 4.11 : /filtrage/ex11-mySqlEscapeStringOutput.php (cf. Fig 4.17)

```php
1 <?php
2   include('./commonFunctions.php');
3   outputEnTeteHTML5('En sortie de BD', 'UTF-8', 'myStyle.css');
4
```

FIGURE 4.16 : Données dans la base après tentative d'injection *SQL* avec échappement

FIGURE 4.17 : Illustration du code source 4.11

```
5   echo "<h1>Application d'<code>htmlentities</code><br/>En sortie de BD</h1>";
6
7   // On se connecte au serveur de bases de données.
8   // Adapter le nom d'hote où se trouve le serveur mysql
9   // mysqli($sql_server_url, $sql_user, $sql_user_password, $database_name)
10  $mysqli = new mysqli('progweb', "testUser", "motdepasse", 'baseTest');
11
12  // Vérification de la connexion :
13  if (mysqli_connect_errno()) {
14      printf("Échec de la connexion : %s\n", mysqli_connect_error());
15      exit();
16  }
17  echo 'Connecté au serveur de bases de données mysql.<br/>';
18
19  // Insertion de la chaîne dans la table Table1 (requête SQL) :
20  $requete = 'SELECT * FROM Table1';
21
22  $result = $mysqli->query($requete) or die('Query failed : ' . mysql_error()."<
        br/>");
23
24  while ($ligneResReq = mysqli_fetch_array($result, MYSQL_ASSOC)){
25      echo "Donnée sortie de la table :<br/><code>".htmlentities($ligneResReq['
            chaine'], ENT_QUOTES, 'UTF-8')."</code><br/>";
26  }
27  // On ferme la connection
28  $mysqli->close();
29
30  outputFinFichierHTML5();
31 ?>
```

4.3 La fonction `filter_var`

4.3.1 Principe de la fonction *PHP* `filter_var`

La fonction *PHP* `filter_var` permet

1. de valider la forme d'une chaine de caractères attendue suivant son usage (exemple : adresse $e-$mail, *URL*, nombre réel, adresse *IP*, etc.).

2. de nettoyer une chaine de caractères attendue suivant son usage (élimination des caractères inattendus compte tenu du type de données (exemple : élimination d'un caractère inattendu $'@'$ dans un nombre entier).

le prototype de la fonction `filter_var` est :

```
mixed filter_var(mixed $variable, int $filter = FILTER_DEFAULT, mixed $options)
```

- La fonction retourne `false` en cas de données invalides avec échec du filtre, ou les données elles mêmes dans le cas de données valides, ou encore les données filtrées en cas de filtres de nettoyage.

- `$variable` est la valeur à filtrer ;

- `$filter` est le type de filtre. Bien qu'il soit en option et qu'il est une valeur par défaut définie dans la configuration du serveur (fichier `php.ini`), il est fortement conseillé de spécifier un, ou plusieurs, filtres)

- `$options` définit les options et/ou les *flags* du filtre, plus ou moins strictes (c'est à dire que ces filtres n'éliminent pas les memes caractères suivant les options choisies).

  En toute généralité, les options et les *flags* sont définis dans un tableau associatif (avec deux clés facultatives `'options'` et/ou `'flags'`) de tableaux associatifs chaqu'un de ces tableaux associatifs définissant les valeurs d'une ou plusieurs options (pour le tableau `$options['options']`) ou d'un ou plusieurs flags (pour le tableau `$options['flags']`).

  Partez pas ! Je mets quelques exemples ci-dessous...

Nous ne ferons pas ici une présentation exhaustive des utilisations des filtres ou des options. Pour celà voyez `php.net`. Nous donnons quelques exemples typiques.

4.3.2 Les filtres de Validation

Les filtres de validation sont les valeurs possibles du deuxième paramètre `filter` de la fonction `filter_var` qui commencent par `FILTER_VALIDATE_`. Le but d'un filtre de validation est de dire si une chaîne satisfait certaines condition ; si la forme de la chaîne est confrome à ce qu'on attend d'un certain type de données.

La liste n'est pas très longue. La plus grosse difficultés vient, comme d'habitude, des conversions automatiques entre formats de nombre et booléens, qui produisent des résultats conte-intuitifs qui peuvent conduire à des *bugs*.

4.3.2.a Le filtre `FILTER_VALIDATE_BOOLEAN`

Ce filtre 'admet pas d'option et admet `FILTER_NULL_ON_FAILURE` pour seul flag.
Retourne `TRUE` pour "1", "true", "on" et "yes". Retourne`FALSE` sinon.
Si le flag `FILTER_NULL_ON_FAILURE` est activé, `FALSE` n'est retourné que pour les valeurs "0", "false", "off", "no", "", et `NULL` est retourné pour les valeurs non-booléennes.

Ce filtre s'applique à une chaine de caractères et ne donne pas le bon résultat sur une variable de type booléen (car les booléens `FALSE` ou `TRUE` ne sont pas des chaines de caractères représentant un booléen) !

Code Source 4.12 :

```
1 if (!is_bool($value)) {
2     $value= filter_var($value, FILTER_VALIDATE_BOOLEAN, FILTER_NULL_ON_FAILURE);
3 }
```

4.3.2.b Le filtre `FILTER_VALIDATE_EMAIL`

Ce filtre valide une adresse $e-$mail. N'admet ni option, ni flag. Rien à signaler sur ce filtre.

4.3.2.c Le filtre `FILTER_VALIDATE_FLOAT`

Ce filtre valide un nombre décimal.

Le nombre flottant égal à zéro (`$x = 0`) est un nombre flottant valide, mais la fonction `filter_var` avec le filtre `FILTER_VALIDATE_FLOAT`, comme les données sont valides, va retourner la variable égale à 0, qui serait dans un test convertie en le booléen `false`). Le bon usage consiste à tester l'*identité* de la donnée retournée par `filter_var` sans conversion, en utilisant les opérateurs `===` (vrai si deux variables ont des valeurs égales et on même type) ou `!==` (vrai si deux variables ont des valeurs différentes ou sont de types différents).

Code Source 4.13 : /filtrage/ex12-filterVarValidateFloat.php

```
1  <!doctype html>
2  <html lang="fr">
3  <head>
4    <meta charset="UTF-8" />
5  <link rel="stylesheet" href="./myStyle.css" />
6    <title>Filtre VALDATE_FLOAT</title>
7  </head>
8  <body>
9    <h1>Tests de <code>filter_var</code><br/>avec filtre <code>
         FILTER_VALIDATE_FLOAT</code></h1>
10   <?php
11     $x = 0;
12     if (filter_var($x, FILTER_VALIDATE_FLOAT)) {
13       echo "$x est un float valide car ".htmlentities("filter_var($x,
             FILTER_VALIDATE_FLOAT)", ENT_QUOTES, "UTF-8")
14            ." vaut ";
15            var_dump(filter_var($x, FILTER_VALIDATE_FLOAT));
16            echo "<br/>";
```

```
17      } else {
18        echo "$x n'est pas un float valide car ".htmlentities("filter_var($x,
              FILTER_VALIDATE_FLOAT)", ENT_QUOTES, "UTF-8")
19            ." vaut ";
20          var_dump(filter_var($x, FILTER_VALIDATE_FLOAT));
21           echo "<br/>";
22      }
23      if (filter_var($x, FILTER_VALIDATE_FLOAT)!== false) {
24        echo "$x est bien sûr un float valide ! <br/>";
25      } else {
26        echo "$x n'est pas un float valide <br/>";
27      }
28    ?>
29  </body>
30  </html>
```

```
if (filter_var($x, FILTER_VALIDATE_FLOAT)!== false) {
    echo "$x est un float valide";
} else {
    echo "$x n'est pas un float valide";
}
```

4.3.2.d Le filtre `FILTER_VALIDATE_INT`

Même remarque que pour les nombres réels à propos de l'identité des variables à tester avec `===` ou `!==`, en raison du problème d'un nombre égal à 0 (zéro) qui, converti en booléen, donnerait `FALSE`.

Il y a des options permettant de tester un intervalle et des flags autorisant les écritures octales (style `0123`) ou hexadécimales (style `0x2c3f`).

4.3.2.e Le filtre `FILTER_VALIDATE_URL`

Ne fonctionne pas avec les *URLs* internationalisées (c'est à dire avec des caractères non *ASCII*). Ces *URLs* doivent etre échappées auparavant.

4.3.2.f Le filtre `FILTER_VALIDATE_IP`

Valide une adresse *IP* ? Admet des flags pour autoriser de manière sélective une adresse *IPV4*, *IPV6*, ou avec des plages d'adresses réservées.

4.3.3 Les filtres de Nettoyage

Les filtres de nettoyage permettent d'appliquer un traitement à une chaîne pour la rendre conforme à la forme attendue des données. C'est aussi une manière de sécuriser des données incorrectes sans renvoyer les données vers l'utilisateur. Les filtres de nettoyage sont les valeurs possibles du deuxième paramètre de `filter_var` qui commencent par `FILTER_SANITIZE`, ce qui signifie en anglais "rendre raisonnable", "rendre hygiennique" ou "ramener à la raison".

Le fait d'appliquer un filtre de nettoyage améliore la sécurité mais, en général, cela ne permet pas de rendre coorectes des données incorrectes. Cela permet juste de rendre les données "raisonnables".

Par exemple, le filtre `FILTER_SANITIZE_NUMBER_INT` Supprime tous les caractères sauf les chiffres, et les signes plus et moins. Cela n'empeche pas que si la donnée en entrée n'est pas un nombre, le programme risque de ne pas fonctionner correctement. Le filtre `FILTER_SANITIZE_STRING` permet de supprimer ou d'encoder diiférents jeux de caractères spéciaux (suivant la valeur du *flag*).

4.3.4 Le filtre personnalisé `FILTER_CALLBACK`

Ce filtre est un exemple dans lequel on va fournir à `filter_var` une *fonction callback* personnalisée, que l'on codera soi-même, et qui sera appelée automatiquement `filter_var` pour valider ou nettoyer une chaîne.

Voici un exemple de validation par expression régulière (voir plus loin dans ce chapitre).

Code Source 4.14 : /filtrage/ex13-filterVarCallback.php

```
1  <!doctype html>
2  <html lang="fr">
3  <head>
4    <meta charset="UTF-8" />
5  <link rel="stylesheet" href="./myStyle.css" />
6    <title>Post un Nom</title>
7  </head>
8  <body>
9    <h1>Tests de <code>filter_var</code><br/>avec filtre <code>FILTER_CALLBACK</
       code></h1>
10   <?php
11     function numeroTelephone($tel){
12          if (preg_match('/^(((\+33 )|0){1}[0-9]{1}([ ]{0,1}[0-9]{2}){4})$/', $tel
                ){
13         return $tel;
14          }else{
15          return false;
16          }
17     }
18
19          $telephone = "+33 1 23 45 689";
20          if (filter_var($telephone, FILTER_CALLBACK,
21                array('options' => 'numeroTelephone'))!==false){
22          echo "numéro de téléphone valide.<br/>";
23          }else{
24          echo "numéro de téléphone invalide.<br/>";
25          }
26   ?>
27 </body>
28 </html>
```

4.4 Expressions régulières

Les expressions régulières sont un moyen de vérifier d'une manière très générale qu'une chaîne de caractère a une certaine forme.

L'extension PRCE en PHP permet, via des fonctions comme `preg_match`, de vérifier qu'une chaîne est conforme à une expression régulière. Un bon tutorial sur la syntaxe et l'utilisation

des expressions régulières en *PHP* se trouve à l'adresse suivante :

`http://php.net/manual/en/book.pcre.php`

Exemples.

Par exemple, pour valider un nom ou un prénom entre 1 et 50 caractères alphabétiques français avec des espaces ou traits d'unions, on peut utiliser quelque-chose du genre :

```
/^(([a-zA-ZàâéèêôùûçÀÂÉÈÊÔÙÛÇäöëüÄÖËÜ](\-|( )*)){1,50})$/
```

Notez que pour que le filtrage puisse servir à éviter les injections, il ne faut pas omettre le^ au début et le $ à la fin de l'expression pour que l'expression régulière représente l'ensemble de l'input utilisateur et non pas simplement une sous-chaîne.

Le filtrage d'une chaine de caractères accentués (au moins en français) peut se faire un plus sytématiquement après échappement par :

```
 '/^([a-zA-Z]'
.'|(\&[a-zA-Z]grave\;)|(\&[a-zA-Z]acute\;)|(\&[a-zA-Z]circ\;)|(\&[a-zA-Z]uml\;)'
.'|(\&[a-zA-Z]cedil\;)|(\&[a-zA-Z][a-zA-Z]lig\;)|(\&szlig\;)|(\&[a-zA-Z]tilde\;)'
.'|(\-)|( )|(\&\#039\;)|(\&quot\;)|(\.))+$/'
```

Le filtrage avec `preg_match` se fait alors par un code du genre :

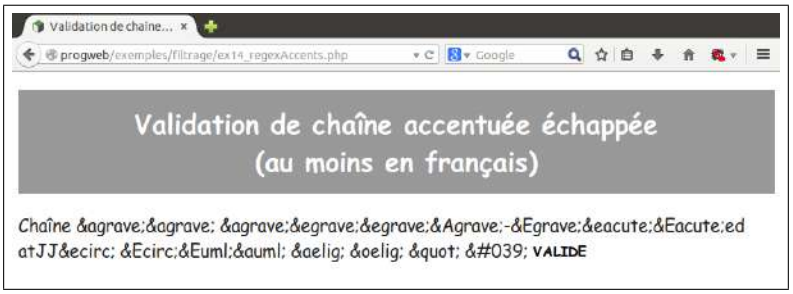


FIGURE 4.18 : Illustration du code source 4.15

Code Source 4.15 : /filtrage/ex14-regexAccents.php (cf. Fig 4.18)

```
1  <!doctype html>
2  <html lang="fr">
3  <head>
4    <meta charset="UTF-8" />
5  <link rel="stylesheet" href="./myStyle.css" />
6    <title>Validation de chaîne accentuée</title>
7  </head>
8  <body>
9    <h1>Validation de chaîne accentuée échappée<br/> (au moins en français)</h1>
10   <?php
11     $chaine = htmlentities("àà àèèÀ-ÈéÉed atJJê ÊËä æ œ \" ' .", ENT_QUOTES);
12     echo "Chaîne ".htmlentities($chaine, ENT_QUOTES);
13         if (preg_match('/^([a-zA-Z]'
14         .'|(\&[a-zA-Z]grave\;)|(\&[a-zA-Z]acute\;)|(\&[a-zA-Z]circ\;)|(\&[a-zA-Z
               ]uml\;)' // caractères accentués
15         .'|(\&[a-zA-Z]cedil\;)|(\&[a-zA-Z][a-zA-Z]lig\;)|(\&szlig\;)|(\&[a-zA-Z]
               tilde\;)' // caractères accentués
```

```
16          . '/(\-)/( )/(\&\#039\;)/(\&quot\;)/(\.))+$/',    // Espaces, trait d'
                union, simples et doubles quotes
17          $chaine)){
18          echo "<strong>valide</strong>";
19          }else{
20          echo "<strong>invalide</strong>";
21          }
22    ?>
23 </body>
24 </html>
```

Pour un numéro de téléphone français sous la forme de 10 chiffres commençant par 0 et par groupes de deux séparés par des espaces :

```
/^(0{1}[0-9]{1}( [0-9]{2}){4})$/
```

En option, avec un +33 devant pour les appels internationaux :

```
/^(((\+33 )|0){1}[0-9]{1}( [0-9]{2}){4})$/
```

En option, avec un +33 devant pour les appels internationaux et les espaces entre chiffres optionnels :

```
/^(((\+33 )|0){1}[0-9]{1}([ ]{0,1}[0-9]{2}){4})$/
```

En option, avec la possibilité de ne pas laisser d'espaces ou de mettre des tires ou des points entre les groupes de chiffres (exemples : `+33-1.02-0304.05`) :

```
/^(((\+33(| |\-|\.))|0){1}[0-9]{1}((| |\-|\.)[0-9]{2}){4})$/
```

Chapitre 5

Conception Objet, Gestion des Erreurs

5.1 *Plain Old PHP Objects* (*Pattern POPO*)

Le *Pattern Plain Old PHP Object* (*POPO*) consiste à créer une classe qui a pour **unique responsabilité de stocker les valeurs des attributs** d'un objet (en général un objet de la logique métier). Une telle classe est similaire à une *structure* en langage *C* :

- Elle expose tous ses attributs en public ;
- Elle n'a pas d'autre comportement que de permettre la création d'instances sans aucune vérification.

L'objectif est de pouvoir très facilement passer des instances d'une classe à l'autre, et de rendre la manipulation des attributs très facile, sans passer par des *setters/getters* qui alourdissent le code. Voici la définition du modèle *POPO* d'une adresse :

Code Source 5.1 : /forms2/classes/Adresse.php

```
1  <?php
2  namespace CoursPHP\Metier;
3  /** @brief Contient les données de l'adresse d'une personne
4   (qui peut être un client, un employé, un fournisseur, etc...) */
5  class Adresse {
6    /** id unique de l'adresse */
7    public $idAdresse;
8    /** id unique de l'agrégat Personne à qui correspond l'adresse */
9    public $idPersonne;
10   /** Numéro dans la rue, ne doit pas être null mais peut être vide */
11   public $numeroRue;
12   /** Nom de la rue, ne doit pas être null mais peut être vide */
13   public $rue;
14   /** Complément (lieu dit, etc. ne doit pas être null mais peut être vide */
15   public $complementAddr;
16   /** code postal */
17   public $codePostal;
18   /** nom de la ville. ne doit pas être null mais peut être vide*/
19   public $ville;
20   /** nom du pays. ne doit pas être null mais peut être vide*/
21   public $pays;
22
23   /** @brief Génère un Id de 10 chiffres hexa aléatoires (soit 5 octets).
```

```php
24      * Dans la pratique, préférer une implémentation d'UUID (ID unique universel).
25      * Un UUID permet de (statistiquement) garantir l'absence de collision
26      * entre notre ID unique de ressource et l'ID d'une autre ressource
27      * quelconque de n'importe quelle application. Cf. la fonction PHP uniqid()*/
28     public static function generateRandomId()
29     {
30       // Génération de 5 octets (pseudo-)aléatoires codés en hexa
31       $cryptoStrong = false; // Variable pour passage par référence
32       $octets = openssl_random_pseudo_bytes(5, $cryptoStrong);
33       return bin2hex($octets);
34     }
35
36     /** @brief Constructeur : initialise les attributs à partir des paramètres.
37      * Les paramètres correspondent aux valeurs à mettre dans les attributs */
38     public function __construct($idAdresse, $idPersonne, $numeroRue, $rue,
           $complementAddr,
39                                 $codePostal, $ville, $pays) {
40       $this->idAdresse = $idAdresse;
41       $this->idPersonne = $idPersonne;
42       $this->numeroRue = $numeroRue;
43       $this->rue = $rue;
44       $this->complementAddr = $complementAddr;
45       $this->codePostal = $codePostal;
46       $this->ville = $ville;
47       $this->pays = $pays;
48     }
49
50     /** Retourne une copie de l'instance.
51      * Utilisé avant d'appeler une méthode qui modifie une instance */
52     public function clone(){
53       return new self($this->idAdresse, $this->idPersonne, $this->numeroRue,
54                       $this->rue, $this->complementAddr,
55                       $this->codePostal, $this->ville, $this->pays);
56     }
57
58     /** @brief construit une instance par défaut
59      * L'ID est généré aléatoirement.
60      * @param $idPersonne ID de la personne qui possède cette adresse */
61     public static function getDefaultInstance($idPersonne, $idAdresse){
62       $adresse = new self($idAdresse, $idPersonne,
63                           "", "", "", "", "", "France");
64       return $adresse;
65     }
66 }
67 ?>
```

Les données membres étant publiques et les constructeurs n'effectuant aucun test, les autres classes de l'application, qui manipulent les objets *POPO*, doivent prendre en charge la sécurité et la garantie de la logique métier.

5.2 Utilitaires pour le filtrage

Nous distinguons **deux types de filtrage des données :**

1. **Le filtrage pour la sécurité**, qui sera géré par des fonctions de validation ou de nettoyage systématique (comme `filter_var`) ou, dans le cas de la couche de persistance basée sur *PDO*, par la préparation systématique des requêtes.

2. **Le filtrage pour la cohérence des données**, qui nécessite généralement un connaissance de la sémantique des objets métiers et de leurs attributs (*Logique Métier*), est se fonde le plus souvent sur des tests d'expressions régulières. Dans notre implémentation, ce filtrage sera réalisé dans une classe de validation et de fabrique d'instances des classes de représentation des données métier.

5.2.1 Prévention des injections *HTML*

Nous proposons ici un utilitaire un peu général qui définit des politiques de filtrage pour éviter les injections *HTML*. Différentes politiques sont prévues :

1. Aucun filtrage ;

2. Élimination des balises (voir `filter_var`) par un filtre `FILTER_SANITIZE_STRING` (avec ou sans échappement des simples ou doubles quotes) ;

3. Échappement systématique (voir `htmlentities`) de toutes les entités *HTML* (y compris les caractères accentués et autres caractères plus ou moins inoffensifs) ;

4. Échappement uniquement des caractères spéciaux *HTML* (voir `htmlspecialchars`).

Deux méthodes permettent respectivement de filtrer la chaîne et d'inverser l'échappement pour restaurer la chaîne (dans la mesure où les caractères n'auraient pas été éliminés par un filtre de nettoyage (de type `SANITIZE`). Le but de l'inversion de l'échappement est de permettre, si besoin, de rétablir la chaîne d'origine, par exemple pour que l'utilisateur puisse la modifier dans un formulaire.

Code Source 5.2 : /forms2/classes/ValidationUtils.php

```php
<?php
namespace CoursPHP\Controleur;
/** @brief Permet la validation des données pour éviter les injections HTML
 * Typiquement, les données reçues via $_REQUEST sont filtrées avant d'être
 * affichées dans une page ou re-soumises dans un formulaire.
 * Plusieurs politiques de filtrage (nettoyage ou échappement) sont prévues. */
class ValidationUtils {
  // Politiques de nettoyage et d'échappement :

  /** 1) Ne rien filtrer ni changer */
  const SANITIZE_POLICY_NONE = 0;
  /** 2) Supprimer les balises HTML uniquement, mais ne pas échapper.
    * Laisser les quotes " et apostrophes ' inchangées */
  const SANITIZE_POLICY_DISCARD_HTML_NOQUOTE = 1;
  /** 3) Supprimer les balises HTML et échapper les quotes " et '.
   * Laisser les quotes et apostrophes inchangées */
  const SANITIZE_POLICY_DISCARD_HTML = 2;
  /** 4) Échapper toutes les caractères spéciaux HTML (<, >, ", etc.) */
  const SANITIZE_POLICY_ESCAPE_SPECIAL_CHARS = 3;
  /** 5) Échapper toutes les entités HTML (y compris les accents, etc.) */
  const SANITIZE_POLICY_ESCAPE_ENTITIES = 4;
```

```
22
23    /** @brief Méthode de Nettoyage et/ou d'échappement
24     * @param $chaine {inOut} la chaîne de caractères à filtrer
25     * @param $policy la politique de filtrage (voir constantes de classe)
26     * @return false en cas d'échec ($chaine n'est pas une chaîne) */
27    private static function filterMethod(&$chaine, $policy){
28      // Si la chaine n'est pas définie, on met une chaîne vide
29      $chaine = isset($chaine) ? $chaine : "";
30      switch($policy){
31        case self::SANITIZE_POLICY_DISCARD_HTML_NOQUOTE:
32          // Supprimer les balises uniquement, mais ne pas échapper
33          $chaine = filter_var($chaine, FILTER_SANITIZE_STRING,
34                                        FILTER_FLAG_NO_ENCODE_QUOTES);
35          break;
36        case self::SANITIZE_POLICY_DISCARD_HTML:
37          // Supprimer les balises et échapper les quotes " et '.
38          $chaine = filter_var($chaine, FILTER_SANITIZE_STRING, ENT_QUOTES);
39          break;
40        case self::SANITIZE_POLICY_ESCAPE_SPECIAL_CHARS:
41          // Échapper toutes les caractères spéciaux HTML (<, >, ", etc.)
42          $chaine = htmlspecialchars($chaine, ENT_QUOTES, 'UTF-8');
43          break;
44        case self::SANITIZE_POLICY_ESCAPE_ENTITIES:
45          // Échapper toutes les entités HTML (y compris les accents, etc.)
46          $chaine = htmlentities($chaine, ENT_QUOTES, 'UTF-8');
47          break;
48        default: // SANITIZE_POLICY_NONE: rien à faire
49      }
50      return $chaine === false ? false : true;
51    }
52
53    /** @brief Méthode d'Inversion d'échappement
54     * @param $chaine la chaîne de caractères à restaurer suite à un échappement
55     * @param $policy la politique de filtrage (voir constantes de classe) */
56    private static function filterMethodReverse(&$chaine, $policy){
57      // Si la chaine n'est pas définie, on met une chaîne vide
58      $chaine = isset($chaine) ? $chaine : "";
59      switch($policy){
60        case self::SANITIZE_POLICY_DISCARD_HTML:
61          // On restaure justes les simples et doubles quotes qui sont html-encodé
                es
62          $tmp = preg_replace('/^\&quot\;$/', "\"", isset($chaine) ? $chaine : "")
                ;
63          $chaine = preg_replace('/^\&39\;$/', "'", $tmp);
64          break;
65        case self::SANITIZE_POLICY_ESCAPE_SPECIAL_CHARS:
66          // On inverse l'encodage des caractères spéciaux HTML
67          $chaine = htmlspecialchars_decode($chaine, ENT_QUOTES, 'UTF-8');
68          break;
69        case self::SANITIZE_POLICY_ESCAPE_ENTITIES:
70          // On inverse l'encodage des entités HTML
71          $chaine = htmlentities_decode($chaine, ENT_QUOTES, 'UTF-8');
72          break;
73        default: // SANITIZE_POLICY_DISCARD_HTML_NOQUOTE: Rien à restaurer
74                 // SANITIZE_POLICY_NONE: rien à faire
75      }
```

```
76    }
77
78    /** @brief Méthode de (Nettoyage et/ou d'échappement)
79     *                    et/ou d'inversion d'échappement
80     * @param $chaine la chaîne de caractères à filtrer
81     * @param $reversed true s'il faut appliquer une inversion d'échappement
82     *                  false s'il faut appliquer un nettoyage et/ou échappement
83     * @param $policy la politique de filtrage (voir constantes de classe) **/
84    public static function filterString(&$chaine, $reversed, $policy){
85      if (!isset($policy)){
86        throw new \Exception("Politique de filtrage non définie");
87      }
88      return $reversed ? self::filterMethodReverse($chaine, $policy) :
89            self::filterMethod($chaine, $policy);
90    }
91  }
92  ?>
```

La classe `AdresseValidation` permet, en utilisant une politique de filtrage de la classe `ValidationUtils`, de fitrer les attributs d'une adresse.

Les isntances d'`Adresse` suivent le modèle *POPO*. Une méthode prend en argument une instance d'`Adresse`, et une autre méthode prend en argument un tableau associatif dont les clefs correspondent aux noms d'attributs d'`Adresse` (typiquement, `$inputArray` sera le tableau `$_POST` ou `$_REQUEST` pour créer une instance à partir des données saisies dans un formulaire.

Code Source 5.3 : /forms2/classes/AdresseValidation.php

```
1   <?php
2   namespace CoursPHP\Metier;
3   use \CoursPHP\Controleur\ValidationUtils as ValidationUtils; // Raccourci classe
4
5   /** @brief Permet la validation initiale des données d'une adresse.
6    * Typiquement, les données qui viennent du client reçues (via $_REQUEST...)
7    * Nettoyage de toutes les chaînes. Initialisation des inputs inexistants */
8   class AdresseValidation {
9     /** @brief Validation et initialisation des données d'une instance Adresse
10     * à partir d'une instance de POPO Adresse
11     * @param $adresse instance d'Adresse
12     * @param $reversed  true pour appliquer la méthode d'inversion d'échappement
13     *                   false pour la méthode de nettoyage et/ou d'échappement
14     * @param $policy l'une des politiques définies par ValidationUtils **/
15    public static function filterAdresse($adresse, $reversed, $policy){
16      ValidationUtils::filterString($adresse->idAdresse, $reversed, $policy);
17      ValidationUtils::filterString($adresse->idPersonne, $reversed, $policy);
18      ValidationUtils::filterString($adresse->numeroRue, $reversed, $policy);
19      ValidationUtils::filterString($adresse->rue, $reversed, $policy);
20      ValidationUtils::filterString($adresse->complementAddr, $reversed, $policy);
21      ValidationUtils::filterString($adresse->codePostal, $reversed, $policy);
22      ValidationUtils::filterString($adresse->ville, $reversed, $policy);
23      ValidationUtils::filterString($adresse->pays, $reversed, $policy);
24    }
25
26    /** @brief Validation et initialisation des données d'une adresse
27     * à partir des données reçues dans les tableau associatif
28     * (typiquement $_REQUEST ou $_POST).
29     * @param $inputArray tableau associatif dont les clefs contiennent les noms
```

```
30      *                    des attributs d'Adresse
31      * @param $adresse {inOut}  Instance d'Adresse à créer ou initialiser
32      * @param $policy l'une des politiques définies par ValidationUtils **/
33    public static function validationInput($inputArray, &$adresse, $policy){
34      // Si $adresse n'est pas une instance d'Adresse, on crée une instance :
35      if (!is_object($adresse) || !preg_match("/Adresse$/", get_class($adresse))){
36        $adresse = \CoursPHP\Metier\Adresse
37                        ::getDefaultInstance($inputArray['idPersonne'],
38                                             $inputArray['idAdresse']);
39      }
40      // Initialisation des attributs
41      $adresse->idAdresse = $inputArray['idAdresse'];
42      $adresse->idPersonne = $inputArray['idPersonne'];
43      $adresse->numeroRue =$inputArray['numeroRue'];
44      $adresse->rue = $inputArray['rue'];
45      $adresse->complementAddr = $inputArray['complementAddr'];
46      $adresse->codePostal = $inputArray['codePostal'];
47      $adresse->ville = $inputArray['ville'];
48      $adresse->pays = $inputArray['pays'];
49      // Filtrage des attributs avec la politique
50      self::filterAdresse($adresse, false, $policy);
51    }
52  }
53  ?>
```

La classe `AdresseView` permet de générer le code *HTML* d'une instance d'`Adresse`, en appliquant une politique de filtrage (par défaut `htmlentities`) avant de générer le code *HTML*. La chaîne retournée peut alors être affichée (pour la politique par défaut) sans risque d'injection *HTML*.

Code Source 5.4 : /forms2/classes/AdresseView.php

```
1   <?php
2   namespace CoursPHP\Vue;
3   /** @brief Implémente la génération d'HTML pour afficher une Adresse dans une
        vue
4     *  dans un navigareur.
5     * Implémente aussi des utilitaires de conversion à partir d'une Adresse
6     * pour obtenir facilement le code HTML pour afficher une Adresse. */
7   class AdresseView {
8     /** @brief Méthode de génération de code HTML. Permet d'afficher une adresse.
9      * Les attributs sont échappés pour éviter tout risque d'injection HTML
10     * @param $adresse l'instance d'adresse à afficher
11     * @param $sanitizePolicy Politique pour encoder ou échapper les attributs
12     *                        (Voir la classe ValidationUtils)
13     * @return  le code HTML pour un affichage développé. */
14    public static function getHtmlDevelopped($adresse,
15                        $sanitizePolicy = \CoursPHP\Controleur\ValidationUtils
16                                          ::SANITIZE_POLICY_ESCAPE_ENTITIES){
17      if (\CoursPHP\Metier\AdresseValidation::filterAdresse(
18                                  $adresse, false, $sanitizePolicy) === false){
19        return "Adresse incorrecte";
20      }
21      $htmlCode = "<strong>Adresse : </strong><br/>\n";
22      $htmlCode .= $adresse->numeroRue;
23      if (!empty($adresse->numeroRue))
24        $htmlCode .= ", ";
```

```
25      $htmlCode .= $adresse->rue;
26      if (!empty($adresse->rue))
27        $htmlCode .= "<br/>";
28      $htmlCode .= $adresse->complementAddr;
29      if (!empty($adresse->complementAddr))
30        $htmlCode .= "<br/>";
31      $htmlCode .= $adresse->codePostal." ";
32      $htmlCode .=  $adresse->ville;
33      if (!empty($adresse->ville) && !empty($adresse->pays))
34        $htmlCode .= "<br/>";
35      $htmlCode .= $adresse->pays."<br/>";
36
37      return $htmlCode;
38    }
39
40    /** @brief Méthode de génération d'HTML. Permet d'afficher une adresse.
41     * Les attributs sont échappés pour éviter tout risque d'injection HTML
42     * @param $adresse l'instance d'adresse à afficher
43     * @param $sanitizePolicy politique de nettoyage/échappement des attributs
44     * @return le code HTML pour un affichage compact. */
45    public static function getHtmlCompact($adresse,
46                            $sanitizePolicy = \CoursPHP\Controleur\ValidationUtils
47                                              ::SANITIZE_POLICY_ESCAPE_ENTITIES){
48      if (\CoursPHP\Metier\AdresseValidation::filterAdresse(
49                                    $adresse, false, $sanitizePolicy) === false){
50        return "Adresse incorrecte";
51      }
52      $htmlCode = "";
53      $htmlCode .= $adresse->numeroRue;
54      if (!empty($adresse->numeroRue))
55        $htmlCode .= ", ";
56      $htmlCode .= $adresse->rue;
57      if (!empty($adresse->rue))
58        $htmlCode .= ", ";
59      $htmlCode .= $adresse->complementAddr;
60      if (!empty($adresse->complementAddr))
61        $htmlCode .= ", ";
62      $htmlCode .= $adresse->codePostal." ";
63      $htmlCode .=  $adresse->ville;
64      if (!empty($adresse->ville) && !empty($adresse->pays))
65        $htmlCode .= ", ";
66      $htmlCode .= $adresse->pays;
67
68      return $htmlCode;
69    }
70  } // end of class AdresseView
71  ?>
```

Voici un fichier de test, qui affiche une instance d'adresse en utilisant différentes politiques de filtrage. Le code *HTML* d'affichage de l'instance, qui apparaît ci-dessous, illustre les différentes politiques de filtrage/nettoyage.

Code Source 5.5 : /forms2/testFiltrageHTML.php (cf. Fig 5.1)

```
1  <?php
2    require_once(dirname(__FILE__).'/classes/VueHtmlUtils.php');
3    require_once(dirname(__FILE__).'/classes/Adresse.php');
```

FIGURE 5.1 : Illustration du code source 5.5

```php
require_once(dirname(__FILE__).'/classes/AdresseValidation.php');
require_once(dirname(__FILE__).'/classes/ValidationUtils.php');
require_once(dirname(__FILE__).'/classes/AdresseView.php');
// Header HTML5
echo CoursPHP\Vue\VueHtmlUtils::enTeteHTML5('Filtrage d\'une Adresse',
                                            'UTF-8', 'myStyle.css');
echo "<h1>Filtrage, nettoyage et échappement d'une Adresse</h1>";
// Création d'une instance via le constructeur de POPO
$adresse = new CoursPHP\Metier\Adresse(
                "54522f6ab1", "9c9efcda32", "2 bis",
                "Rue de l'avenir < Companie", // avec un caractère '<' !!!
                "Résid. \"Les Flots Bleus\"", "63000", "Clermont-Ferrand", "");

// Politique par défaut : htmlentities
// (on clone pour préserver l'adresse d'origine)
echo "<p>ESCAPE ENTITIES : <br/>\n<strong> "
                .CoursPHP\Vue\AdresseView::getHtmlCompact($adresse->clone())
                ."\n</strong></p>\n";
// Politique special chars, NO_ENCODE_QUOTES (on clone...)
echo "<p>ESCAPE SPECIAL CHARS, NO_ENCODE_QUOTES : <br/>\n<strong> "
                .CoursPHP\Vue\AdresseView::getHtmlCompact($adresse->clone(),
                                        \CoursPHP\Controleur\ValidationUtils
                                        ::SANITIZE_POLICY_ESCAPE_SPECIAL_CHARS)
                        ."\n</strong></p>\n";
// Politique special chars, NO_ENCODE_QUOTES (on clone...)
echo "<p>DISCARD SPECIAL CHARS, NO_ENCODE_QUOTES : <br/>\n<strong> "
        .CoursPHP\Vue\AdresseView::getHtmlCompact($adresse->clone(),
                                        \CoursPHP\Controleur\ValidationUtils
                                        ::SANITIZE_POLICY_DISCARD_HTML_NOQUOTE)
                        ."\n</strong></p>\n";
// Politique special chars, ENCODE_QUOTES (on clone...)
echo "<p>DISCARD SPECIAL CHARS, ENCODE_QUOTES : <br/>\n<strong> "
        .CoursPHP\Vue\AdresseView::getHtmlCompact($adresse->clone(),
                                        \CoursPHP\Controleur\ValidationUtils
                                        ::SANITIZE_POLICY_DISCARD_HTML)
                        ."\n</strong></p>\n";
// Politique "ne rien faire"
```

```
41    echo "<p>NE RIEN FAIRE : <br/>\n<strong> "
42          .CoursPHP\Vue\AdresseView::getHtmlCompact($adresse,
43                                                    \CoursPHP\Controleur\ValidationUtils
44                                                    ::SANITIZE_POLICY_NONE)
45                                  ."\n</strong></p>";
46    echo CoursPHP\Vue\VueHtmlUtils::finFichierHTML5();
47  ?>
```

Voici la partie de la sortie standard (code *HTML*) du *CGI* qui montre le résultat des différentes formes de filtrage et d'échappement :

Sortie *HTML* du *CGI*

```
1  <p>ESCAPE ENTITIES : <br/>
2  <strong> 2 bis, Rue de l&#039;avenir &lt; Companie,
3              R&eacute;sid. &quot;Les Flots Bleus&quot;, 63000 Clermont-Ferrand
4  </strong></p>
5  <p>ESCAPE SPECIAL CHARS, NO_ENCODE_QUOTES : <br/>
6  <strong> 2 bis, Rue de l&#039;avenir &lt; Companie,
7              Résid. &quot;Les Flots Bleus&quot;, 63000 Clermont-Ferrand
8  </strong></p>
9  <p>DISCARD SPECIAL CHARS, NO_ENCODE_QUOTES : <br/>
10 <strong> 2 bis, Rue de l'avenir ,
11             Résid. "Les Flots Bleus", 63000 Clermont-Ferrand
12 </strong></p>
13 <p>DISCARD SPECIAL CHARS, ENCODE_QUOTES : <br/>
14 <strong> 2 bis, Rue de l&#39;avenir ,
15             Résid. &#34;Les Flots Bleus&#34;, 63000 Clermont-Ferrand
16 </strong></p>
17 <p>NE RIEN FAIRE : <br/>
18 <strong> 2 bis, Rue de l'avenir < Companie,
19             Résid. "Les Flots Bleus", 63000 Clermont-Ferrand
20 </strong></p>
```

5.2.2 Garantie de la Logique Métier

Nous proposons tout d'abord des méthodes génériques de test d'expressions régulières pour les langues d'Europe occidentale (jeu de caractères *Latin-1* tel que défini par la norme `ISO 8859-1`), ce qui inclut la langue française, avec ou sans chiffres. Notons que le jeu de caractères que nous utilisons pour l'encodage des caractères est toujours le standard *UTF-8*. Le je de caractères *Latin-1* est juste utilisé dans les expressions régulières.

Code Source 5.7 : /forms2/classes/ExpressionsRegexUtils.php

```
1  <?php
2  namespace CoursPHP\Metier;
3  /** @brief Classe de définitions d'Expressions Régulières d'usage général.
4   * Définit quelques expressions régulières utiles pour la langue locale supporté
       e
5   * et les routines de test sur une chaîne correspondant. */
6  class ExpressionsRegexUtils{
7    /** @brief : expression régulière pour la langue Française avec accents */
8    private static function getRegexLatin1(){
9      return '/^[a-zA-ZÀÁÂÃÄÅÆÇÈÉÊËÌÍÎÏÐÑÒÓÔÕÖ×ØÙÚÛÜÝÞßàáâãäåæçèéêëìíîïðñòóôõö÷øùú
         ûĀāüýþÿ \"\'\-\ ]*$/';
```

```
10      }
11
12      /** @brief : expression régulière pour la langue Française avec accents,
13       * et chiffres */
14      private static function getRegexLatin1WithNumbers(){
15       return '/^[0-9a-zA-ZÀÁÂÃÄÅÆÇÈÉÊËÌÍÎÏÐÑÒÓÔÕÖ×ØÙÚÛÜÝÞßàáâãäåæçèéêëìíîïðñòóô
             õö÷øùúûÃãüýþÿ \"\'\-\ ]*$/';
16      }
17
18      /** @brief : expression régulière pour la langue Française avec accents,
19       * chiffres et ponctuation */
20      private static function getRegexLatin1WithNumbersAndPunctuation(){
21       return '/^[\!\.\:\;\?\,0-9a-zA-ZÀÁÂÃÄÅÆÇÈÉÊËÌÍÎÏÐÑÒÓÔÕÖ×ØÙÚÛÜÝÞßàáâãäåæçèéêë
             ìíîïðñòóôõö÷øùúûÃãüýþÿ \"\'\-\ ]*$/';
22      }
23
24      /** @brief : Test expression régulière pour la langue Française avec accents
25       * avec conditions de longueur (par exemple pour un champ obligatoire) */
26      public static function isValidLatin1($chaine, $minLength, $maxLength){
27        return (isset($chaine) &&
28                strlen($chaine) >= $minLength && strlen($chaine) <= $maxLength
29                && preg_match(self::getRegexLatin1(), $chaine));
30      }
31
32      /** @brief : Test expression régulière pour la langue Française avec accents
33       * et chiffres, avec conditions de longueur
34       * (par exemple pour un champ obligatoire) */
35      public static function isValidLatin1WithNumbers($chaine,
36                                                      $minLength, $maxLength){
37        return (isset($chaine) &&
38                strlen($chaine) >= $minLength && strlen($chaine) <= $maxLength
39                && preg_match(self::getRegexLatin1WithNumbers(), $chaine));
40      }
41
42      /** @brief : Test expression régulière pour la langue Française avec accents,
43       * chiffres et ponctuation, avec conditions de longueur
44       * (par exemple pour un champ obligatoire) */
45      public static function isValidLatin1WithNumbersAndPunctuation($chaine,
46                                                      $minLength, $maxLength){
47        return (isset($chaine) &&
48                strlen($chaine) >= $minLength && strlen($chaine) <= $maxLength
49           && preg_match(self::getRegexLatin1WithNumbersAndPunctuation(), $chaine));
50      }
51
52      /** @brief : Test expression régulière passée en paramètre
53       * avec conditions de longueur (par exemple pour un champ obligatoire) */
54      public static function isValidString($chaine, $regExp, $minLength, $maxLength)
           {
55        return (isset($chaine) &&
56                strlen($chaine) >= $minLength && strlen($chaine) <= $maxLength
57                && preg_match($regExp, $chaine));
58      }
59  }
60  ?>
```

5.2.3 Fabrique d'`Adresse`

Nous présentons aussi une fabrique d'adresses qui construit des instances à partir de données issues d'un formulaire, tout en construisant un tableau d'erreurs en cas d'exception générée au niveau des setters. Le tableau d'erreurs `$dataErrors`, passé par références, contiendra des couples clé/valeur, dont la clé sera le nom de l'attribut de la classe `Adresse` et la valeur sera le messages de l'exception générée dans le *setter* de cet attribut. Si aucune exception n'est générée dans les *setter*, le tableau d'erreurs est vide.

Code Source 5.8 : /forms2/classes/AdresseFabrique.php

```
1  <?php
2  namespace CoursPHP\Metier;
3  /** @brief Implémente la construction d'une instance d'Adresse validée
4    * à partir des données, par exemple saisies dans un formulaire.
5    * Les erreurs générées dans les validateurs des attributs,
6    * qui reflètent la logique métier, qui sont reçues via des exceptions,
7    * sont accumulées dans un tableau associatif d'erreurs,
8    * pour générer le cas échéant une vue d'erreur. */
9  class AdresseFabrique {
10   /** @brief permet de valider l'ID unique.
11    * @param $idAdresse {inOut} identifiant unique à valider/réinitialiser
12    * @param $dataErrors {inOut} tabeau associatif de remontée d'erreurs. */
13   protected static function validateIdAdresse(&$idAdresse, &$dataErrors) {
14     if (!isset($idAdresse) || !preg_match("/^[0-9a-f]{10}$/", $idAdresse)){
15       $dataErrors['idAdresse'] = "Erreur, identifiant d'Adresse \""
16                                  .$idAdresse."\" incorrect";
17       $idAdresse = "";
18     }
19   }
20
21   /** @brief permet de valider l'ID unique de l'agrégat Personne.
22    * @param $idPersonne {inOut} identifiant unique à valider/réinitialiser
23    * @param $dataErrors {inOut} tabeau associatif de remontée d'erreurs. */
24   protected static function validateIdPersonne(&$idPersonne, &$dataErrors) {
25     if (!isset($idPersonne) || !preg_match("/^[0-9a-f]{10}$/", $idPersonne)){
26       $dataErrors['idPersonne'] = "Erreur, identifiant de Personne \""
27                                   .$idPersonne."\" incorrect";
28       $idPersonne = "";
29     }
30   }
31
32   /** @brief permet de valider le numéro dans la rue/place.
33    * @param $numeroRue {inOut} numéro à valider et/ou réinitialiser
34    * @param $dataErrors {inOut} tabeau associatif de remontée d'erreurs. */
35   protected static function validateNumeroRue(&$numeroRue, &$dataErrors) {
36     if (!ExpressionsRegexUtils::isValidLatin1WithNumbers($numeroRue, 0, 50)){
37       $dataErrors['numeroRue'] = "Erreur, le numéro de la rue \"".$numeroRue."\"
         "
38                                  ." devrait comporter au plus 50  caractères "
39                                  ."(alphabétiques, chiffres sans ponctuation)";
40       $numeroRue = "";
41     }
42   }
43
44   /** @brief permet de valider le nom de la rue/place/lieu-dit.
```

```
45      * @param $rue {inOut} nom à valider et, en cas d'erreur, réinitialiser
46      * @param $dataErrors {inOut} tabeau associatif de remontée d'erreurs. */
47     protected static function validateRue(&$rue, &$dataErrors) {
48       if (!ExpressionsRegexUtils::isValidLatin1WithNumbers($rue, 1, 150)){
49         $dataErrors['rue'] = "Erreur, le nom de la rue \"".$rue."\", "
50                              ."obligatoire devrait comporter au plus 150"
51                    ." caractères (alphabétiques, chiffres sans ponctuation)";
52         $rue = "";
53       }
54     }
55
56     /** @brief permet de valider le complément d'adresse.
57      * @param $complementAddr {inOut} chaîne à valider et/ou réinitialiser
58      * @param $dataError {inOut} tabeau associatif de remontée d'erreurs.  */
59     protected static function validateComplementAddr(&$complementAddr,
60                                                      &$dataErrors) {
61       if (!ExpressionsRegexUtils::isValidLatin1WithNumbersAndPunctuation(
62                                                        $complementAddr, 0, 150)){
63         $dataErrors['complementAddr'] = "Erreur, le Complément d'adresse \""
64                         .$complementAddr."\" devrait comporter au plus 150 "
65                         ."caractères (alphabétiques, chiffres ou ponctuation)";
66         $complementAddr = "";
67       }
68     }
69
70     /** @brief permet de valider le code postal.
71     * @param $codePostal {inOut} code à valider et/ou réinitialiser.
72      * @param $dataErrors {inOut} tabeau associatif de remontée d'erreurs.  */
73     protected static function validateCodePostal(&$codePostal, &$dataErrors) {
74       if (!ExpressionsRegexUtils::isValidString($codePostal, "/^[0-9]{5}$/", 5, 5)
           ){
75         $dataErrors['codePostal'] = "Erreur, code postal \"".$codePostal."\"
              invalide "
76                    ."invalide (code postal de france métropolitaine sans cedex ni B.P
                        .)";
77         $codePostal = "";
78       }
79     }
80
81     /** @brief permet de valider le nom de la ville.
82      * @param $ville {inOut} nom à valider et, en cas d'erreur, réinitialiser
83      * @param $dataErrors {inOut} tabeau associatif de remontée d'erreurs. */
84     protected static function validateVille(&$ville, &$dataErrors) {
85       if (!ExpressionsRegexUtils::isValidLatin1($ville, 1, 150)){
86         $dataErrors['ville'] = "Erreur, le nom de la ville \"".$ville."\", "
87                              ."obligatoire, devrait comporter au plus 150 "
88                              ."caractères (alphabétiques sans ponctuation)";
89         $ville = "";
90       }
91     }
92
93     /** @brief permet de valider le nom du pays
94      * @param $pays {inOut} nom à valider et, en cas d'erreur, réinitialiser
95      * @param $dataErrors {inOut} tabeau associatif de remontée d'erreurs. */
96     protected static function validatePays(&$pays, &$dataErrors) {
97       if (!ExpressionsRegexUtils::isValidLatin1($pays, 1, 100)){
```

```
98        $dataErrors['pays'] = "Erreur, le nom du Pays \"".$pays."\", obligatoire,
              "
99                            ."devrait comporter au plus 100 caractères"
100                           ." (alphabétiques sans ponctuation)";
101       $pays = "";
102     }
103   }
104
105   /** @brief Validation des attributs d'une instance d'Adresse
106    * (par exemple à partir des données saisies dans un formulaire via $_REQUEST)
107    * Pour chaque attribut de la classe, si une exception est générée lors de la
108    * validation de l'attribut, le message d'exception est ajouté dans un tableau
109    * associatif d'erreurs.
110    * Ces messages d'exception sont ainsi retournés vers le contrôleur.
111    * @param $dataErrors {out} tableau associatif d'erreurs
112    * @param $adresse {inOut} instance d'Adresse */
113   public static function validateInstance(&$dataErrors, &$adresse){
114     // Création du tableau des erreurs vide :
115     $dataErrors = array();
116     // Validation des différents attributs
117     self::validateIdAdresse($adresse->idAdresse, $dataErrors);
118     self::validateIdPersonne($adresse->idPersonne, $dataErrors);
119     self::validateNumeroRue($adresse->numeroRue, $dataErrors);
120     self::validateRue($adresse->rue, $dataErrors);
121     self::validateComplementAddr($adresse->complementAddr, $dataErrors);
122     self::validateCodePostal($adresse->codePostal, $dataErrors);
123     self::validateVille($adresse->ville, $dataErrors);
124     self::validatePays($adresse->pays, $dataErrors);
125   }
126
127   /** @brief Obtention d'une instance validée de la classe Adresse
128    * (par exemple à partir des données saisies dans un formulaire).
129    * Cette méthode attend un tableau associatif (typiquement $_REQUEST).
130    * Les valeurs du tableau sont nettoyées ou échappées
131    * par une politique de filtrage définie dans ValidationUtils.
132    * Un paramètre passé par référence retourne les messages d'exceptions.
133    * La méthode crée une instance et valide ses attributs avec
134    * self::validateInstance().
135    * Ces messages d'exception sont ainsi retournés vers le contrôleur.
136    * @param $dataErrors {out} tableau associatif d'erreurs
137    * @param $inputArray tableau associatif dont les clefs contiennent les noms
138    *                    des attributs d'Adresse.
139    *                    Passage par référence pour éviter une recopie.
140    * @param $policy = SANITIZE_POLICY_DISCARD_HTML politique de nettoyage
141    * @return une instance d'Adresse validée
142    * @see AdresseValidation
143    * @see ValidationUtils */
144   public static function getValidInstance(&$dataErrors, &$inputArray,
145                         $policy = \CoursPHP\Controleur\ValidationUtils
146                               ::SANITIZE_POLICY_DISCARD_HTML_NOQUOTE){
147     // Construction d'une instance filtrée suivant la politique
148     AdresseValidation::validationInput($inputArray,$adresse, $policy);
149     // Validation suivant la logique métier
150     self::validateInstance($dataErrors, $adresse);
151     return $adresse;
152   }
```

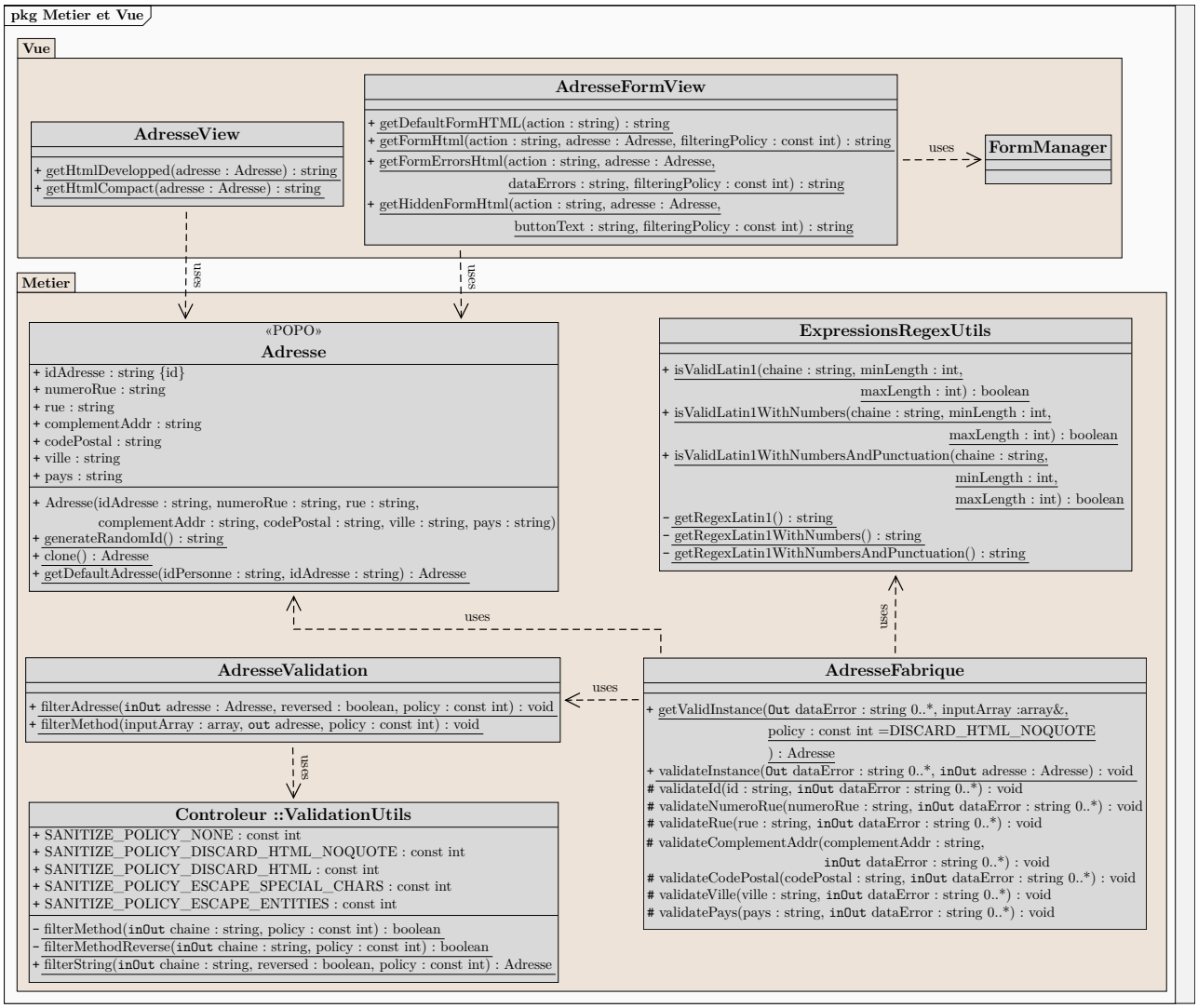
```
153 }
154 ?>
```

Dans ce chapitre, nous proposons une conception de classes métiers et classes d'utilitaires pour générer des vues *HTML* comportatnt des formulaires. La gestion des données saisies par l'utilisateur nous conduit, au vu du chapitre 4, à gérer plus rigoureusement le filtrage. Nous nous appuierons pour cela sur les utilitaires présentés dans la partie 5.2.

5.3 Modélisation : Diagrammes de Classes

Nous reprenons la classe `Adresse` de la partie 5.1, mais nous ajoutons des classes utilitaires pour le filtrage (classe `ExpressionsRegexUtils`) et la génération du code *HTML* de formulaires (la classe `AdresseFormView` et la classe utilitaire générale `FormManager`).

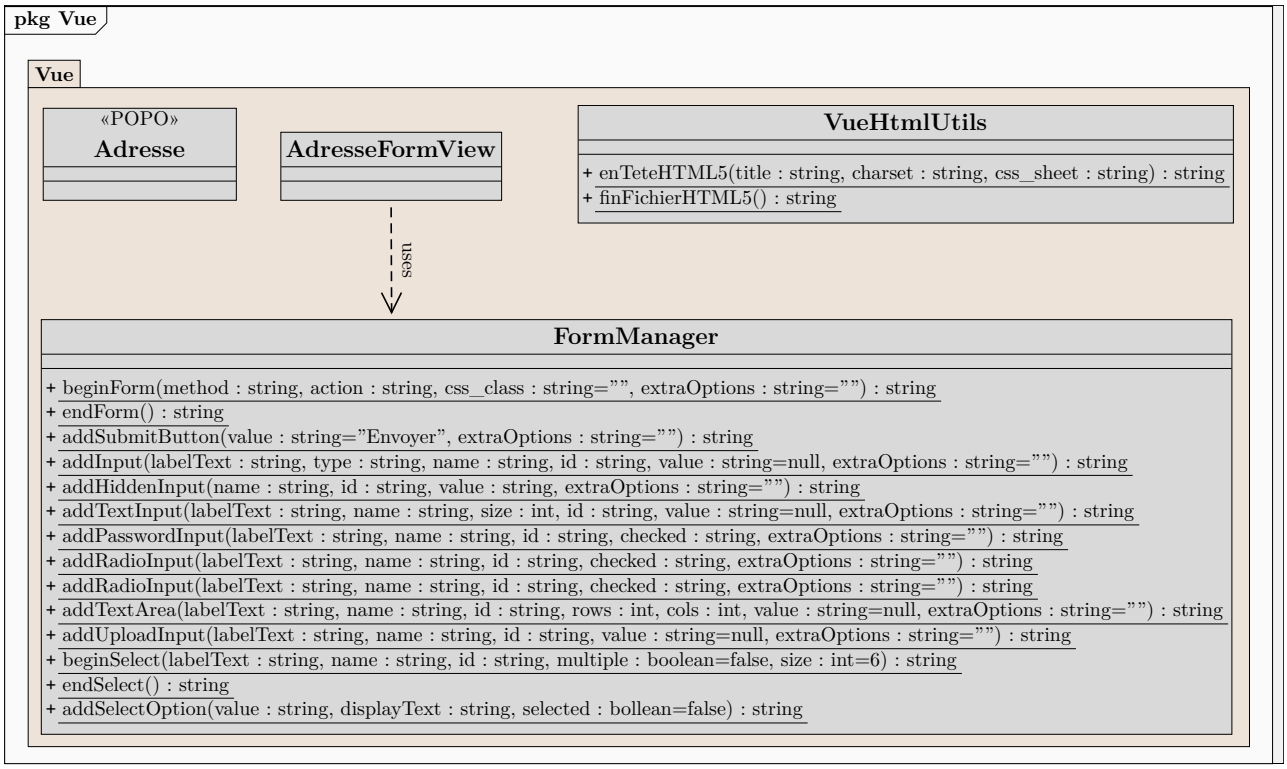
La classe `FormManager` permet la génération de code *HTML* pour chaque *input*, *select*, ainsi que la balise `<form>` elle même, ou encore un bouton *submit*.



Diag 2. Diagramme de Classes des Package `Metier` et `Vue`

Notez que ce diagramme ne fait pas apparaître l'agrégat `Personne`, ni les classes fabrique et

validation associées, qui sont très similaires à celles concernant les adresses. Par ailleurs, la classe `Adresse` est aussi dottée d'un attribut `idPersonne` pour représenter la clef étrangère pour l'éagrégat `Personne`. Voir la figure 8.3 pour la modélisation des données (diagramme de relation d'entités : *ERD*).



Diag 3. Diagramme de Classes du Package `Vue` avec le détail de la classe `FormManager`

5.4 Génération de Formulaires *HTML*

Nous définissons ensuite la classe `AdresseFormView` qui présente (essentiellement) deux méthodes pour générer des formulaires, avec ou sans gestion d'erreur dans des vues. Un méthode permet aussi de générer un formulaire avec tous les champs cachés pour transmettre tous les champs d'une adresse par la méthode post de manière transparente pour l'utilisateur.

Pour générer des formulaires, on utilise une classe *template* de génération de formulaires dont le code est donné ci-dessous.

Dans la méthode permettant de générer le formulaire avec un éventuel message d'erreur pour chaque attribut, on utilise le format du tableau d'erreurs construit par la fabrique d'adresse (partie 5.2.3).

Code Source 5.9 : /forms2/classes/AdresseFormView.php

```
1 <?php
2 namespace CoursPHP\Vue;
3 /** @brief Implémente la génération de formulaire HTML de saisie d'Adresse
4   * Les formulaires peuvent être vierges ou pré-remplis avec d'éventuels
```

```
5     * messages d'erreur sur la forme des attributs. */
6   class AdresseFormView {
7
8     /** @brief Méthode de génération d'un formulaire HTML vierge. */
9     public static function getDefaultFormHTML($action, $idPersonne, $idAdresse){
10      return self::getFormHtml($action,
11                               \CoursPHP\Metier\Adresse
12                               ::getDefaultInstance($idPersonne, $idAdresse));
13    }
14
15    /** @brief Méthode de génération d'un formulaire HTML pré-rempli. */
16    public static function getFormHtml($action, $adresse,
17                        $filteringPolicy = \CoursPHP\Controleur\ValidationUtils
18                                                        ::SANITIZE_POLICY_NONE){
19      $dataErrors = array(); // Aucun message d'erreur
20      return self::getFormErrorsHtml($action, $adresse, $dataErrors,
21                                                          $filteringPolicy);
22    }
23
24    /** Génère un message d'erreur associé à un attribut, s'il en existe un.
25     * Le texte du message est formé du message d'erreur contenu dans dataError.
26     * @param $dataError Tableau associatif (nom d'attribut => message d'erreur)
27     * @param $fieldName nom de l'attribut considéré */
28    private static function addErrorMsg($dataError, $fieldName){
29      $htmlCode = "";
30      if (!empty($dataError[$fieldName])){
31        $htmlCode .=  "<span class=\"errorMsg\">"
32                    .htmlentities($dataError[$fieldName], ENT_COMPAT, "UTF-8")
33                    ."</span>";
34      }
35      return $htmlCode;
36    }
37
38    /** @brief Méthode de génération de formulaire HTML pré-rempli avec erreurs.
39     * Génère des messages d'erreur associés aux attribut, s'il en existe. */
40    public static function getFormErrorsHtml($action, $adresse, $dataError,
41                      $filteringPolicy = \CoursPHP\Controleur\ValidationUtils
42                                                        ::SANITIZE_POLICY_NONE){
43      \CoursPHP\Metier\AdresseValidation::filterAdresse($adresse, false,
44                                                          $filteringPolicy);
45      $htmlCode = FormManager::beginForm("post", $action);
46        $htmlCode .= FormManager::addHiddenInput("idAdresse", "idAdresse",
47                                                 $adresse->idAdresse);
48        $htmlCode .= FormManager::addHiddenInput("idPersonne", "idPersonne",
49                                                 $adresse->idPersonne);
50        $htmlCode .= self::addErrorMsg($dataError, "numeroRue");
51        $htmlCode .= FormManager::addTextInput("Numéro", "numeroRue", "numeroRue",
52                                               "4", $adresse->numeroRue);
53        $htmlCode .= self::addErrorMsg($dataError, "rue");
54        $htmlCode .= FormManager::addTextInput("Rue/place", "rue", "rue", "30",
55                                    $adresse->rue);
56        $htmlCode .= self::addErrorMsg($dataError, "complementAddr");
57        $htmlCode .= FormManager::addTextInput("Complément d'adresse",
58                                    "complementAddr", "complementAddr",
59                                    "30", $adresse->complementAddr);
60        $htmlCode .= self::addErrorMsg($dataError, "codePostal");
```

```
61          $htmlCode .= FormManager::addTextInput("Code postal", "codePostal",
62                                      "codePostal", "10",
63                                      $adresse->codePostal);
64          $htmlCode .= self::addErrorMsg($dataError, "ville");
65          $htmlCode .= FormManager::addTextInput("Ville", "ville", "ville", "20",
66                                      $adresse->ville, ENT_QUOTES, "UTF-8");
67          $htmlCode .= self::addErrorMsg($dataError, "pays");
68          $htmlCode .= FormManager::addTextInput("Pays", "pays", "pays", "15",
69                                      $adresse->pays);
70          $htmlCode .= FormManager::addSubmitButton("Envoyer",
71                                                    "class=\"sansLabel\"");
72      $htmlCode .= FormManager::endForm();
73
74      return $htmlCode;
75    }
76
77    /** @brief Méthode de génération d'un formulaire HTML caché prérempli.
78     * Permet de transmettre les données d'une instance via $_POST.
79     * Tous les attributs du formulaire sont de type "hidden". */
80    public static function getHiddenFormHtml($action, $adresse, $buttonText,
81                          $filteringPolicy = \CoursPHP\Controleur\ValidationUtils
82                                                    ::SANITIZE_POLICY_NONE){
83      \CoursPHP\Metier\AdresseValidation::filterAdresse($adresse, false,
84                                                        $filteringPolicy);
85      $htmlCode = FormManager::beginForm("post", $action);
86          $htmlCode .= FormManager::addHiddenInput("idAdresse", "idAdresse",
87                                                   $adresse->idAdresse);
88          $htmlCode .= FormManager::addHiddenInput("idPersonne", "idPersonne",
89                                                   $adresse->idPersonne);
90          $htmlCode .= FormManager::addHiddenInput("numeroRue", "numeroRue",
91                                                   $adresse->numeroRue);
92          $htmlCode .= FormManager::addHiddenInput("rue", "rue", $adresse->rue);
93          $htmlCode .= FormManager::addHiddenInput("complementAddr",
94                                                   "complementAddr",
95                                                   $adresse->complementAddr);
96          $htmlCode .= FormManager::addHiddenInput("codePostal", "codePostal",
97                                                   $adresse->codePostal);
98          $htmlCode .= FormManager::addHiddenInput("ville", "ville",
99                                                   $adresse->ville);
100         $htmlCode .= FormManager::addHiddenInput("pays", "pays", $adresse->pays);
101         $htmlCode .= FormManager::addSubmitButton($buttonText,
102                                                   "class=\"sansLabel\"");
103         $htmlCode .= FormManager::endForm();
104     return $htmlCode;
105   }
106 }
107 ?>
```

La classe *template* de génération de formulaires utilisée ci-dessous est la suivante :

Code Source 5.10 : /forms2/classes/FormManager.php

```
1 <?php
2 namespace CoursPHP\Vue;
3 /** @brief Cette classe sert à faciliter la génération de formulaires HTML.
4   * Elle fournit de méthodes pour générer le début, la fin du formulaire,
5   * ainsi que les inputs, textarea et select avec les options de base.
```

```
6     * Les options complémentaires des inputs peuvent être sélectionnées
7     * via une variable $extraOptions des méthodes. */
8   class FormManager {
9   /** @brief génère la balise &lt;form&gt; avec méthode (Post, Get) et action (url
        ) */
10    public static function beginForm($method, $action, $css_class="",
11                                                          $extraOptions=""){
12      $css_class_option = "";
13      if (!empty($css_class)){
14        $css_class_option = "class=\"".$css_class."\" ";
15      }
16      return "<form method=\"".$method."\" action=\"".$action."\" "
17          ." accept-charset=\"utf-8\" " // On force le charset
18          .$css_class_option.$extraOptions.">\n";
19    }
20
21    /** @brief ferme le formulaire */
22    public static function endForm() {
23      return "</form>";
24    }
25
26    /** méthode générique de génération d'un input
27      * @param $labelText texte du label correspondant à l'input
28      * @param $type type d'input : texte, date, checkbox, radio, submit...
29      * @param $name name de l'input pour la correspondance label/input
30      * @param $id ID de l'input pour la correspondance label/input
31      * @param $value valeur initiale du champs de l'input
32      * @param $extraOptions chaine de caractères contenant les options
33      *                supplémentaires de l'input suivant la syntaxe HTML. */
34    public static function addInput($labelText, $type, $name, $id, $value=null,
35                                                        $extraOptions=""){
36      // On échappe pour éviter tout injection
37      $value = ($value == null) ? "" : filter_var($value, FILTER_SANITIZE_STRING)
            ;
38      $valueOption = " value=\"".$value."\" ";
39      if ($extraOptions == null) {
40        $extraOptions="";
41      }
42      $returnText = "<span class=\"formField\">";
43      if ($labelText!=null && $labelText!=""){
44        $returnText .= "<label for=\"".$id."\">".$labelText."</label>\n";
45      }
46      $returnText .= "<input type=\"".$type."\" name=\"".$name."\" id=\""
47                    .$id."\" ".$valueOption." ".$extraOptions." />\n";
48      $returnText .= "</span>";
49
50      return $returnText;
51    }
52  //! @cond Doxygen_Suppress
53    /** @brief méthode pour générer un input de type text */
54    public static function addTextInput($labelText, $name, $id, $size,
55                                        $value=null, $extraOptions=""){
56      return self::addInput($labelText, "text", $name, $id, $value,
57                                              "size =\"".$size."\" ".$extraOptions
                                                    );
58    }
```

```
59
60    /** @brief méthode pour générer un input de type password */
61    public static function addPasswordInput($labelText, $name, $id, $size,
62                                            $value=null, $extraOptions=""){
63      return self::addInput($labelText, "password", $name, $id, $value,
64                            "size =\"".$size."\" ".$extraOptions);
65    }
66
67    /** @brief méthode pour générer un input de type radio */
68    public static function addRadioInput($labelText, $name, $id, $checked,
69                                         $value=null, $extraOptions=""){
70      return self::addInput($labelText, "radio", $name, $id, $value,
71                (strcmp($checked, 'checked')==0)? "checked =\"checked\" "
72                                                :" ".$extraOptions);
73    }
74
75    /** @brief méthode pour générer un input de type checkbox */
76    public static function addCheckboxInput($labelText, $name, $id, $checked,
77                                            $value, $extraOptions=""){
78      return self::addInput($labelText, "checkbox", $name, $id, $value,
79                (strcmp($checked, 'checked')==0)? "checked =\"checked\" "
80                                                :" ".$extraOptions);
81    }
82
83    /** @brief méthode pour générer une zone de saisie &lt;textarea&gt; */
84    public static function addTextArea($labelText, $name, $id, $rows, $cols,
85                                       $value=null, $extraOptions=""){
86      // On échappe, au moins pour les quotes, mais aussi
87      // pour éviter tout injection
88      $value = ($value == null) ? "" : filter_var($value, FILTER_SANITIZE_STRING)
            ;
89      $valueOption = " value=\"".$value."\" ";
90      if ($extraOptions == null) {
91        $extraOptions="";
92      }
93      $returnText .= "<p>\n";
94      if ($labelText!=null && $labelText!=""){
95        $returnText .= "<label for=\"".$id."\">".$labelText."</label>\n";
96      }
97      $returnText .= "<textarea name=\"".$name."\" id=\"".$id."\" rows=\"".$rows
98                     ."\" cols=\"".$cols."\" ".$extraOptions." >".$valueOption
99                     ."</textarea>\n";
100     $returnText .= "</p>\n";
101     return $returnText;
102   }
103
104   /** @brief méthode pour générer un input de type file (upload) */
105   public static function addUploadInput($labelText, $name, $id, $size,
106                                         $value="", $extraOptions=""){
107     $valueOption = ($value == null) ? "value=\"\"" : " value=\"".$value."\" ";
108     if ($extraOptions == null) {
109       $extraOptions="";
110     }
111     return self::addInput($labelText, "file", $name, $id, $value,
112                           "size =\"".$size."\" ".$valueOption." ".$extraOptions)
                              ;
```

```
113     }
114
115     /** @brief méthode  pour commencer une liste d'options &lt;select&gt; */
116     public static function beginSelect($labelText, $name, $id, $multiple=false,
117                                        $size=6){
118       $returnText = "";
119       if ($multiple){
120         $multipleOption="multiple=\"multiple\" size=\"".$size."\"";
121       }else{
122         $multipleOption="";
123       }
124       if ($labelText!=null && $labelText!=""){
125         $returnText .= "<label for=\"".$id."\">".$labelText."</label>\n";
126       }
127       $returnText .= "<select name=\"".$name.($multiple === true ? "[]" : "")."\"
              id=\"".$id."\" "
128                     .$multipleOption.">\n";
129       return $returnText;
130     }
131
132     /** @brief méthode simplifiée pour terminer une liste d'options &lt;select&gt;
            */
133     public static function endSelect(){
134       $returnText = "</select></p>\n";
135       return $returnText;
136     }
137
138     /** @brief méthode simplifiée pour ajouter une &lt;option&gt; de liste &lt;
          select&gt;
139      **/
140     public static function addSelectOption($value,$displayText, $selected=false){
141       $returnText = "";
142       if ($selected){
143         $selectedOption="selected=\"selected\"";
144       }else{
145         $selectedOption="";
146       }
147       $returnText .= "<option value=\"".$value."\" ".$selectedOption.">"
148                     .$displayText."</option>\n";
149       return $returnText;
150     }
151
152     /** @brief méthode simplifiée pour générer un input de type radio */
153     public static function addHiddenInput($name, $id, $value, $extraOptions=""){
154       return self::addInput("", "hidden", $name, $id, "".$value, $extraOptions);
155     }
156
157     /** @brief méthode simplifiée pour générer un bouton submit */
158     public static function addSubmitButton($value="Envoyer", $extraOptions=""){
159       return self::addInput(null, "submit", "", "", $value, " ".$extraOptions);
160     }
161   //! @endcond
162   }
163   ?>
```

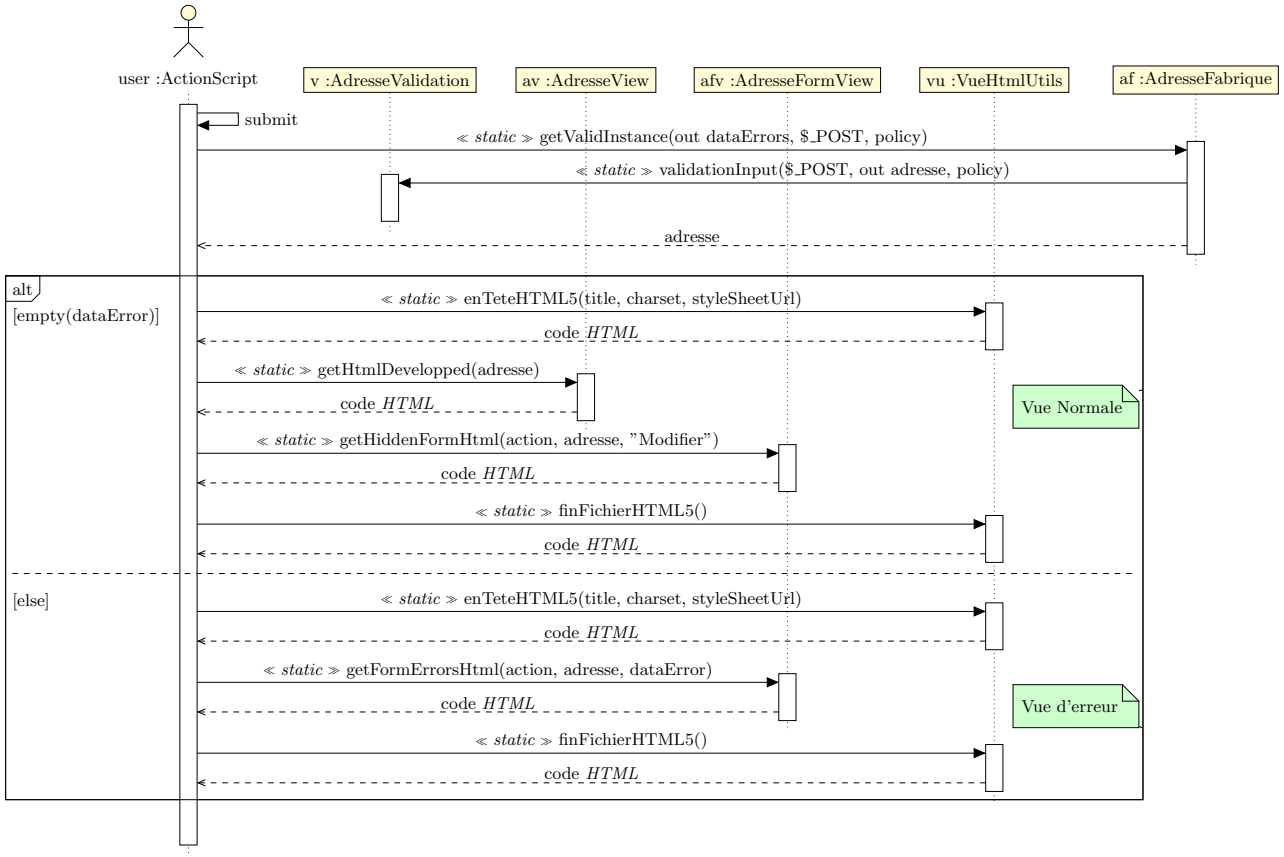
Remarque. Notons que les valeurs affectées dans l'attribut `value` des formulaires sont sys-

tématiquement nettoyées au moyen d'un filtre de type `FILTER_SANITIZE_STRING` dans la méthode `addInput`.

5.5 Enchaînement de la saisie à la vue

Nous montrons enfin comment enchaîner le filtrage initial, la construction des instances, la détection des erreurs et la génération de la vue *HTML* qui convient.

Nous proposerons aussi de modifier une adresse, en transmettant tous les attributs d'une instance dans des champs cachés (*hidden*) (méthode `getHiddenFormHtml` de la classe `AdresseFormView`). Voici le diagramme de séquence de l'implémentation de l'action suite à validation (bouton *submit*) d'un formulaire.



Diag 4. Diagramme de séquence de l'action "validation d'un formulaire"

5.5.1 Saisie et Soumission du Formulaire

Voici tout d'abord la vue permettant de saisir une adresse dans un formulaire vierge :

Code Source 5.11 : /forms2/vueFormulaireAdresse.php

```
1 <?php
2   require_once(dirname(__FILE__).'/classes/VueHtmlUtils.php');
3   require_once(dirname(__FILE__).'/classes/FormManager.php');
4   require_once(dirname(__FILE__).'/classes/Adresse.php');
5   require_once(dirname(__FILE__).'/classes/AdresseValidation.php');
```

```php
require_once(dirname(__FILE__).'/classes/ValidationUtils.php');
require_once(dirname(__FILE__).'/classes/AdresseFormView.php');

echo CoursPHP\Vue\VueHtmlUtils::enTeteHTML5('Saisie d\'une Adresse',
                                            'UTF-8', 'myStyle.css');
echo "<h1>Saisie d'une Adresse</h1>";
echo \CoursPHP\Vue\AdresseFormView
                    ::getDefaultFormHTML("./receptionAdresseRequest.php",
                                         "0123454321", "54321abcde");
echo \CoursPHP\Vue\VueHtmlUtils::finFichierHTML5();
?>
```

Le script qui reçoit les données saisies dans le formulaire, construit l'instance d'adresse et appelle une vue (vue normale ou vue d'erreur, selon le cas).

Remarque. Dans certaines archirectures d'applications, un filtrage des données reçues est effectué dè la réception des données issues d'un formulaire.

Code Source 5.12 : /forms2/receptionAdresseRequest.php

```php
<?php
require_once(dirname(__FILE__).'/classes/ExpressionsRegexUtils.php');
require_once(dirname(__FILE__).'/classes/Adresse.php');
require_once(dirname(__FILE__).'/classes/AdresseValidation.php');
require_once(dirname(__FILE__).'/classes/AdresseFabrique.php');
require_once(dirname(__FILE__).'/classes/ValidationUtils.php');
// On crée une instance à partir du tableau $_POST
// Les attributs "name" des inputs de l'adresse doivent correspondre
// aux noms des attributs de la classe \CoursPHP\Metier\Adresse
// Les attributs de l'addresse sont aussi valédés pour la logique métier.
$adresse = CoursPHP\Metier\AdresseFabrique::getValidInstance(
                    $dataError, $_POST,
                    \CoursPHP\Controleur\ValidationUtils::SANITIZE_POLICY_NONE
                        );
// Appel de la vue ou de la vue d'erreur :
if (empty($dataError)){
  require(dirname(__FILE__).'/vueAfficheAdresse.php');
}else{
  require(dirname(__FILE__).'/vueErreurSaisieAdresse.php');
}
?>
```

Code Source 5.13 : /forms2/vueErreurSaisieAdresse.php (cf. Fig 5.2)

```php
<?php
require_once(dirname(__FILE__).'/classes/VueHtmlUtils.php');
require_once(dirname(__FILE__).'/classes/FormManager.php');
require_once(dirname(__FILE__).'/classes/AdresseFormView.php');

echo CoursPHP\Vue\VueHtmlUtils::enTeteHTML5('Erreur de Saisie d\'une Adresse',
                                            'UTF-8', 'myStyle.css');
echo "<h1>Erreur(s) de saisie d'une Adresse</h1>";
// Affichage d'un formulaire pré-rempli (politique de filtrage par défaut)
echo CoursPHP\Vue\AdresseFormView::getFormErrorsHtml(
        "./receptionAdresseRequest.php", $adresse, $dataError);
echo CoursPHP\Vue\VueHtmlUtils::finFichierHTML5();
?>
```

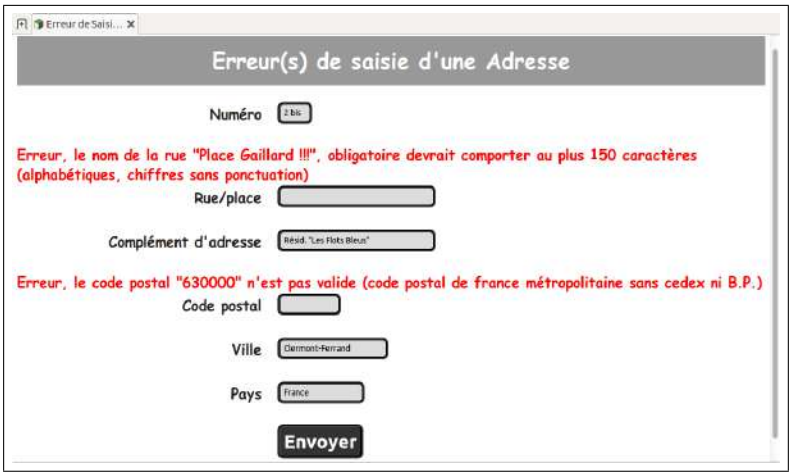


FIGURE 5.2 : Illustration du code source 5.13

Compléments

√ Pour la validation, nous faisons le choix de modifier le moins possible la chaîne, supprimant uniquement les caractères utilisés pour les balises *HTML*, mais *uniquement pour la génération de code HTML*. Nous souhaitons en effet conserver les données brutes dans le système d'information, enutilisant, comme nous le verrons plus loin, la sécurité apportée par les requêtes préparées de *PDO* avec le codage *UTF-8*.

La raison de ce choix est que, pour une utilisation de notre code de filtrage (notamment les expressions régulières) dans une *API*, les codes d'échappement liés à la technologie *HTML* ne sont pas forcément pertinents. Nous validons par `filter_var` dans le cadre de notre application *WEB*, mais nos objets métier et notre couche persistance ne sera pas liée à *HTML*.

5.5.2 Modification d'une `Adresse`

Lors de l'affichage de la vue normale avec l'adresse saisie, un formulaire avec toutes les valeurs des attributs de l'adresse sous forme de champs cachés est inclus dans la vue, avec un bouton *submit* pour éditer les valeurs. On affiche alors une vue avec un formulaire pré-rempli pour modifier l'adresse. Il se trouve que cette vue avec un formulaire pré-rempli a déjà été implémentée dans la vue d'erreur de saisie, que nous ré-utilisons donc ici (bien que l'édition des données par l'utilisateur ne constitue pas une erreur).

La vue normale d'affichage d'une adresse avec un formulaire caché pour modifier l'adresse se présente comme suit :

Code Source 5.14 : /forms2/vueAfficheAdresse.php (cf. Fig 5.3)

```
1 <?php
2   require_once(dirname(__FILE__).'/classes/VueHtmlUtils.php');
3   require_once(dirname(__FILE__).'/classes/FormManager.php');
4   require_once(dirname(__FILE__).'/classes/AdresseView.php');
5   require_once(dirname(__FILE__).'/classes/AdresseFormView.php');
6   // Header HTML5
7   echo CoursPHP\Vue\VueHtmlUtils::enTeteHTML5('Saisie d\'une Adresse',
```

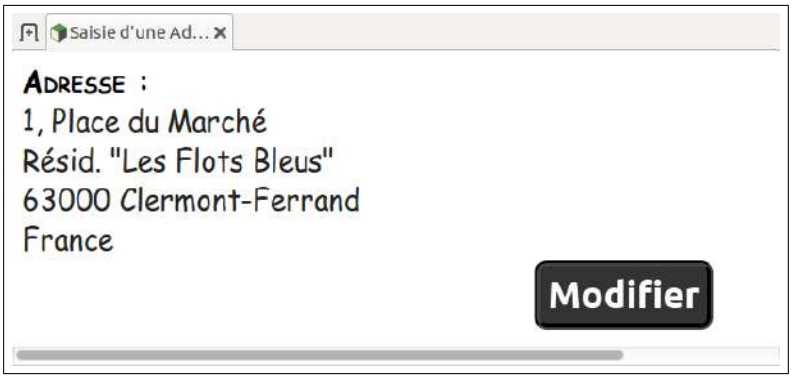


FIGURE 5.3 : Illustration du code source 5.14

```
8                                                    'UTF-8', 'myStyle.css');
9     echo CoursPHP\Vue\AdresseView::getHtmlDeveloppped($adresse);
10    echo CoursPHP\Vue\AdresseFormView::getHiddenFormHtml(
11                          "./editAdresseRequest.php", $adresse, "Modifier");
12    echo CoursPHP\Vue\VueHtmlUtils::finFichierHTML5();
13 ?>
```

La réception des données du formulaire caché via le tableau `$_POST` et l'affichage du formulaire pré-rempli est codé comme suit :

Code Source 5.15 : /forms2/editAdresseRequest.php

```
1  <?php
2     require_once(dirname(__FILE__).'/classes/ExpressionsRegexUtils.php');
3     require_once(dirname(__FILE__).'/classes/Adresse.php');
4     require_once(dirname(__FILE__).'/classes/AdresseValidation.php');
5     require_once(dirname(__FILE__).'/classes/AdresseFabrique.php');
6     require_once(dirname(__FILE__).'/classes/ValidationUtils.php');
7     // On crée une instance à partir du tableau $_POST
8     // Les attributs "name" des inputs de l'adresse doivent correspondre
9     // aux noms des attributs de la classe \CoursPHP\Metier\Adresse
10    // Les attributs de l'addresse sont aussi valédés pour la logique métier.
11    $adresse = CoursPHP\Metier\AdresseFabrique::getValidInstance(
12                          $dataError, $_POST,
13                          \CoursPHP\Controleur\ValidationUtils::SANITIZE_POLICY_NONE
                                );
14    // Appel systématique de la vue d'erreur, qui renvoie un formulaire pré-rempli
          :
15    require(dirname(__FILE__).'/vueErreurSaisieAdresse.php');
16 ?>
```

Troisième partie

Persistance

Table of Contents

Introduction : Mécanismes de la Persistence

Le protocole *HTTP* est un protocole *sans état*. Cela signifie que, dans le protocole *HTTP*, aucune information n'est conservée entre deux transaction. Théoriquement, suivant un tel protocole, il faudrait qu'un client qui se reconnecte rerpenne tout depuis le début. Ce comportement peut être problématique pour certaine application, dans lequelles le serveur doit se souvenir des données personnelles du clients, comme son profil, son adresse, etc.

Pour cette raison, les développeurs *Web* doivent implémenter eux meme la *persistence*, qui permet aux programmes de conserver des données d'un connexion à l'autre. Il y a trois types de mécanismes permettant d'implémenter la persistence :

- **L'authentification par *login* et mot de passe**. C'est la manière la plus sûre d'identifier le client lorsqu'il se représente. Cependant, redemander le mot de passe du client à chaque chargement de script pour lui réattribuer ses données peut vite devenir exaspérant pour le client. Il y a généralement nécessité d'ajouter un mécanisme de persistence temporaire des données tant que le client passe d'une page à l'autre au cours d'une même visite.

- **Les *cookies***, qui sont des données (généralement peu volumineuses accessibles dans le tebleau associatif superglobal `$_COOKIE`) stockées sur la machine du client. Le client a la possibilité de refuser le stockage du cookie, ou d'éliminer le cookie entre deux connections. Un pirate a des fois la possibilité de suptiliser un cookie. En général, on ne peut pas se fier de manière sûre au mécanisme des cookies, ce qui n'empeche pas d'exploiter les cookies. Les cookies sont surtout utilisés pour identifier les client d'une connexion à l'autre, pour pouvoir lui associer les données venant d'une précédente connexion. Pour éviter un piratage des données, l'utilisation de cookie doit être accompagnée d'une politique de sécurité pour éviter l'usurpation d'identité.

  La politique de sécurité, qui dépend de l'application considérée, va s'appuyer notamment sur :

  - Une date limite de validié du cookie, qui permet de limiter la fenêtre d'opportunité pour un pirate.
  - Le *hashage* ou le *chiffrage*, qui permet de ne pas révéler en clair les données stokées dans un cookie.
  - Éventuellement l'adresse *IP* du client. Cependant, sachant que les clients mobiles peuvent changer régulièrement d'adresse *IP*, sachant aussi que des adresses *IP* sur internet peuvent etre partagées par plusieurs clients utilisant la même passerelle, on ne peut pas s'appuyer uniquement sur l'adresse *IP*, même si un changement d'adresse

IP peut être l'un des critères pour redemander une authentification par mot de passe, surtout en présence d'autres indices d'une éventuelle usurpation d'identité.

- Des systèmes de jetons aléatoires à usage unique dans le cookie et, d'une manière générale, l'usage unique des données du cookie, qui permet de limiter la fenêtre d'opportunité pour un pirate.

En général, il faut trouver un compromis, dépendant de l'application, entre le confort du client et la sécurité, et éviter les politiques de sécurité trop "bâteau", qu'un pirate pourra facilement deviner.

- **Les *sessions*** permettent de stocker des données coté serveur. Les données mémorisées sont des couples clé/valeur (accessibles en *PHP* dans un tableau associatif superglobal `$_SESSION`). Pour le stockage sont sous la forme de chaîne de caractère. La *sérialisation* permet de coder des données complexes (instances de classes, etc.) dans une session.

  Selon la configuration du serveur, les sessions peuvent etre stockées dans des fichiers sur le disque, dans une base de données, ou encore (généralement sous forme chiffrée) via un cookie sur le poste client. Ces différentes formes de stockage ont un impact sur la charge du serveur, notamment en cas de grosses applications nécessitant que plusieurs serveurs partagent les données de session, et sur la sécurité et la confidentialité des données.

  Une session, caractérisée par un nom et un identifiant de session (*SID*), doit etre réattribuée à un client d'une page à l'autre, et éventuellement d'une visite à l'autre. Elles sont donc combinées à la transmission de données d'un script à l'autre, soit par la méthode *GET* dans l'*URL*, soit par la méthode *POST* dans un champs caché, soit par un cookie.

- **Les Bases de données**, qui permettent de stocker de manière durable de grandes quantités de données, structurées de manière relationnelle. Pour pouvoir associer des données dans une base de données à un client, il faut identifier le client, généralement via une *clé primaire* dans une table. Cela nécessite de conserver l'information de l'identité du client par les autres mécanismes (login, cookie, session) ci-dessus.

Chapitre 6

Cookies

6.1 Création d'un *cookie*

On crée un *cookie* en *PHP* à l'aide de la fonction `setcookie`. Cette fonction a le prototypes suivant (seul le premier argument est obligatoire) :

Code Source 6.1 :

```
bool setcookie(string $name, string $value, int $expire = 0,
               string $path, string $domain, bool $secure = false,
               bool $httponly = false)
```

Il existe une autre fonction, `setrawcookie`, de même signature, mais qui n'applique pas d'*encodage URL* (voie documentation de la fonction `urlencode`) aux données du *cookie*, contrairement à `setcookie`.

Code Source 6.2 :

```
bool setrawcookie(string $name, string $value, int $expire = 0,
               string $path, string $domain, bool $secure = false,
               bool $httponly = false)
```

La signification des paramètres est la suivante :

- `Name` : nom du *cookie*, qui permet de stocker plusieurs *cookies* sur un même client avec différentes fonctions. On récupère ensuite les valeurs des *cookie* grâce au tableau associatif `$_COOKIE`, qui est indexé par les attributs `name` des différents *cookie*.
- `value` : La valeur du *cookie*, qui est stockée sur l'ordinateur du client. Ne stockez pas d'informations sensibles chez le client, ou alors en chiffrant les données (mais un chiffrement peut toujours se casser...).
- `expire` : la date (*timestamp Unix* : nombre de secondes depuis le 1er janvier 1970 à 0h00) limite de validité du *cookie*. La fonction *time* retourne le *timestamp* de la date actuelle, permettant de fixer la date d'expiration en ajoutant un certain nombre de secondes par rapport au présent. La fonction `mktime` retourne le *timestamp* d'une date donnée par son heure, minute, seconde, année, etc.
- `path` : chemin vers le répertoire sur le serveur dans lequel le *cookie* est disponible dans les scripts *PHP* (Exemple : */* pour la racine du site). La valeur par défaut est le répertoire contenant le script courant définissant le *cookie* (donné par `dirname(__FILE__)`.

- `secure` : permet de ne créer le *cookie* seulement si la connexion est sécurisé par *SSL* (protocole `https`).
- `httponly` : lorsqu'il vaut `true`, ce paramètre empêche l'accès direct du *cookie* via des langages de scripts comme *javascripts*. C'est discutable dans la mesure où, par exemple, ça dépend de l'implémentation du navigateur client.

La fonction `setcookie` retourne `true` en cas de succès de création du *cookie*. Cela ne permet pas d'être sur que le *cookie* a été accepté chez le client. On ne peut pas savoir si le *cookie* a été accepté avant d'avoir chargé un nouveau script essayant d'accéder au *cookie*.

On peut éventuellement stocker un tableau de plusieurs chaînes dans un *cookie*, mais cela crée de fait plusieurs *cookies* chez le client. Mieux vaut créer un *cookie* avec un séparateur de son choix (caractère ou patron de type chaîne. Éviter les caractères spéciaux *HTML*...), puis utiliser le fonction `explode`, qui va retourner un tableau en coupant la chaîne suivant les occurrences du séparateur.

Voici le code de création d'un cookie, valable une heure, dans tous les scripts *PHP* du répertoire courant :

FIGURE 6.1 : Illustration du code source 6.3

Code Source 6.3 : /cookies/ex01-setCookie.php (cf. Fig 6.1)

```php
<?php
  // Fonction qui retourne l'heure locale sous fomr de string :
  function getLocalTimeFrenchFormat(){
    $heureLocaleArray = localtime(time(),true);
    $dayOfMonth = str_pad(intval($heureLocaleArray['tm_mday']), 10), 2,"0",
        STR_PAD_LEFT);
    $monthOfYear = str_pad(intval($heureLocaleArray['tm_mon']), 10)+1, 2,"0",
        STR_PAD_LEFT);
    $yearSinceEra = str_pad(intval($heureLocaleArray['tm_year']), 10)+1900, 4,"0"
        ,STR_PAD_LEFT);
    $hourOfDay = str_pad(intval($heureLocaleArray['tm_hour']), 10), 2,"0",
        STR_PAD_LEFT);
    $minOfHour = str_pad(intval($heureLocaleArray['tm_min']), 10), 2,"0",
        STR_PAD_LEFT);
    $secOfMin = str_pad(intval($heureLocaleArray['tm_sec']), 10), 2,"0",
        STR_PAD_LEFT);

     $heureLocaleFormatee = $dayOfMonth."/".$monthOfYear."/".$yearSinceEra
                            ." à ".$hourOfDay.":".$minOfHour.":".
                              $secOfMin;
    return $heureLocaleFormatee;
  }

```

```php
17    // trois chaîne séparées par des virgules (c'est juste un exemple avec explode
          ...)
18    $valeur = "ma chaîne 1, ma chaîne 2, ma chaîne 3, ".getLocalTimeFrenchFormat()
          ;
19    setcookie("essaiCookie", $valeur, time()+3600);
20
21    // Code de la de la vue :
22    require_once('classes/VueHtmlUtils.php');
23    echo Vue\VueHtmlUtils::enTeteHTML5("Création d'un Cookie", 'UTF-8', 'myStyle.
          css');
24    echo "<h1>Création d'un <i>cookie</i></h1>";
25    echo "<p><a href=\"ex02-retrieveCookie.php\">Cliquez ici</a> "
26          ."pour voir si le <i>cookie</i> a bien été stocké chez le client.</p>";
27    echo Vue\VueHtmlUtils::finFichierHTML5();
28  ?>
```

6.2 Récupération d'un *cookie*

Les *cookies* peuvent être obtenus, jusqu'à expiration, dans le tableau associatif (qui est un *superglobal*) $_COOKIE. Les clés de ce tableau associatif sont les noms des *cookies*, et les valeurs du tableau sont les valeurs respectives des *cookies*.

FIGURE 6.2 : Illustration du code source 6.4

Code Source 6.4 : /cookies/ex02-retrieveCookie.php (cf. Fig 6.2)

```php
1   <?php
2     if (isset($_COOKIE['essaiCookie'])) {
3       // Le contenu d'un cookie doit être filtré comme les inputs
4       $valeur = $_COOKIE['essaiCookie'];
5       $valeur = filter_var($valeur, FILTER_SANITIZE_STRING);
6       $tabChaines = explode(',', $valeur);
7     }else{
8       $dataError = array('cookie' => '"essaiCookie" introuvable !');
9     }
10
11    // Appel des vues :
12    require_once('classes/VueHtmlUtils.php');
13    if (empty($dataError)){ // Code de la vue normale :
14      echo Vue\VueHtmlUtils::enTeteHTML5("Récupération d'un Cookie", 'UTF-8', '
            myStyle.css');
15      echo "<h1>Récupération d'un <i>cookie</i></h1>";
16      echo "Les chaînes contenues dans le <i>cookie</i> sont : ";
17      echo "<ul>";
```

```
18      foreach ($tabChaines as $chaine){
19        echo "<li>".$chaine."</li>";
20      }
21      echo "</ul>";
22      echo Vue\VueHtmlUtils::finFichierHTML5();
23    }else{ // Appel de la vue d'erreur :
24      require('ex02-vueErreur.php');
25    }
26
27
28  ?>
```

FIGURE 6.3 : Illustration du code source 6.5

Code Source 6.5 : /cookies/ex02-vueErreur.php (cf. Fig 6.3)

```
1  <?php
2    echo Vue\VueHtmlUtils::enTeteHTML5("Problème de récupération d'un Cookie", '
         UTF-8', 'myStyle.css');
3    echo "<h1>Erreur de récupération de <i>cookie</i></h1>";
4    echo "<p>";
5    foreach ($dataError as $field => $message){
6      echo "<i>".$field."</i> : ".$message."<br/>";
7    }
8    echo "</p>";
9    echo Vue\VueHtmlUtils::finFichierHTML5();
10 ?>
```

6.3 Suppression d'un *cookie*

Pour supprimer un *cookie*, on le recrée, avec le même nom, une valeur `false` (ou chaîne vide), et une date d'expiration antérieure au présent.

Code Source 6.6 : /cookies/ex03-unsetCookie.php

```
1  <?php
2    // On met une valeur vide et une date d'expiration antérieure au présent
3    setcookie("essaiCookie", "", time() - 3600);
4
5    // Code de la vue :
6    require_once('classes/VueHtmlUtils.php');
7    echo Vue\VueHtmlUtils::enTeteHTML5("Suppression d'un Cookie", 'UTF-8', '
         myStyle.css');
```

```
8   echo "<h1>Suppression d'un <i>cookie</i></h1>";
9   echo "<p><a href=\"ex02-retrieveCookie.php\">Cliquez ici</a> "
10      ."pour vérifier que le <i>cookie</i> a bien été supprimé chez le client.</
          p>";
11  echo Vue\VueHtmlUtils::finFichierHTML5();
12 ?>
```

Le fait de faire `unset($_COOKIE['essaiCookie'])` ne modifiera pas, et ne supprimera pas, le *cookie* chez le client.

6.4 Mise à jour d'un *cookie*

Il n'y a pas d'autre méthodes pour mettre à jour un *cookie* que d'en créer une nouveau, de même nom, avec la fonction `setcookie`.

On peut par exemple mettre à jour la date d'expiration à chaque chargement de page, pour prolonger la validité tant que l'utilisateur est actif, sans changer la valeur du *cookie* :

FIGURE 6.4 : Illustration du code source 6.7

Code Source 6.7 : /cookies/ex04-setAndProlongCookie.php (cf. Fig 6.4)

```
1  <?php
2    if (isset($_COOKIE['essaiCookie'])){ // si le cookie existe
3      $valeur = $_COOKIE['essaiCookie'];
4      $valeur = filter_var($valeur, FILTER_SANITIZE_STRING);
5    }else{ // si le cookie n'existe pas, on le crée avec la date :
6      $valeur = "Je suis la valeur dans le cookie : ".time();
7    }
8    // Le coockie est prolongé tant que l'utilsateur ne reste pas inactif 15mn
9    setcookie("essaiCookie", $valeur, time()+15*60);
10
11   // Code de la vue :
12   require_once('classes/VueHtmlUtils.php');
13   echo Vue\VueHtmlUtils::enTeteHTML5("Création d'un Cookie", 'UTF-8', 'myStyle.
         css');
14   echo "<h1>Prolongement de la Validité d'un <i>cookie</i></h1>";
15   echo "<p>Valeur courante du cookie : <i>".$valeur."</i></p>";
16   // Lien sur ce même script (en enlevant la "query string" avec basename par sé
         curité) :
17   echo "<p><a href=\"".basename($_SERVER[REQUEST_URI], ".php").".php\">Cliquez
         ici</a> "
18      ."pour voir si le <i>cookie</i> a bien été stocké chez le client.</p>";
19   echo Vue\VueHtmlUtils::finFichierHTML5();
20 ?>
```

Chapitre 7

Sessions

7.1 Concept de Session et Problèmes de Sécurité

Les sessions sont des données, mémorisées sur le serveur, qui peuvent rester accessible d'un script à l'autre. Pour utiliser les sessions en *PHP*, il faut démarrer la session dans chaque script qui devra utiliser les données ce cette session. Si la session n'existe pas, elle sera créée et ne contiendra initialement aucune donnée. Si une session existe et est active, les données de cette session seront chargées dans le tableau associatif superglobal `$_SESSION`. Ce tableau associatif est aussi accessible en écriture pour ajouter des données en session. Les données de session doivent être sérialisées pour être stockées sur le serveur. La sauvegarde des sessions a un comportement par défaut, qui peut être modifié, sur le serveur.

Pour retrouver une session d'un script à l'autre, un numéro de session (*SID*) doit être transmis entre les scripts. Il y a trois manières de transmettre le numéro de session, qui doivent chacune s'accompagner d'une politique de sécurité, pour éviter l'usurpation malveillante de l'accès à une session. Voici (par ordre décroissant de sécurité supposée) ces trois manières :

- Les *cookies*, stockés par le navigateur du client, et éventuellement accessible côté client via un langage de script comme *javascript* ;
- La méthode *POST*, sous la forme d'un champs caché de formulaire en clair dans le source *HTML* ;
- La méthode *GET*, en clair dans l'*URL* accessible au client ;

Du fait qu'il y a toujours une possibilité pour une personne malveillante d'accéder aux données transmises, il est généralement déconseillé de transmettre en clair l'identifiant de session. Tout au moins, des données doivent permettre de vérifier que l'utilisateur qui nous transmet un numéro de session est bien identifié, avant de lui donner accès aux données de session. En effet, les données de session permettent entre autre de maintenir un utilisateur connecté considéré comme authentifié, sans qu'il ait besoin de rentrer son mot de passe à chaque chargement de script.

En dehors du risque d'usurpation d'identité lié à la transmission de l'identité de l'utilisateur ou du numéro de session, les données de session elles mêmes, n'étant pas transmises au client, sont relativement sûres (dans la mesure ou le serveur lui-même est sécurisé).

7.2 Cycle de vie d'une Session

7.2.1 Création d'une session commune à tous les utilisateurs

Voici un exemple de session contenant un compteur du nombre de chargement du script, global pour tous les utilisateurs. Les fonctions utilisées pour la gestion de la session sont :

- `session_id` : permet de définir l'identifiant (*SID*) de session (ou d'y accéder) ;
- `session_start` : permet la création d'une session (avec le *SID* précédemment défini), ou son chargement si la session existe sur le disque et n'a pas expiré.
- `session_write_close` : Permet d'écrire immédiatement la session et de la fermer, permettant éventuellement à d'autres scripts de charger la session. (si on n'appelle pas explicitement la fonction `session_write_close`, la session sera quand même écrite (sauf erreur), mais il peut y avoir une latence pour les accès concurrents.

FIGURE 7.1 : Illustration du code source 7.1

Code Source 7.1 : /sessions/ex01-createSessionForDummies.php (cf. Fig 7.1)

```php
<?php
  // Création d'un identifiant de session valable pour tous les utilisateurs
  session_id("all-users-session");

  // Le démarage de session doit avoir lieu avant toute sortie de code HTML via
      echo, print, etc.
  session_start();

  // Si la variable de session du jour existe (il y a déjà eu un script chargé
      aujourd'hui)
  if (isset($_SESSION['counter'])){
    $counter = intval($_SESSION['counter'], 10);
    $counter++;
    $_SESSION['counter'] = "".$counter;
  }else{
    $_SESSION['counter'] = "1";
  }

  // Mémorisation de la donnée de session avant fermeture
  $counterValue = $_SESSION['counter'];
  // Flush des Données de Session, (sauvegarde immédiate sur le disque)
  // Libère instantanément le verrou pour l'accès à la session par d'autres
      scripts ou clients
```

```
21    session_write_close ();
22
23    // Code de la vue :
24    require_once('classes/VueHtmlUtils.php');
25    echo CoursPHP\Vue\VueHtmlUtils::enTeteHTML5("Utilisation basique d'une session
          ", 'UTF-8', 'myStyle.css');
26    echo "<h1>Utilisation Basique d'une Session<br/>Commune à Tous les Clients</h1
          >";
27    echo "<p>Le script a été chargé ".$counterValue." fois depuis la création de
          la session.</p>";
28    echo CoursPHP\Vue\VueHtmlUtils::finFichierHTML5();
29  ?>
```

7.2.2 Identifiant de Session (*SID*)

FIGURE 7.2 : Illustration du code source 7.2

Code Source 7.2 : /sessions/ex02-createSessionBasicId.php (cf. Fig 7.2)

```
1   <?php
2     // Création d'un identifiant de session valable pour tous les utilisateurs
3     session_id("all-users-session");
4
5     // Le démarage de session doit avoir lieu avant toute sortie de code HTML via
          echo, print, etc.
6     session_start();
7
8     // Chaîne qui code la date d'aujourd'hui
9     $heureLocaleArray = localtime(time(),true);
10    $dayOfMonth = str_pad(intval($heureLocaleArray['tm_mday'], 10), 2,"0",
          STR_PAD_LEFT);
11        $monthOfYear = str_pad(intval($heureLocaleArray['tm_mon'], 10)+1, 2,"0",
              STR_PAD_LEFT);
12        $yearSinceEra = str_pad(intval($heureLocaleArray['tm_year'], 10)+1900, 4,"0
              ",STR_PAD_LEFT);
13    $dateSting = $yearSinceEra."-".$monthOfYear."-".$dayOfMonth;
14
15    // Si la variable de session du jour existe (il y a déjà eu un script chargé
          aujourd'hui)
16    if (isset($_SESSION['counter-'.$dateSting])){
17      $counter = intval($_SESSION['counter-'.$dateSting], 10);
18      $counter++;
19      $_SESSION['counter-'.$dateSting] = "".$counter;
```

```
20    }else{
21      // Chaîne qui code la date d'hier
22      $yesterdayLocaleArray = localtime(time()-60*60*24,true);
23      $yestadayDayOfMonth = str_pad(intval($yesterdayLocaleArray['tm_mday'], 10),
            2,"0",STR_PAD_LEFT);
24      $yestadayMonthOfYear = str_pad(intval($yesterdayLocaleArray['tm_mon'], 10)
            +1, 2,"0",STR_PAD_LEFT);
25      $yestadayYearSinceEra = str_pad(intval($yesterdayLocaleArray['tm_year'], 10)
            +1900, 4,"0",STR_PAD_LEFT);
26      $yesterdayDateString = $yestadayYearSinceEra."-".$yestadayMonthOfYear."-".
            $yestadayDayOfMonth;
27      // On efface le compteur de la veille pour éviter que les compteurs s'
            accumulent...
28      unset ($_SESSION['counter-'.$yesterdayDateString]);
29      // On initialise le compteur du jour
30      $_SESSION['counter-'.$dateSting] = "1";
31    }
32    // Mémorisation de la donnée de session avant fermeture
33    $counterValue = $_SESSION['counter-'.$dateSting];
34    // Flush des Données de Session, (sauvegarde immédiate sur le disque)
35    // Libère instantanément le verrou pour l'accès à la session par d'autres
          scripts ou clients
36    session_write_close ();
37
38    // Code de la vue :
39    require_once('classes/VueHtmlUtils.php');
40    echo CoursPHP\Vue\VueHtmlUtils::enTeteHTML5("Utilisation basique d'une session
          ", 'UTF-8', 'myStyle.css');
41    echo "<h1>Utilisation d'une Session<br/>Commune à Tous les Clients</h1>";
42
43    echo "<p>Le script a été chargé ".$counterValue." fois aujourd'hui (le ".
          $dayOfMonth."/".$monthOfYear."/".$yearSinceEra.").</p>";
44    echo CoursPHP\Vue\VueHtmlUtils::finFichierHTML5();
45  ?>
```

7.2.3 Destruction d'une Session

Code Source 7.3 : /sessions/ex03-destroySession.php

```
1  <?php
2    // Récupération de la session à l'aide d'un identifiant de session
3    session_id("all-users-session");
4
5    // Le démarage de session doit avoir lieu avant toute sortie de code HTML via
         echo, print, etc.
6    session_start();
7
8    // Détruit toutes les variables de session
9    session_unset();
10   // Détruit toute la session (aussi les données préalablement sauvegardées)
11   session_destroy();
12 ?>
```

7.3 Gestion de l'Identifiant de Session (*SID*)

7.3.1 Exemple de Session avec *SID* aléatoire transmis par *GET*

Voici un exemple qui crée une session spécifique pour chaque utilisateur pour la configuration de sa langue préférée (français ou anglais) (voir la figure 7.3). Il s'agit d'une donnée peu sensible, donc l'essentiel est de ne pas mélanger les utilisateur. Dans cet exemple, l'usurpation d'identité n'aura pas de conséquences. On ne s'intéresse donc pas à la question d'un sniffeur qui piraterait l'*URL* passée par *GET*, permettant au sniffeur d'obtenir le *SID*.

(a) Pas de session en cours

(b) Langue préférée en anglais

(c) Cas d'un *SID* incorrect

FIGURE 7.3 : Accueil d'un client suivant trois cas de figure de session

Code Source 7.4 : /sessions/ex04-sessionRandomIdGET.php

```php
<?php
  require(dirname(__FILE__) . "/ex04-function GetGreeting.php");
  $dataError = array();
  // Test pour voir si l'identifiant de session existe et si la donnée a la
      bonne forme
  // (10 chiffres hexa entre 0 et f)
  if (isset($_GET['session-id-test']) && preg_match("/^[0-9a-fA-F]{20}$/",
                                                    $_GET['session-id-test'])){
    // On a bien vérifié la forme par expression régulière donc, pas d'autre pré
        caution
    $mySid = $_GET['session-id-test'];
  }else{
    if (isset($_GET['session-id-test'])){
      $dataError['session-id-test'] = "Identifiant de session incorrect. Pirates
          s'abstenir...";
    }
    // Génération d'un SID par des octets (pseudo-)aléatoires codés en hexa
    $cryptoStrong = false; // Variable pour passage par référence
    $octets = openssl_random_pseudo_bytes(10, $cryptoStrong);
    $mySid = bin2hex($octets);
```

```
18    }
19    session_id($mySid);
20
21    // Le démarage de session doit avoir lieu avant toute sortie de code HTML via
          echo, print, etc.
22    session_start();
23
24    // Initialisation des paramètres régionaux
25
26    // Si un choix de langage est précisé dans l'URL, on modifie la variable de
          session
27      if (isset($_GET['pref_lang'])){
28      if ($_GET['pref_lang'] == "en" || $_GET['pref_lang'] == "fr"){
29        // Les données entrées en session doivent être filtrées,
30        // même si, dans ce cas, il n'y a pas de danger car on a testé avec ==
31        $_SESSION['preferred_language'] = filter_var($_GET['pref_lang'],
              FILTER_SANITIZE_STRING);
32      }else{
33        // Paramètre imprévu, on détruit la donnée de session, au cas où
34        unset($_SESSION['preferred_language']);
35      }
36    }
37    // Si une préférence de langage a été définie, soir dans l'URL, soit en
          session
38    if (isset($_SESSION['preferred_language'])){
39      $PREFERRED_LANG = $_SESSION['preferred_language'];
40    }else{
41      $PREFERRED_LANG = "undef";
42    }
43    // Flush des Données de Session, (sauvegarde simmédiate ur le disque)
44    session_write_close();
45    // Code de la vue
46    if (empty($dataError)){
47      require('ex04-vueNormale.php');
48    }else{ // Appel de la vue d'erreur :
49      require('ex04-vueErreur.php');
50    }
51  ?>
```

Code Source 7.5 : /sessions/ex04-vueNormale.php

```
1   <?php
2     require_once('classes/VueHtmlUtils.php');
3     echo CoursPHP\Vue\VueHtmlUtils::enTeteHTML5("Session avec SID Aléatoire", 'UTF
          -8', 'myStyle.css');
4     echo "<h1>Session avec <i>SID</i> Aléatoire<br/>Transmis par <code>GET</code
          ></h1>";
5     echo "<p>";
6     // Message de bienvenue dans la langue sélectionnée ou en multilingue si undef
7     echo getGreeting($PREFERRED_LANG)."<br/>";
8     $urlClean = explode("?", $_SERVER['SCRIPT_NAME'])[0];
9     echo "<a href=\"".$_SERVER['SCRIPT_NAME']."?session-id-test=".$mySid."&
          pref_lang=fr\">Français</a> ou "
10        ."<a href=\"".$_SERVER['SCRIPT_NAME']."?session-id-test=".$mySid."&
              pref_lang=en\">English</a>";
11    echo "</p>";
```

```
12    echo CoursPHP\Vue\VueHtmlUtils::finFichierHTML5();
13  ?>
```

Code Source 7.6 : /sessions/ex04-functionGetGreeting.php

```
1   <?php
2     /** Fonction qui retourne un message de bienvenue dans la langue choisie */
3     function getGreeting($PREF_LANG){
4       $htmlGreeting = "";
5       switch ($PREF_LANG){
6           case "en" : $htmlGreeting .= "Hi, guys, Welcome to <code>mySite.com</
                code> !";
7         break;
8           case "fr" : $htmlGreeting .= "Salut la compagnie, bienvenue sur <code>
                monSite.fr</code> !";
9         break;
10
11        default : $htmlGreeting .= "Wilkommen, Bienvenue, Welcomme !<br/>";
12        break;
13      }
14      return $htmlGreeting;
15    }
16  ?>
```

7.3.2 Exemple de Session avec *SID* aléatoire transmis par *COOKIE*

Comme dans l'exemple précédent, le script crée une session spécifique pour chaque utilisateur pour la configuration de sa langue préférée (français ou anglais). La différence est dans le mode de transmission du *SID* par *cookie*.

Les données de session sont peu sensibles, donc l'usurpation d'identité n'aura pas de conséquences. On ne s'intéresse pas à la question d'un vol de *cookie* qui permettrait à un pirate d'obtenir le *SID*.

Code Source 7.7 : /sessions/ex05-sessionRandomIdCOOKIE.php

```
1   <?php
2     require(dirname(__FILE__)."/ex04-functionGetGreeting.php");
3     $dataError = array();
4     // Test pour voir si l'identifiant de session existe et si la donnée a la
          bonne forme
5     // (10 chiffres hexa entre 0 et f)
6     if (isset($_COOKIE['session-id-test']) && preg_match("/^[0-9a-fA-F]{20}$/",
7                                                          $_COOKIE['session-id-test'])){
8       // On a bien vérifié la forme par expression régulière donc, pas d'autre pré
            caution
9       $mySid = $_COOKIE['session-id-test'];
10    }else{
11      if (isset($_COOKIE['session-id-test'])){
12        $dataError['session-id-test'] = "Identifiant de session incorrect. Pirates
              s'abstenir...";
13      }
14      // Génération d'un SID par des octets (pseudo-)aléatoires codés en hexa
15      $cryptoStrong = false; // Variable pour passage par référence
16      $octets = openssl_random_pseudo_bytes(10, $cryptoStrong);
17      $mySid = bin2hex($octets);
```

```php
    }
    session_id($mySid);
    // Création (ou mise à jour) du cookie. Nouvelle validité du cookie : 10 jours
    setcookie("session-id-test", $mySid, time()+60*60*24*10);

    // Le démarage de session doit avoir lieu avant toute sortie de code HTML via echo, print, etc.
    session_start();

    // Initialisation des paramètres régionaux

    // Si un choix de langage est précisé dans l'URL, on modifie la variable de session
      if (isset($_GET['pref_lang'])){
      if ($_GET['pref_lang'] == "en" || $_GET['pref_lang'] == "fr"){
        // Les données entrées en session doivent être filtrées,
        // même si, dans ce cas, il n'y a pas de danger car on a testé avec ==
        $_SESSION['preferred_language'] = htmlentities($_GET['pref_lang'], ENT_QUOTES, "UTF-8");
      }else{
        // Paramètre imprévu, on détruit la donnée de session, au cas où
        unset($_SESSION['preferred_language']);
      }
    }
    // Si une préférence de langage a été définie, soir dans l'URL, soit en session
    if (isset($_SESSION['preferred_language'])){
      $PREFERRED_LANG = $_SESSION['preferred_language'];
    }else{
      $PREFERRED_LANG = "undef";
    }
    // Flush des Données de Session, (sauvegarde simmédiate sur le disque)
    session_write_close();
    // Code de la vue
    if (empty($dataError)){
      require('ex05-vueNormale.php');
    }else{ // Appel de la vue d'erreur :
      require('ex04-vueErreur.php');
    }
?>
```

Code Source 7.8 : /sessions/ex05-vueNormale.php

```php
<?php
    require_once('classes/VueHtmlUtils.php');
    echo CoursPHP\Vue\VueHtmlUtils::enTeteHTML5("Session avec SID Aléatoire", 'UTF-8', 'myStyle.css');
    echo "<h1>Session avec <i>SID</i> Aléatoire<br/>Transmis par <code>GET</code></h1>";
    echo "<p>";
    // Message de bienvenue dans la langue sélectionnée ou en multilingue si undef
    echo getGreeting($PREFERRED_LANG)."<br/>";
    $urlClean = explode("?", $_SERVER['SCRIPT_NAME'])[0];
    echo "<a href=\"".$_SERVER['SCRIPT_NAME']."?&pref_lang=fr\">Français</a> ou "
        ."<a href=\"".$_SERVER['SCRIPT_NAME']."?pref_lang=en\">English</a>";
    echo "</p>";
```

```
12    echo CoursPHP\Vue\VueHtmlUtils::finFichierHTML5();
13 ?>
```

7.4 *Login/Password* : Exemple de Politique de Sécurité

Nous voyons maintenant un exemple d'utilisation d'une session et de *cookie* un peu mieux sécurisé.

Il s'agit d'un exemple à vocation pédagogique. L'auteur décline toute responsabilité en cas d'utilisation, telle quelle ou avec adaptation, de cette politique de sécurité.
This example is to be taken on an "as is" basis. We accept no liability for consequences of direct or indirect use of this security policy whatsoever.

Dans cet exemple,

- Le numéro de session (*SID*) est aléatoire ;
- On effectue un contrôle par l'adresse *IP* du client. Cette adresse *IP* est stockée en session, et est ensuite testée pour vérifier que le client n'a pas changé d'adresse IP.

Le numéro de session est envoyé chez le client via un *cookie*. De plus, le *cookie* (et la session associée) ont une durée de validité de $2mn$, temps laissé au client pour charger le script suivant.

Lors du chargement du script suivant, le numéro de session récupéré via le *cookie*.

Enfin, à chaque chargement de script, on change le *SID* aléatoire, en copiant les données de session dans une nouvelle, et on re-génère le *cookie*. Le *SID*, ainsi que le *cookie*, n'est ainsi valable qu'une seule fois.

Notons qu'une application sensible pourrait aussi effectuer d'autres contrôles, par exemple sur le navigateur, système d'exploitation, ou encore la cohérence du *referer* avec la structure du site et de ses liens internes.

La vue d'authentification (saisie de *login* et du mot de passe) est la suivante :

Code Source 7.9 : /sessions/ex07-authentification.php (cf. Fig 7.4)

```
1  <?php
2     require_once(dirname(__FILE__).'/classes/VueHtmlUtils.php');
3     echo CoursPHP\Vue\VueHtmlUtils::enTeteHTML5("Login Form", 'UTF-8',
4                                                 'myStyle.css');
5     echo "<h1>Page d'Authentification</h1>";
6     echo CoursPHP\Vue\VueHtmlUtils::getHTML_LoginForm("ex07-receivePassword.php");
7     echo CoursPHP\Vue\VueHtmlUtils::finFichierHTML5();
8  ?>
```

FIGURE 7.4 : Illustration du code source 7.9

Le code *HTML* du formulaire est généré dans la méthode de `VueHtmlUtils` ci-dessous :

Code Source 7.10 : /sessions/classes/VueHtmlUtils.php

```
27    public static function getHTML_LoginForm($formAction){
28
29      $htmlCode = "";
30      // Test de connexion SSL et le cas échéant, warning.
31      if (!isset($_SERVER['HTTPS']) || $_SERVER['HTTPS'] == "off"){
32        $htmlCode .= "<p><strong>Warning :</strong> Vous n'êtes pas "
33                    ."sur une connexion sécurisée <i>HTTPS</i> avec <i>SSL</i>."
34                    ."<br/>Votre confidentialité n'est pas garantie !!!</p>";
35      }
36          // Code du formulaire :
37      $htmlCode .= '<form method="POST" action="'.$formAction.'">';
38      $htmlCode .= '<input type="hidden" name="action" value="validateAuth"/>';
39      $htmlCode .= '<p><label for="e-mail">e-mail</label>'
40                  .'<input type="email" name="email" size="25"/></p>';
41      $htmlCode .= '<p><label for="motdepasse">Mot de passe</label>'
42                  .'<input type="password" name="motdepasse" size="25"/></p>';
43      $htmlCode .= '<input class="sansLabel" value="Envoyer" type="submit"/>';
44      $htmlCode .= '</form>';
45      $htmlCode .=  "<p>L'adresse <i>e-mail</i> doit être valide et votre "
46                    ."mot de passe doit contenir au moins 8 caractères, une "
47                    ."minuscule, une majuscule, un chiffre, et un caractère parmis
                        "
48                    .htmlentities("#-|.@[]=!&", ENT_QUOTES, "UTF-8")
49                    .", merci de votre compréhension...</p>";
50      return $htmlCode;
51    }
52  }
```

Si le mot de passe est trop simple (test dans `validationPasswd`php), on appelle une vue d'erreur qui demande un nouveau mot de passe :

FIGURE 7.5 : Illustration du code source 7.11

Code Source 7.11 : /sessions/ex07-vueErreur.php (cf. Fig 7.5)

```php
<?php
  require_once('classes/VueHtmlUtils.php');
  echo CoursPHP\Vue\VueHtmlUtils::enTeteHTML5("Échec d'authentification", 'UTF-8
      ', 'myStyle.css');
  echo "<h1>Problème d'Authentification</h1>";
  echo "<p>";
      foreach ($dataError as $errorMsg){
    echo "<strong>".$errorMsg."</strong><br/>";
  }
  echo "</p>";
  echo CoursPHP\Vue\VueHtmlUtils::getHTML_LoginForm("ex07-receivePassword.php");
  echo CoursPHP\Vue\VueHtmlUtils::finFichierHTML5();
?>
```

Code Source 7.12 : /sessions/classes/ValidationRequest.php

```php
<?php
namespace CoursPHP\Auth;
/** @brief Validation les données de login/password reçues via $_REQUEST.
 * Nettoyage de toutes les chaînes.
 * Initialisation à vide des inputs inexistants */
class ValidationRequest {
  /** @brief Nettoie une chaîne avec filter_var et FILTER_SANITIZE_STRING
   */
  private static function sanitizeString($chaine){
    return isset($chaine) ? filter_var($chaine, FILTER_SANITIZE_STRING) : "";
  }

  /** @brief Validation et initialisation des données du login/password
   * à partier des données reçues dans les tableau superglobal $_REQUEST.
   */
   public static function validationLogin(&$dataError, &$email, &$password){
    if (!isset($dataError)){
      $dataError = array();
    }
    // Test sur la forme des données de login et mot de passe :
    $wouldBePasswd = $_POST['motdepasse'];
```

```
22      if (empty($wouldBePasswd) || !\CoursPHP\Auth\AuthUtils::isStrongPassword(
            $wouldBePasswd)){
23        $password = "";
24        $dataError["login"] = "Mot de passe incorrect"
25                              ."votre mot de passe doit contenir au moins 8
                                    caractères, "
26                                          ."une minuscule, une majuscule, un
                                                chiffre, "
27                            ."et un caractère parmis "
28                           .htmlentities("#-|.@[]=!&", ENT_QUOTES, "UTF-8")."</p>";
29       }else{
30        $password = $wouldBePasswd;
31       }
32
33      if (filter_var($_POST["email"], FILTER_VALIDATE_EMAIL) == FALSE){
34        $email = "";
35        $dataError["login"] = "Adresse e-mail invalide.";
36      }else{
37        $email = $_POST["email"];
38      }
39    }
40  }
41  ?>
```

Le test sur la forme du mot de passe, ainsi que la génération du numéro de session (*SID*) sont effectués par une classe d'utilitaires `AuthUtils`

Code Source 7.13 : /sessions/classes/AuthUtils.php

```
1   <?php
2   namespace CoursPHP\Auth;
3   /** @brief Utilitaires de connection (validation REGEX du mot de passe) */
4   class AuthUtils{
5     /** Fonction qui teste si un mot de passe est suffisemment difficile
6       * @param $wouldBePasswd mot de passe (non hashé) saisi par l'utilisateur */
7     public static function isStrongPassword($wouldBePasswd){
8       $lengthCondition = (strlen($wouldBePasswd) >= 8 &&
9                           strlen($wouldBePasswd) <= 35);
10      // On peut sûrement faire plus efficace pour l'évaluation des
11      // expressions régulières...
12      $CharacterDiversityCondition = preg_match("/[a-z]/", $wouldBePasswd)
13                    && preg_match("/[A-Z]/", $wouldBePasswd)
14                    && preg_match("/[0-9]/", $wouldBePasswd)
15                    && preg_match("/[\#\-\|\.\@\[\]\=\!\&]/", $wouldBePasswd);
16      return $lengthCondition && $CharacterDiversityCondition;
17    }
18  }
19  ?>
```

Si tout se passe bien, on crée une session d'*ID* aléatoire qui contient l'adresse *IP* du client, pour contrôle par adresse *IP* lors de la prochaine visite.

Code Source 7.14 : /sessions/ex07-receivePassword.php (cf. Fig 7.6)

```
1   <?php
2     require_once(dirname(__FILE__).'/classes/AuthUtils.php');
3     require_once(dirname(__FILE__).'/classes/SessionUtils.php');
4     require_once(dirname(__FILE__).'/classes/ValidationRequest.php');
```

FIGURE 7.6 : Illustration du code source 7.14

```
5
6   // Fonction à implémenter : test d'existance du login/mot de passe en BD
7   // Voir le chapitre sur les bases de données...
8   function userPasswordCheckInDatabase($email, $hashedPassword){
9     // TODO : tester si le couple e-mail et mot de passe (après hashage SHA512)
10    // sont bien présents dans la base de données
11    return true;
12  }
13  // Test de la forme (regex) du mot de passe et de l'e-mail
14  \CoursPHP\Auth\ValidationRequest::validationLogin($dataError, $email,
        $password);
15
16  if (empty($dataError)){ // les données d'authentification ont la bonne forme.
17    // On vérifie que le mot de passe (après hashage SHA512)
18    // est bien celui en base de donnée.
19    if (!userPasswordCheckInDatabase($email, hash("sha512", $password))){
20      // Renvoi d'une erreur de login
21      $dataError["login"] = "Erreur : login ou mot de passe incorrect";
22    }else{
23      \CoursPHP\Auth\SessionUtils::createSession($email);
24      // Flush des Données de Session, (sauvegarde simmédiate ur le disque)
25      session_write_close();
26    }
27  }
28  // Appel de la vue :
29  if (empty($dataError)){
30    require('ex07-vueNormale.php');
31  }else{ // Appel de la vue d'erreur :
32    require('ex07-vueErreur.php');
33  }
34 ?>
```

Code Source 7.15 : /sessions/ex07-vueNormale.php

```
1 <?php
2   require_once('classes/VueHtmlUtils.php');
3   echo CoursPHP\Vue\VueHtmlUtils::enTeteHTML5("Welcome Page", 'UTF-8',
4                                               'myStyle.css');
5   echo "<h1>Persistance de connexion<br/>Exemple de politique de sécurité</h1>";
6   echo "Bienvenue ! Vous êtes convenablement authentifié.<br/>";
7   echo "Pour accéder encore à des données sensibles, "
```

```
8       ."<a href=\"./ex08-sessionTestIP-RandomIdCookie.php\">cliquez ici</a>.";
9     echo \CoursPHP\Vue\VueHtmlUtils::finFichierHTML5();
10  ?>
```

La création de la session (avec son *ID*), contenant l'*e-mail* et l'adresse *IP* du client est effectuée par une classe utilitaire `SessionUtils` :

Code Source 7.16 : /sessions/classes/SessionUtils.php

```
1   <?php
2   namespace CoursPHP\Auth;
3   /** @brief Gère le cycle de vie de la session (Identification utilisateur)
4    * génère des SID aléatoires, crée et met à jour le cookie pour le SID */
5   class SessionUtils{
6     /** Durée du cookie en secondes */
7     const DUREE_COOKIE = 120;
8
9     /** @brief fonction de génération de l'ID de session aléatoire */
10    public static function generateSessionId(){
11      // Génération de 10 octets (pseudo-)aléatoires codés en hexa
12      $cryptoStrong = false; // Variable pour passage par référence
13      $octets = openssl_random_pseudo_bytes(10, $cryptoStrong);
14      $mySid = bin2hex($octets);
15      return $mySid;
16    }
17
18    /** Création d'une session de SID aléatoire avec l'e-mail (login unique)
19     *  @param $email e-mail servant de login (identifiant unique de l'utilisateur
          )
20     *  @param $role rôle de l'utilisteur (admin, visiteur, gestionnaire...)
21     * (voir le chapitre sur le Front Controller) **/
22    public static function createSession($email, $role="visitor"){
23      // Dans le cas improbable d'une collision sur le SID,
24      // Mais surtout d'une usurpation d'identité, on détruit la session
25      // avant de redémarer une session vide
26      session_write_close();
27      session_start();
28      session_destroy();
29      // Le numéro de session aléatoire
30      $mySid =  self::generateSessionId();
31      session_id($mySid);
32      // Destruction du coockie avant de le recréer
33      setcookie("session-id", "", time()-60, '/');
34      // Création du cookie avec SID aléatoire. Validité du cookie : 2mn
35      // Un pirate aura besoin de temps pour voler le cookie...
36      setcookie("session-id", $mySid, time()+self::DUREE_COOKIE, '/');
37      // Démarrage de la session
38      session_start();
39      // On échappe, même si on sait qu'on a validé l'adresse e-mail....
40      $_SESSION['email'] = htmlentities($email, ENT_QUOTES, "UTF-8");
41      // On échappe, même si on sait qu'on a validé l'adresse e-mail....
42      $_SESSION['role'] = htmlentities($role, ENT_QUOTES, "UTF-8");
43      $_SESSION['ipAddress'] = $_SERVER['REMOTE_ADDR'];
44      session_write_close();
45    }
46  }
47  ?>
```

Lorsque l'utilisateur poursuit la navigation, il reste reconnu et peut accéder à ses données personnelles. Dans notre implémentation, pour plus de sécurité, le numéro de session est à usage unique. Une nouvelle session, avec son nouveau *SID* est créée à chaque chargement de script.

FIGURE 7.7 : Illustration du code source 7.17

Code Source 7.17 : /sessions/ex08-sessionTestIP-RandomIdCookie.php (cf. Fig 7.7)

```php
<?php
  require_once(dirname(__FILE__).'/classes/AuthUtils.php');
  require_once(dirname(__FILE__).'/classes/SessionUtils.php');

  $dataError = array();
  // Test pour voir si l'identifiant de session existe et si la donnée a la
      bonne forme
  // (10 chiffres hexa entre 0 et f)
  if (!isset($_COOKIE['session-id']) ||
          !preg_match("/^[0-9a-fA-F]{20}$/", $_COOKIE['session-id'])){
    $dataError['no-cookie'] = "Votre cookie a peut-être expirée, "
                              ."Merci de vous connecter à nouveau...";
  }else{
    // On récupère l'ID de session
    $mySid = $_COOKIE['session-id'];

    // On récupère les données de session :
    session_id($mySid);
    session_start();

    // Test sur les données de session et contrôle par IP
    if (empty($_SESSION['email']) || empty($_SESSION['ipAddress'])
              || $_SESSION['ipAddress'] != $_SERVER['REMOTE_ADDR']){
      $dataError["no-session"] = "Votre session a peut-être expirée, "
                                 ."Merci de vous connecter à nouveau...";
      session_destroy();
    }else{
      // Raffinement : on change le SID aléatoire, en copiant
```

```
28          // la session dans une nouvelle. On regénère ensuite le cookie
29          // Comme ça, le cookie n'est valable qu'une fois, et l'ID de session aussi
30          // ce qui limite beaucoup la possibilité d'un éventuel hacker
31          $backupSession_Email = $_SESSION['email'];
32          // On détruit l'ancienne session
33          session_destroy();
34          // On recrée une session :
35          CoursPHP\Auth\SessionUtils::createSession($backupSession_Email);
36          // Flush des Données de Session, (sauvegarde simmédiate ur le disque)
37          session_write_close();
38        }
39    }
40    // Code de la vue :
41    if (empty($dataError)){ // Code de la vue normale
42      require('ex08-vueNormale.php');
43    }else{ // Appel de la vue d'erreur :
44      require('ex07-vueErreur.php');
45    }
46  ?>
```

Code Source 7.18 : /sessions/ex08-vueNormale.php

```
1  <?php
2    require_once('classes/VueHtmlUtils.php');
3    echo CoursPHP\Vue\VueHtmlUtils::enTeteHTML5("Consultation des Données
         Personnelles", 'UTF-8', 'myStyle.css');
4    echo "<h1>Consultation des Données Personnelles</h1>";
5    echo "<p>";
6    echo "Ne le dite à personne : le <i>SID</i> est  : <br/>".$mySid;
7    echo "</p><p>";
8    echo "Votre adresse e-mail est : ".$_SESSION['email']."<br/>";
9    echo "Grâce à votre adresse e-mail, le serveur peur retrouver vos données
         personnelles<br/>";
10   echo "et vous les afficher. Et d'ailleurs, les voici...<br/>";
11   echo "Pour accéder encore à des données sensibles, "
12     ."<a href=\"./ex08-sessionTestIP-RandomIdCookie.php\">cliquez ici</a>.";
13   echo "</p>";
14   echo CoursPHP\Vue\VueHtmlUtils::finFichierHTML5();
15 ?>
```

FIGURE 7.8 : Re-chargement du script. La connexion persiste, mais le *SID* a changé.

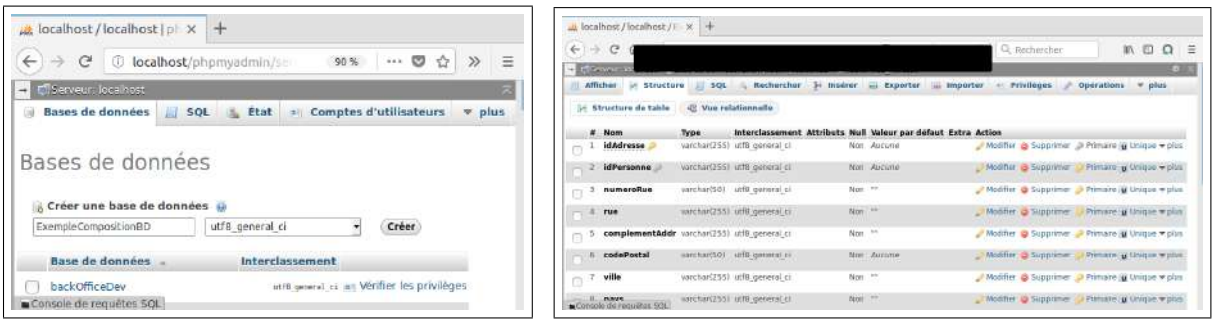
FIGURE 7.9 : Re-chargement du script. Expiration du *cookie*.

Chapitre 8

Bases de Données et *PHP Data Objects*

8.1 Créer un Base de Données dans *phpmyadmin*

8.1.1 Création d'une Base de Données Relationnelle



(a) Création d'une nouvelle Base de Données (b) Création d'une nouvelle table

FIGURE 8.1 : Processus de Création d'une Base de Données et des Tables

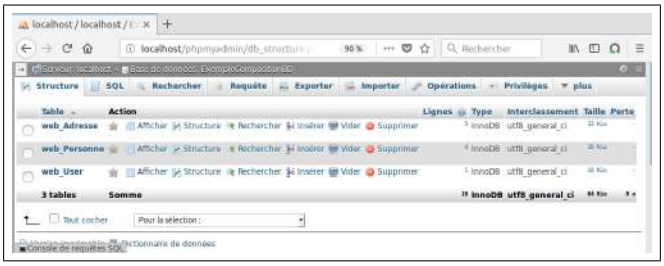


FIGURE 8.2 : Exemple de base de données avec trois tables

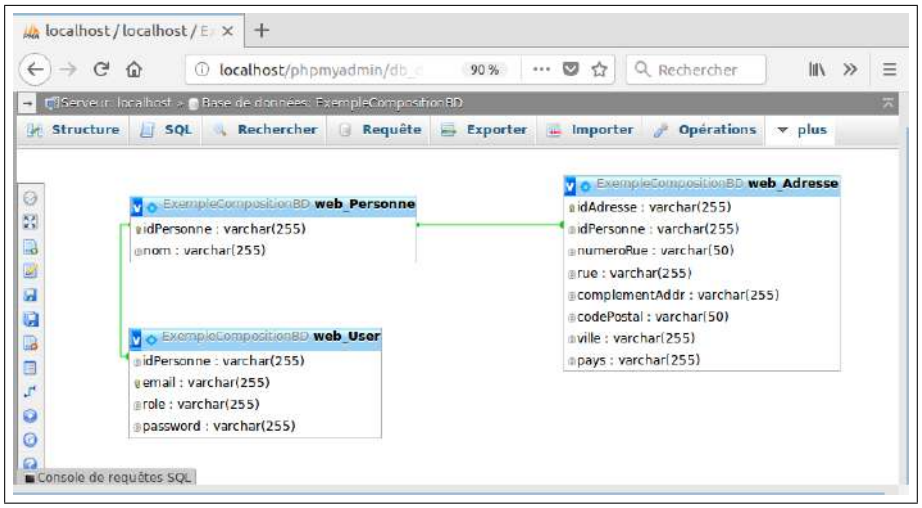
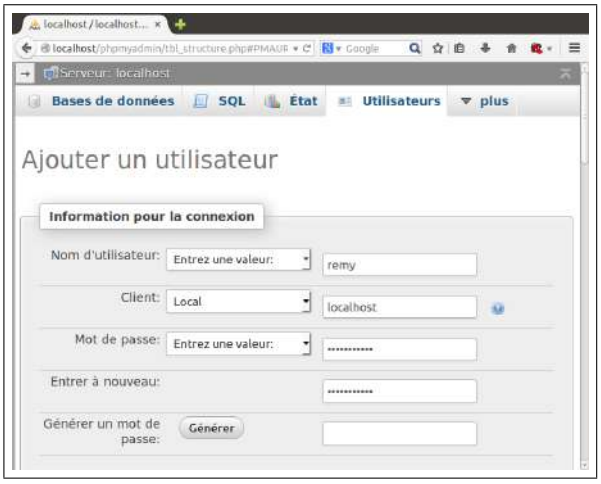


FIGURE 8.3 : Vue Relationnelle de base de données avec trois tables. La définition des clés étrangères doit être cohérente avec les clés primaire.

8.1.2 Créer un Utilisateur *MySql* Responsable d'une *BD*

Pour éviter qu'un éventuel piratage de la configuration de l'authentification pour l'accès aux bases de données, qui se trouve en clair dans les sources *PHP*, ne donnée accès à toutes les bases de données, nous pratiquons des droits "étanches" entre nos différentes bases de données. Dans notre cas, l'utilisateur `remy` aura les droits uniquement sur la base `ExampleDataBase`.

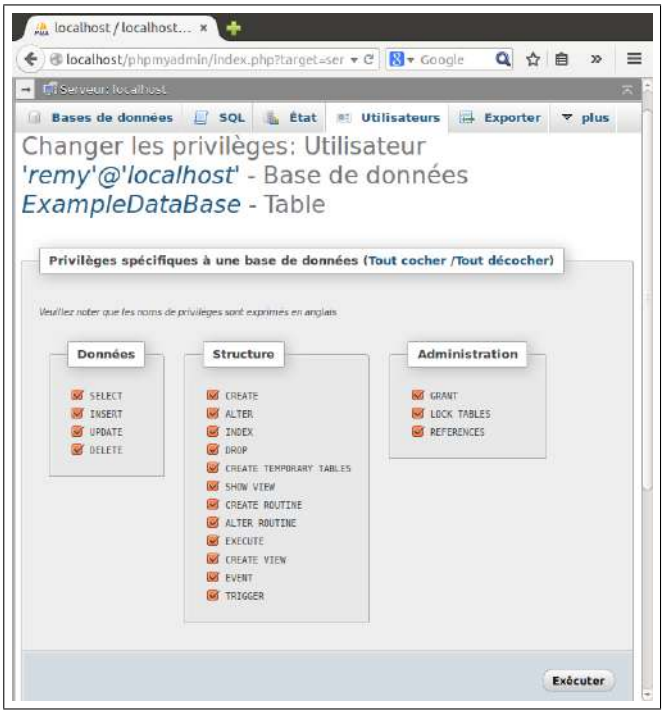
(a) Ajout d'un Utilisateur *MySQL*



(b) Accès *MySQL* Uniquement sur le Serveur Local

FIGURE 8.4 : Ajout d'un utilisateur *MySQL* avec uniquement sur le Serveur Local

(a) Privilèges Spécifiques d'un Utilisateur sur une Base



(b) L'Utilisateur aura tous les droits sur une Base

FIGURE 8.5 : Donner tous les Droits à l'Utilisateur sur une Base spécifique

8.2 Initiation à *PDO* : connexion, query, destruction

L'extension du langage *PHP* appelée *PHP Data Objects* fournit une couche d'abstraction pour accéder à des données, à l'aide de différents *drivers*. Comme exemples de *drivers*, on peut citer *MySQL*, *Oracle*, *SQLite*, *PostgreSQL*, *MS SQL Sever*, etc.

L'intérêt de *PDO* est de permettre d'utiliser tous ces *drivers*, qui accèdent à des bases de données différentes, avec les mêmes fonction : les méthodes de *PDO*.

8.2.1 Établir la Connexion à une Base de Données

La connexion à la base de données se fait avec le nom du driver (ici *mysql*), le nom d'hôte, le nom de la base de données, le nom d'utilisateur ayant les droits sur la base de données, et son mot de passe.

Une éventuelle exception issue du constructeur de `PDO` doit absolument être gérée avec `try...catch` (ou avec un *handler*), car sinon le message de l'exception s'affiche, révélant le nom et le mot de passe de l'utilisateur ayant les droits sur la base.

Code Source 8.1 : /pdo/ex01-testConnexionPDO.php (cf. Fig 8.6)

```
1 <?php
2     $mySqlUser = "remy";
3     $mySqlPassword = "my_password";
4     $dataBase = "ExempleCompositionBD";
5
6     $dataError = array();
7
```

FIGURE 8.6 : Illustration du code source 8.1

```php
  // ON DOIT ABSOLUMENT GÉRER CETTE EXCEPTION, FAUTE DE QUOI
  // L'UTILISATEUR DE LA BASE DE DONNÉES ET LE MOT DE PASSE
  // APPARAÎSSENT EN CLAIR !!!!
  try {
    // Création de l'instance de PDO (database handler).
    $dbh = new PDO('mysql:host=localhost;dbname='.$dataBase, $mySqlUser,
                   $mySqlPassword);
  } catch (PDOException $e) {
    $dataError["connexion"] = "Erreur de connexion à la base de données."
                              ."Vous n'avez pas besoin d'en savoir plus...";
    require("vueErreur.php");
    die();
  }
  // Requête : chaîne de caractères contenant une requête valide en SQL
  $requete = 'SELECT * from web_Adresse';

  // Exécution de la requête et mémorisation du résultat :
  $resultExec = $dbh->exec($requete);

  if ($resultExec === false){
    $dataError["requete"] = "Problème d'exécution de la requête. "
          ."(par exemple, un nom de champs est incorrect, "
          ."ou encore la requête n'est pas valide sur la base...)";
    require("vueErreur.php");
  }else{
    // Code de la vue :
    require('ex01-vueConnexionPDO.php');
  }
  // Fermeture de la connexion (connexion non persistante)
  $resultExec = null;
  $dbh = null;
?>
```

Code Source 8.2 : /pdo/ex01-vueConnexionPDO.php

```php
<?php
  require_once('classes/VueHtmlUtils.php');
  echo CoursPHP\Vue\VueHtmlUtils::enTeteHTML5("Ma Première Connexion PDO",
                                              'UTF-8', 'myStyle.css');
  echo "<h1>Initier une Connexion <i>PDO</i></h1>";
  echo "La requête a bien été exécutée...";
  echo CoursPHP\Vue\VueHtmlUtils::finFichierHTML5();
?>
```

FIGURE 8.7 : Illustration du code source 8.3

Code Source 8.3 : /pdo/vueErreur.php (cf. Fig 8.7)

```php
<?php
   require_once('classes/VueHtmlUtils.php');
   echo \CoursPHP\Vue\VueHtmlUtils::enTeteHTML5("Erreur BD", 'UTF-8',
                                                'myStyle.css');
   echo "<h1>Une erreur s'est produite</h1>";
   foreach ($dataError as $errorMsg){
     echo "<p>".$errorMsg."</p>";
   }
   echo \CoursPHP\Vue\VueHtmlUtils::finFichierHTML5();
?>
```

Voici un exemple où l'exception générée par le constructeur de *PDO* n'est pas gérée. Les données d'authentification pour l'accès à la base de données apparaîssent en clair chez le client.

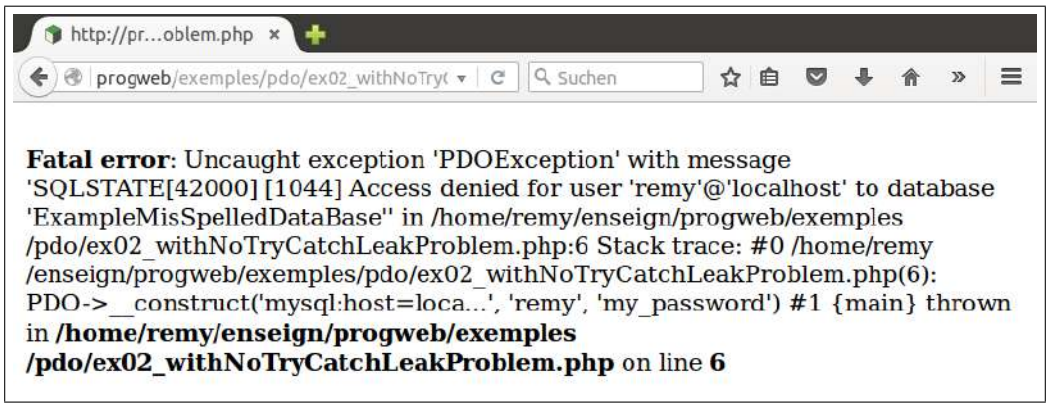


FIGURE 8.8 : Illustration du code source 8.4

Code Source 8.4 : /pdo/ex02-withNoTryCatchLeakProblem.php (cf. Fig 8.8)

```php
<?php
   $mySqlUser = "remy";
   $mySqlPassword = "my_password";
   $dataBase = "ExampleMisSpelledDataBase";

   $dbh = new PDO('mysql:host=localhost;dbname='.$dataBase, $mySqlUser,
                  $mySqlPassword);

   $requete = 'INSERT INTO web_Adresse(id, numeroRue, rue, complementAddr, '
            .'codePostal, ville, pays) '
```

```
11              . 'VALUES ("0fda5a80a", "11", "Allée des Pies Jaunes", '
12              . '"Bâtiment 2D", "63000", "Clermont-Ferrand", "France")';
13    $dbh->query($requete);
14
15      // Code de la vue :
16    require_once('classes/VueHtmlUtils.php');
17    echo CoursPHP\Vue\VueHtmlUtils::enTeteHTML5("Ma Première Connexion PDO",
18                                                'UTF-8', 'myStyle.css');
19    // Code de la vue :
20    foreach($dbh->query('SELECT * from Adresse') as $row) {
21      print_r($row);
22    }
23    $dbh = null;
24    echo CoursPHP\Vue\VueHtmlUtils::finFichierHTML5();
25  ?>
```

Dans les exemples des deux parties suivantes, nous incluerons (par un `require`) le fichier suivant qui réalisera la connexion à la base de données :

Code Source 8.5 : /pdo/ex03-connectToDatabasePDO.php

```
1   <?php
2     $mySqlUser = "remy";
3     $mySqlPassword = "my_password";
4     $dataBase = "ExempleCompositionBD";
5
6     // ON DOIT ABSOLUMENT GÉRER CETTE EXCEPTION, FAUTE DE QUOI
7     // L'UTILISATEUR DE LA BASE DE DONNÉES ET LE MOT DE PASSE
8     // APPARAÎSSENT EN CLAIR !!!!
9     try {
10      // Création de l'instance de PDO (database handler).
11      $dbh = new PDO('mysql:host=localhost;dbname='.$dataBase, $mySqlUser,
            $mySqlPassword);
12    } catch (PDOException $e) {
13      $dataError['connexion-bd'] = "Erreur de connexion à la base de données."
14              ."Vous n'avez pas besoin d'en savoir plus...";
15      require("vueErreur.php");
16    }
17  ?>
```

8.2.2 Parcourir les Résultats d'une Requête

Voici un exemple qui récupère les lignes des résultats d'une requête (de type `SELECT`) sous une forme associative. Les clés du tableau associatif, pour chaque ligne, sont les noms de colonnes du résultat de la requête.

Code Source 8.6 : /pdo/ex03-testQueryForeachModeAssoc.php (cf. Fig 8.9)

```
1   <?php
2     // Connexion à la base de données :
3     require_once(dirname(__FILE__).'/ex03-connectToDatabasePDO.php');
4
5     // Stockage des données résultat de la requête dans une variable
6     // De type PDOStatement
7     $statement = $dbh->query('SELECT * from web_Adresse');
8     // On veut récupérer les lignes comme des tableaux Associatifs
```

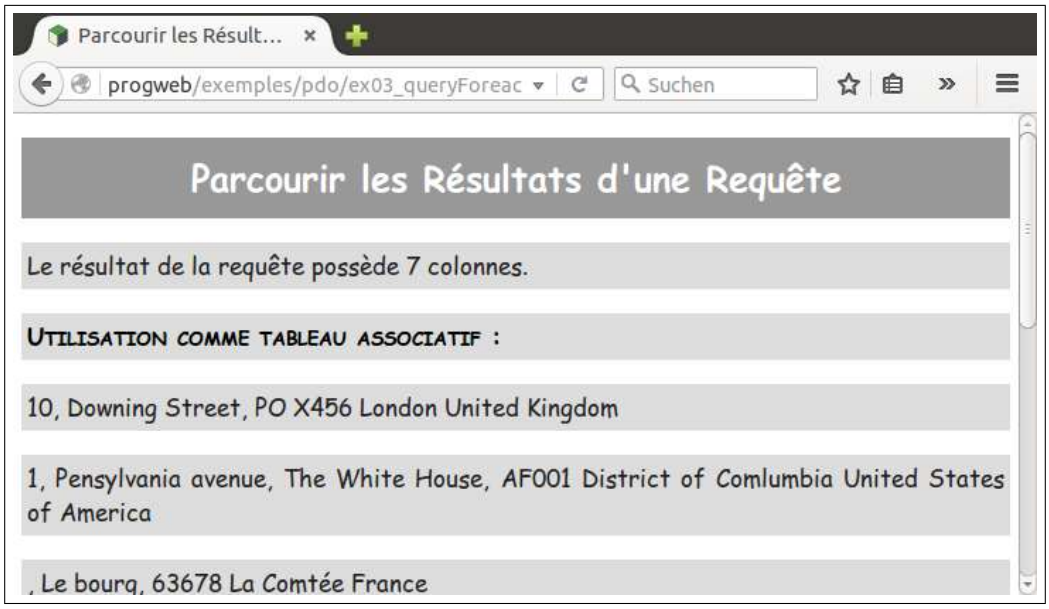


FIGURE 8.9 : Illustration du code source 8.6

```
9   $statement->setFetchMode(PDO::FETCH_ASSOC);
10
11  require_once('classes/VueHtmlUtils.php');
12  echo CoursPHP\Vue\VueHtmlUtils::enTeteHTML5("Parcourir les Résultats",
13                                              'UTF-8', 'myStyle.css');
14  echo "<h1>Parcourir les Résultats d'une Requête</h1>";
15
16  echo "<p>Le résultat de la requête a ".$statement->columnCount()." colonnes.</
        p>";
17
18  echo "<p>";
19  echo "<strong>Utilisation comme tableau associatif :</strong>";
20  echo "</p>";
21  foreach ($statement as $row){
22    echo "<p>";
23    echo $row['numeroRue'].", ".$row['rue'].", ";
24    if (!empty($row['complementAddr']))
25      echo $row['complementAddr'].", ";
26      echo $row['codePostal']." ";
27      echo $row['ville']." ";
28      echo $row['pays'];
29      echo "</p>";
30  }
31
32  // Connexion non persistante : on ferme la connexion
33  // il faut détruire tous les objects liés à la connexion SANS EN OUBLIER
34  // (en les mettant à null, pour que la connexion se ferme).
35  $statement = null;
36  $dbh = null;
37
38  echo CoursPHP\Vue\VueHtmlUtils::finFichierHTML5();
39 ?>
```

Voici un exemple qui récupère les lignes des résultats d'une requête (de type `SELECT`) sous

une forme soit associative, soit numérique. Les clés du tableau associatif, pour chaque ligne, sont les noms de colonnes du résultat de la requête. Les clés du tableau numérique, pour chaque ligne, sont les numéros de colonnes du résultat de la requête (commençant à 0).

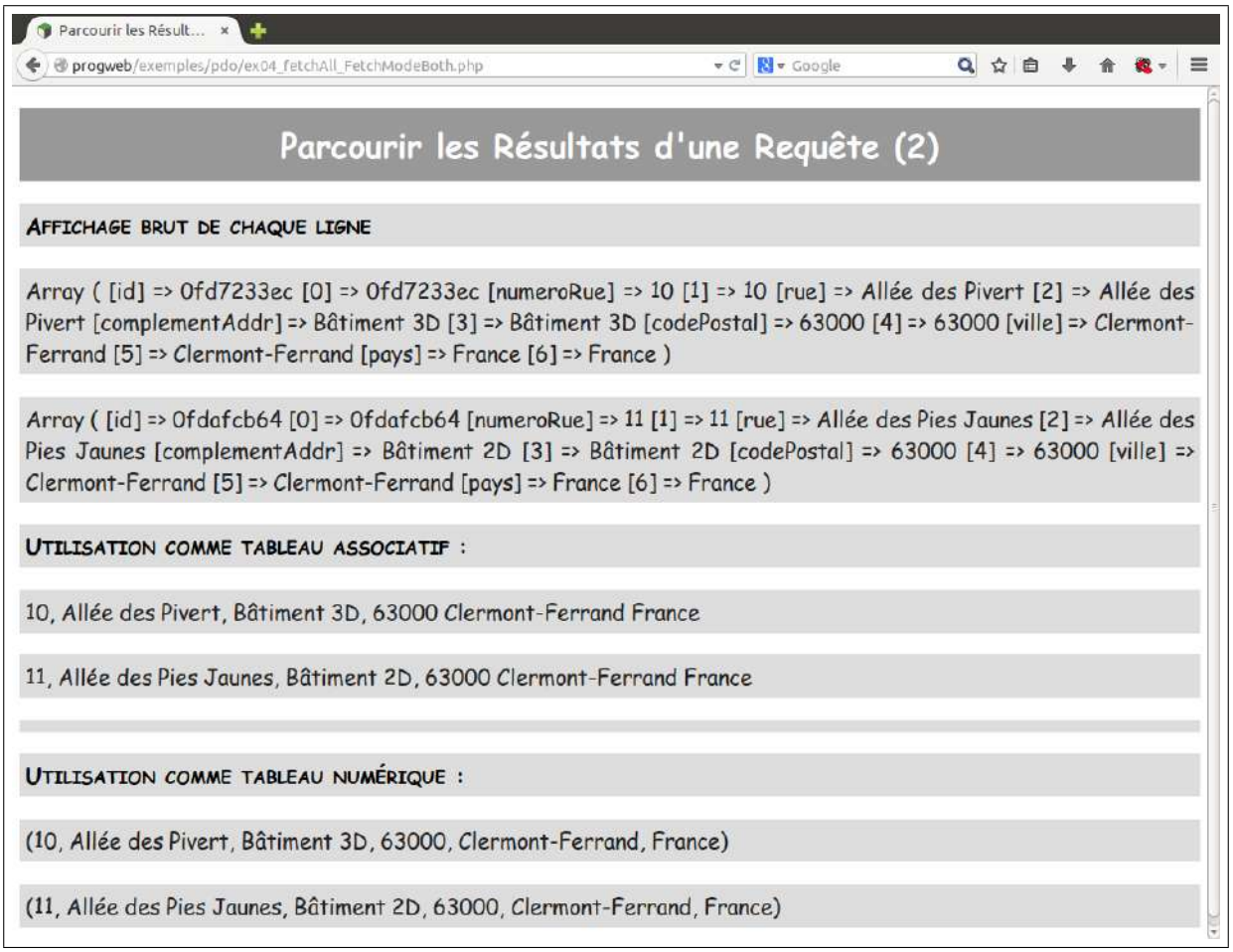


FIGURE 8.10 : Illustration du code source 8.7

Code Source 8.7 : /pdo/ex04-testFetchAll-FetchModeBoth.php (cf. Fig 8.10)

```
1  <?php
2    // Connexion à la base de données :
3    require_once(dirname(__FILE__).'/ex03-connectToDatabasePDO.php');
4
5    // Stockage des données résultat de la requête dans une variable
6    // De type PDOStatement
7    $statement = $dbh->query('SELECT * from web_Adresse');
8    if ($statement === false){
9      $dataError["query"] = "Problème d'exécution de la requête. "
10           ."(par exemple, la connexion n'est pas ouverte ou la table n'existe
                pas...)";
11           die();
12   }
13
14   $statement->setFetchMode(PDO::FETCH_BOTH);
15
16   // Pour pouvoir parcourir trois fois les résultats, on copie ceux-ci
```

```php
17  // dans un grand tableau.
18  // Ça peut utiliser beaucoup de mémoire s'il y a beaucoup de lignes...
19  $tabResultats = $statement->fetchAll();
20
21  require_once('classes/VueHtmlUtils.php');
22  echo CoursPHP\Vue\VueHtmlUtils::enTeteHTML5("Parcourir les Résultats d'une
        Requête", 'UTF-8', 'myStyle.css');
23  echo "<h1>Parcourir les Résultats d'une Requête (2)</h1>";
24  echo "<p>";
25  echo "<strong>Affichage brut de chaque ligne</strong>";
26  echo "</p>";
27
28  foreach($tabResultats as $row) {
29    echo "<p>";
30    print_r($row)."<br/><br/>";
31    echo "</p>";
32  }
33
34  echo "<p>";
35  echo "<strong>Utilisation comme tableau associatif :</strong>";
36  echo "</p>";
37  foreach($tabResultats as $row) {
38    echo "<p>";
39    echo $row['numeroRue'].", ".$row['rue'].", ";
40    if (!empty($row['complementAddr']))
41      echo $row['complementAddr'].", ";
42    echo $row['codePostal']." ";
43    echo $row['ville']." ";
44    echo $row['pays'];
45    echo "</p>";
46  }
47  echo "</p>";
48
49  echo "<p>";
50  echo "<strong>Utilisation comme tableau numérique :</strong>";
51  echo "</p>";
52  foreach($tabResultats as $row) {
53    echo "<p>(";
54    for ($i = 1 ; $i < $statement->columnCount() ; $i++){
55      if ($i>1){
56        echo ", ";
57      }
58      echo $row[$i];
59    }
60    echo ")</p>";
61  }
62  echo CoursPHP\Vue\VueHtmlUtils::finFichierHTML5();
63
64  // Connexion non persistante : on ferme la connexion
65  // il faut détruire tous les objects liés à la connexion SANS EN OUBLIER
66  // (en les mettant à null, pour que la connexion de ferme.
67  $statement = null;
68  $dbh = null;
69 ?>
```

D'une manière générale, les méthodes `fetch` (renvoie une seule ligne des résultats d'une

requête) et `fetchAll` (renvoie toutes les lignes des résultats d'une requête) ont des options sur la structure des données retournées. Une liste (non exaustive !!!) de ces options est :

- `PDO` :: `FETCH_ASSOC` : lignes sous forme d'un tableau associatif indexé par le nom de la colonne ;
- `PDO` :: `FETCH_BOTH` : lignes sous forme d'un tableau à la fois associatif indexé par le nom de la colonne et numérique indexé par le numéro de la colonne ;
- `PDO` :: `FETCH_OBJ` : lignes sous la forme d'objets anonymes dont les propriétés sont les noms de la colonne ;

8.3 Requêtes Préparées

8.3.1 Principe des Requêtes Préparées

L'idée des requêtes préparées est la suivante :

1. On indique à *PDO* la requête *SQL*, sauf que les valeurs (attributs des tables...) ne sont pas précisées (ce sont des `?`).
2. Cela permet déjà à *PDO* d'analyser une fois pour toute la requête, même si on doit exécuter la requête plusieurs fois avec des valeurs différentes. C'est ce qu'on appelle préparer la requête. Cela améliore généralement l'efficacité, réduisant la charge du serveur et les délais d'exécution des requêtes.
3. Avant d'exécuter la requête préparée, ou au moment de l'exécution de la requête préparée, on spécifie les valeurs (attributs des tables...) qui viennent remplace les `?` dans la requête. **Ces valeurs, qui peuvent correspondre à des inputs utilisateur, sont automatiquement filtrée, évitant tout risque d'injection *SQL*.**

Le mécanisme des requêtes préparées repose sur un lien effectué (avec la méthode `bindParam`) entre une variable *PHP* (donnée par sa référence), et une valeur non fixée (`?`) dans la requête.

Il peut y avoir plusieurs syntaxes pour les requêtes préparées.

8.3.2 Syntaxe avec des Points d'Interrogation (`?`)

Voyons déjà un exemple d'insertion d'une adresse dans une table. L'adresse est saisie dans un formulaire :

Code Source 8.8 : /pdo/ex05-testFormAdresse.php

```
1 <!doctype html>
2 <html lang="fr">
3 <head>
4   <meta charset="UTF-8" />
5 <link rel="stylesheet" href="./myStyle.css" />
6   <title>Saisie d'une Adresse</title>
7 </head>
8 <body>
9   <h1>Saisie d'une adresse</h1>
```

```
10 <form method="post" action="ex07-requetesPreparees.php">
11   <p>
12   <label for="numeroRue">Numéro</label>
13   <input type="text" name="numeroRue" id="numeroRue"  size="4"/><br/>
14     </p>
15   <p>
16   <label for="rue">Place/Rue*</label>
17   <input type="text" name="rue"    id="rue" size="30"/>
18 </p>
19
20 <p>
21   <label for="complementAddr">Complément d'adresse</label>
22   <input type="text" name="complementAddr" id="complementAddr" size="30"/><br/>
23 </p>
24   <p>
25   <label for="codePostal">Code postal*</label>
26   <input type="text" name="codePostal"     id="codePostal"  size="10"/><br/>
27     </p>
28   <p>
29   <label for="ville">Ville*</label>
30   <input type="text" name="ville" id="ville" size="10"/><br/>
31     </p>
32 <p>
33   <input type="submit" value="Envoyer" class="sansLabel"></input>
34   </p>
35 </form>
36 </body>
37 </html>
```

Les valeurs saisies par l'utilisateur seront récupérées du tableau $_POST dans un fichier *PHP*, qui sera inclus par un `require` juste avant d'exécuter la requête de type `INSERT` :

Code Source 8.9 : /pdo/ex06-retrieveInputPosts.php

```
1  <?php
2    $numeroRue="";
3    if (isset($_POST['numeroRue'])){
4      $numeroRue = filter_var($_POST['numeroRue'], FILTER_SANITIZE_STRING);
5    }
6    $rue="";
7    if (isset($_POST['rue'])){
8      $rue = filter_var($_POST['rue'], FILTER_SANITIZE_STRING);
9    }
10   $complementAddr="";
11   if (isset($_POST['complementAddr'])){
12     $complementAddr = filter_var($_POST['complementAddr'],
           FILTER_SANITIZE_ENCODED, FILTER_FLAG_ENCODE_LOW|FILTER_FLAG_ENCODE_HIGH)
           ;
13   }
14   $codePostal="";
15   if (isset($_POST['codePostal'])){
16     $codePostal = filter_var($_POST['codePostal'], FILTER_SANITIZE_STRING);
17   }
18   $ville="";
19   if (isset($_POST['ville'])){
20     $ville = filter_var($_POST['ville'], FILTER_SANITIZE_STRING);
21   }
```

```php
  $pays="France";
  if (isset($_POST['pays']) && $_POST['pays'] != ""){
    $pays = filter_var($_POST['pays'], FILTER_SANITIZE_STRING);
  }
?>
```

Voici enfin l'exemple qui effectue :

1. La préparation de la requête de type `INSERT` (avec des "`?`" à la place des attributs de l'adresse) ;
2. Définit (avec `bindValue`) le lien entre les "`?`" et des variables *PHP* ;
3. Exécute la requête (en effectuant les tests d'erreur).

Code Source 8.10 : /pdo/ex07-requetesPreparees.php

```php
<?php
  // Connexion à la base de données
  require_once(dirname(__FILE__).'/ex03-connectToDatabasePDO.php');
  $requete = 'INSERT INTO web_Adresse(idAdresse, idPersonne, '
                    .'numeroRue, rue, complementAddr, codePostal, ville, pays)
                    '
                    .'VALUES (?, ?, ?, ?, ?, ?, ?, ?)';
  // Préparation de la requête (chaîne représentant une requête SQL
  // sauf que les valeurs à insérer dont des ?)
  $statement = $dbh->prepare($requete);
  // Test en supposant le mode de gestion d'erreurs PDO::ERRMODE_SILENT
  // (sinon, en mode PDO::ERRMODE_EXCEPTION il faudrait utiliser try...catch)
  if ($statement === false){
    $dataError['preparation-query'] = "Problème de préparation de la requête."
     ."(par exemple, la syntaxe de la requête est invalide "
     ."pour le driver utilisé...)";
  }else{
    // Liaison de variables avec les "?" de la requête préparée :
    // Le premier paramètre de bindParam est ici le numéro du "?"
    // en commençant par 1.
    // Lors de l'exécution de la requête, chaque "?" sera remplacé par
    // le contenu de la variable correspondante.
    $statement->bindParam(1, $idAdresse);
    $statement->bindParam(2, $idPersonne);
    $statement->bindParam(3, $numeroRue);
    $statement->bindParam(4, $rue);
    $statement->bindParam(5, $complementAddr);
    $statement->bindParam(6, $codePostal);
    $statement->bindParam(7, $ville);
    $statement->bindParam(8, $pays);

    // Récupération des données du formulaires et affectation des variables
    // $numeroRue , $rue, $complementAddr, $codePostal, $ville, $pays
    // à partir des données utilisateur (tableau $_POST)
    require(dirname(__FILE__).'/ex06-retrieveInputPosts.php');

    // Génération d'un $id difficile à deviner.
    $idAdresse = hash("sha512", $numeroRue.$rue.$complementAddr.$codePostal.
        $ville.$pays);
```

```php
    $idAdresse = substr($idAdresse, 0, 10); // respect de la forme des ID (10 chiffres hexa)
    $idPersonne = "69a1666c6c";
    // Exécution de la requête. (Tous les "?" de la requête ont été liés à des variables)
    if ($statement->execute() === false){
      $dataError["execute-query"] = "Problème d'exécution de la requête. "
          ."(par exemple, une ligne avec cette clé primaire $idAdresse existe déjà...)";
    }
  }

  // Appel de la vue (ou vue d'erreur)
  if (!empty($dataError)){
    require("vueErreur.php");
  }else{  // Code de la vue :
    require("vueNormale.php");
  }
  // Fermeture de la connexion (connexion non persistante)
  $statement = null;
  $dbh = null;
?>
```

Voici un autre exemple de requête préparée, avec une requête de type `SELECT`.

Code Source 8.11 : /pdo/ex08-testRequetesPrepareesSelect.php

```php
<?php
  // Connexion à la base de données :
  require_once(dirname(__FILE__).'/ex03-connectToDatabasePDO.php');

  // Préparation de la requête (chaîne représentant une requête SQL
  // sauf que les valeurs à insérer dont des ?
  $statement = $dbh->prepare('SELECT * FROM web_Adresse WHERE codePostal = ?');

  // Test en supposant le mode de gestion des erreurs PDO::ERRMODE_SILENT
  // (sinon, en mode PDO::ERRMODE_EXCEPTION il faudrait utiliser try...catch)
  if ($statement === false){
    $dataError['preparation-query'] = "Problème de préparation de la requête. "
     ."(par exemple, la syntaxe de la requête est invalide "
     ."pour le driver utilisé...)";
  }else{
    // Liaison de la variable $_GET['codePostal'] avec le "?" de la requête :
    // Le premier paramètre de bindParam est ici le numéro du "?", à savoir 1
    $statement->bindParam(1, $_GET['codePostal']);

    // Test d'erreur en supposant le mode de gestion des erreurs PDO::ERRMODE_SILENT
    // (sinon, avec le mode PDO::ERRMODE_EXCEPTION il faudrait utiliser avec try...catch)
    if ($statement->execute() === false){   // Code de la vue :
      $dataError["execute-query"] = "Problème d'exécution de la requête. ";
    }
  }
  // Appel de la vue (ou vue d'erreur)
  if (!empty($dataError)){
    require("vueErreur.php");
```

```
29    }else{  // Code de la vue :
30      require("ex08-vueRequetesPrepareesSelect.php");
31    }
32    // Fermeture de la connexion (connexion non persistante)
33    $statement = null;
34    $dbh = null;
35  ?>
```

Code Source 8.12 : /pdo/ex08-vueRequetesPrepareesSelect.php

```
1   <?php
2     require_once('classes/VueHtmlUtils.php');
3     echo CoursPHP\Vue\VueHtmlUtils::enTeteHTML5("Requêtes Préparées",
4                                                 'UTF-8', 'myStyle.css');
5     echo "<h1>Requêtes Préparées (2) (Donnez un Code Postal)</h1>";
6     // Affichage des résultats de la requête
7     foreach ($statement as $row){
8       echo "<p>";
9       echo $row['numeroRue'].", ".$row['rue'].", ";
10      if (!empty($row['complementAddr']))
11        echo $row['complementAddr'].", ";
12        echo $row['codePostal']." ";
13        echo $row['ville']." ";
14        echo $row['pays'];
15        echo "</p>";
16    }
17    echo CoursPHP\Vue\VueHtmlUtils::finFichierHTML5();
18  ?>
```

8.3.3 Syntaxe avec des `:name`

Un autre syntaxe consiste à donner des noms aux variables à substituer, au lieu des points d'interrogation "`?`" qui sont anonymes. La syntaxe avec les "`?`" repose sur l'ordre des variables dans un tableau indexé, et il y a plus de risque de se mélanger les pinceaux. Dans la syntaxe avec des `:name`, chaque `:quelqueChose` dans la requête devra être liée (avec `bindParam`) à une variable (ou élément de tableau).

Code Source 8.13 : /pdo/ex09-testRequetesPrepareesV2.php

```
1   <?php
2     // Connexion à la base de données
3     require_once(dirname(__FILE__).'/ex03-connectToDatabasePDO.php');
4
5     // Préparation de la requête (chaîne représentant une requête SQL
6     // sauf que les valeurs à insérer dont des ?)
7     $statement = $dbh->prepare('REPLACE INTO web_Adresse(idAdresse, idPersonne, '
8                                .'numeroRue, rue, complementAddr, codePostal, ville,
                                    pays) '
9                                .'VALUES (:idAdresse, :idPersonne, :numeroRue, :rue,
                                    '
10                               .':complementAddr, :codePostal, :ville, :pays)'
11                               );
12
13    // Test en supposant le mode de gestion d'erreurs PDO::ERRMODE_SILENT
14    // (sinon, en mode PDO::ERRMODE_EXCEPTION il faudrait utiliser try...catch)
```

```php
  if ($statement === false){
    $dataError['preparation-query'] = "Problème de préparation de la requête."
     ."(par exemple, la syntaxe de la requête est invalide "
     ."pour le driver utilisé...)";
  }else{
    // Création d'un tableau associatif avec les valeurs :
    $inputArray = array(
            "idAdresse" => "0123456788",
            "idPersonne" => "69a1666c6c",
            "numeroRue" => "2 bis",
            "rue" => "Rue de la Paix",
            "complementAddr" => "Résid. \"Les Flots\"",
            "codePostal" => "63000",
            "ville" => "Clermont-Ferrand",
            "pays" => "France");
    // Liaison de variables ":quelqueChose" de la requête préparée :
    // Lors de l'exécution de la requête, chaque ":quelqueChose" sera remplacé par
    // le contenu de la variable correspondante $inputArray["quelqueChose"].
    $statement->bindParam(":idAdresse", $inputArray["idAdresse"]);
    $statement->bindParam(":idPersonne", $inputArray["idPersonne"]);
    $statement->bindParam(":numeroRue", $inputArray["numeroRue"]);
    $statement->bindParam(":rue", $inputArray["rue"]);
    $statement->bindParam(":complementAddr", $inputArray["complementAddr"]);
    $statement->bindParam(":codePostal", $inputArray["codePostal"]);
    $statement->bindParam(":ville", $inputArray["ville"]);
    $statement->bindParam(":pays", $inputArray["pays"]);

    // Exécution de la requête.
    // (Tous les ":quelqueChose" de la requête ont été liés à des variables)
    if ($statement->execute() === false){
      $dataError["execute-query"] = "Problème d'exécution de la requête. "
           ."(par exemple, une ligne avec cette clé primaire "
           .$inputArray["idAdresse"]." existe déjà...)";
    }
  }
  // Appel de la vue (ou vue d'erreur)
  if (!empty($dataError)){
    require("vueErreur.php");
  }else{  // Code de la vue :
    require("vueNormale.php");
  }
  // Fermeture de la connexion (connexion non persistante)
  $statement = null;
  $dbh = null;
?>
```

L'un des avantages de la syntaxe avec des `:name` est de permettre une automatisation aisée de la préparation et de l'exécution de la requête à partir d'un tableau associatif contenant les valeurs. Dans l'exemple suivant le tableau associatif contient les attributs d'une `Adresse`. On pourrait aussi lier et exécuter automatiquement la requête à partir d'un tableau `$_REQUEST` directement issu d'un formulaire.

Code Source 8.14 : /pdo/ex10-testRequetesPrepareesV3.php

```php
<?php
  // Connexion à la base de données
```

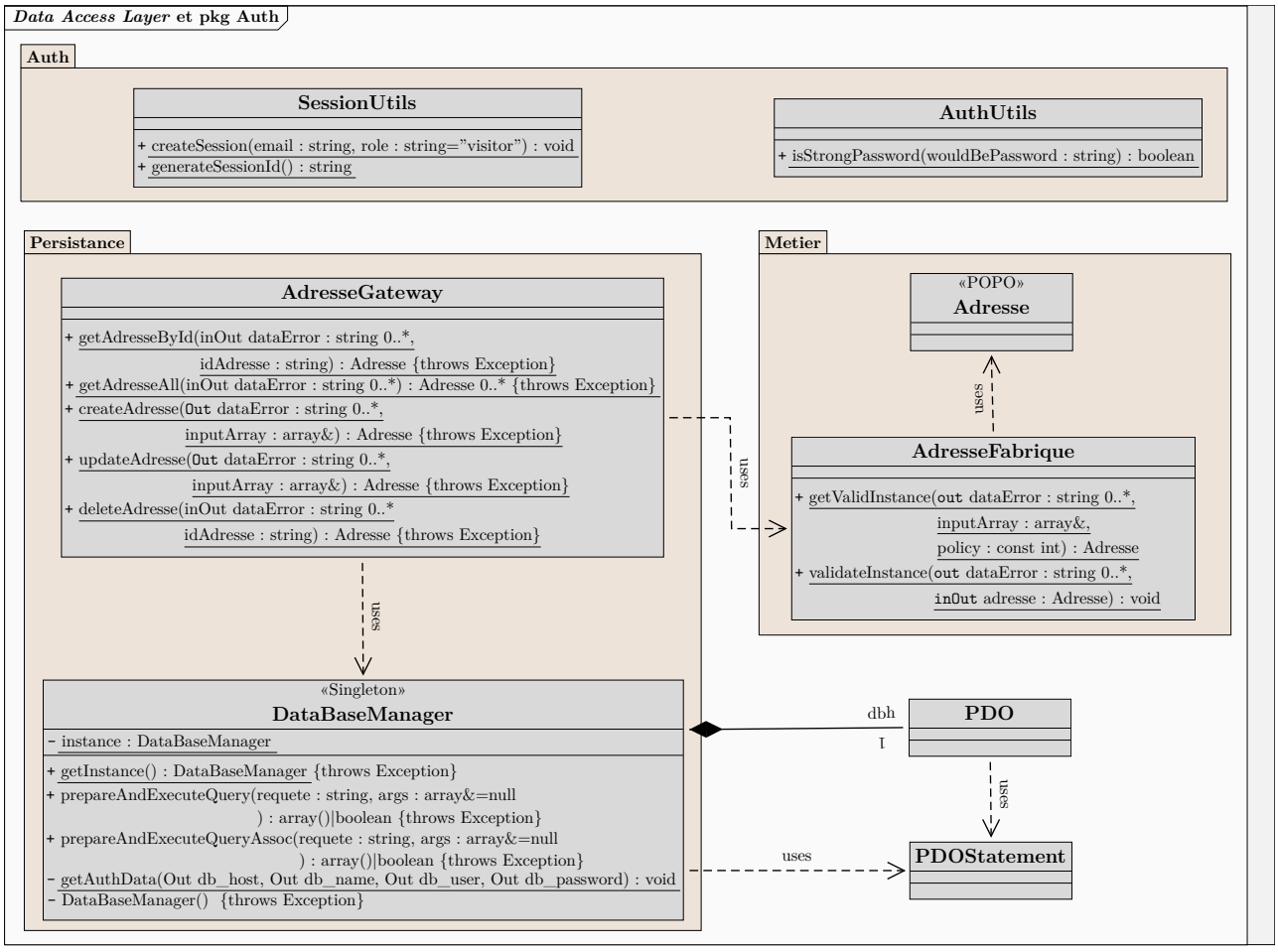
```php
require_once(dirname(__FILE__).'/ex03-connectToDatabasePDO.php');

// La requête à préparer avec des ":quelqueChose" à la place des valeurs
$requete = 'REPLACE INTO web_Adresse(idAdresse, idPersonne, '
                        .'numeroRue, rue, complementAddr, codePostal, ville, pays) '
                        .'VALUES (:idAdresse, :idPersonne, :numeroRue, :rue, '
                        .':complementAddr, :codePostal, :ville, :pays)';

// Préparation de la requête (chaîne représentant une requête SQL
// sauf que les valeurs à insérer dont des ?)
$statement = $dbh->prepare($requete);

// Test en supposant le mode de gestion d'erreurs PDO::ERRMODE_SILENT
// (sinon, en mode PDO::ERRMODE_EXCEPTION il faudrait utiliser try...catch)
if ($statement === false){
  $dataError['preparation-query'] = "Problème de préparation de la requête.";
}else{
  // Création d'un tableau associatif avec les valeurs :
  $inputArray = array(
          "idAdresse" => "9876543211",
          "idPersonne" => "69a1666c6c",
          "numeroRue" => "2 Ter",
          "rue" => "Rue de la Sérénité",
          "complementAddr" => "Complément utile",
          "codePostal" => "63001",
          "ville" => "Clermont-Ferrand",
          "pays" => "France");
  // Liaison de variables ":quelqueChose" de la requête préparée :
  // Lors de l'exécution de la requête, chaque ":quelqueChose" sera remplacé par
  // le contenu de la variable correspondante $inputArray["quelqueChose"].

  //1) On recherche dans la requete les chaînes de la forme ":quelqueChose"
  preg_match_all("/\:[a-zA-Z][a-zA-Z0-9]+/", $requete,
                                  $keyCollection, PREG_PATTERN_ORDER);
  // On parcours les arguments de la requête
  foreach ($keyCollection[0] as $key){
    $associativeKey = substr($key, 1); // clé dans le tableau $args
    $statement->bindParam($key, $inputArray[$associativeKey]);
  }

  // Exécution de la requête.
  // (Tous les ":quelqueChose" de la requête ont été liés à des variables)
  if ($statement->execute() === false){
        $dataError["execute-query"] = "Problème d'exécution de la requête. "
          ."(par exemple, une ligne avec cette clé primaire ".
           $inputArray['idAdresse']." existe déjà...)";
  }
}
// Appel de la vue (ou vue d'erreur)
if (!empty($dataError)){
  require("vueErreur.php");
}else{  // Code de la vue :
  require("vueNormale.php");
```

```
56    }
57    // Fermeture de la connexion (connexion non persistante)
58    $statement = null;
59    $dbh = null;
60  ?>
```

Chapitre 9

Couche d'Accès aux données (*DAL*)

9.1 Diagrammes de Conception



Diag 5. Diagramme de classes de la *Data Access Layer* et des utilitaires d'authentification

9.2 Classe de Connexion à une Base de Données

9.2.1 Principe du Singleton et Construction (classe `Config`)

Nous présentons ici une classe `DataBaseManager` gestionnaire de connexion à une base de données. La connexion est persistante, c'est à dire que l'on ne va pas réinitialiser la connexion sans arrêt. Cette gestion de la connexion permet l'exécution plus rapide de requêtes, en évitant d'établir à chaque fois la connexion.

Pour cela, la classe suit me *Design Pattern* du Singleton. Cela garantit que nous n'aurons qu'une seule instance de la classe à la fois. La construction de l'instance, qui initialise la connexion à la base de données, fait appel à une classe `Config`, qui contiendra à l'avenir tous les paramètres relatifs à l'installation de notre application. Ici, la classe `Config` permet d'initialiser les paramètres de connexion (utilisateur *MySql*/mot de passe pour la *BD*), ainsi que le préfixe commun des tables.

Code Source 9.1 : /pdo/classes/Config.php

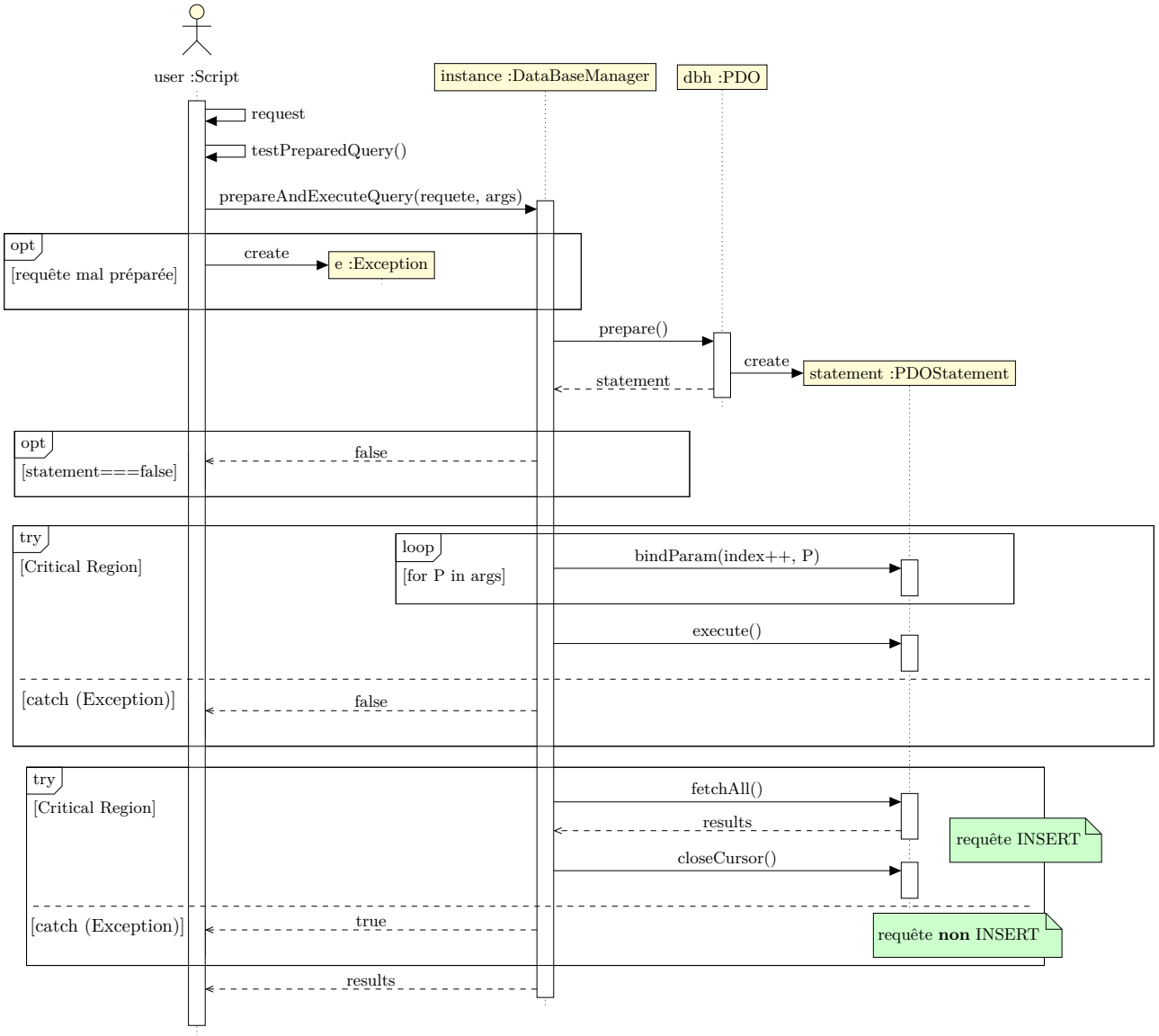
```
1  <?php
2  namespace CoursPHP\Config;
3  /** @brief Classe de configuration
4   * Donne accès aux paramères spécifiques concernant l'application
5   * telles que les chemins vers les vues, les vues d'erreur,
6   * les hash pour les ID de sessions, etc. */
7  class Config {
8    /** @brief Données nécessaires à la connexion à la base de données.
9     * Les valeurs pourraient être initialisées à partir d'un
10    * fichier de configuration séparé (require('configuration.php'))
11    * pour faciliter la maintenance par le webmaster. */
12   public static function getAuthData(&$db_host, &$db_name, &$db_user,
13                                      &$db_password){
14     $db_host="mysql:host=localhost;";
15     $db_name="dbname=ExempleCompositionBD";
16     $db_user="remy";
17     $db_password="my_password";
18   }
19
20   /** @return Le préfixe commun aux tables de la BD de l'application */
21   public static function getTablesPrefix(){
22     return "web_";
23   }
24 }
```

9.2.2 Requêtes préparées avec des ?

La méthode `prepareAndExecuteQuery` prend deux arguments `$requete` et `$args` :

1. la requête avec des ?

2. un tableau des arguments qui doivent remplacer les " ?" dans la requête.



Diag 6. Diagramme de séquence de la méthode `DataBaseManager::prepareAndExecuteQuery()`

Code Source 9.2 : /pdo/classes/DataBaseManager.php

```php
<?php
namespace CoursPHP\Persistance;
/** @brief Permet de gérer la connexion à une base de données (ici MySQL)
 * L'exécution de requêtes SQL avec préparation "offerte service compris".
 * La classe est gérée avec le pattern SINGLETON, qui permet
 * d'avoir un exemplaire unique du gestionaire de connexion,
 * pour une connexion persistante.
 * La classe encapsule complètement PDO, y compris les exceptions. */
class DataBaseManager{
  /** Gestionnaire de connexion à la base de données avec PDO */
  private  $dbh = null;

  /** Référence de l'unique instance de la classe suivant le modèle Singleton.
   * Initialement null */
  private static $instance=null;
```

```
16
17    /** @brief Constructeur qui crée une instance de PDO avec données UTF8
18     * Le constructeur est privé : EnsembleClass ne peut créer des instances
19     * car dans le singleton il ne doit y avoir qu'une seule instance.
20     * Récupère les exception PDO et établit le mode d'erreur "EXCEPTION".
21     * @throws exception personnalisée en cas d'exception PDO */
22    private function __construct(){
23      try {
24        \CoursPHP\Config\Config::getAuthData($db_host, $db_name, $db_user,
              $db_password);
25        // Création de l'instance de PDO (database handler).
26        $this->dbh = new \PDO($db_host.$db_name, $db_user, $db_password);
27              // Rendre les erreurs PDO détectables et gérables par exceptions :
28        $this->dbh->setAttribute(\PDO::ATTR_ERRMODE, \PDO::ERRMODE_EXCEPTION);
29        $this->dbh->setAttribute(\PDO::MYSQL_ATTR_INIT_COMMAND, 'SET NAMES UTF8');
30      }catch (\PDOException $e){
31        throw new \Exception("Désolé, une erreur s'est produite. Si le problème
              persiste, merci de le signaler à Rémy.");
32      }
33    }
34
35    /** @brief Méthode statique publique d'accès à l'unique instance.
36     * Si l'instance n'existe pas, elle est crée. On retourne l'unique instance */
37    public static function getInstance()
38    {
39      if (null === self::$instance) {
40        self::$instance = new self;
41      }
42      return self::$instance;
43    }
44
45    /** @brief Prépare et exécute une requête.
46     * @param $requete requête avec des ?
47     * @param $args arguments à lier (binder) aux ? dans la requête
48     *        Passage par référence pour éviter une recopie.
49     * @return false si la requête échoue,
50     *         true si succès ET requête différente de SELECT,
51     *         ou résultats du SELECT dans un array à double entrée PHP standard
52     * @throws exception personnalisée en cas d'exception PDO */
53    public function prepareAndExecuteQuery($requete, &$args = null){
54      if ($args === null){
55          $args = array();
56      }
57      // récupération du nombre d'arguments :
58      $numargs = count($args);
59      // Une requête préparée ne doit pas contenir de guillemets !!!
60      if (empty($requete) || !is_string($requete) ||
61                                  preg_match('/(\"|\')+/', $requete) !== 0){
62        throw new \Exception("Erreur concernant la sécurité. "
63                             ."Requête incomplètement préparée.");
64      }
65      // On ne laisse pas remonter d'exceptions PDO
66      try{
67        // Préparation de la requête
68        $statement = $this->dbh->prepare($requete);
69        if ($statement !== false){
```

```php
        // On parcours les arguments en commençant au deuxième
        // on commence après le paramètre $requete
        for ($i=1 ; $i <= $numargs; $i++){
          // Lien entre l'argument et le "?" numéro i
          // (rappel : les "?" sont numérotés à partir de 1)
          $statement->bindParam($i, $args[$i-1]);
        }
        // Exécution de la requête préparée :
        $statement->execute();
      }
    }catch (\Exception $e){
      return false;
    }
    // Si la requête échoue :
    if ($statement === false){
      return false;
    }
    // Tentative d'obtenir des résultats de SELECT en array
    try{
      $results = $statement->fetchAll(\PDO::FETCH_ASSOC);
      // destruction des données du PDOstatement
      $statement->closeCursor();
    } catch (\PDOException $e) {
      // La requête a été exécutée mais pas de résultats
      // La requête n'est pas de type SELECT...
      $results = true;
    }

    // Libération via la grabage collector
    $statement = null;

    return $results; // retour des données de requête
  }

  /** @brief Prépare et exécute une requête.
   * @param $requete  requête avec des ":name" pour PDO::prepare
   * @param $args tableau associatif des valeurs à lier aux ":name"
   *              Les clés "quelqueChose" du tableau associatif args
   *        doivent correspondre aux ":quelqueChose" de requete
   *              Passage par référence pour éviter une recopie.
   * @return false si la requête échoue,
   *         true si succès ET requête différente de SELECT,
   *         ou résultats du SELECT dans un array à double entrée PHP standard
   * @throws exception personnalisée en cas d'exception PDO */
  public function prepareAndExecuteQueryAssoc($requete, &$args = null){
    if ($args === null){
        $args = array();
    }
    // récupération du nombre d'arguments :
    $numargs = count($args);
    // Une requête préparée ne doit pas contenir de guillemets !!!
    if (empty($requete) || !is_string($requete) ||
                              preg_match('/(\"|\')+/', $requete) !== 0){
      throw new \Exception("Erreur concernant la sécurité. "
                           ."Requête incomplètement préparée.");
    }
```

```php
    // On ne laisse pas remonter d'exceptions PDO du wrapper
    try{
      // Préparation de la requête
      $statement = $this->dbh->prepare($requete);
      if ($statement !== false){ // si la syntaxe est correcte
        // On recherche dans la requete les valeurs à associer via bindParam
        // Chaînes de la forme ":quelqueChose"
        preg_match_all("/\:[a-zA-Z][a-zA-Z0-9\_]+/", $requete,
                       $keyCollection, PREG_PATTERN_ORDER);
        // On parcours les arguments de la requête
        foreach ($keyCollection[0] as $key){
          $associativeKey = substr($key, 1); // clé dans le tableau $args
          $statement->bindParam($key, $args[$associativeKey]);
        }
        // Exécution de la requête préparée :
        $statement->execute();
      }
    }catch (\Exception $e){
      return false;
    }
    // Si la requête échoue :
    if ($statement === false){
      return false;
    }
    // Tentative d'obtenir des résultats de SELECT en array
    try{
      $results = $statement->fetchAll(\PDO::FETCH_ASSOC);
      // destruction des données du PDOstatement
      $statement->closeCursor();
    } catch (\PDOException $e){
      // La requête a été exécutée mais pas de résultats
      // La requête n'est pas de type SELECT...
      $results = true;
    }
    // Libération via la garbage collector
    $statement = null;

    return $results; // retour des données de requête
  }

  /** @brief on interdit le clonage (pour le pattern singleton). */
  private function __clone(){}
}
?>
```

Voici une utilisation de cette classe de gestion de la base de données, avec une requête qui affiche les adresses dont le code postal est passé par la méthode *GET*.

Code Source 9.3 : /pdo/ex13-testSingletonPDO.php

```php
<?php
  require_once(dirname(__FILE__).'/classes/Config.php');
  require_once(dirname(__FILE__).'/classes/DataBaseManager.php');

  try{
    // Arguments de la requête (tableau indexé 0 => valeur0, 1 => valeur1)
    $args = array(isset($_GET['codePostal']) ? $_GET['codePostal'] : "");
```

```
8
9       $tableName = \CoursPHP\Config\Config::getTablesPrefix().'Adresse';
10      // Requête avec des ? (méthode prepareAndExecuteQuery)
11      $queryResults = CoursPHP\Persistance\DataBaseManager
12                              ::getInstance()->prepareAndExecuteQuery(
13                                                  'SELECT * FROM '.$tableName
14                                                  .' WHERE codePostal = ?',
15                                                  $args // valeurs
16                                                );
17    }catch (Exception $e){
18      $dataError[] =  $e->getMessage();
19      require("vueErreur.php");
20    }
21
22    if ($queryResults === false){
23      // Erreur lors de l'exécution de la requête
24      $dataError[] = "Problème lors de la préparation de la requête. "
25       ."(par exemple, la donnée codePostal passée par GET est invalide...)";
26       require("vueErreur.php");
27    }else{
28      // Code de la vue :
29      require("vueTestSingletonPDO.php");
30    }
31  ?>
```

Code Source 9.4 : /pdo/vueTestSingletonPDO.php

```
1   <?php
2     require_once('classes/VueHtmlUtils.php');
3     echo \CoursPHP\Vue\VueHtmlUtils::enTeteHTML5("Classe Singleton de Connection",
4                                                  'UTF-8', 'myStyle.css');
5     echo "<h1>Test de la Classe de Connection (Donnez un Code Postal)</h1>";
6     // Affichage des résultats de la requête
7     foreach ($queryResults as $row){
8       echo "<p>";
9       echo $row['numeroRue'].", ".$row['rue'].", ";
10      if (!empty($row['complementAddr']))
11        echo $row['complementAddr'].", ";
12        echo $row['codePostal']." ";
13        echo $row['ville']." ";
14        echo $row['pays'];
15        echo "</p>";
16    }
17    \CoursPHP\Vue\VueHtmlUtils::finFichierHTML5();
18  ?>
```

9.2.3 Requêtes préparées avec des `:nomDeChamp`

Code Source 9.5 : /pdo/ex14-testSingletonPDO-V2.php

```
1   <?php
2     require_once(dirname(__FILE__).'/classes/Config.php');
3     require_once(dirname(__FILE__).'/classes/DataBaseManager.php');
4
5     try{
```

```php
    // Arguments de la requête (tableau associatif nomChamp => valeurChamp)
    $args = array('codePostal' => isset($_GET['codePostal'])
                                    ? $_GET['codePostal'] : ""
            );

    $tableName = \CoursPHP\Config\Config::getTablesPrefix().'Adresse';
    // Requête avec des :nomChamp (méthode prepareAndExecuteQueryAssoc)
    $queryResults = CoursPHP\Persistance\DataBaseManager
                        ::getInstance()->prepareAndExecuteQueryAssoc(
                                        'SELECT * FROM '.$tableName
                                        .' WHERE codePostal=:codePostal',
                                        $args // valeurs
                                       );
  }catch (Exception $e){
    $dataError[] =  $e->getMessage();
    require("vueErreur.php");
  }

  if ($queryResults === false){
     // Erreur lors de l'exécution de la requête
    $dataError[] = "Problème lors de la préparation de la requête. "
     ."(par exemple, la donnée codePostal passée par GET est invalide...)";
     require("vueErreur.php");
  }else{
    // Code de la vue :
    require("vueTestSingletonPDO.php");
  }
?>
```

9.3 Classes *Gateway* : Persistance des Objets Métiers

9.3.1 Notion de classe *Gateway* et Opérations *CRUD*

La *Gateway* pour les instances d'`Adresse` est une fabrique concrète qui permet de construire les instances d'objets métiers (ici de type `Adresse`) obtenues par des requêtes. Dans un tel contexte de modélisation des données, les instances sont aussi appelées *entités*.

Ici des requêtes préparées (basées sur *PDO*) exécutées sur la classe de connexion `DataBaseManager`, sur la base de données *SQL* (voir la partie 9.2). Les méthodes de la classe `AdresseGateway` construisent les requêtes *SQL* nécessaires pour implémenter les différentes fonctionnalités concernant les adresses, demande leur exécution par la classe de connexion, puis appellent la fabrique d'adresse (partie 5.2.3) pour retourner les résultats (ou les erreurs) à des fin, par exemple, d'affichage. On parle d'un rôle de *génération de code SQL*.

Remarques.

1. Les principales opérations implémentées dans une classe *Gateway* sont les opérations dite *CRUD*, c'est à dire :
   - la Création d'une entité (*C* comme *Create*) ;
   - la lecture d'entité (*R* comme *Read*) ;
   - la mise à jour d'une entité (*U* comme *Update*) ;

- et la suppression d'une entité (*D* comme *Delete*)

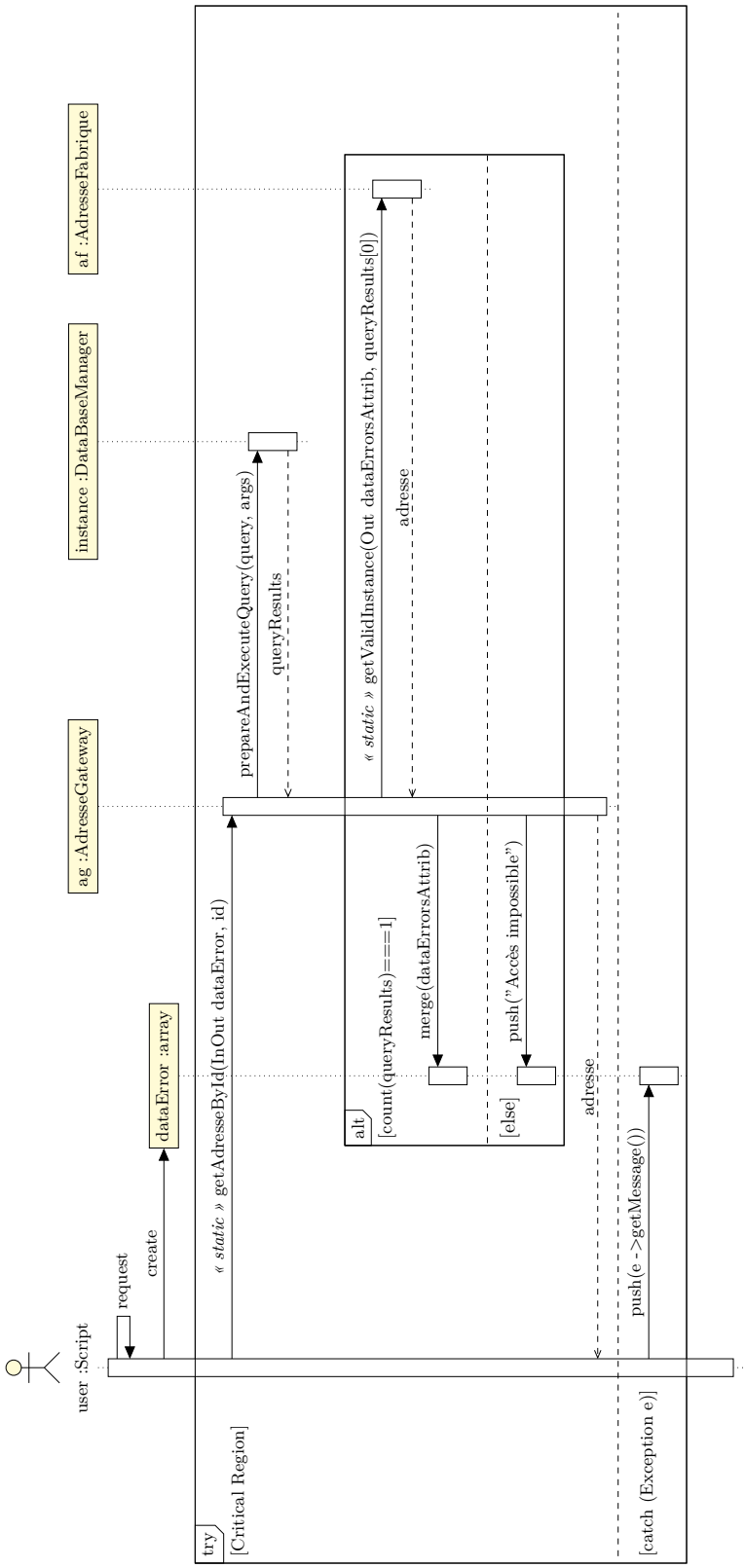
2. En cas d'erreur imprévisible qui ne relève pas de l'erreur de programmation (comme le cas d'un serveur de bases de données inaccessible), les méthodes de la couche d'accès aux données rejettent une exception personnalisée (pas d'exception *PDO*). Nous choisissons ici de rejeter un message d'exception correspondans à un code d'erreur (*status code*) standard du protocole *HTTP* (par exemple 404 pour une resource introuvable). Ceci facilitera l'implémentation de *Web Services* en s'appuyant sur cette couche *DAL*.

3. Dans l'implémentation ci-dessous, pour la création d'une nouvelle instance, nous proposons une recette de cuisine pour la génération de l'*id* unique de l'adresse. Signalons que cette technique ne passe pas à l'échelle, notamment pour une utilisation dans le cadre de *Web Services*. Dans ce cas, des techniques robustes, appelées *Universal Unique Identifier* (*UUID*) peuvent être utilisées, et des implémentations en *PHP* existent, par exemple dans la fonction `uniqid` du langage.

4. Dans le cas de classes métiers comportant (par exemple) une agrégation (voir partie 2.2.2), la *Gateway* de l'agrégat comportera généralement des méthodes permettant d'effectuer des résultats de jointures entre les tables correspondantes (voir la partie 9.3.2 pour une classe *Gateway* avec jointure et la partie 13.5 pour une exemple d'application comportant une agrégation/jointure).

Code Source 9.6 : /pdo/classes/AdresseGateway.php

```php
<?php
namespace CoursPHP\Persistance;
/** @brief Permet d'accéder/mettre à jour les données de la table Adresse
 * dans la base de données (au moins les opérations CRUD).
 * Les méthodes génèrent le code SQL pour des requêtes préparée, puis
 * font appel à la classe Connection (DataBaseManager) pour préparer et
 * exécuter les requêtes.
 * Les méthodes retournent, selon la requête considérée, des instances
 * ou collections d'instances d'Adresse, résultats d'une requête SELECT,
 * ou ligne impactée par la requête (INSERT, UPDATE, DELETE).
 * Les méthodes retournent les erreurs sur les données incorrectes
 * dans un tableau associatif dataError, et pevent rejeter des exceptions
 * en cas de problèmes d'accès à la BD (serveur inaccessible par exemple) */
class AdresseGateway {
  /** Permet d'obtenir le nom complet de la table contenant les adresses
   * @return le nom de la table avec le préfixe commun aux tables */
  public static function getTableNameAdresse(){
    return \CoursPHP\Config\Config::getTablesPrefix().'Adresse';
  }

  /** Permet de récupérer une adresse à partir de son ID.
   * @param $dataError : données d'erreurs (couple nomChamp => message)
   * @param $idAdresse : clé primaire de l'adresse à récupérer
   * @return instance d'Adresse en cas de succès, undefined sinon.
   * @throws en case de problème d'accès à la base de données */
  public static function getAdresseById(&$dataError, $idAdresse){

    $adresse = null; // Initialisation pour tester a posteriori

    if (isset($idAdresse)){
```



Diag 7. Diagramme de séquence de la méthode `AdresseGateway::getAdresseById()`

```
31        // Exécution de la requête via la classe de connexion (singleton)
32        // Les exceptions éventuelles, personnalisées, sont gérées plus haut
33        $args=array($idAdresse);
34        $queryResults = DataBaseManager::getInstance()->prepareAndExecuteQuery(
35                        'SELECT * FROM '.self::getTableNameAdresse()
36                        .' WHERE idAdresse=?', $args);
37        // Si l'exécution de la requête n'a pas fonctionné
38        if (!isset($queryResults) || !is_array($queryResults)) {
39          throw new \Exception("500"); // "Internal Server Error"
40        }
41        // si une et une seule adresse a été trouvée
42        if (count($queryResults) === 1){
43          $row = $queryResults[0];
44          $adresse = \CoursPHP\Metier\AdresseFabrique
45                          ::getValidInstance($dataError, $row);
46        }
47      }
48      // Test si adresse introuvable :
49      if ($adresse === null) {
50        throw new \Exception("404"); // "Not Found"
51      }
52      return $adresse;
53    }
54
55    /** Permet de récupérer une collection d'adresses présentes dans la table.
56     * @param $dataError : données d'erreurs (couple nomChamp => message)
57     * @return collection d'Adresses en cas de succès, collection vide sinon.
58     * @throws en case de problème d'accès à la base de données */
59    public static function getAdresseAll(&$dataError){
60      // Exécution de la requête via la classe de connexion (singleton)
61      // Les exceptions éventuelles, personnalisées, sont gérées plus haut
62      $args=array();
63      $queryResults = DataBaseManager::getInstance()->prepareAndExecuteQuery(
64                    'SELECT * FROM '.self::getTableNameAdresse(),
65                    $args);
66
67      // Construction de la collection des résultats (fabrique)
68      $collectionAdresse = array();
69      // Si l'exécution de la requête a fonctionné
70      if ($queryResults !== false){
71        // Parcours des lignes du résultat de la requête :
72        foreach ($queryResults as $row){
73          // Ajout d'une adresse dans la collection :
74          $adresse = \CoursPHP\Metier\AdresseFabrique
75                          ::getValidInstance($dataError, $row);
76          $collectionAdresse[] = $adresse;
77        }
78      }else{  // Échec de la requête SQL
79          throw new \Exception("500"); // "Internal Server Error"
80      }
81      return $collectionAdresse;
82    }
83
84    /** @brief Met à jour une adresse (Update)
85     * @param $dataError : données d'erreurs (couple nomChamp => message)
86     * @param $inputArray tableau associatif dont les clefs correspondent aux
```

```
     nom_contact_sportsClubs
87    *                   des attributs d'Adresse
88    * @return l'instance d'Adresse (erreurs ET instance de l'adresse modifiée)
89    * @throws en case de problème d'accès à la base de données */
90   public static function updateAdresse(&$dataError, &$inputArray){
91     // Tentative de construction d'une instance (et filtrage)
92     $adresse = \CoursPHP\Metier\AdresseFabrique
93                             ::getValidInstance($dataError, $inputArray);
94     // Si la forme des attributs sont incorrects (expressions régulières)
95     if (!empty($dataError)){
96       return $adresse;
97     }
98     // On teste si la donnée existe. Sinon Status 404 (Not Found)
99     $queryResultsExists = DataBaseManager::getInstance()
100                                 ->prepareAndExecuteQueryAssoc(
101                   'SELECT idAdresse FROM '.self::getTableNameAdresse().
102                   ' WHERE idAdresse=:idAdresse',
103                  $inputArray
104                );
105    if ($queryResultsExists === false){ // Échec de la requête SQL
106      throw new \Exception("500"); // "Internal Server Error"
107    }
108    if (count($queryResultsExists) < 1){ // Donnée introuvable
109      throw new \Exception("404"); // "Internal Server Error"
110    }
111    // Exécution de la requête de mis à jour :
112    $queryResults = DataBaseManager::getInstance()->prepareAndExecuteQueryAssoc(
113                        'UPDATE '.self::getTableNameAdresse()
114                        .' SET idPersonne=:idPersonne, '
115                        .'numeroRue=:numeroRue, rue=:rue, '
116                        .'complementAddr=:complementAddr, codePostal=:codePostal
                              , '
117                        .'ville=:ville, pays=:pays WHERE idAdresse=:idAdresse',
118                        $inputArray
119                      );
120    if ($queryResults === false){  // Échec de la requête SQL
121      throw new \Exception("500"); // "Internal Server Error"
122    }
123    return $adresse;
124  }
125
126  /** @brief Insère une nouvelle adresse (Create)
127    * @param $dataError : données d'erreurs (couple nomChamp => message)
128    * @param $inputArray tableau associatif dont les clefs correspondent aux
         nom_contact_sportsClubs
129    *                   des attributs d'Adresse
130    * @return l'instance d'Adresse (erreurs ET instance de l'adresse créée)
131    * @throws en case de problème d'accès à la base de données */
132  public static function createAdresse(&$dataError, &$inputArray){
133    // Tentative de construction d'une instance (et filtrage)
134    $adresse = \CoursPHP\Metier\AdresseFabrique
135                    ::getValidInstance($dataError, $inputArray);
136    // Si la forme des attributs sont incorrects (expressions régulières)
137    if (!empty($dataError)){
138      return $adresse;
139    }
```

```php
    // Exécution de la requête d'insertion :
    $queryResults = DataBaseManager::getInstance()->prepareAndExecuteQueryAssoc(
                        'REPLACE INTO '.self::getTableNameAdresse()
                        .'(idAdresse, idPersonne, numeroRue, rue, '
                        .'complementAddr, codePostal, ville, pays) '
                        .'VALUES (:idAdresse, :idPersonne, :numeroRue, :rue, '
                        .':complementAddr, :codePostal, :ville, :pays)',
                        $inputArray
                    );
    if ($queryResults === false){  // Échec de la requête SQL
      throw new \Exception("500"); // "Internal Server Error"
    }
    return $adresse;
  }

  /** @brief Supprime une adresse à partir de son ID.
   * Retourne le modèle de données (erreurs ET instance de l'Adresse supprimée)
   * @param $dataError : données d'erreurs (couple nomChamp => message)
   * @param $idAdresse : clé primaire de l'adresse à récupérer
   * @return instance d'Adresse en cas de succès, undefined sinon.
   * @throws en case de problème d'accès à la base de données */
  public static function deleteAdresse(&$dataError, $idAdresse){
    // Test si l'adresse existe et récupérations données à supprimer
    $dataErrorIdSearch = array();
    // On teste si l'adresse existe
    try {
      $adresse = self::getAdresseById($dataErrorIdSearch, $idAdresse);
    } catch (\Exception $e) {
      return null; // Pas d'erreur suivant les recommandations HTTP
    }
    // L'adresse existe
    $args=array($idAdresse);
    $queryResults = DataBaseManager::getInstance()->prepareAndExecuteQuery(
                        'DELETE FROM '.self::getTableNameAdresse()
                        .' WHERE idAdresse=?', $args
                    );
    if ($queryResults === false){ // Impossible d'exécuter la requête
      throw new \Exception("500"); // "Internal Server Error"
    }
    return $adresse;
  }
}
?>
```

Voici un script de test des fonctionnalités (et gestion des erreurs) de la classe *Gateway* :

Code Source 9.7 : /pdo/ex15-testAdresseGateway.php (cf. Fig 9.1)

```php
<?php
  require_once(dirname(__FILE__).'/classes/Config.php');
  require_once(dirname(__FILE__).'/classes/DataBaseManager.php');
  require_once(dirname(__FILE__).'/classes/ValidationUtils.php');
  require_once(dirname(__FILE__).'/classes/ExpressionsRegexUtils.php');

  require_once(dirname(__FILE__).'/classes/Adresse.php');
  require_once(dirname(__FILE__).'/classes/AdresseValidation.php');
  require_once(dirname(__FILE__).'/classes/AdresseFabrique.php');
```

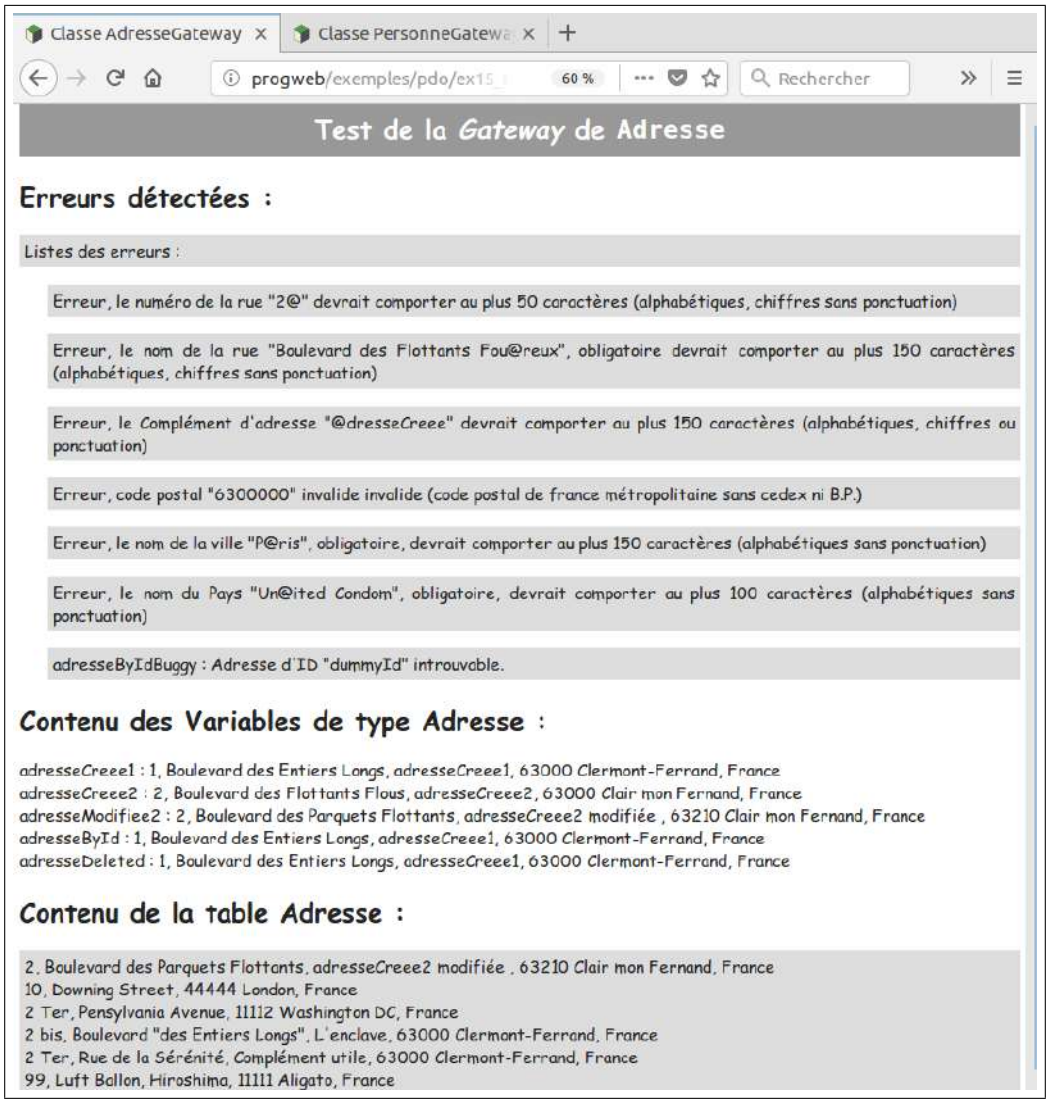


FIGURE 9.1 : Illustration du code source 9.7

```
10  require_once(dirname(__FILE__).'/classes/AdresseGateway.php');
11  require_once(dirname(__FILE__).'/classes/AdresseView.php');
12
13  require_once(dirname(__FILE__).'/classes/Personne.php');
14  require_once(dirname(__FILE__).'/classes/PersonneValidation.php');
15  require_once(dirname(__FILE__).'/classes/PersonneFabrique.php');
16  require_once(dirname(__FILE__).'/classes/PersonneGateway.php');
17
18  $dataError = array();
19
20  // Création d'une Adresse "adresseCreee1"
21  $idAdresse1 = "165dfe8623";
22  // ID de la Personne "personneCreee1" (doit être préalablement créée)
23  $idPersonne1 = "165dfe8790";
24  try{
25    // Création de la Personne
26    $argsPersonne = array("idPersonne" => $idPersonne1,
27                          "nom" => "personneCreee1 - Titi le tout p'tit");
```

```
28      $personneCreee1 = CoursPHP\Persistance\PersonneGateway::createPersonne(
29                                                          $dataError, $argsPersonne);
30      // Test 1 de création d'une adresse pour cette Personne
31      $argsAdresse = array("idAdresse" => $idAdresse1,
32                           "idPersonne" => $idPersonne1,
33                           "numeroRue" => "1",
34                           "rue" => "Boulevard des Entiers Longs",
35                           "complementAddr" =>"adresseCreee1",
36                           "codePostal" => "63000",
37                           "ville" => "Clermont-Ferrand",
38                           "pays" => "France");
39      $adresseCreee1 = CoursPHP\Persistance\AdresseGateway::createAdresse(
40                                                          $dataError, $argsAdresse);
41    }catch (Exception $e){
42      $dataError[] =  "adresseCreee1 : ".$e->getMessage();
43    }
44    // Création d'une Adresse 2
45    $idAdresse2 = "165dfe1254";
46    try{
47      // Test 2 de création d'une adresse
48      $argsAdresse = array("idAdresse" => $idAdresse2,
49                           "idPersonne" => $idPersonne1,
50                           "numeroRue" => "2",
51                           "rue" => "Boulevard des Flottants Flous",
52                           "complementAddr" =>"adresseCreee2",
53                           "codePostal" => "63000",
54                           "ville" => "Clair mon Fernand",
55                           "pays" => "France");
56      $adresseCreee2 = CoursPHP\Persistance\AdresseGateway::createAdresse(
57                                                          $dataError, $argsAdresse);
58    }catch (Exception $e){
59      $dataError[] =  "adresseCreee2 : ".$e->getMessage();
60    }
61    // Création d'une adresse avec un erreur sur l'attribut "nom"
62    try{
63      $argsAdresse = array("idAdresse" => "165dfe1342",
64                           "idPersonne" => $idPersonne1,
65                           "numeroRue" => "2@",
66                           "rue" => "Boulevard des Flottants Fou@reux",
67                           "complementAddr" =>"@dresseCreee",
68                           "codePostal" => "6300000",
69                           "ville" => "P@ris",
70                           "pays" => "Un@ited Condom");
71      $adresseCreeeBuggy = CoursPHP\Persistance\AdresseGateway
72                                    ::createAdresse($dataError, $argsAdresse);
73    }catch (Exception $e){
74      $dataError[] =   "adresseCreeeBuggy : ".$e->getMessage();
75    }
76    // Mise à jour d'une adresse
77    try{
78      $argsAdresse = array("idAdresse" => $idAdresse2,
79                           "idPersonne" => $idPersonne1,
80                           "numeroRue" => "2",
81                           "rue" => "Boulevard des Parquets Flottants",
82                           "complementAddr" =>"adresseCreee2 modifiée ",
83                           "codePostal" => "63210",
```

```
84                              "ville" => "Clair mon Fernand",
85                              "pays" => "France");
86      $adresseModifiee2 = CoursPHP\Persistance\AdresseGateway
87                                      ::updateAdresse($dataError, $argsAdresse);
88    }catch (Exception $e){
89      $dataError[] =  "adresseModifiee2 : ".$e->getMessage();
90    }
91    // Mise à jour d'une adresse avec un erreur sur les attributs
92    try{
93      $argsAdresse = array("idAdresse" => $idAdresse2,
94                              "idPersonne" => $idPersonne1,
95                              "numeroRue" => "2@",
96                              "rue" => "Boulevard des Flottants Fou@reux",
97                              "complementAddr" =>"@dresseCreee",
98                              "codePostal" => "6300000",
99                              "ville" => "P@ris",
100                             "pays" => "Un@ited Condom");
101     $adresseModifieeBuggy = CoursPHP\Persistance\AdresseGateway
102                                     ::createAdresse($dataError, $argsAdresse);
103   }catch (Exception $e){
104     $dataError[] =  "adresseModifieeBuggy : ".$e->getMessage();
105   }
106   // Recherche de Adresse par ID :
107   try{
108     $adresseById = CoursPHP\Persistance\AdresseGateway::getAdresseById(
109                             $dataError, $idAdresse1);
110   }catch (Exception $e){
111     $dataError[] =  "adresseById : ".$e->getMessage();
112   }
113     // Recherche d'adresse avec ID inexistant :
114   try{
115     $adresseByIdBuggy = CoursPHP\Persistance\AdresseGateway
116                                     ::getAdresseById($dataError, "dummyId");
117   }catch (Exception $e){
118     $dataError[] =  'adresseByIdBuggy : '.$e->getMessage();
119   }
120   // Suppression d'une Adresse (adresseCree1)
121   try{
122     $adresseDeleted = CoursPHP\Persistance\AdresseGateway
123                                       ::deleteAdresse($dataError,
124                                                       $idAdresse1);
125   }catch (Exception $e){
126     $dataError[] =  $e->getMessage();
127   }
128   // Récupération de toutes les adresses
129   try{
130     $adresseAll= CoursPHP\Persistance\AdresseGateway
131                                                 ::getAdresseAll($dataError);
132   }catch (Exception $e){
133     $dataError[] =  "".$e->getMessage();
134   }
135   // Code de la vue :
136   require('ex15-vueAdresseGateway.php');
137 ?>
```

Code Source 9.8 : /pdo/ex15-vueAdresseGateway.php

```php
<?php

  require_once('classes/VueHtmlUtils.php');
  echo CoursPHP\Vue\VueHtmlUtils::enTeteHTML5("Classe AdresseGateway",
                                              'UTF-8', 'myStyle.css');
  echo "<h1>Test de la <i>Gateway</i> de <code>Adresse</code></h1>";

  // Liste des erreurs :
  echo "<h2>Erreurs détectées :</h2>";
  if (empty($dataError)){
    echo "<p>Aucune erreur.</p>";
  }else{
    echo "<p>Listes des erreurs :</p><ol>";
    foreach ($dataError as $errorMsg){
      echo "<p>".$errorMsg."</p>";
    }
    echo "</ol>";
  }
  echo "<h2>Contenu des Variables de type Adresse :</h2>";

  echo "adresseCreee1 : ".CoursPHP\Vue\AdresseView
                                    ::getHtmlCompact($adresseCreee1)."<br/>";
  echo "adresseCreee2 : ".CoursPHP\Vue\AdresseView
                                    ::getHtmlCompact($adresseCreee2)."<br/>";
  echo "adresseModifiee2 : ".CoursPHP\Vue\AdresseView
                                    ::getHtmlCompact($adresseModifiee2)."<br/>";
  echo "adresseById : ".CoursPHP\Vue\AdresseView
                                    ::getHtmlCompact($adresseById)."<br/>";
  echo "adresseDeleted : ".CoursPHP\Vue\AdresseView
                                    ::getHtmlCompact($adresseDeleted)."<br/>";

  echo "<h2>Contenu de la table Adresse :</h2>";

  echo "<p>";
  foreach ($adresseAll as $adresse){
    echo CoursPHP\Vue\AdresseView::getHtmlCompact($adresse)."<br/>";
  }
  echo "</p>";

  CoursPHP\Vue\VueHtmlUtils::finFichierHTML5();
?>
```

9.3.2 Classe *Gateway* d'un Agrégat et Jointures Naturelles

Code Source 9.9 : /pdo/classes/PersonneGateway.php

```php
<?php
namespace CoursPHP\Persistance;
/** @brief Permet d'accéder/mettre à jour les données de la table Personne
 * dans la base de données (au moins les opérations CRUD).
 * Les méthodes génèrent le code SQL pour des requêtes préparée, puis
 * font appel à la classe Connection (DataBaseManager) pour préparer et
 * exécuter les requêtes.
 * Les méthodes retournent, selon la requête considérée, des instances
```

```php
 * ou collections d'instances de Personne, résultats d'une requête SELECT,
 * ou ligne impactée par la requête (INSERT, UPDATE, DELETE).
 * Les méthodes retournent les erreurs sur les données incorrectes
 * dans un tableau associatif dataError, et pevent rejeter des exceptions
 * en cas de problèmes d'accès à la BD (serveur inaccessible par exemple) */
class PersonneGateway {
  /** Permet d'obtenir le nom complet de la table contenant les personnes
   * @return le nom de la table avec le préfixe commun aux tables */
  public static function getTableNamePersonne(){
    return \CoursPHP\Config\Config::getTablesPrefix().'Personne';
  }

  /** Permet de récupérer une personne à partir de son ID.
   * @param $dataError : données d'erreurs (couple nomChamp => message)
   * @param $idPersonne : clé primaire de la personne à récupérer
   * @return instance de Personne (avec ses adresses) en cas de succès.
   * @throws en case de problème d'accès à la base de données */
  public static function getPersonneById(&$dataError, $idPersonne){

    $currentPersonne = null; // Initialisation pour tester a posteriori

    if (isset($idPersonne)) {
      $args=array($idPersonne);
      // Jointure pour récupérer les Adresse
      $queryResults = DataBaseManager::getInstance()->prepareAndExecuteQuery(
        'SELECT * FROM '.self::getTableNamePersonne()
        .' NATURAL LEFT JOIN '.AdresseGateway::getTableNameAdresse()
        .' WHERE '.self::getTableNamePersonne().'.idPersonne=?',
        $args);
      // Si l'exécution de la requête n'a pas fonctionné
      if (!isset($queryResults) || !is_array($queryResults)) {
        throw new \Exception("500");// "Internal Server Error"
      }
      // Pour chaque ligne
      foreach ($queryResults as $row) {
        // Si on est passé à la personne suivante
        if ($currentPersonne === null){
          $currentPersonne = \CoursPHP\Metier\PersonneFabrique
                    ::getValidInstance($dataError, $row);
        }
        if ($currentPersonne->idPersonne !== $row['idPersonne']){
          // L'identifiant de la personne doit être unique
          throw new \Exception("500"); // "Internal Server Error"
        }
        // Si la Personne possède au moins un Adresse
        if ($row['idAdresse'] !== null) {
          // Ajout d'une Adresse dans la collection :
          $adresse = \CoursPHP\Metier\AdresseFabrique
            ::getValidInstance($dataError, $row);
          $currentPersonne->addAdresse($adresse);
        }
      }
    }
    // Test si personne introuvable :
    if ($currentPersonne === null){
      throw new \Exception("404"); // "Not Found"
```

```
65      }
66      return $currentPersonne;
67    }
68
69    /** Permet de récupérer une collection d'personnes présentes dans la table.
70     * @param $dataError : données d'erreurs (couple nomChamp => message)
71     * @return collection de Personnes en cas de succès, collection vide sinon.
72     * @throws en case de problème d'accès à la base de données */
73    public static function getPersonneAll(&$dataError){
74
75      $collectionPersonne = array();
76      $currentPersonne = null;
77      // Jointure pour récupérer les Adresse
78      $queryResults = DataBaseManager::getInstance()->prepareAndExecuteQuery(
79        'SELECT * FROM '. self::getTableNamePersonne()
80        . ' NATURAL LEFT JOIN '. AdresseGateway::getTableNameAdresse()
81        . ' ORDER BY '.self::getTableNamePersonne(). '.nom, '
82        .self::getTableNamePersonne(). '.idPersonne');
83
84      // Si l'exécution de la requête n'a pas fonctionné
85      if (!isset ($queryResults) || !is_array($queryResults)) {
86          throw new \Exception("500"); // "Internal Server Error"
87      }
88      // Pour chaque ligne
89      foreach ($queryResults as $row) {
90        // Si on est passé à la personne suivante
91        if ($currentPersonne === null ||
92          $currentPersonne->idPersonne !== $row['idPersonne']) {
93
94          $currentPersonne = \CoursPHP\Metier\PersonneFabrique
95                  ::getValidInstance($dataError, $row);
96          $collectionPersonne[] = $currentPersonne;
97        }
98        if ($row['idAdresse'] !== null) {
99          // Ajout d'une Adresse dans la collection :
100         $adresse = \CoursPHP\Metier\AdresseFabrique
101                                   ::getValidInstance($dataError, $row);
102         $currentPersonne->addAdresse($adresse);
103       }
104     }
105     return $collectionPersonne;
106   }
107
108   /** @brief Met à jour une personne (Update)
109    * @param $dataError  données d'erreurs (couple nomChamp => message)
110    * @param $inputArray tableau associatif dont les clefs correspondent aux noms
111    *                    des attributs de Personne
112    * @return l'instance de Personne (erreurs ET l'instance modifiée)
113    * @throws en case de problème d'accès à la base de données */
114   public static function updatePersonne(&$dataError, &$inputArray){
115
116     // Tentative de construction d'une instance (et filtrage)
117     $personne = \CoursPHP\Metier\PersonneFabrique
118                   ::getValidInstance($dataError, $inputArray);
119     // Si la forme des attributs sont incorrects (expressions régulières)
120     if (!empty($dataError)){
```

```php
        return $personne;
      }
      // On teste si la donnée existe. Sinon Status 404 (Not Found)
      $queryResultsExists = DataBaseManager::getInstance()
                                  ->prepareAndExecuteQueryAssoc(
                                      'SELECT idPersonne FROM '.
                                       self::getTableNamePersonne().
                                       ' WHERE idPersonne=:idPersonne',
                                       $inputArray
                                    );
      if ($queryResultsExists === false){ // Échec de la requête SQL
        throw new \Exception("500"); // "Internal Server Error"
      }
      if (count($queryResultsExists) < 1){ // Donnée introuvable
        throw new \Exception("404"); // "Internal Server Error"
      }
      // Exécution de la requête de mis à jour :
      $queryResults = DataBaseManager::getInstance()->prepareAndExecuteQueryAssoc(
                          'UPDATE '.
                      self::getTableNamePersonne().' SET nom=:nom'.
                      ' WHERE idPersonne=:idPersonne',
                         $inputArray
                       );
      if ($queryResults === false){ // Échec de la requête SQL
        throw new \Exception("500"); // "Internal Server Error"
      }
      return $personne;
    }

    /** @brief Insère une nouvelle personne (Create)
     * @param $dataError  données d'erreurs (couple nomChamp => message)
     * @param $inputArray tableau associatif dont les clefs correspondent aux noms
     *                    des attributs de Personne
     * @return l'instance de Personne (erreurs ET instance de la personne créée)
     * @throws en case de problème d'accès à la base de données */
    public static function createPersonne(&$dataError, &$inputArray){
      // Tentative de construction d'une instance (et filtrage)
      $personne = \CoursPHP\Metier\PersonneFabrique
                      ::getValidInstance($dataError, $inputArray);
      // Si la forme des attributs sont incorrects (expressions régulières)
      if (!empty($dataError)){
        return $personne;
      }
      // Exécution de la requête d'insertion :
      $queryResults = DataBaseManager::getInstance()->prepareAndExecuteQueryAssoc(
                          'REPLACE INTO '.
                      self::getTableNamePersonne().'(idPersonne, nom) '
                           .'VALUES (:idPersonne, :nom)',
                           $inputArray
                         );
      if ($queryResults === false){ // Échec de la requête SQL
        throw new \Exception("500"); // "Internal Server Error"
      }
      return $personne;
    }

```

```php
  /** @brief Supprime une personne à partir de son ID, ainsi que ses Adresses.
   * @param $dataError : données d'erreurs (couple nomChamp => message)
   * @param $idPersonne : clé primaire de la personne à récupérer
   * @return le modèle de données (erreurs ET instance de Personne supprimée)
   * @throws en case de problème d'accès à la base de données */
  public static function deletePersonne(&$dataError, $idPersonne){
    // Test si la personne existe et récupérations données à supprimer
    $dataErrorIdSearch = array();
    try {
      $personne = self::getPersonneById($dataErrorIdSearch, $idPersonne);
    } catch (\Exception $e) {
      return null; // Pas d'erreur suivant les recommandations HTTP
    }
    // La Personne existe
    // Suppression en cascade des Adresses associées à la Personne
    $args=array($idPersonne);
    if(DataBaseManager::getInstance()->prepareAndExecuteQuery(
                        'DELETE FROM '.
                      AdresseGateway::getTableNameAdresse().' WHERE idPersonne=?'
                        ,
                        $args
                      ) === false){
      throw new \Exception("Problème d'exécution de la requête.");
    }
    // Suppression de la personne elle même
    $args=array($idPersonne);
    $queryResults = DataBaseManager::getInstance()->prepareAndExecuteQuery(
                        'DELETE FROM '. self::getTableNamePersonne()
                        .' WHERE idPersonne=?',
                        $args
                      );
    if ($queryResults === false){ // Échec de la requête SQL
      throw new \Exception("500"); // "Internal Server Error"
    }
    return $personne;
  }
}
?>
```

Code Source 9.10 : /pdo/ex16-testPersonneGateway.php (cf. Fig 9.2)

```php
<?php
  require_once(dirname(__FILE__).'/classes/Config.php');
  require_once(dirname(__FILE__).'/classes/DataBaseManager.php');
  require_once(dirname(__FILE__).'/classes/ValidationUtils.php');
  require_once(dirname(__FILE__).'/classes/ExpressionsRegexUtils.php');

  require_once(dirname(__FILE__).'/classes/Personne.php');
  require_once(dirname(__FILE__).'/classes/PersonneValidation.php');
  require_once(dirname(__FILE__).'/classes/PersonneFabrique.php');
  require_once(dirname(__FILE__).'/classes/PersonneGateway.php');
  require_once(dirname(__FILE__).'/classes/PersonneView.php');

  require_once(dirname(__FILE__).'/classes/Adresse.php');
  require_once(dirname(__FILE__).'/classes/AdresseValidation.php');
  require_once(dirname(__FILE__).'/classes/AdresseFabrique.php');
```

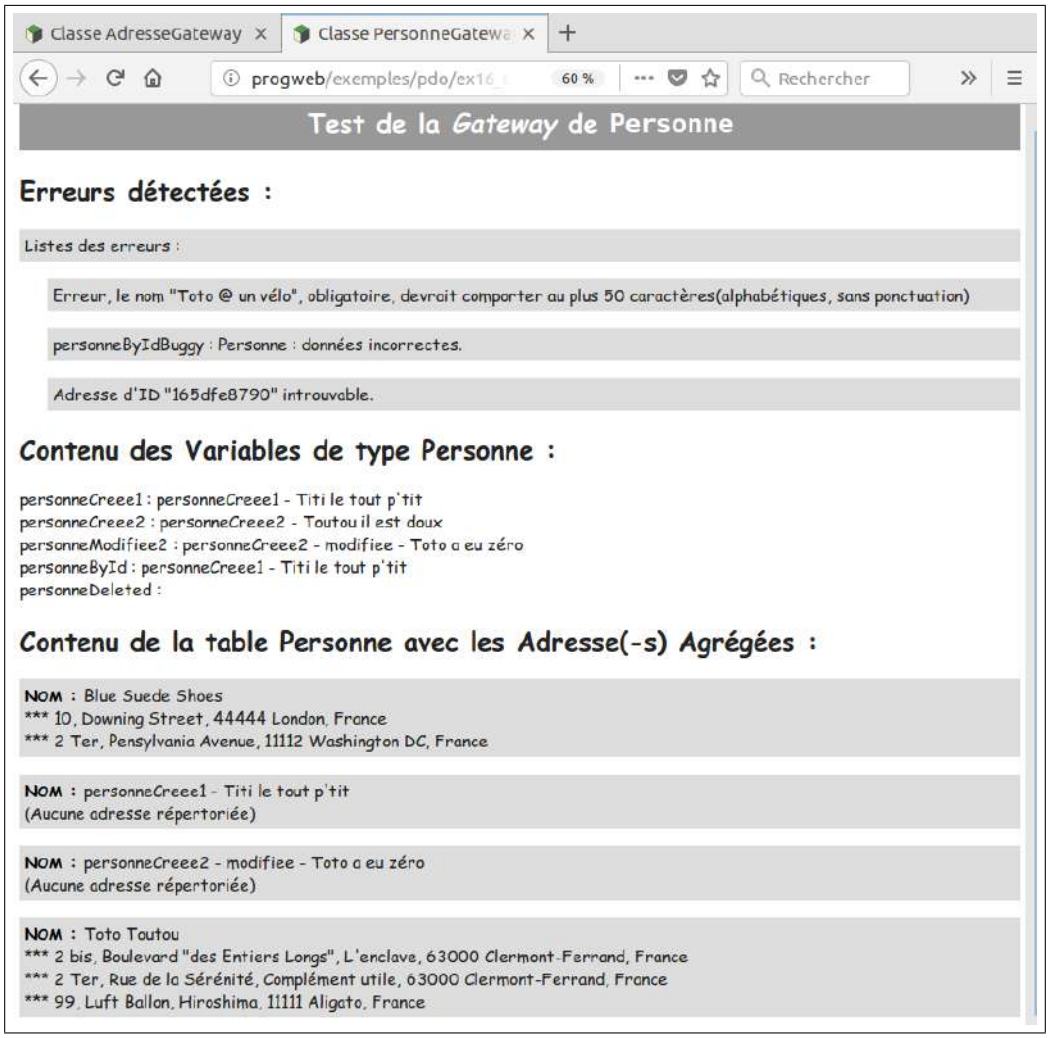


FIGURE 9.2 : Illustration du code source 9.10

```php
require_once(dirname(__FILE__).'/classes/AdresseGateway.php');
require_once(dirname(__FILE__).'/classes/AdresseView.php');

$dataError = array();

// Création d'une Personne "personneCreee1"
$idPersonne1 = "165dfe8790";
try{
  // Test 1 de création d'une personne
  $argsPersonne = array("idPersonne" => $idPersonne1,
                        "nom" => "personneCreee1 - Titi le tout p'tit");
  $personneCreee1 = CoursPHP\Persistance\PersonneGateway::createPersonne(
                                                    $dataError, $argsPersonne);
}catch (Exception $e){
  $dataError[] = "personneCreee1 : ".$e->getMessage();
}
// Création d'une Personne 2
$idPersonne2 = "165dfe8784";
try{
  // Test 2 de création d'une personne
```

```
36      $argsPersonne = array("idPersonne" => $idPersonne2,
37                            "nom" => "personneCreee2 - Toutou il est doux");
38      $personneCreee2 = CoursPHP\Persistance\PersonneGateway::createPersonne(
39                                                          $dataError, $argsPersonne);
40    }catch (Exception $e){
41      $dataError[] = "personneCreee2 : ".$e->getMessage();
42    }
43    // Création d'une personne avec un erreur sur l'attribut "nom"
44    try{
45      $argsPersonne = array("idPersonne" => "165dfe8782",
46                            "nom" => "T@rte @ l@ crême");
47      $personneCreeeBuggy = CoursPHP\Persistance\PersonneGateway
48                                    ::createPersonne($dataError, $argsPersonne);
49    }catch (Exception $e){
50      $dataError[] = "personneCreeeBuggy : ".$e->getMessage();
51    }
52    // Mise à jour d'une personne
53    try{
54      $argsPersonne = array("idPersonne" => $idPersonne2,
55                            "nom" => "personneCreee2 - modifiee - Toto a eu zéro")
                                  ;
56      $personneModifiee2 = CoursPHP\Persistance\PersonneGateway
57                                    ::updatePersonne($dataError, $argsPersonne);
58    }catch (Exception $e){
59      $dataError[] = "personneModifiee2 : ".$e->getMessage();
60    }
61    // Mise à jour d'une personne avec un erreur sur l'attribut "nom"
62    try{
63      $argsPersonne = array("idPersonne" => $idPersonne2,
64                            "nom" => "Toto @ un vélo");
65      $personneModifieeBuggy = CoursPHP\Persistance\PersonneGateway
66                                    ::createPersonne($dataError, $argsPersonne);
67    }catch (Exception $e){
68      $dataError[] = "personneModifieeBuggy : ".$e->getMessage();
69    }
70    // Recherche de Personne par ID :
71    try{
72      $personneById = CoursPHP\Persistance\PersonneGateway::getPersonneById(
73                          $dataError, $idPersonne1);
74    }catch (Exception $e){
75      $dataError[] = "personneById : ".$e->getMessage();
76    }
77      // Recherche d'adresse avec ID inexistant :
78    try{
79      $personneByIdBuggy = CoursPHP\Persistance\PersonneGateway
80                                    ::getPersonneById($dataError, "dummyId");
81    }catch (Exception $e){
82      $dataError[] = 'personneByIdBuggy : '.$e->getMessage();
83    }
84    // Suppression d'une Personne (personneCree1)
85    try{
86      $personneDeleted = CoursPHP\Persistance\PersonneGateway
87                                      ::deletePersonne($dataError,
88                                                       $idPersonne1);
89    }catch (Exception $e){
90      $dataError[] = $e->getMessage();
```

```
91    }
92    // Récupération de toutes les adresses
93    try{
94      $personneAll= CoursPHP\Persistance\PersonneGateway
95                                                    ::getPersonneAll($dataError);
96    }catch (Exception $e){
97      $dataError[] =  "".$e->getMessage();
98    }
99
100   // Code de la vue :
101   require('ex16-vuePersonneGateway.php');
102 ?>
```

Code Source 9.11 : /pdo/ex16-vuePersonneGateway.php

```
1   <?php
2     require_once('classes/VueHtmlUtils.php');
3     echo \CoursPHP\Vue\VueHtmlUtils::enTeteHTML5("Classe PersonneGateway",
4                                                   'UTF-8', 'myStyle.css');
5     echo "<h1>Test de la <i>Gateway</i> de <code>Personne</code></h1>";
6
7     // Liste des erreurs :
8     echo "<h2>Erreurs détectées :</h2>";
9     if (empty($dataError)){
10      echo "<p>Aucune erreur.</p>";
11    }else{
12      echo "<p>Listes des erreurs :</p><ol>";
13      foreach ($dataError as $errorMsg){
14        echo "<p>".$errorMsg."</p>";
15      }
16      echo "</ol>";
17    }
18    echo "<h2>Contenu des Variables de type Personne :</h2>";
19
20    echo "personneCreee1 : ".CoursPHP\Vue\PersonneView
21                                    ::getHtmlCompact($personneCreee1)."<br/>";
22    echo "personneCreee2 : ".CoursPHP\Vue\PersonneView
23                                    ::getHtmlCompact($personneCreee2)."<br/>";
24    echo "personneModifiee2 : ".CoursPHP\Vue\PersonneView
25                                    ::getHtmlCompact($personneModifiee2)."<br/>";
26    echo "personneById : ".CoursPHP\Vue\PersonneView
27                                    ::getHtmlCompact($personneById)."<br/>";
28    echo "personneDeleted : ".CoursPHP\Vue\PersonneView
29                                    ::getHtmlCompact($personneDeleted)."<br/>";
30
31    echo "<h2>Contenu de la table Personne avec les Adresse(-s) Agrégées :</h2>";
32
33
34    foreach ($personneAll as $personne){
35      echo "<p><strong>Nom : </strong>"
36            .CoursPHP\Vue\PersonneView::getHtmlCompact($personne)."<br/>";
37      if (empty($personne->getAdresses())){
38        echo "(Aucune adresse répertoriée)";
39      }
40      foreach ($personne->getAdresses() as $adresse){
41        echo "*** ".CoursPHP\Vue\AdresseView::getHtmlCompact($adresse)."<br/>";
```

```
42          }
43          echo "</p>";
44      }
45      \CoursPHP\Vue\VueHtmlUtils::finFichierHTML5();
46  ?>
```

Quatrième partie

Conception d'Architectures Avancées

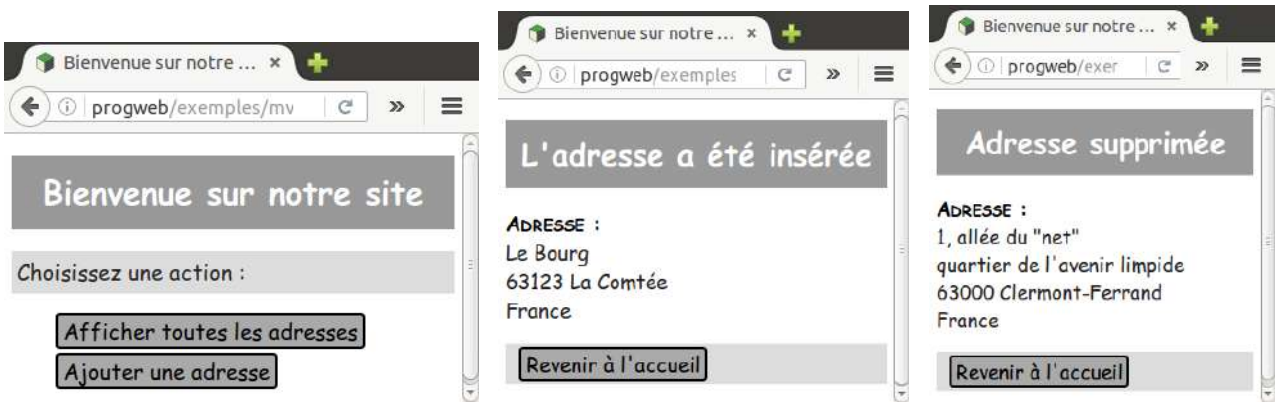
Table of Contents

Chapitre 10

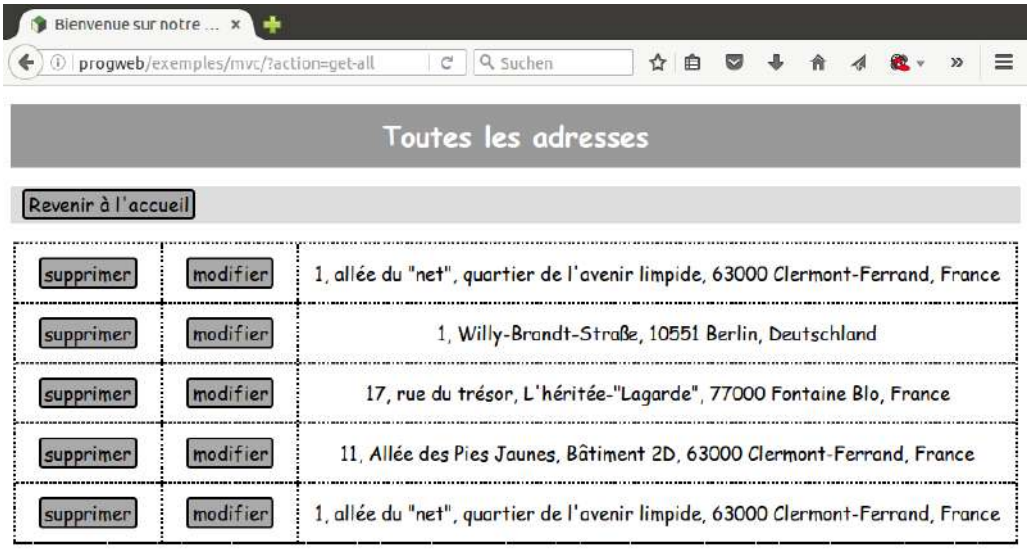
Analyse Fonctionnelle

10.1 *Storyboards*

Les *Storyboards* sont des croquis, élaborés avec un expert métier, qui représentent les différentes vues d'une application.

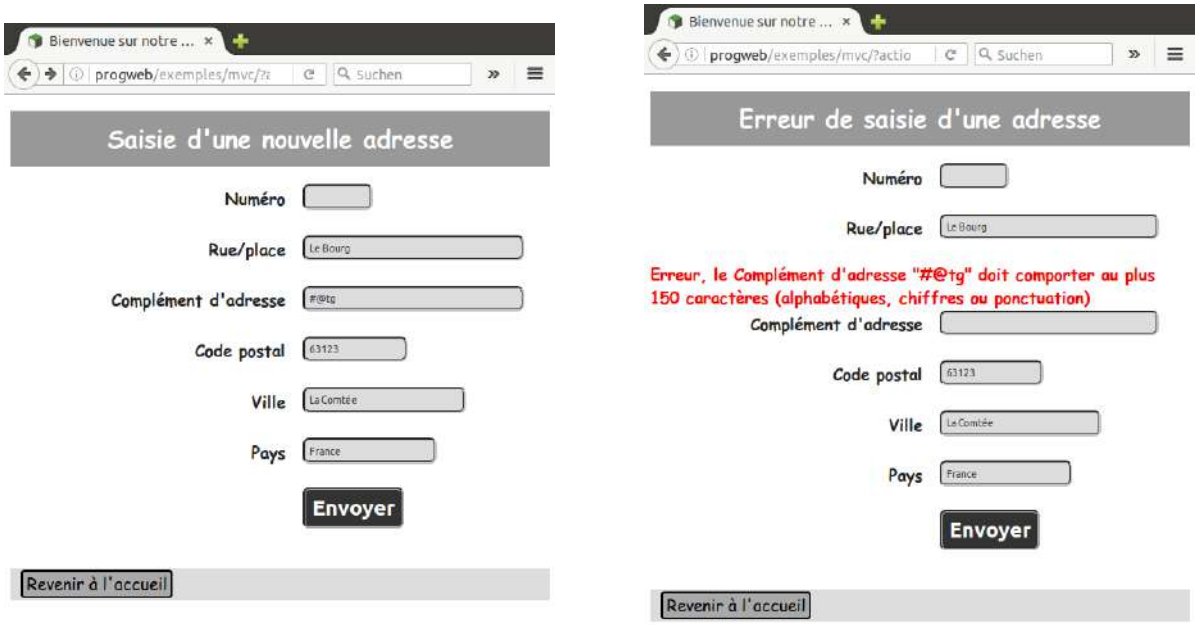


(a) Page d'accueil (b) Après création d'une instance (c) Suppression d'instance



(d) La vue affichant toutes les adresses

FIGURE 10.1 : *Storyboards* : Vues d'affichage d'instances et de collection d'instances d'`Adresse`



(a) La vue de saisie d'une instance

(b) La vue d'erreur de saisie

FIGURE 10.2 : *Storyboards* : Vue normale et vue d'erreur de saisie d'une instance d'`Adresse`

10.2 Diagrammes de Cas d'Utilisations

Dans les *storyboards* précédents, tous les liens et les boutons correspondent à des actions (événements utilisateur) que nous résumons dans un diagramme de cas d'utilisation.

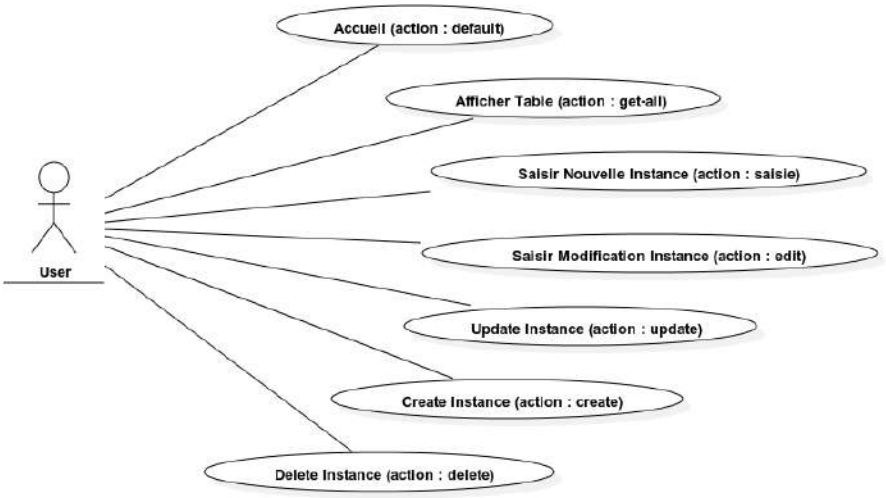


FIGURE 10.3 : *Use Case Diagram* : Les actions possibles pour l'utilisateur

Chapitre 11

Organisation des Répertoires et Configuration

11.1 Organisation des Répertoires

Nous allons adopter une organisation des répertoire très précise (voir arborescence ci-contre) pour permettre à la classe `Autoload`, décrite dans la partie 11.2, de charger automatiquement les classes.
La liste des sous-répertoire contenant des classes (ici `Config/`, `Controleur/`, `Metier/`, etc.) correspond aux sous-*namespaces* de ces classes, ce qui permet à l'*autoload* de trouver directement les fichiers sources des classes, à partir du nom complet (incluant les *namespaces*) de la classe.
Les classes du répertoire `Metier/` on été étudiées dans le chapitre 5. Les classes du répertoire `Vue/` ont été étudiées dans ce même chapitre 5.
La classe `DataBaseManager` de connexion à la base de données, du répertoire `Persistance/` a été étudiée dans la partie 9.2. La classe `AdresseGateway` qui gère, via la classe `DataBaseManager`, la génération et l'exécution des requêtes *SQL* concernant la table `Adresse`, a été étudiée dans la partie 9.3. Ces deux classes constituent dans notre application la couche d'accès aux données (*DAL*).
À l'exception de la classe `ValidationUtils`, qui a été abordée dans les parties 5.2.1 pour le nettoyage/échappement des données, les classes des répertoires `Controleur`, `Modele` et les vues du répertoire `Vue/vues/`, qui constitue le coeur de l'architecture trois tiers dite *Modèle, Vue, Contrôleur* (*MVC*), seront étudiées au chapitre 12.

```
|-- index.php
|-- Config
|   |-- Autoload.php
|   |-- Config.php
|-- Controleur
|   |-- Controleur.php
|   |-- ValidationUtils.php
|-- css
|   |-- defaultStyle.css
|-- Metier
|   |-- AdresseFabrique.php
|   |-- Adresse.php
|   |-- AdresseValidation.php
|   |-- ExpressionsRegexUtils.php
|-- Modele
|   |-- ModelAdresse.php
|   |-- ModelCollectionAdresse.php
|   |-- Model.php
|-- Persistance
|   |-- AdresseGateway.php
|   |-- DataBaseManager.php
|-- Vue
    |-- AdresseFormView.php
    |-- AdresseView.php
    |-- FormManager.php
    |-- VueHtmlUtils.php
    |-- vues
        |-- vueAccueil.php
        |-- vueAfficheAdresse.php
        |-- vueCollectionAdresse.php
        |-- vueErreurDefault.php
        |-- vueErreurSaisieCreate.php
        |-- vueErreurSaisieUpdate.php
        |-- vueSaisieAdresseCreate.php
        |-- vueSaisieAdresseUpdate.php
```

11.2 *Autoload*

La classe `Autoload` déclare et implémente une méthode *callback* qui sera appelée lors de l'utilisation dans le programme d'une classe (ou d'un *trait*) inconnu(e). La méthode *callback* en question cherche alors dans les répertoires un fichier source *PHP* dont le nom correspond à celui de la classe en question, et charge ce fichier source pour définir la classe « à la volée ».

Nous présentons une classe `Autoload` conforme à la norme *PSR-4* (pour *PHP Standard Recommendations*, voir la partie 2.1.2). Suivant cette norme, les *namespaces* du modules comportent tous un préfixe, appelé *Vendor Name* du module. Dans nos exemples, ce préfixe est le *namespace* : `\CoursPHP`. Le but de ce préfixe est de garantir que des *frameworks* différents ne produiront pas de collisions dans les noms de *namespaces*, de classe, etc. qui poseraient des problèmes d'interopérabilité.

En outre, le chemin relatif complet (définissant aussi le sous-répertoire du répertoire racine de notre module) vers le fichier source de la classe doit être nommé suivant le nom complet de la classe, en tenant compte de ses sous-*namespaces* (voir la partie 11.1). Ces conventions sur les *namespaces* et les chemins vers classes permettent de déterminer automatiquement l'emplacement du fichier source de la classe à partir de son nom complet.
La transformation du nom complet de la classe en chemin vers le fichier source est illustrée ci-dessous :

$$\text{Nom Complet de la classe : } \longrightarrow \underbrace{\texttt{\textbackslash CoursPHP}}_{\text{prefix}}\ \underbrace{\texttt{\textbackslash Persistance}}_{\text{sub-namespace}}\ \underbrace{\texttt{\textbackslash DataBaseManager}}_{\text{class name}}$$

$$\text{Chemin vers le fichier source : } \underbrace{\texttt{/path/to/mvc/root}}_{\textit{MVC}\text{ root dir}}\,/\,\underbrace{\texttt{Persistance}}_{\text{sub-dir}}\,/\,\underbrace{\texttt{DataBaseManager}\textit{php}}_{\text{class source file}}$$

Code Source 11.1 : /mvc//Config/Autoload.php

```php
<?php
namespace CoursPHP\Config;
/** @brief classe Autoload : permet de charger automatiquement les classes.
 * La méthode autoloadCallback() permer de charger le code source
 * d'une classe dont le nom est passé en paramètre.
 * Pour cela, la méthode load() déclare autoloadCallback()
 * par un appel à spl_autoload_register()                    */
class Autoload{

  /** Préfixe principal des namespaces du projet */
  public static $vendorNamespace;

  /** @brief Enregistrement du callback d'autoload.
   * @param $vendorNamespace Préfixe principal des namespaces du projet */
  public static function load_PSR_4($vendorNamespace){
    self::$vendorNamespace = $vendorNamespace;
    // Enregistrement d'un callback chargé d'inclure les classes.
    // Il peut y en avoir plusieurs, appelés successivement
    // en cas d'échec des premier...
    spl_autoload_register('CoursPHP\Config\Autoload::autoloadCallback_PSR_4');
  }

  /** @brief Callback d'Autoload suivant la norme PSR-4,
   * Cette méthode est appelée automatiquement en cas d'instanciation
```

```
25      * d'une classe inconnue. La méthode charge alors la classe en question.
26      *
27      * @param $class : nom complet de la classe à charger.
28      *
29      * @note L'arborescence des répertoires et les noms de fichiers PHP
30      * contenant les classes doivent coincider avec les sous-namespace
31      * de la classe pour trouver directement le répertoire contenant
32      * le fichier source de la classe.                                  */
33     public static function autoloadCallback_PSR_4($class) {
34       // La classe a-t-elle le bon préfixe de namespace ?
35       $longueurVendorNamespace = strlen(self::$vendorNamespace);
36       if (strncmp(self::$vendorNamespace, $class, $longueurVendorNamespace) !== 0)
             {
37           // Echec de l'autoloader. Espérons qu'il y en a un deuxième...
38           return;
39       }
40       // On enlève le préfixe ''Vendor Namespace".
41       $relativeClass = substr($class, $longueurVendorNamespace);
42       // Chemin vers le fichier source de la classe :
43       global $rootDirectory; // Voir début de index.php
44       $filePath = $rootDirectory.str_replace('\\', '/', $relativeClass).'.php';
45       // si le fichier existe
46       if (file_exists($filePath)) {
47         // Chargement de la classe :
48         require($filePath);
49       }
50     }
51 } // fin de la classe Autoload
52 ?>
```

Voici un exemple de test de cette fonctionnalité d'auto-chargement de classes (en plaçant le fichier script de test à la racine du *MVC*) :

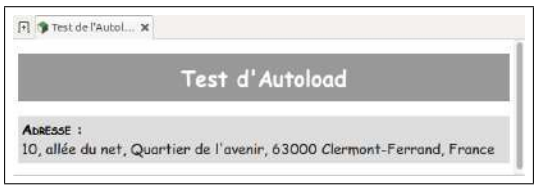


Figure 11.1 : Illustration du code source 11.2

Code Source 11.2 : /mvc//testAutoload.php (cf. Fig 11.1)

```
1  <?php
2    // Répertoire racine du MVC
3    $rootDirectory = dirname(__FILE__)."/";
4
5    // chargement de l'autoload pour autochargement des classes
6    require_once($rootDirectory.'/Config/Autoload.php');
7    \CoursPHP\Config\Autoload::load_PSR_4('CoursPHP\\');
8
9    // Création d'une instance d'adresse :
10   $adresse1 = new CoursPHP\Metier\Adresse("0af46d3bd9", '10', 'allée du net',
11               'Quartier de l\'avenir', '63000', 'Clermont-Ferrand', 'France');
12
```

```
13    echo \CoursPHP\Vue\VueHtmlUtils::enTeteHTML5("Test de l'Autoload",
14              'UTF-8', 'http://progweb/exemples/mvc/css/defaultStyle.css');
15    echo "<h1>Test d'Autoload</h1>";
16    echo "<p>";
17    echo "<strong>Adresse :</strong><br/>".
18              \CoursPHP\Vue\AdresseView::getHtmlCompact($adresse1)."<br/>";
19    echo "</p>";
20    echo \CoursPHP\Vue\VueHtmlUtils::finFichierHTML5();
21  ?>
```

11.3 La classe `Config` : éviter les *URL* en dûr

La classe `Config` permet de compléter les informations sur l'arborescence des répertoires, en indiquant les chemins vers les vues, les vues d'erreurs, ou des ressources comme des feuilles de style *CSS*, à partir de la racine du site. Ceci permet ensuite, dans le code, d'éviter les *URL* en dur en obtenant le chemin dans un tableau.

Ainsi, si on change l'emplacement des fichiers de ressources ou de vues par la suite, il n'y a alors pas besoin de chercher toutes les occurrences de liens cassés dans tous les fichiers sources, il suffit de changer la classe `Config`. Cela facilite fortement la maintenance du site.

Pour gérer les liens vers des vues, qui sont des fichiers sources *PHP*, on se base sur la donnée `$rootDirectory`, obtenue par `dirname(__FILE__)."/"` dans le fichier `indexphp`, à la racine du répertoire contenant notre module basé sur l'architecture *MVC* (voir le chapitre 12).

La classe `Config` permet aussi de gérer les *URL* de ressources (styles *CSS*, images, bannières, logos, etc.). La différence est que les *URL* sont ici gérées en absolu (commençant par `http[s]://`), et ne peuvent donc pas se baser sur un nom de répertoire (comme `$rootDirectory`). Si nous souhaitons placer nos ressources dans notre répertoire contenant notre module basé sur l'architecture *MVC* (par exemple dans un répertoire `$rootDirectory/css`), nous pouvons obtenir l'*URL* absolue de la racine (correspondant à notre répertoire `$rootDirectory`) comme suit (toujours au niveau du fichier `indexphp`).
Voici un fichier qui illustre le calcul des différents parties de l'*URL* complète du *script* :

FIGURE 11.2 : Illustration du code source 11.3

Code Source 11.3 : /mvc//testURI.php (cf. Fig 11.2)

```php
<?php
   // Répertoire racine du MVC
   $rootDirectory = dirname(__FILE__)."/";

   // Calcul portable de l'URI de la racine du MVC (sans la query string)
   // 1) on coupe l'URL du script au niveau de l'extension ".php"
   $scriptWithoutExtention = explode(".php", $_SERVER['PHP_SELF'])[0];
   // 2) puis on s'arrête au dernier slash (pour enlever la basename du script)
   $longueurRootURI = strrpos($scriptWithoutExtention, '/');
   // 3) On prend le début de l'URL en coupant à la bonne longueur
   $rootURI = substr($_SERVER['REQUEST_URI'], 0, $longueurRootURI);

   // chargement de l'autoload pour autochargement des classes
   require_once($rootDirectory.'/Config/Autoload.php');
   \CoursPHP\Config\Autoload::load_PSR_4('CoursPHP\\');

   echo \CoursPHP\Vue\VueHtmlUtils::enTeteHTML5("Calcul de l'URI",
                        'UTF-8', 'http://'.$_SERVER['SERVER_NAME']
                                          .$rootURI.'/css/defaultStyle.css');

   echo "<h1>Calcul de l'<i>URI</i> de la racine du <i>MVC</i></h1>";
   echo "<ul>";
   echo "<li><strong>\$_SERVER['REQUEST_URI'] :</strong> <code>"
                                    .$_SERVER['REQUEST_URI']."</code></li>";
   echo "<li><strong>\$_SERVER['PHP_SELF'] :</strong> <code>"
                                    .$_SERVER['PHP_SELF']."</code></li>";
   echo "<li><strong>Script sans extension :</strong> <code>"
                                    .$scriptWithoutExtention."</code></li>";
   echo "<li><strong><i>URI</i> de la racine du <i>MVC</i> :</strong> <code>"
                                    .$rootURI."</code></li>";
   echo "<li><strong>\$_SERVER['SERVER_NAME'] :</strong> <code>"
                                    .$_SERVER['SERVER_NAME']."</code></li>";
   echo "<li><strong><i>URL</i> de la racine du <i>MVC</i> :</strong> <code>"
                                    ."http://".$_SERVER['SERVER_NAME']
                                    .$rootURI."/</code></li>";
   echo "<li><strong>Nom du fichier source <i>PHP</i> :</strong> <code>"
                                    .basename($_SERVER['PHP_SELF'])."</code></li>";
   echo "<li><strong><i>URL</i> absolue complète de la requête :</strong> <code>"
                                    ."http://".$_SERVER['SERVER_NAME']
                                    .$rootURI."/".basename($_SERVER['PHP_SELF'])."?"
                                    .$_SERVER['QUERY_STRING']."</code></li>";
   echo "</ul>";
   echo \CoursPHP\Vue\VueHtmlUtils::finFichierHTML5();
?>
```

Enfin, la classe `Config` contiendra aussi les données d'identification du serveur de bases de données, qui seront utilisées en remplaçant de la méthode `DataBaseManager::getAuthData`, utilisée dans le constructeur de la classe `DataBaseManager` (voir le chapitre 9.2) par une méthode similaire `Config::getAuthData` de la classe `Config`.

Code Source 11.4 : /mvc//Config/Config.php

```php
<?php
namespace CoursPHP\Config;
/** @brief Classe de configuration
```

```
4   * Donne accès aux paramères spécifiques concernant l'application
5   * telles que les chemins vers les vues, les vues d'erreur,
6   * les hash pour les ID de sessions, etc. */
7  class Config
8  {
9    /** @brief Données nécessaires à la connexion à la base de données.
10    * Les valeurs pourraient être initialisées à partir d'un
11    * fichier de configuration séparé (require('configuration.php'))
12    * pour faciliter la maintenance par le webmaster. */
13   public static function getAuthData(&$db_host, &$db_name,
14                                      &$db_user, &$db_password){
15     $db_host="mysql:host=localhost;";
16     $db_name="dbname=ExempleBD";
17     $db_user="remy";
18     $db_password="my_password";
19   }
20
21   /** @return Le préfixe commun aux tables de la BD de l'application */
22   public static function getTablesPrefix(){
23     return "web_";
24   }
25
26   /** @brief retourne le tableau des (chemins vers les) vues */
27   public static function getVues(){
28     // Racine du site
29     global $rootDirectory;
30     // Répertoire contenant les vues
31     $vueDirectory = $rootDirectory."Vue/vues/";
32     return array(
33         "default" => $vueDirectory."vueAccueil.php",
34         "saisieAdresseCreate" => $vueDirectory."vueSaisieAdresseCreate.php",
35         "saisieAdresseUpdate" => $vueDirectory."vueSaisieAdresseUpdate.php",
36         "afficheAdresse" => $vueDirectory."vueAfficheAdresse.php",
37         "afficheCollectionAdresse" => $vueDirectory."vueCollectionAdresse.php"
38       );
39   }
40
41   /** @brief retourne le tableau des (chemins vers les) vues d'erreur */
42   public static function getVuesErreur(){
43     // Racine du site
44     global $rootDirectory;
45     // Répertoire contenant les vues d'erreur
46     $vueDirectory = $rootDirectory."Vue/vues/";
47     return array(
48         "default" => $vueDirectory."vueErreurDefault.php",
49         "saisieAdresseCreate" => $vueDirectory."vueErreurSaisieCreate.php",
50         "saisieAdresseUpdate" => $vueDirectory."vueErreurSaisieUpdate.php"
51       );
52   }
53
54   /** @brief retourne l'URI (sans le nom d'hôte du site et sans la query string)
55    * du répertoire la racine de notre architecture MVC.
56    * Exemple : pour l'URL http://example.org/path/to/my/mvc/?action=goToSleep,
57    * l'URI est : /path/to/my/mvc/                                    */
58   public static function getRootURI(){
59     global $rootURI; // variable globale initialisée dans l'index.php
```

```
60      return $rootURI;
61    }
62
63    /** @brief retourne le tableau des (URLS vers les) feuilles  de style CSS */
64    public static function getStyleSheetsURL(){
65      // Répertoire contenant les styles css
66      // Le nettoyage par filter_var évite tout risque d'injection XSS
67      $cssDirectoryURL = filter_var(
68          "http://".$_SERVER['SERVER_NAME'].self::getRootURI()."/css/",
69          FILTER_SANITIZE_URL);
70      return array("default" => $cssDirectoryURL."defaultStyle.css");
71    }
72
73    /** @brief Génère 10 chiffres hexa aléatoires (soit 5 octets) : */
74    public static function generateRandomId()
75    {
76      // Génération de 5 octets (pseudo-)aléatoires codés en hexa
77      $cryptoStrong = false; // Variable pour passage par référence
78      $octets = openssl_random_pseudo_bytes(5, $cryptoStrong);
79      return bin2hex($octets);
80    }
81  }
82  ?>
```

Chapitre 12

Architecture Modèle-Vue-Contrôleur

12.1 Principe Général du *MVC* et Modélisation

Le Diagramme 8 montre le diagramme de classes de notre implémentation de l'architecture trois-tiers Modèle-Vue-Controlleur (*MVC*).

12.2 Le Contrôleur

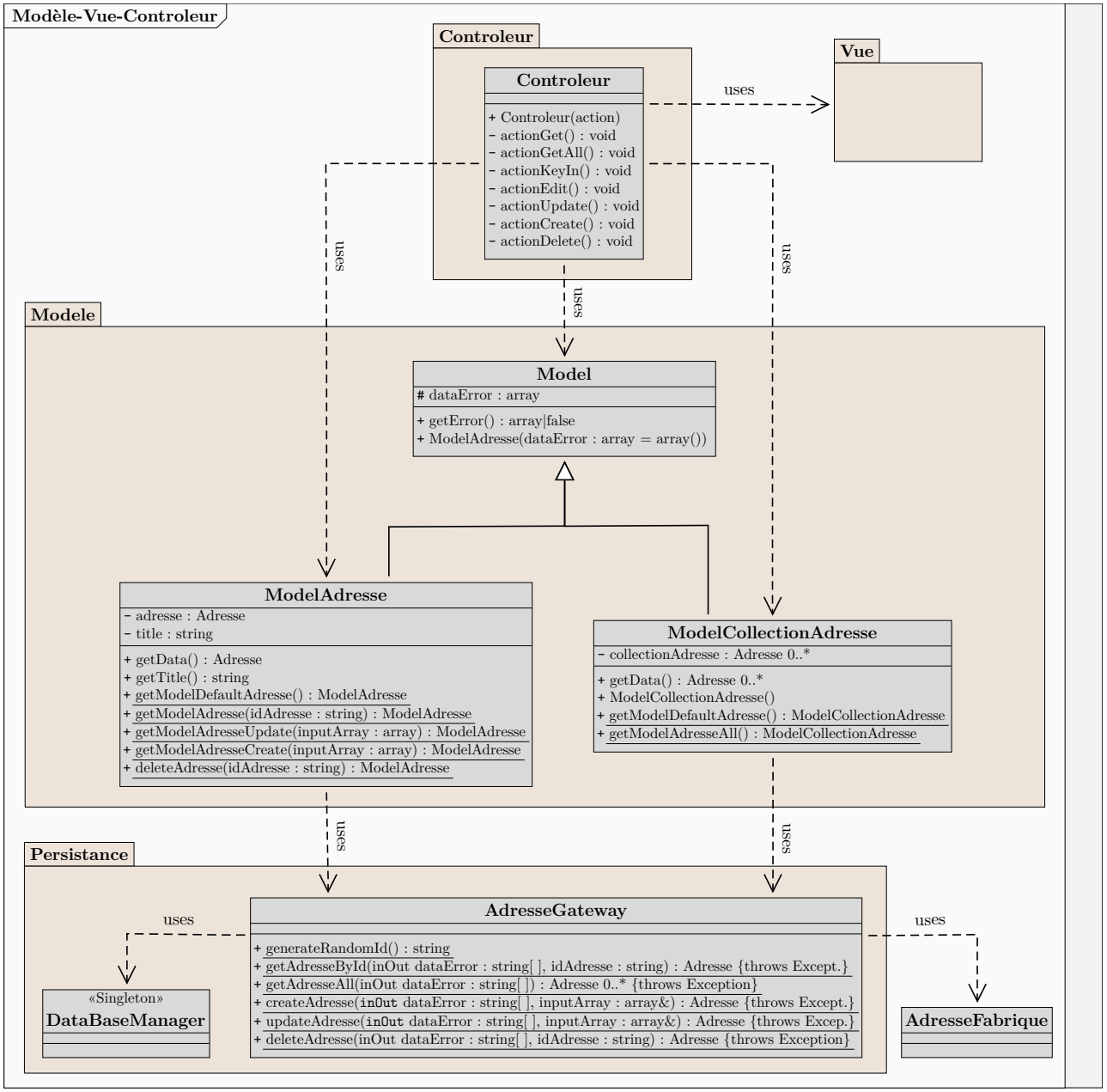
Le contrôleur est chargé de :

1. Reconnaître l'action (événement) à réaliser pour l'exécuter.
2. Demander la construction du modèle (données à renvoyer ou afficher en sortie).
3. Tester les erreurs et récupérer les éventuelles exceptions pour éviter un caca.
4. Appeler la vue (ou la vue d'erreur) pour afficher le résultat de l'action.

L'index (script `indexphp`) initialise la donnée du répertoire racine `rootDirectory`, charge le code source de l'`Autoload`, puis exécute la méthode `Autoload::load_PSR_4` qui déclare le *callback* chargé d'inclure le code source des classes utilisées dans le programme. L'index crée ensuite une instance du contrôleur. C'est le constructeur du contrôleur qui fait le reste du travail.

Code Source 12.1 : /mvc/index.php

```
1  <?php
2    // Répertoire racine du MVC
3    $rootDirectory = dirname(__FILE__)."/";
4
5    // Calcul portable de l'URI de la racine du MVC (sans la query string)
6    // 1) On enlève la "query string" : ?action=blabla&id=03456789
7    $urlWithoutQueryString = explode("?", $_SERVER['REQUEST_URI'])[0];
8    // 2) on coupe l'URL du script au niveau de l'extension ".php"
9    $scriptWithoutExtention = explode(".php", $urlWithoutQueryString)[0];
10   // 3) puis on s'arrête au dernier slash (pour enlever la basename du script)
11   $longueurRootURI = strrpos($scriptWithoutExtention, '/');
12   // 4) On prend le début de l'URL en coupant à la bonne longueur
13   $rootURI = substr($_SERVER['REQUEST_URI'], 0, $longueurRootURI);
14
```



Diag 8. Diagramme de classes du *Pattern* Modèle-Vue-Controlleur

```
15    // chargement de l'autoload pour auto-chargement des classes
16    require_once($rootDirectory.'/Config/Autoload.php');
17    CoursPHP\Config\Autoload::load_PSR_4('CoursPHP\\');
18
19    // Création de l'instance du contrôleur (voir Controleur.php)
20    $cont = new CoursPHP\Controleur\Controleur();
21  ?>
```

Le constructeur du contrôleur récupère dans le tableau `$_REQUEST` (réunion de `$_GET`, de `$_POST` et de `$_COOKIE`), l'action à réaliser. Ces actions sont déterminées lors de l'analyse dans le diagramme de cas d'utilisation (voir la partie 10.2). On fait un `switch` pour distinguer tous les cas correspondant aux actions, sans oublier le cas `default`, qui renverra généralement vers la page d'accueil, ou encore une vue d'erreur par défaut, en cas d'action non définie.

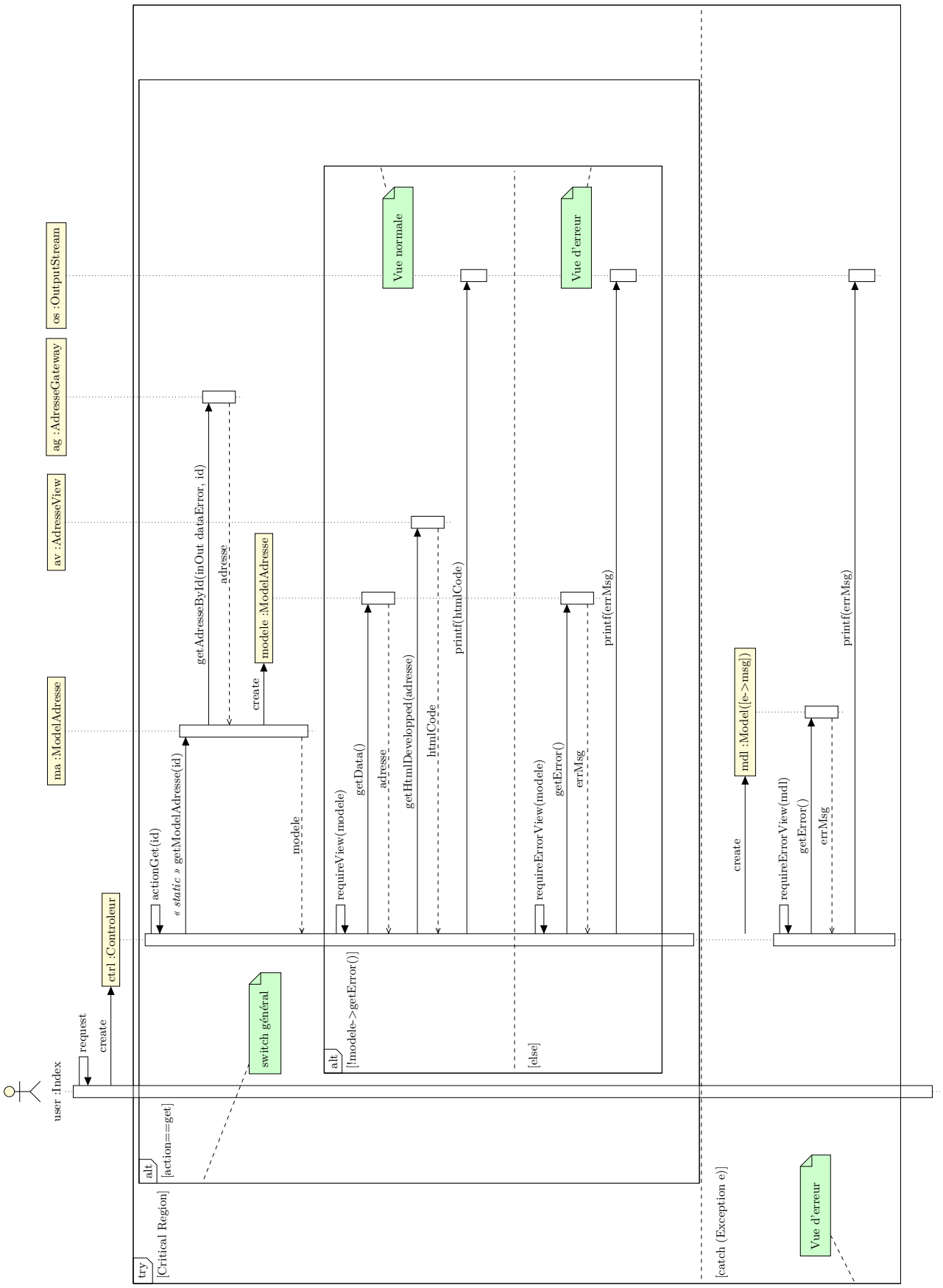
Code Source 12.2 : /mvc/Controleur/Controleur.php

```
1  <?php
2  namespace CoursPHP\Controleur;
3  /**
4   * @brief La classe Controleur identifie l'action et appelle la méthode
5   * pour construire le modèle correspondant à l'action.
6   * Le controleur appelle aussi la vue correspondante.
7   * Il gère aussi les exceptions et appelle le cas échéant une vue d'erreur.
8   */
9  class Controleur {
10   /**
11    * @brief C'est dans le contructeur que le contrôleur fait son travail.
12    */
13   function __construct() {
14     try{
15       // Récupération de l'action
16       $action = isset($_REQUEST['action']) ? $_REQUEST['action'] : "";
17
18       // On distingue des cas d'utilisation, suivant l'action
19       switch($action){
20         case "get": // Affichage d'une Adresse à partir de son ID
21           $this->actionGet();
22           break;
23         case "get-all": // Affichage de toutes les Adresse's
24           $this->actionGetAll();
25           break;
26         case "saisie": // Saisie d'une nouvelle Adresse
27           $this->actionKeyIn();
28           break;
29         case "edit": // Saisie des modifications d'une Adresse
30           $this->actionEdit();
31           break;
32         case "update": // Met à jour une Adresse dans la BD
33           $this->actionUpdate();
34           break;
35         case "create": // Cration d'une nouvelle Adresse dans la BD
36           $this->actionCreate();
37           break;
38         case "delete": // Supression d'une Adresse à partir de son ID
39         $this->actionDelete();
40           break;
41         default: // L'action indéfinie (page par défaut, ici accueil)
```

```
42            require (\CoursPHP\Config\Config::getVues()["default"]);
43          break;
44        }
45      }catch (\Exception $e){ // Page d'erreur par défaut
46        // Modèle contenant uniquement un tableau de messages d'erreur :
47        $modele = new \CoursPHP\Modele\Model(
48                          array('exception' => $e->getMessage())
49                        );
50        require (\CoursPHP\Config\Config::getVuesErreur()["default"]);
51      }
52    }
53
54    /** @brief Implemente l'action "get" :récupère une instance à partir de ID
55      */
56    private function actionGet(){
57      // ID de l'instance à récupérer
58      $rawId = isset($_REQUEST['idAdresse']) ? $_REQUEST['idAdresse'] : "";
59      $idAdresse = filter_var($rawId, FILTER_SANITIZE_STRING);
60      // Construction du modèle et implémentation de la persistance :
61      $modele = \CoursPHP\Modele\ModelAdresse::getModelAdresse($idAdresse);
62      // test d'erreur :
63      if ($modele->getError() === false){ // Appel de la vue
64        require (\CoursPHP\Config\Config::getVues()["afficheAdresse"]);
65      }else{ // Appel de la vue d'erreur
66        require (\CoursPHP\Config\Config::getVuesErreur()["default"]);
67      }
68    }
69
70    /** @brief Implemente l'action "get-all" : récupère toutes les instances
71      */
72    private function actionGetAll(){
73      // Construction du modèle et implémentation de la persistance :
74      $modele = \CoursPHP\Modele\ModelCollectionAdresse::getModelAdresseAll();
75      // test d'erreur :
76      if ($modele->getError() === false){ // Appel de la vue
77        require (\CoursPHP\Config\Config::getVues()["afficheCollectionAdresse"]);
78      }else{ // Appel de la vue d'erreur
79        require (\CoursPHP\Config\Config::getVuesErreur()["default"]);
80      }
81    }
82
83    /** @brief Implemente l'action "saisie" : affiche un formulaire vide de saisie
84      */
85    private function actionKeyIn(){
86        // Construction du modèle et implémentation de la persistance :
87        $modele = \CoursPHP\Modele\ModelAdresse::getModelDefaultAdresse();
88        // Appel de la vue
89        require (\CoursPHP\Config\Config::getVues()["saisieAdresseCreate"]);
90    }
91
92    /** @brief Implemente l'action "edit" : affiche un formulaire de modification
93      */
94    private function actionEdit(){
95      // ID de l'instance à modifier
96      $rawId = isset($_REQUEST['idAdresse']) ? $_REQUEST['idAdresse'] : "";
97      $idAdresse = filter_var($_REQUEST['idAdresse'], FILTER_SANITIZE_STRING);
```

```
98      // Construction du modèle et implémentation de la persistance :
99      $modele = \CoursPHP\Modele\ModelAdresse::getModelAdresse($idAdresse);
100     // test d'erreur :
101     if ($modele->getError() === false){ // Appel de la vue
102       require (\CoursPHP\Config\Config::getVues()["saisieAdresseUpdate"]);
103     }else{ // Appel de la vue d'erreur
104       require (\CoursPHP\Config\Config::getVuesErreur()["default"]);
105     }
106   }
107
108   /** @brief Implemente l'action "update" : met à jour une instance dans la BD
109    */
110   private function actionUpdate(){
111     // Construction du modèle de données validées et implémentation persistance
112     $modele = \CoursPHP\Modele\ModelAdresse::getModelAdresseUpdate($_REQUEST);
113     // test d'erreur :
114     if ($modele->getError() === false){ // Appel de la vue
115         require (\CoursPHP\Config\Config::getVues()["afficheAdresse"]);
116     }else{ // Appel de la vue d'erreur
117       // Erreur de saisie (attribut incorrect)
118       require (\CoursPHP\Config\Config::getVuesErreur()["saisieAdresseUpdate"]);
119     }
120   }
121
122   /** @brief Implemente l'action "create" : crée une instance dans la BD
123    */
124   private function actionCreate(){
125     // Construction du modèle de données validées et implémentation persistance
126     $modele = \CoursPHP\Modele\ModelAdresse::getModelAdresseCreate($_REQUEST);
127     // test d'erreur :
128     if ($modele->getError() === false){ // Appel de la vue
129         require (\CoursPHP\Config\Config::getVues()["afficheAdresse"]);
130     }else{ // Appel de la vue d'erreur
131       // Erreur de saisie (attribut incorrect)
132       require (\CoursPHP\Config\Config::getVuesErreur()["saisieAdresseCreate"]);
133     }
134   }
135
136   /** @brief Implemente l'action "delete" : supprime une instance via son ID
137    */
138   private function actionDelete(){
139     // ID de l'instance à supprimer
140     $idAdresse = filter_var($_REQUEST['idAdresse'], FILTER_SANITIZE_STRING);
141     // Construction du modèle et implémentation de la persistance :
142     $modele = \CoursPHP\Modele\ModelAdresse::deleteAdresse($idAdresse);
143     // test d'erreur :
144     if ($modele->getError() === false){ // Appel de la vue
145       require (\CoursPHP\Config\Config::getVues()["afficheAdresse"]);
146     }else{ // Appel de la vue d'erreur
147       require (\CoursPHP\Config\Config::getVuesErreur()["default"]);
148     }
149   }
150 }
151 ?>
```

Le Diagramme 9 montre le diagramme de séquence de l'action `get` qui permet d'afficher



Diag 9. Diagramme de séquence de l'action `get` qui permet d'afficher une adresse à partir de son *ID*

une adresse à partir de son *ID*.

Comme expliqué au chapitre 5, si les données issues du tableau `$_REQUEST` sont utilisées, elles doivent être systématiquement filtrées (validées ou nettoyées, voir partie 5.5).

Un appel de méthode construit un *modèle*, instance de classe qui contiendra les données nécessaires à la vue, et un tableau d'erreur, éventuellement non vide. La construction du modèle est décrite dans la partie 12.3 ci-dessous. Le contrôleur appelle ensuite la vue, qui sera éventuellement une vue d'erreur en cas de tableau d'erreurs non vide.

12.3 Le Modèle

Le but du modèle est de stocker les données nécessaires à la vue pour afficher le résultat d'une action. Par exemple, pour l'action `get-all` affichant toutes les adresses de la base de données, le modèle doit contenir une collection d'instances d'`Adresse`, qui contient toutes les adresses. Pour l'action `create` de création d'une nouvelle adresse, la vue doit afficher l'adresse saisie pour confirmation, et le modèle contiendra donc une seule instance d'`Adresse`.

Dans notre implémentation, le modèle contient aussi les données d'erreurs. Une classe de base appelée `Model`, générique, contient le tableau des erreurs et son accesseur (qui renvoit le booléen `false` en cas de tableau vide). Le tableau des erreurs est un tableau associatif dont les clés sont les types d'erreurs (champs de formulaire incorrect, login, accès à la base de données, etc.) et la valeur un message d'erreur pour informer l'utilisateur ou stocker dans un fichier *log*.

Code Source 12.3 : /mvc/Modele/Model.php

```
1  <?php
2  namespace CoursPHP\Modele;
3  /** @brief Classe de base pour toutes les classes contenant des modèles.
4   * Cette classe vise seulement à factoriser le code concernant les données
5   * d'erreurs (tableau associatif dont les valeurs sont des messages d'erreur) */
6  class Model{
7    /** @brief Dictionnaire d'erreurs (couples type => message) */
8    protected $dataError;
9
10   /** @brief return false en l'absence d'erreurs, la collection d'erreurs sinon
11    * @return un tableau associatif dont les valeurs sont des messages d'erreur.
          */
12   public function getError(){
13     if (empty($this->dataError)){
14       return false;
15     }
16     return $this->dataError;
17   }
18
19   /** @brief Constructeur
20    * @param $dataError un tableau associatif dont les valeurs sont des messages
          d'erreur.
21    * (par exemple un tableau vide, au début d'un traitement) */
22   public function __construct($dataError = array()){
23     $this->dataError = $dataError;
24   }
25 }
26 ?>
```

La classe `ModelAdresse`, qui hérite de `Model` contient les données d'une instance d'`Adresse`, avec son accesseur `getData()`. Dans notre implémentation, le modèle contient aussi du texte (ici le titre) à afficher dans la vue.

Les méthodes de la classe `ModelAdresse` correspondent aux différentes actions qui ne portent que sur une seule adresse (suppression d'une adresse, saisie ou modification d'une adresse, affichage des détails d'une adresse, etc.) Ces méthodes appellent des méthodes d'accès aux données de la classe `AdresseGateway` pour implémenter la persistence.

Code Source 12.4 : /mvc/Modele/ModelAdresse.php

```php
<?php
namespace CoursPHP\Modele;
/** @brief Classe Modèle pour stocker une Adresse
 * Construit un modèle de données pour les vues affichant une unique adresse.
 * Les données peuvent venir d'un formulaire ou d'un accès à la BD. */
class ModelAdresse extends Model
{
  /** Instance d'adresse, données métier du modèle */
  private $adresse;

  /** Titre principal de la vue */
  private $title;

  /** Donne accès à la donnée d'adresse */
  public function getData(){
    return $this->adresse;
  }

  /** Donne accès au titre de la vue à afficher */
  public function getTitle(){
    return $this->title;
  }

  /** @brief Retourne un modèle avec une instance d'Adresse par défaut
   * (par exemple pour créer un formulaire vide) */
  public static function getModelDefaultAdresse(){
    $model = new self(array());
    // Appel de la couche d'accès aux données :
    $model->adresse = \CoursPHP\Metier\Adresse
              ::getdefaultInstance("0000000000",
                                   \CoursPHP\Metier\Adresse
                                       ::generateRandomId());
    $model->title = "Saisie d'une adresse";
    return $model;
  }

  /** @brief Retourne un modèle avec une instance d'Adresse à partir
   * de son ID par accès à la couche Persistance.
   * @param $idAdresse Identifiant unique de l'adresse à consulter */
  public static function getModelAdresse($idAdresse){
    $model = new self(array());
    // Appel de la couche d'accès aux données :
    $model->adresse = \CoursPHP\Persistance\AdresseGateway::getAdresseById(
                        $model->dataError, $idAdresse);
    $model->title = "Affichage d'une adresse";
    return $model;
```

```
47      }
48
49      /** @brief Modifie une adresse dans la couche Persistance.
50       * @param $inputArray tableau associatif dont les clefs correspondent
51       *                    aux noms des attributs d'Adresse */
52      public static function getModelAdresseUpdate($inputArray){
53        $model = new self(array());
54        // Appel de la couche d'accès aux données :
55        $model->adresse = \CoursPHP\Persistance\AdresseGateway::updateAdresse(
56                                                  $model->dataError, $inputArray);
57        $model->title = "L'adresse a été mise à jour";
58        return $model;
59      }
60
61      /** @brief Insère une adresse en créant un nouvel ID dans la BD
62       * @param $inputArray tableau associatif dont les clefs correspondent aux noms
63       *                    des attributs d'Adresse (à l'exveption de l'ID) */
64      public static function getModelAdresseCreate($inputArray){
65        $model = new self(array());
66        // Appel de la couche d'accès aux données :
67        $model->adresse = \CoursPHP\Persistance\AdresseGateway::createAdresse(
68                                                  $model->dataError, $inputArray);
69        $model->title = "L'adresse a été insérée";
70        return $model;
71      }
72
73      /** @brief Supprime une adresse dans la BD et retourne l'adresse.
74       * @param $idAdresse Identifiant unique de l'adresse à supprimer */
75      public static function deleteAdresse($idAdresse){
76      $model = new self(array());
77          // Appel de la couche d'accès aux données :
78      $model->adresse = \CoursPHP\Persistance\AdresseGateway::deleteAdresse(
79                                                 $model->dataError, $idAdresse);
80      $model->title = "Adresse supprimée";
81      return $model;
82      }
83  }
84  ?>
```

La classe `ModelCollectionAdresse`, qui hérite aussi de `Model` contient les données d'une collection d'instances d'`Adresse`, avec son accesseur `getData()`, qui cette fois renvoie la collection en question. Les méthodes de la classe `ModelCollectionAdresse` correspondent aux différentes actions qui ne portent que sur une collection d'adresse (ici, uniquement "afficher toute la table", mais on pourrait, par exemple, faire des requêtes avec `LIMIT` et `OFFSET` pour paginer). Ces méthodes appelle des méthodes d'accès aux données de la classe `AdresseGateway` pour implémenter la persistence.

Code Source 12.5 : /mvc/Modele/ModelCollectionAdresse.php

```
1  <?php
2  namespace CoursPHP\Modele;
3  /**
4   * @brief Classe Modèle pour stocker une collection de Adresse
5   */
6  class ModelCollectionAdresse extends Model
7  {
```

```php
8     /** Collection d'adresses, données métier du modèle */
9     private $collectionAdresse;
10
11    /** Donne accès à la collection d'adresses */
12    public function getData(){
13      return $this->collectionAdresse;
14    }
15
16    /** @brief Constructeur par défaut (privé, crée des collections vides) */
17    private function __contruct(){
18      $this->collectionAdresse = array();
19      $this->dataError = array();
20    }
21
22   /** @brief Retourne un modèle avec la collection de toutes les adresses
23     * par accès à la base de données. */
24    public static function getModelAdresseAll(){
25      $model = new self(array());
26      // Appel de la couche d'accès aux données :
27      $model->collectionAdresse = \CoursPHP\Persistance\AdresseGateway
28                                          ::getAdresseAll($model->dataError);
29      return $model;
30    }
31  }
32  ?>
```

Notons qu'il ne s'agit pas vraiment ici de polymorphisme. L'héritage de la classe de base `Model`, qui correspond bien à une relation de spécialisation, n'a pas la propriété de substitution, car les données dans les classes spécialisées (instance dans un cas et collection dans l'autre, ne se correspondent pas. L'héritage est ici utilisé pour factoriser uniquement.
Ce type de relation pourrait aussi (devraient plutôt ?) être implémenté par une composition.
On pourrait aussi implémenter l'instance d'adresse comme une collection avec un seul élément pour n'avoir qu'une seule classe.

12.4 Les Vues

Les vues ne font aucun test et se contentent d'afficher le modèle qui, en l'absence de bug, doit s'afficher correctement. Voyons tout d'abord la vue qui affiche une adresse (figures 10.1b, 10.1c par exemple) :

Code Source 12.6 : /mvc/Vue/vues/vueAfficheAdresse.php

```php
1  <?=\CoursPHP\Vue\VueHtmlUtils::enTeteHTML5('Bienvenue sur notre site',
2          'UTF-8', \CoursPHP\Config\Config::getStyleSheetsURL()['default']) ?>
3
4  <h1><?=$modele->getTitle() ?></h1>
5
6  <?=\CoursPHP\Vue\AdresseView::getHtmlDevelopped($modele->getData()) ?>
7  <p>
8    <a href="<?=\CoursPHP\Config\Config::getRootURI() ?>">Revenir à l'accueil</a>
9  </p>
```

```
10 <?=\CoursPHP\Vue\VueHtmlUtils::finFichierHtml5();?>
```

Voyons maintenant la vue qui affiche toutes les adresses (figure 10.1d) :

Code Source 12.7 : /mvc/Vue/vues/vueCollectionAdresse.php

```
1  <?=\CoursPHP\Vue\VueHtmlUtils::enTeteHTML5('Bienvenue sur notre site', '
2        UTF-8', \CoursPHP\Config\Config::getStyleSheetsURL()['default']) ?>
3  <h1>Toutes les adresses</h1>
4  <p>
5    <a href="<?=\CoursPHP\Config\Config::getRootURI() ?>">Revenir à l'accueil</a>
6  </p>
7  <?php
8    echo "<table><tbody>";
9    foreach ($modele->getData() as $adresse){
10     echo "<tr>";
11     echo "<td><a href=\"?action=delete&idAdresse=".$adresse->idAdresse
12                                                    ."\">supprimer</a></td>";
13     echo "<td><a href=\"?action=edit&idAdresse=".$adresse->idAdresse
14                                                    ."\">modifier</a></td>";
15     echo "<td>".\CoursPHP\Vue\AdresseView::getHtmlCompact($adresse)."</td>";
16     echo "<tr>";
17   }
18   echo "</tbody></table>";
19 ?>
20 <?=\CoursPHP\Vue\VueHtmlUtils::finFichierHtml5();?>
```

Voyons enfin, par exemple, la vue d'erreur concernant la saisie incorrecte d'une adresse (figure 10.2b).

Code Source 12.8 : /mvc/Vue/vues/vueErreurSaisieCreate.php

```
1  <?=\CoursPHP\Vue\VueHtmlUtils::enTeteHTML5('Bienvenue sur notre site',
2        'UTF-8', \CoursPHP\Config\Config::getStyleSheetsURL()['default']) ?>
3
4  <h1>Erreur de saisie d'une adresse</h1>
5  <?=\CoursPHP\Vue\AdresseFormView::getFormErrorsHtml("?action=create",
6        $modele->getData(), $modele->getError()) ?>
7  <p>
8    <a href="<?=\CoursPHP\Config\Config::getRootURI() ?>">Revenir à l'accueil</a>
9  </p>
10 <?=\CoursPHP\Vue\VueHtmlUtils::finFichierHtml5();?>
```

Chapitre 13

Utilisateurs et *Front Controller*

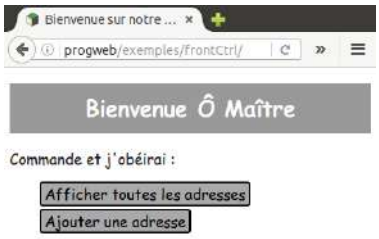
13.1 *Storyboards*

(a) Vues d'accueil

(b) Vue de toutes les adresses

(c) La vue d'authentification

FIGURE 13.1 : *Storyboards* : Vues accessible avec le rôle de visiteur



(a) Vue d'accueil

(b) Vue *admin* de toutes les adresses

FIGURE 13.2 : *Storyboards* : Vue accessibles avec le rôle d'administrateur.
Voir les autres vues sur les figures 10.1b, 10.1c et 10.2

13.2 Diagramme de Cas d'Utilisation

Dans cette partie, nous proposons un *pattern* d'architecture *WEB* pour gérer plusieurs catégories d'utilisateurs avec des rôles différents. Dans notre application démo, nous aurons deux types d'utilisateurs :

1. Les visiteurs (utilisateurs non authentifiés) ;
2. Les administrateurs (nécessairement authentifiés).

Dans le diagramme de cas d'utilisation de la figure 13.3, nous proposons unn type d'utilisateurs générique *User*, avec deux spécialisations : *Visitor* et *Admin*.

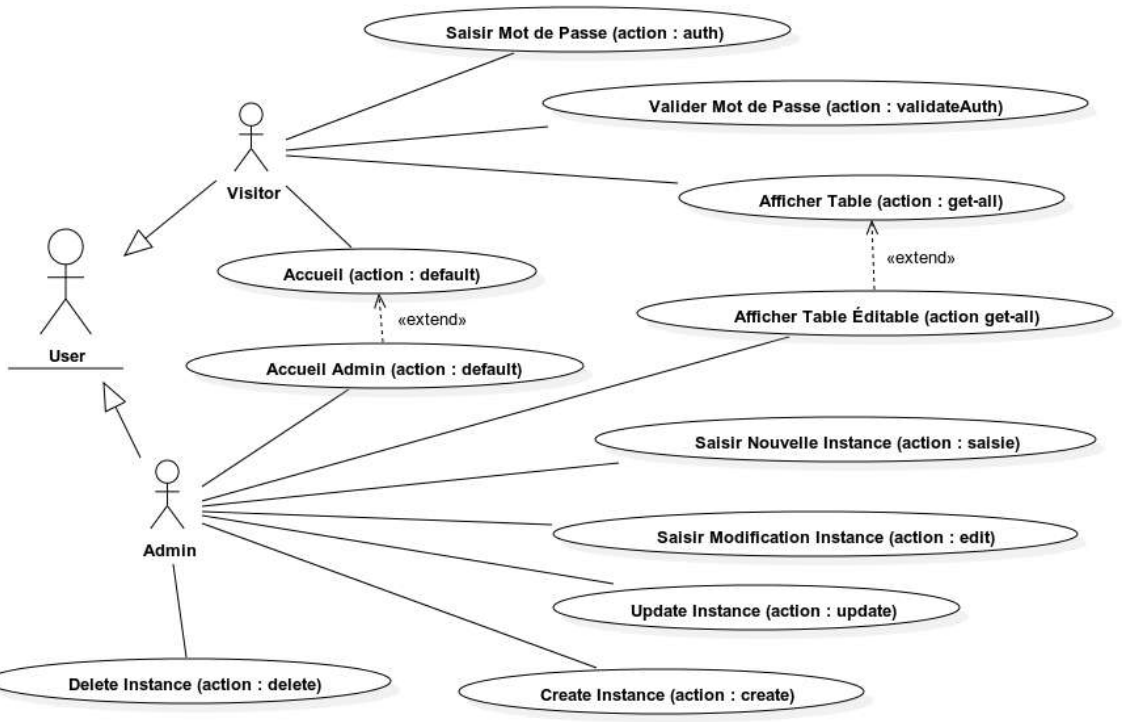


FIGURE 13.3 : *Use Case Diagram* : Les actions possibles pour les deux types d'utilisateurs

13.3 Le *Front-Controller*

Lorsque nous souhaitons gérer plusieurs rôles d'utilisateurs, il est commode d'avoir un contrôleur par rôle, qui gère les actions disponibles pour les utilisateurs de ce rôle. Cependant, sans le *Front-Controller*, cela nécessiterait de gérer plusieurs *URL* d'index, correspondant à des droits d'accès différents. Le rôle du *Front-Controller* est :

1. De distinguer les droits d'accès des différentes actions ;
2. De tester si l'utilisateur a des droits suffisants (si les droits sont insuffisants, on affiche selon le cas une page d'erreur ou une page d'authentification.) ;
3. De créer une instance du contrôleur adapté pour l'action avec le rôle de l'utilisateur.

Code Source 13.1 : /frontCtrl/index.php

```php
<?php
  // Répertoire racine du MVC
  $rootDirectory = dirname(__FILE__)."/";

  // Calcul portable de l'URI de la racine du MVC (sans la query string)
  // 1) On enlève la "query string" : ?action=blabla&id=03456789
  $urlWithoutQueryString = explode("?", $_SERVER['REQUEST_URI'])[0];
  // 2) on coupe l'URL du script au niveau de l'extension ".php"
  $scriptWithoutExtention = explode(".php", $urlWithoutQueryString)[0];
  // 3) puis on s'arrête au dernier slash (pour enlever la basename du script)
  $longueurRootURI = strrpos($scriptWithoutExtention, '/');
  // 4) On prend le début de l'URL en coupant à la bonne longueur
  $rootURI = substr($_SERVER['REQUEST_URI'], 0, $longueurRootURI);

  // chargement de l'autoload pour autochargement des classes
  require_once($rootDirectory.'/Config/Autoload.php');
  CoursPHP\Config\Autoload::load_PSR_4('CoursPHP\\');

  // Création de l'instance du contrôleur (voir Controleur.php)
  $ctrl = new \CoursPHP\Controleur\ControleurFront();
?>
```

Code Source 13.2 : /frontCtrl/Controleur/ControleurFront.php

```php
<?php
namespace CoursPHP\Controleur;
/**
 * @brief Le ControleurFront identifie l'action et le rôle de l'utilisateur
 * Dans le cas où l'utilisateur a des droits insuffisants pour l'action,
 * le ControleurFront affiche une vue d'autentification ou un vue d'erreur.
 * Sinon, ControleurFront instancie le contrôleur adapté pour les rôle et action
 * Il gère aussi les exceptions et appelle le cas échéant une vue d'erreur.
 */
class ControleurFront {
  /**
   * @brief C'est dans le contructeur que le contrôleur fait son travail.
   */
  function __construct() {
    // Récupération d'une évéentuelle exception, d'où qu'elle vienne.
    try{
      // Récupération de l'action
      $action = isset($_REQUEST['action']) ? $_REQUEST['action'] : "";

      // L'utilisateur est-il identifié ? Si oui, quel est son rôle ?
      $modele = \CoursPHP\Auth\Authentication::restoreSession();
      $role = ($modele->getError() === false) ? $modele->getRole() : "";

      // On distingue des cas d'utilisation, suivant l'action et le rôle
      switch($action){

        // 1) Actions accessibles uniquement aux visiteurs (rôle par défaut)
        case "auth": // Vue de saisie du login/password
        case "validateAuth": // Validation du login/password
          $publicCtrl = new ControleurVisitor($action);
          break;
```

```
32
33          // 2) Actions accessibles uniquement aux administrateurs :
34          case "saisie": // Saisie d'une nouvelle Adresse
35          case "edit": // Saisie des modifications d'une Adresse
36          case "update": // Met à jour une Adresse dans la BD
37          case "create": // Cration d'une nouvelle Adresse dans la BD
38          case "delete": // Supression d'une Adresse à partir de son ID
39            // L'utilisateur a-t-il des droits suffisants ?
40            if ($role === "admin"){
41              $adminCtrl = new ControleurAdmin($action);
42            }else{
43              require (\CoursPHP\Config\Config::getVues()["authentification"]);
44            }
45            break;
46
47          // 3) Actions accessibles aux visiteurs et aux administrateurs :
48          case "get": // Affichage d'une Adresse à partir de son ID
49          case "get-all": // Affichage de toutes les Adresse's
50          default: // L'action par défaut
51            // L'implémentation (donc le contrôleur) dépend du rôle
52            if ($role === "admin"){
53              $adminCtrl = new ControleurAdmin($action);
54            }else{
55              $publicCtrl = new ControleurVisitor($action);
56            }
57        }
58      }catch (Exception $e){ // Page d'erreur par défaut
59        $modele = new \CoursPHP\Modele\Model(
60                                  array('exception' => $e->getMessage()));
61          require (\CoursPHP\Config\Config::getVuesErreur()["default"]);
62      }
63    }
64  }
65  ?>
```

Voici le contrôleur spécialisé pour les actions accessibles à un visiteur non authentifié.

Code Source 13.3 : /frontCtrl/Controleur/ControleurVisitor.php

```
1  <?php
2  namespace CoursPHP\Controleur;
3  /**
4   * @brief ControleurVisitor identifie l'action et appelle la méthode
5   * pour construire le modèle correspondant à l'action et au rôle "visistor".
6   * Le controleur appelle aussi la vue correspondante.
7   * Il ne gère pas les exceptions, qui remontent au Front Controller.
8   */
9  class ControleurVisitor {
10   /**
11    * @brief C'est dans le contructeur que le contrôleur fait son travail.
12    */
13   function __construct($action) {
14     // On distingue des cas d'utilisation, suivant l'action
15     switch($action){
16       case "auth":
17         $this->actionAuth();
18         break;
```

```
19          case "validateAuth":
20            $this->actionValidateAuth();
21            break;
22          case "get": // Affichage d'une Adresse à partir de son ID
23            $this->actionGet();
24            break;
25          case "get-all": // Affichage de toutes les Adresse's
26            $this->actionGetAll();
27            break;
28          default: // L'action indéfinie (page par défaut, ici accueil)
29            require (\CoursPHP\Config\Config::getVues()["default"]);
30            break;
31      }
32    }
33
34    /**
35     * @brief Implemente l'action "auth" : Saisie du login/mot de passe
36     */
37    private function actionAuth(){
38      $modele = new \CoursPHP\Modele\Model(array());
39      require (\CoursPHP\Config\Config::getVues()["authentification"]);
40    }
41
42    /**
43     * @brief Implemente l'action "validateAuth"
44     * Validation du login/password et création de session.
45     */
46    private function actionValidateAuth(){
47      \CoursPHP\Auth\ValidationRequest::validationLogin($dataError, $email,
            $password);
48      $modele = \CoursPHP\Auth\Authentication::checkAndInitiateSession(
49                                              $email, $password, $dataError);
50      if ($modele->getError() === false){
51        require (\CoursPHP\Config\Config::getVues()["defaultAdmin"]);
52      }else{
53        require (\CoursPHP\Config\Config::getVues()["authentification"]);
54      }
55    }
56
57    /**
58     * @brief Implemente l'action "get" : récupère une instance à partir de ID
59     */
60    private function actionGet(){
61      // ID de l'instance à récupérer
62      $rawId = isset($_REQUEST['idAdresse']) ? $_REQUEST['idAdresse'] : "";
63      $idAdresse = filter_var($rawId, FILTER_SANITIZE_STRING);
64      $modele = \CoursPHP\Modele\ModelAdresse::getModelAdresse($idAdresse);
65      if ($modele->getError() === false){
66        require (\CoursPHP\Config\Config::getVues()["afficheAdresse"]);
67      }else{
68        require (\CoursPHP\Config\Config::getVuesErreur()["default"]);
69      }
70    }
71
72    /**
73     * @brief Implemente l'action "get-all"  : Récupère toutes les instances
```

```
74      */
75    private function actionGetAll(){
76      $modele = \CoursPHP\Modele\ModelCollectionAdresse::getModelAdresseAll();
77      if ($modele->getError() === false){
78        require (\CoursPHP\Config\Config::getVues()["afficheCollectionAdresse"]);
79      }else{
80        require (\CoursPHP\Config\Config::getVuesErreur()["default"]);
81      }
82    }
83  }
84  ?>
```

Voici le contrôleur spécialisé pour les actions accessibles à un utilisateur authentifié avec le rôle `admin`.

Code Source 13.4 : /frontCtrl/Controleur/ControleurAdmin.php

```
1   <?php
2   namespace CoursPHP\Controleur;
3   /**
4    * @brief ControleurVisitor identifie l'action et appelle la méthode
5    * pour construire le modèle correspondant à l'action et au rôle "admin".
6    * Le controleur appelle aussi la vue correspondante.
7    * Il ne gère pas les exceptions, qui remontent au Front Controller.
8    */
9
10  class ControleurAdmin {
11    /**
12      * @brief C'est dans le contructeur que le contrôleur fait son travail.
13      */
14    function __construct($action) {
15      // On distingue des cas d'utilisation, suivant l'action
16      switch($action){
17        case "get": // Affichage d'une Adresse à partir de son ID
18          $this->actionGet();
19          break;
20        case "get-all": // Affichage de toutes les Adresse's
21          $this->actionGetAll();
22          break;
23        case "saisie": // Saisie d'une nouvelle Adresse
24          $this->actionKeyIn();
25          break;
26        case "edit": // Saisie des modifications d'une Adresse
27          $this->actionEdit();
28          break;
29        case "update": // Met à jour une Adresse dans la BD
30          $this->actionUpdate();
31          break;
32        case "create": // Cration d'une nouvelle Adresse dans la BD
33          $this->actionCreate();
34          break;
35        case "delete": // Supression d'une Adresse à partir de son ID
36        $this->actionDelete();
37          break;
38        default: // L'action indéfinie (page par défaut, ici accueil)
39          require (\CoursPHP\Config\Config::getVues()["defaultAdmin"]);
40        break;
```

```
41        }
42    }
43
44    /** @brief Implemente l'action "get" : Récupère une instance à partir de ID
45     */
46    private function actionGet(){
47        // ID de l'instance à récupérer
48        $rawId = isset($_REQUEST['idAdresse']) ? $_REQUEST['idAdresse'] : "";
49        $idAdresse = filter_var($rawId, FILTER_SANITIZE_STRING);
50        $modele = \CoursPHP\Modele\ModelAdresse::getModelAdresse($idAdresse);
51        if ($modele->getError() === false){
52            require (\CoursPHP\Config\Config::getVues()["afficheAdresseAdmin"]);
53        }else{
54            require (\CoursPHP\Config\Config::getVuesErreur()["default"]);
55        }
56    }
57
58    /** @brief Implemente l'action "get-all" : Récupère toutes les instances
59     */
60    private function actionGetAll(){
61        $modele = \CoursPHP\Modele\ModelCollectionAdresse::getModelAdresseAll();
62        if ($modele->getError() === false){
63            require (\CoursPHP\Config\Config::getVues()["afficheCollectionAdresseAdmin
                "]);
64        }else{
65            require (\CoursPHP\Config\Config::getVuesErreur()["default"]);
66        }
67    }
68
69    /** @brief Implemente l'action "saisie" : Affiche un formulaire vierge
70     */
71    private function actionKeyIn(){
72            $modele = \CoursPHP\Modele\ModelAdresse::getModelDefaultAdresse();
73            require (\CoursPHP\Config\Config::getVues()["saisieAdresseCreate"]);
74    }
75
76    /**  @brief Implemente l'action "edit" : Affiche un formulaire de modification
77     */
78    private function actionEdit(){
79        // ID de l'instance à modifier
80        $rawId = isset($_REQUEST['idAdresse']) ? $_REQUEST['idAdresse'] : "";
81        $idAdresse = filter_var($_REQUEST['idAdresse'], FILTER_SANITIZE_STRING);
82        $modele = \CoursPHP\Modele\ModelAdresse::getModelAdresse($idAdresse);
83        if ($modele->getError() === false){
84            require (\CoursPHP\Config\Config::getVues()["saisieAdresseUpdate"]);
85        }else{
86            require (\CoursPHP\Config\Config::getVuesErreur()["default"]);
87        }
88    }
89
90    /** @brief Implemente l'action "update" : Met à jour une instance dans la BD
91     */
92    private function actionUpdate(){
93        // Construction du modèle avec l'adresse validée
94        $modele = \CoursPHP\Modele\ModelAdresse::getModelAdresseUpdate($_POST);
95        if ($modele->getError() === false){
```

```
96            require (\CoursPHP\Config\Config::getVues()["afficheAdresse"]);
97        }else{
98            // Erreur de saisie
99            require (\CoursPHP\Config\Config
100                                ::getVuesErreur()["saisieAdresseUpdate"]);
101       }
102   }
103
104   /** @brief Implemente l'action "create" : Crée une instance dans la BD
105    */
106   private function actionCreate(){
107       // Construction du modèle avec l'adresse validée
108       $modele = \CoursPHP\Modele\ModelAdresse::getModelAdresseCreate($_POST);
109       if ($modele->getError() === false){
110           require (\CoursPHP\Config\Config::getVues()["afficheAdresse"]);
111       }else{
112           // Erreur de saisie
113           require (\CoursPHP\Config\Config
114                                ::getVuesErreur()["saisieAdresseCreate"]);
115       }
116   }
117
118   /** @brief Implemente l'action "delete" : Supprime une instance via son ID
119    */
120   private function actionDelete(){
121       // ID de l'instance à supprimer
122       $idAdresse = filter_var($_REQUEST['idAdresse'], FILTER_SANITIZE_STRING);
123       $modele = \CoursPHP\Modele\ModelAdresse::deleteAdresse($idAdresse);
124       if ($modele->getError() === false){
125           require (\CoursPHP\Config\Config::getVues()["afficheAdresse"]);
126       }else{
127           require (\CoursPHP\Config\Config::getVuesErreur()["default"]);
128       }
129   }
130 }
131 ?>
```

13.4 Gestion de l'Authentification

La vue d'authentification, ainsi que les utilitaires de génération d'*HTML* correspondants sont exactement les mêmes que dans la partie 7.4 et les autres éléments (sessions, *cookies*) en sont fortement inspirés. La plus grande différence par rapport à la partie 7.4 est l'implémentation effective de la vérification d'existence du couple *login/password* dans la base de données, utilisant une *Gateway* d'utilisateur sur le modèle de la *DAL* (partie 9.3).

13.4.1 Modèle et *Gateway* de la table *User*

Nous considérons dans notre modélisation que toutes les classes concernant l'authentification des utilisateurs (y compris les modèles et *gateway*) font partie du *package* `Auth`. Nous verrons plus loin dans ce chapitre comment articuler cela avec les données métier concernant l'utilisateur (données personnelles comme le nom, l'adresse, le numéro de téléphone, etc.).

Code Source 13.5 : /frontCtrl/Auth/ModelUser.php

```php
<?php
namespace CoursPHP\Auth;
/** @brief Classe Modèle pour les données de l'utilisateur
 * e-mail (qui sert ici de login), rôle (visitor, admin, etc.)
 * Les données peuvent venir d'une session ou d'un accès à la BD. */
class ModelUser extends \CoursPHP\Modele\Model
{
  /** adresse e-mail de l'utilisateur */
  private $email;
  /** role de l'utilisateur */
  private $role;

  /** Constructeur par défaut (Init. du tableau d'erreurs à vide) */
  public function __construct($dataError){
    parent::__construct($dataError);
  }

  /** Permet d'obtenir l'adresse e-mail (login) */
  public function getEmail(){
    return $this->email;
  }

  /** Permet d'obtenir le rôle (et donc les droits) */
  public function getRole(){
    return $this->role;
  }

  /** @brief Remplie les données de l'utilisateur à partir
   * du login/password par accès à la BD (UserGateway)
   * @param $email e-mail de l'utilisateur  servant d'ID unique
   * @param $hashedPassword mot de passe après hashage */
  public static function getModelUser($email, $hashedPassword){
    $model = new self(array());
    // Appel de la couche d'accès aux données :
    $model->role = UserGateway::getRoleByPassword($model->dataError,
                                                  $email, $hashedPassword);
    if ($model->role !== false){
      $model->email = $email;
    }else{
      $model->dataError['login'] = "Login ou mot de passe incorrect.";
    }
    return $model;
  }

  /** @brief Remplie des données de l'utilisateur à partir de la session
   * @param $email e-mail de l'utilisateur servant d'ID unique
   * @param $role Rôle de l'utilisateur */
  public static function getModelUserFromSession($email, $role){
    $model = new self(array());
    $model->role = $role;
    $model->email = $email;
    return $model;
  }
}
?>
```

Code Source 13.6 : /frontCtrl/Auth/UserGateway.php

```php
<?php
namespace CoursPHP\Auth;
/** Permet d'accéder et mettre à jour les données de la table User
 * dans la base de données (au moins les opérations CRUD). */
class UserGateway{
  /**
   * Vérifie que le couple login/password existe dans la table User
   * @return le rôle de l'utilisateur si login/password valide, une erreur sinon
   */
  public static function getRoleByPassword(&$dataError, $email,
                                           $hashedPassword){
    // Exécution de la requête via la classe de connexion (singleton)
    $args=array($email);
    $queryResults = \CoursPHP\Persistance\DataBaseManager
                    ::getInstance()->prepareAndExecuteQuery(
                      'SELECT * FROM '.\CoursPHP\Config\Config::
                          getTablesPrefix()
                      .'User WHERE email = ?',
                       $args
                    );
    // Si la requête a fonctionné
    if ($queryResults !== false){
      // Si un utilisateur avec cet email existe
      if (count($queryResults) == 1){
        $row = $queryResults[0];
      }
        // Si l'email n'exite pas en BD ou le mot de passe ne correspond pas
      if (count($queryResults) != 1 || $row['password'] != $hashedPassword){
        $dataError['login'] = "Adresse e-mail ou mot de passe incorrect";
        return "";
      }
      return $row['role'];
    }else{
      $dataError['login'] = "Impossible d'accéder à la table des utilisateurs";
      return "";
    }
  }
}
?>
```

13.4.2 Gestion des sessions et des *cookies*

Nous suivons en gros la gestion des numéros de session par *cookie* avec un contrôle par adresse *IP* expliqué dans la partie 7.4. Nous enveloppons le tout dans une nouvelle classe.

Code Source 13.7 : /frontCtrl/Auth/Authentication.php

```php
<?php
namespace CoursPHP\Auth;
/** @brief Permet d'initier une session après saisie du login/password.
 * Permet aussi de restaurer la session d'un utilisateur déjà authentifié. */
class Authentication {
  /**
   * @brief Test du login/password dans la table User et création d'une session
```

```
8       * @return Un modèle avec les données de l'utilisateur pour gestion des rôles
9       * Le modèle contient un tableau d'erreur non vide si l'identification échoue
10      */
11     public static function checkAndInitiateSession($login, $password, $dataError){
12       // On vérifie que le mot de passe (après hashage SHA512)
13       // est bien celui en base de donnée.
14       if (!empty($dataError)){
15         return new \CoursPHP\Modele\Model($dataError);
16       }
17       // On applique le hashage sur le mot de passe :
18       $hashedPassword = hash("sha512", $password);
19       $userModel = ModelUser::getModelUser($login, $hashedPassword);
20       if ($userModel->getError() !== false){
21         return $userModel;
22       }
23       // On crée une session avec les données de l'utilisateur :
24       SessionUtils::createSession($userModel->getEmail(),
25                                   $userModel->getRole());
26       session_write_close();
27       return $userModel;
28     }
29
30     /** @brief Restore la session si l'identificateur a déjà été identifié
31      * @return Un modèle de données de l'utilisateur pour gestion des rôles
32      * Le modèle contient un tableau d'erreur si la restauration de session échoue
33      */
34     public static function restoreSession(){
35       $dataError = array();
36       // Test pour voir si l'identifiant de session existe et a la bonne forme
37       // (10 chiffres hexa entre 0 et f)
38       if (!isset($_COOKIE['session-id']) ||
39         !preg_match("/^[0-9a-fA-F]{20}$/", $_COOKIE['session-id'])){
40           $dataError['no-cookie'] = "Votre cookie a peut-être expirée, "
41                                     ."Merci de vous connecter à nouveau...";
42           $userModel = new \CoursPHP\Modele\Model($dataError);
43       }else{
44         // On a bien vérifié la forme par expression régulière
45         $mySid = $_COOKIE['session-id'];
46         // On récupère les données de session :
47         session_id($mySid);
48         // Le démarage de session
49         session_start();
50
51         // Test sur les données de session et contrôle par IP
52         if (!isset($_SESSION['email']) || !isset($_SESSION['role']) ||
53             !isset($_SESSION['ipAddress']) ||
54             ($_SESSION['ipAddress'] != $_SERVER['REMOTE_ADDR'])){
55           $dataError['session'] = "Unable to recover user session.";
56           $userModel = new \CoursPHP\Modele\Model($dataError);
57         }else{
58           // Création du modèle d'utilisateur :
59           $userModel = ModelUser::getModelUserFromSession($_SESSION['email'],
60                                                           $_SESSION['role']);
61         }
62         // Raffinement : on change le SID aléatoire, en copiant
63         // la session dans une nouvelle. On regénère ensuite le cookie
```

```
64          // Comme ça, le cookie n'est valable qu'une fois, et l'ID de session aussi
65          // ce qui limite beaucoup la possibilité d'un éventuel hacker
66          $backupSessionEmail = $_SESSION['email'];
67          $backupSessionRole = $_SESSION['role'];
68          // On recrée une session :
69          SessionUtils::createSession($backupSessionEmail, $backupSessionRole);
70          // Flush des Données de Session, (sauvegarde simmédiate ur le disque)
71          session_write_close();
72        }
73        return $userModel;
74      }
75  }
76  ?>
```

13.5 Gestion de plusieurs classes métier

13.5.1 Exemple de classes métiers avec agrégation

Voici un exemple dans lequel nous sommes en présence de deux classes métiers avec une agrégation. Dans notre cas, une personne possède un nom et peux avoir plusieurs adresses. L'organisation des classes métier ressemble alors au diagramme de classes de la partie 2.2.2, sauf que la multiplicité de agrégé `Adresse` est `1..*` et non pas `1`.

La vue générale affichant toutes les personnes, avec les droits d'administrateur permettant de modifier les données, ressemble à la figure 13.4. Chaque adresse de chaque personne est modifiable et supprimable. On peut ajouter une adresse dans chaque personne. Enfin, le nom de la personne est modifiable, et la personne est supprimable (avec la suppression des adresses associées en cascade).

13.5.2 Structuration des contrôleurs

Nous proposons, outre la séparation des contrôleurs par rôle de l'utilisateur, de découper, pour chaque rôle, les contrôleurs par classe métier. Nous introduisons en outre un contrôleur spécialisé pour valider le mot de passe lors de l'authentification d'un utilisateur.

Dans notre exemple, nous obtenons donc les contrôleurs suivants :

- `ControleurFront` sera notre *Front Controler*, qui, en fonction de l'action et du rôle de l'utilisateur, construit le contrôleur adapté (ou affiche une vue d'authentification si les droits sont insuffisants pour l'action).
- `ControleurAuth`php pour la gestion des actions concernant l'authentification (saisie ou validation du *login/password*).
- `ControleurVisitorAdresse`php pour la gestion des actions concernant les adresses dans le rôle de visiteur.
- `ControleurVisitorPersonne`php pour la gestion des actions concernant les personnes dans le rôle de visiteur.
- `ControleurAdminAdresse`php pour la gestion des actions concernant les adresses dans le rôle d'administrateur.

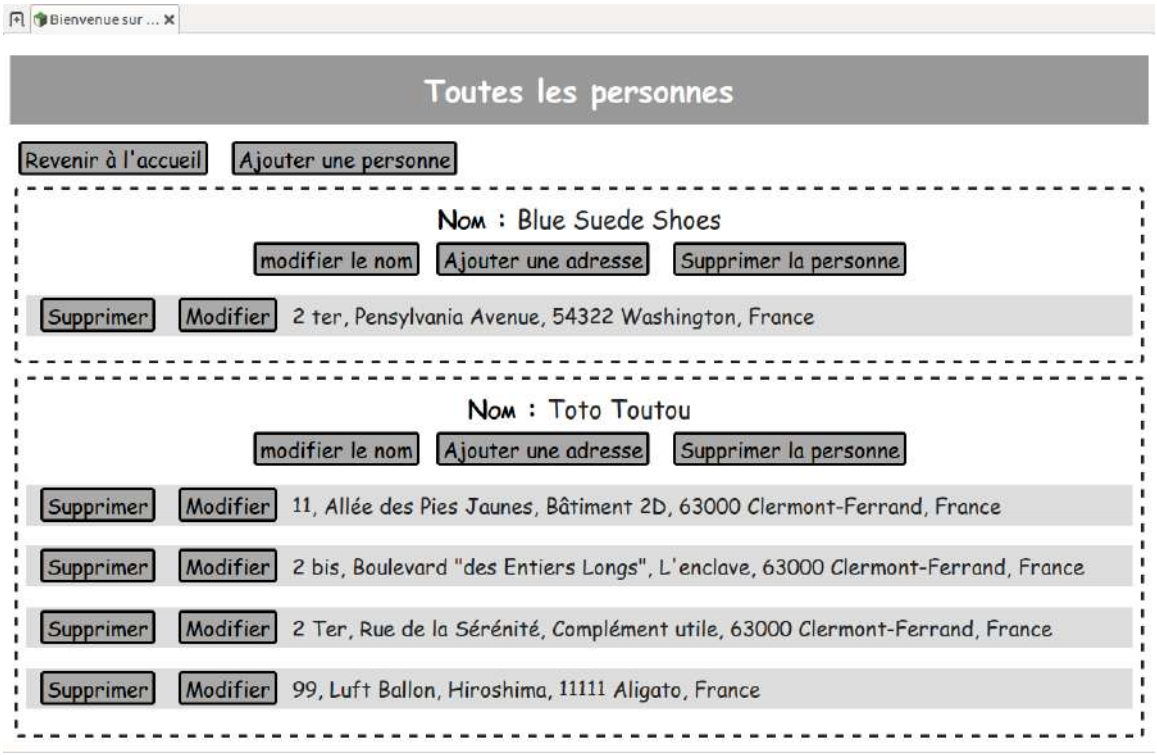


FIGURE 13.4 : Vue général affichant une collection de personnes.

- `ControleurAdminPersonne`php pour la gestion des actions concernant les personnes dans le rôle d'administrateur.

Voici le code du *Front Controller* :

Code Source 13.8 : /metierFrontArchi/Controleur/ControleurFront.php

```php
<?php
namespace CoursPHP\Controleur;
/**
 * @brief Identifie l'action et le rôle de l'utilisateur.
 * Dans le cas où l'utilisateur a des droits insuffisants pour l'action,
 * le ControleurFront affiche une vue d'autentification ou un vue d'erreur.
 * Sinon, ControleurFront instancie le contrôleur adapté pour les rôle et action
 * Il gère aussi les exceptions et appelle le cas échéant une vue d'erreur.
 */
class ControleurFront {
  /**
   * @brief C'est dans le contructeur que le contrôleur fait son travail.
   */
  function __construct() {
    try{
        // Récupération de l'action
        $action = isset($_REQUEST['action']) ? $_REQUEST['action'] : "";

        // L'utilisateur est-il identifié ? Si oui, quel est son rôle ?
        $modele = \CoursPHP\Auth\Authentication::restoreSession();
        $role = ($modele->getError() === false) ? $modele->getRole() : "";

        // On distingue des cas d'utilisation, suivant l'action
```

```
24          switch($action){
25            // 1) Actions concernant l'authentification :
26            case "auth": // Vue de saisie du login/password
27            case "validateAuth": // Validation du login/password
28              $authCtrl = new ControleurAuth($action);
29              break;
30
31            // 2) Actions accessibles uniquement aux administrateurs :
32            // 2.a) Concernant les adresses
33            case "adresse-saisie": // Saisie d'une nouvelle Adresse
34            case "adresse-edit": // Saisie des modifications d'une Adresse
35            case "adresse-update": // Met à jour une Adresse dans la BD
36            case "adresse-create": // Cration d'une nouvelle Adresse dans la BD
37            case "adresse-delete": // Supression d'une Adresse à partir de son ID
38              if ($role == "admin"){
39                $adminCtrl = new ControleurAdminAdresse($action);
40              }else{
41                require (\CoursPHP\Config\Config::getVues()["authentification"]);
42              }
43              break;
44            // 2.b) Concernant les personnes
45            case "personne-saisie": // Saisie d'une nouvelle Personne
46            case "personne-edit": // Saisie des modifications d'une Personne
47            case "personne-update": // Met à jour une Personne dans la BD
48            case "personne-create": // Cration d'une nouvelle Personne dans la BD
49            case "personne-delete": // Supression d'une Personne à partir de son
                ID
50              if ($role == "admin"){
51                $adminCtrl = new ControleurAdminPersonne($action);
52              }else{
53                require (\CoursPHP\Config\Config::getVues()["authentification"]);
54              }
55              break;
56
57            // 3) Actions accessibles aux visiteurs et aux administrateurs :
58            // 3.a) Concernant les adresses
59            case "adresse-get": // Affichage d'une Adresse à partir de son ID
60            case "adresse-get-all": // Affichage de toutes les Adresse's
61                          // L'implémentation (donc le contrôleur) dépend du rôle
62              if ($role == "admin"){
63                  $adminCtrl = new ControleurAdminAdresse($action);
64                }else{
65                  $publicCtrl = new ControleurVisitorAdresse($action);
66                }
67              break;
68            // 3.b) Concernant les personnes
69            case "personne-get-all": // Affichage de toutes les Personne's
70              if ($role == "admin"){
71                  $adminCtrl = new ControleurAdminPersonne($action);
72                }else{
73                  $publicCtrl = new ControleurVisitorPersonne($action);
74                }
75              break;
76            default:
77              if ($role == "admin"){
78                require (\CoursPHP\Config\Config::getVues()["defaultAdmin"]);
```

```
79              }else{
80                require (\CoursPHP\Config\Config::getVues()["default"]);
81              }
82        } // fin du switch
83      }catch (\Exception $e){ // Page d'erreur par défaut
84          $modele = new \CoursPHP\Modele\Model(
85                                  array('exception' => $e->getMessage()));
86          require (\CoursPHP\Config\Config::getVuesErreur()["default"]);
87      }
88    }
89  }
90  ?>
```

Voici à titre d'exemple, le code du *ControllerAuth* gérant les actions associées à l'authentification :

Code Source 13.9 : /metierFrontArchi/Controleur/ControleurAuth.php

```
1   <?php
2   namespace CoursPHP\Controleur;
3   /**
4    * @brief Identifie l'action concernant l'authentification
5    * et appelle la méthode pour construire le modèle pour l'action.
6    * Le controleur appelle aussi la vue correspondante.
7    * Il ne gère pas les exceptions, qui remontent au Front Controller.
8    */
9   class ControleurAuth {
10    /**
11     * @brief C'est dans le contructeur que le contrôleur fait son travail.
12     */
13    function __construct($action) {
14        // On distingue des cas d'utilisation, suivant l'action
15        switch($action){
16          case "auth":
17            $this->actionAuth();
18            break;
19          case "validateAuth":
20            $this->actionValidateAuth();
21            break;
22          default: // L'action indéfinie (page par défaut, ici accueil)
23            require (\CoursPHP\Config\Config::getVues()["default"]);
24            break;
25      }
26    }
27
28    /** @brief Implemente l'action "auth" : saisie du login/password
29     */
30    private function actionAuth(){
31      $modele = new \CoursPHP\Modele\Model(array());
32      require (\CoursPHP\Config\Config::getVues()["authentification"]);
33    }
34
35    /**
36     * @brief Implemente l'action "validateAuth"
37     * Validation du login/password et création de session.
38     */
39    private function actionValidateAuth(){
```

```
40      \CoursPHP\Auth\ValidationRequest::validationLogin($dataError, $email,
            $password);
41      $modele = \CoursPHP\Auth\Authentication::checkAndInitiateSession(
42                                              $email, $password, $dataError);
43      if ($modele->getError() === false){
44        require (\CoursPHP\Config\Config::getVues()["defaultAdmin"]);
45      }else{
46        require (\CoursPHP\Config\Config::getVues()["authentification"]);
47      }
48    }
49  }
50  ?>
```

Voici, toujours à titre d'exemple, le code du *ControllerAdminPersonne* gérant les actions associées aux personnes :

Code Source 13.10 : /metierFrontArchi/Controleur/ControleurAdminPersonne.php

```
1   <?php
2   namespace CoursPHP\Controleur;
3   /**
4    * @brief Identifie l'action concernant des Personne avec le rôle admin
5    * et appelle la méthode pour construire le modèle correspondant à l'action
6    * avec le rôle "admin". Le controleur appelle aussi la vue correspondante.
7    * Il ne gère pas les exceptions, qui remontent au Front Controller.
8    */
9   class ControleurAdminPersonne {
10    /**
11     * @brief C'est dans le contructeur que le contrôleur fait son travail.
12     */
13    function __construct($action) {
14      // On distingue des cas d'utilisation, suivant l'action
15      switch($action){
16        case "personne-get": // Affichage d'une Personne à partir de son ID
17          $this->actionGet();
18          break;
19        case "personne-get-all": // Affichage de toutes les Personne's
20          $this->actionGetAll();
21          break;
22        case "personne-saisie": // Saisie d'une nouvelle Adresse
23          $this->actionSaisie();
24          break;
25        case "personne-edit": // Saisie des modifications d'une Adresse
26          $this->actionEdit();
27          break;
28        case "personne-update": // Met à jour une Adresse dans la BD
29          $this->actionUpdate();
30          break;
31        case "personne-create": // Cration d'une nouvelle Adresse dans la BD
32          $this->actionCreate();
33          break;
34        case "personne-delete": // Supression d'une Adresse à partir de son ID
35        $this->actionDelete();
36          break;
37        default: // L'action indéfinie (page par défaut, ici accueil)
38          require (\CoursPHP\Config\Config::getVues()["defaultAdmin"]);
39        break;
```

```
40      }
41    }
42
43    /** @brief Implemente l'action "get" : récupère une instance à partir de ID
44     */
45    private function actionGet(){
46      // ID de l'instance à récupérer
47      $rawId = isset($_REQUEST['idPersonne']) ? $_REQUEST['idPersonne'] : "";
48      $idPersonne = filter_var($rawId, FILTER_SANITIZE_STRING);
49      $modele = \CoursPHP\Modele\ModelPersonne::getModelPersonne($idPersonne);
50      if ($modele->getError() === false){
51        require (\CoursPHP\Config\Config::getVues()["affichePersonneAdmin"]);
52      }else{
53        require (\CoursPHP\Config\Config::getVuesErreur()["default"]);
54      }
55    }
56
57    /** @brief Implemente l'action "get-all" : récupère toutes les instances
58     */
59    private function actionGetAll(){
60      $modele = \CoursPHP\Modele\ModelCollectionPersonne::getModelPersonneAll();
61      if ($modele->getError() === false){
62        require (\CoursPHP\Config\Config::getVues()["
              afficheCollectionPersonneAdmin"]);
63      }else{
64        require (\CoursPHP\Config\Config::getVuesErreur()["default"]);
65      }
66    }
67
68    /** @brief Implemente l'action "saisie" : affiche un formulaire vierge
69     */
70    private function actionSaisie(){
71      $idPersonne = isset($_REQUEST['idPersonne']) ?
72                  filter_var($_REQUEST['idPersonne'], FILTER_SANITIZE_STRING) : ""
                        ;
73      $modele = \CoursPHP\Modele\ModelPersonne::getModelDefaultPersonne(
            $idPersonne);
74      require (\CoursPHP\Config\Config::getVues()["saisiePersonneCreate"]);
75    }
76
77    /** @brief Implemente l'action "edit" : affiche un formulaire de modification
78     */
79    private function actionEdit(){
80      // ID de l'instance à modifier
81      $rawId = isset($_REQUEST['idPersonne']) ? $_REQUEST['idPersonne'] : "";
82      $idPersonne = filter_var($_REQUEST['idPersonne'], FILTER_SANITIZE_STRING);
83      $modele = \CoursPHP\Modele\ModelPersonne::getModelPersonne($idPersonne);
84      if ($modele->getError() === false){
85        require (\CoursPHP\Config\Config::getVues()["saisiePersonneUpdate"]);
86      }else{
87        require (\CoursPHP\Config\Config::getVuesErreur()["default"]);
88      }
89    }
90
91    /** @brief Implemente l'action "update" : met à jour une instance dans la BD
92     */
```

```php
93    private function actionUpdate(){
94      // Construire le modèle de Personne mise à jour
95      $modele = \CoursPHP\Modele\ModelPersonne::getModelPersonneUpdate($_POST);
96      if ($modele->getError() === false){
97          require (\CoursPHP\Config\Config::getVues()["affichePersonne"]);
98      }else{
99        // Erreur de saisie
100       require(\CoursPHP\Config\Config::getVuesErreur()["saisieUpdatePersonne"]);
101     }
102   }
103
104   /** @brief Implemente l'action "create" : crée une instance dans la BD
105    */
106   private function actionCreate(){
107     // Construire le modèle de Personne mise à jour
108     $modele = \CoursPHP\Modele\ModelPersonne::getModelPersonneCreate($_POST);
109     if ($modele->getError() === false){
110         require (\CoursPHP\Config\Config::getVues()["affichePersonne"]);
111     }else{
112       // Erreur de saisie
113       require(\CoursPHP\Config\Config::getVuesErreur()["saisieCreatePersonne"]);
114     }
115   }
116
117   /** @brief Implemente l'action "delete" : supprime une instance via son ID
118    */
119   private function actionDelete(){
120     // ID de l'instance à supprimer
121     $idPersonne = filter_var($_REQUEST['idPersonne'], FILTER_SANITIZE_STRING);
122     $modele = \CoursPHP\Modele\ModelPersonne::deletePersonne($idPersonne);
123     if ($modele->getError() === false){
124       require (\CoursPHP\Config\Config::getVues()["affichePersonne"]);
125     }else{
126       require (\CoursPHP\Config\Config::getVuesErreur()["default"]);
127     }
128   }
129 }
130 ?>
```

Voici enfin la vue affichant une collection de personnes avec les adresses agrégées (voir figure 13.4) :

Code Source 13.11 : /metierFrontArchi/Vue/vues/vueCollectionPersonneAdmin.php

```php
1  <?=\CoursPHP\Vue\VueHtmlUtils::enTeteHTML5('Bienvenue sur notre site', 'UTF-8',
2                          \CoursPHP\Config\Config::getStyleSheetsURL()['default'])?>
3  <h1>Toutes les personnes</h1>
4
5  <a href="<?=\CoursPHP\Config\Config::getRootURI()?>">Revenir à l'accueil</a>
6  <a href="?action=personne-saisie&idPersonne=<?=
7      \CoursPHP\Metier\Personne::generateRandomId()
8      ?>">Ajouter une personne</a>
9
10 <?php
11     foreach ($modele->getData() as $personne){
12       echo "<div class=\"displayPersonne\">";
13       echo \CoursPHP\Vue\PersonneView::getHtmlDevelopped($personne, true);
```

```
14            echo "</div>";
15        }
16    ?>
17    <?=\CoursPHP\Vue\VueHtmlUtils::finFichierHtml5();?>
```

Cinquième partie

Web Services

Table of Contents

Chapitre 14

API Restful

14.1 Qu'est-ce qu'une *API REST* (ou systèmes *Restful*) ?

L'architecture *REST (representational state transfer)* est, dans notre cadre, une architecture d'application client-serveur, qui permet le lien entre une application côté client en *Javascript* et un serveur web sur lequel s'exécutent des *CGI*.

Le serveur permettra (au moins) d'effectuer au moins les opérations *CRUD* (*Create*, *Read*, *Update*, *Delete*) sur des instances d'objets *métier*, aussi appelées *entités* ou *ressources* :

- **Opération *Create***. De créer une ressource (ici une ligne d'une table de base de données) **avec ou sans son identifiant** unique.
  Exemple 1 : Créer une ressource de type `Adresse` en spécifiant les données de l'adresse, en laissant au serveur le choix de l'*Id* de la ressource créée. Le serveur retourne l'*Id* généré pour que le client le connaisse.
  Exemple 2 : Créer une ressource de type `Adresse` en spécifiant les données de l'adresse **ET** l'identifiant unique de l'instance à créer, par exemple parce que cet *Id* doit être généré par un algorithme dépendant du client, ou parce que cet *Id* doit correspondre à l'*Id* de la même entité ailleurs sur le réseau (comme l'*ISBN* d'un livre, qui ne peut pas être choisi au hasard par le serveur).

- **Opération *Read***. De lire toutes les ressources (ici d'une table de base de données).
  Exemple : Lire toutes les personnes de la table `Personne`, avec une collection d'adresses pour chaque personne (résultat d'une jointure en *SQL* qui correspond à un agrégat sur les objets métiers).

- **Opération *Read* avec *Id* ou prédicat**. De lire **ou bien** une ressource identifiée de manière unique par un identifiant unique (une ligne d'une table de base de données) **ou bien** un certain nombre de ressources données par le résultat d'une requête (comme un `SELECT` en *SQL*) ou par les données d'une jointure (par exemple avec l'identifiant d'un agrégat).
  Exemple 1 : Lire l'adresse d'identifiant unique (clé primaire de la table `Adresse` égaler à `af49bc053de73a0`.
  Exemple 2 : Lire toutes les adresses de la personne d'identifiant unique `bd56bc053de12b3`.
  Exemple 3 : Lire toutes les personnes de la table `Personne` qui ont une adresse avec le code postal commençant par les deux chiffre `63`.

- **Opération *Update***. De mettre à jour une ressource (ici une ligne d'une table de base de données) identifiée de manière unique (par un identifiant unique), avec des données (partielles ou complètes) à modifier.
  Exemple : Modifier le code postal d'une adresse d'identifiant unique égal à `af49bc053de73a0`.

- **Opération *Delete***. De détruire une ressource (ici une ligne d'une table de base de données) identifiée de manière unique (par un identifiant unique) ;
  Exemple : Détruire la personne d'identifiant unique `bd56bc053de12b3`, ainsi que (s'agissant d'une composition) toutes ses adresses de la table adresse (utilisation d'une clé étrangère).

En utilisant cette interface (*service web*), l'application côté client pourra accéder à la couche persistance du serveur.

Nous verrons aussi comment implémenter ces opérations sur le serveur en spécifiant les identifiants et les actions au moyen d'une *URI* (*Universal Ressource Identifier*) et des *verbes* (aussi appelés *méthodes*, `GET`, `PUT`, `POST`, `PATCH` ou `DELETE` du protocole *HTTP* (norme *RFC 2616* puis *RFC 7230*).

Des problèmes de sécurité peuvent se poser. Aussi, les opérations ne sont généralement pas toutes acessibles à un même utilisateur. La sécurité des *Web Service* se fonde sur des protocoles d'*Autorisation sur des Domaines de données et des verbes*, et non pas d'identification de l'utilisateur. Le standard *OAuth 2.0* s'appuie sur la notion de *fournisseur de service* et de *fournisseur d'identité*, qui communiquent par un système de redirection *HTTP*. La sécurité des *Web Service* n'est pour le moment pas traitée dans ce cours, et nos exemples d'*API* sont ouvertes à tous vents.

14.2 Les Points d'Entrée d'une *API Restful*

14.2.1 Qu'est-ce qu'un *Points d'Entrée*

Les *Points d'Entrée* d'une *API Restful* sont définis par des *Uniform Resource Identifier* (*URI*) qui identifient des sortes d'entités, des *verbes*, qui définissent des actions obéissant à une sémantique, et des paramètres (comme, typiquement, des identifiants de ressources, ou des attributs d'entités).

Ces points d'entrée caractérisent les opérations qui peuvent être réalisées sur le *Web Service*. Les points d'entrée sont en principe documentés formellement par des *Schémas* (*Schémas XML*, *Schémas JSON*, etc.) auxquels on accède avec le verbe *HTTP* dedié *OPTIONS* sur les *URI* des points d'entrée.

14.2.2 Ce que dit le Protocole *HTTP*

Le protocole *HTTP* prévoit un certain nombre de *méthodes*, aussi appelées *verbes*, qui produisent des actions différentes sur une même *URI*. Notons :

- L'implémentation d'une méthode est dite *idempotente* sur l'application plusieurs fois de la même requête (même verbe, même *URI*, et même paramètres) laisse le serveur dans le même état qu'une seule application de la requête.

- Une méthode est dite *safe* (sans effets de bord) si une requête suivant ce verbe ne modifie pas l'état du serveur.

Voici une liste (non exhaustive) des verbes *HTTP*. Les usages mentionnés sont liés aux *Web Services Restful*, mais certaines contraintes sont générales aux recommandations *RFC* concernant *HTTP*.

- `GET` Permet uniquement d'accéder aux données et ne doit pas modifier l'état du serveur (méthode dite *safe*).
- `HEAD` Doit retourner le même en-tête *HTTP* que `GET`, mais aucun *Body*. Ceci permet de consulter le type de sortie obtenue par la méthode `GET`, comme le format du *Body* (par exemple `application:json;charset=utf-8`)), sans avoir à transporter toutes les données sur le réseau.
- `POST` Typiquement utilisée pour créer une nouvelle entité. Elle doit accepter la création de données, et peut être idempotente ou non, suivant le choix du serveur. Une implémentation idempotente permet de simplifier le cas où un client soumet deux fois les mêmes données de formulaire, sans créer deux entités. On notera par exemple l'emploi de la requête *SQL* `REPLACE` dans les méthodes *update* des classes *Gateway* de la *DAL* (voir la partie 9.3).
- `PUT` Typiquement utilisée pour modifier une entité existante, ou éventuellement pour créer une nouvelle entité. Cette méthode **doit être idempotente**. Dans notre implémentation de la *DAL* (voir par exemple la partie 9.3), la méthode rejette une erreur 404 (ressource introuvable) si aucune ressource avec le même *ID* n'est connue du serveur. Il s'agit d'un choix.
- `DELETE` Supprime une ressource spécifiée par son *ID*)
- `OPTIONS` Retourne un schéma définissant les méthodes disponibles sur un point d'entrée, avec éventuellement la spécification des types (simples ou complexes) de paramètres et le type de la sortie.

Il n'est généralement pas correct d'identifier deux à deux des verbes comme `GET`, `PUT`, `POST` ou `DELETE` avec les opérations *CRUD* (*Create*, *Read*, *Update*, *Delete*). La correspondance éventuelle dépend de l'implémentation est ne permet pas de caractériser les *API Restful*.

14.2.3 Parser les Données Issues de la Requête *HTTP*

En *PHP*, les données de l'en-tête *HTTP* sont accessible dans le tableau *super-global* `$_SERVER` (par exemple : `$_SERVER['REQUEST_METHOD']` permet d'obtenir le verbe).

Les données du *Body*, qui correspondent à un flot (similaire à un fichier) peuvent être obtenu via le flot `"php://input"`.

Nous présentons maintenant une classe qui permet de *parser*, puis de rendre accessible, les données d'une requête, en supposant que les identifiants sont des nombres hexadécimaux, et que les noms des entités (dans nos exemples `adresse` et `personne` ne sont pas des nombres hexadécimaux.

14.2.3.a Ré-écriture d'*URL* par le serveur

Une redirection *HTTP* par ré-écriture d'*URL* sera d'abord appliquée par configuration du serveur, pour transformer l'*URI* globale, par exemple :
`http://progweb/exemples/apiResful/personne/027ad64fab`
en une *URL* d'un script *PHP* (implémentation du *Web Service* en *PHP*), et une paramètre d'entrée donnant séparément la partie de l'*URI* spécifiant le point d'entrée (par exemple `/personne/027ad64fab`). Voici par exemple la mise en oeuvre par `.htaccess` :

Code Source 14.1 : exemples/apiRestful/.htaccess

```
<IfModule mod_rewrite.c>
RewriteEngine On
RewriteBase "/exemples/apiRestful
RewriteCond %{REQUEST_URI} !-f
RewriteCond %{REQUEST_URI} !-d
RewriteRule "^(.*)$" "index.php?request_uri=$1" [QSA,NC,L]
</IfModule>
```

14.2.3.b Classe Récupérant les données de la Requête *HTTP*

La classe `ParseHttpRequest` suivante va permettre de *parser* les données de la nouvelle requête *HTTP* (après redirection), pour obtenir, dans des attributs, les actions du contrôleur, recensées dans notre *Front Controller* (voir la partie 14.4.1) et qui tiennent compte du verbe *HTTP* et de la spécification du point d'entrée (opération *CRUD* sur un type d'entité). La classe `ParseHttpRequest` va aussi *parser* le *Body* pour obtenir tous les paramètres de la requête *HTTP*.

Code Source 14.2 : /apiRestful/Config/ParseHttpRequest.php

```
<?php
namespace CoursPHP\Config;

class ParseHttpRequest {

  private $action;

  private $parsedBodyData;

  private $headVerb;

  public function getAction(){
    return $this->action;
  }

  public function getParsedBodyData(){
    return $this->parsedBodyData;
  }

  public function isHeadVerb(){
    return $this->headVerb;
  }

  public function __construct($requestURI, &$httpStatusCode){
    $this->action = "";
```

```
26        $this->error = array();
27        $this->headVerb = false;
28
29        // On récupère le contenu du body du message HTTP sous forme de chaîne.
30        $bodyDataString = file_get_contents('php://input');
31        if ($bodyDataString === false){
32          $httpStatusCode = 400; // "Bad Request"
33          return;
34        }
35        // On suppose les caractères spéciaux codés avec "x-www-form-urlencoded"
36        $bodyDataStringDecoded = urldecode($bodyDataString);
37
38        // On parse les données du body pour créer de tableaux associatifs :
39        // Données de la requête de forme semblable au tableaux $_GET, $_POST, etc.
40        $this->parsedBodyData = array();
41        parse_str($bodyDataStringDecoded, $this->parsedBodyData);
42
43        // On analyse l'URI et on ajoute les éventuels IDs au tableau
44        // des données de la requête.
45
46        // On met d'abord l'URI sous forme standard :
47        $argumentsURI = explode("/", strtolower(trim($requestURI, "/")));
48        $currentEntity = "";
49        foreach ($argumentsURI as $arg){
50          // S'il s'agit d'un ID en hexa (différent de 'adresse', 'personne'...)
51          if (preg_match("/^[0-9a-fA-F]{1,}$/", $arg)){
52            // Par exemple parsedBodyData['adresse']['idAdresse']
53            $this->parsedBodyData[$currentEntity]['id'.$currentEntityUpCaseInit]
54                                = $arg;
55          }else{
56            // Il doit s'agir d'un nom d'entité ('adresse', 'personne'...)
57            // L'entité préfixe l'action (exemple : action = 'personne-get-all')
58            $this->action .= $arg."-";
59            $currentEntity = $arg;
60            // 'adresse' ---> 'Adresse'
61            $currentEntityUpCaseInit = strtoupper(substr($currentEntity, 0, 1))
62                                              .substr($currentEntity, 1);
63          }
64        }
65
66        // On détermine l'action ('personne-get-all', 'adresse-update, etc.)
67        // suivant le verbe HTTP et l'URL (présence ou non d'un ID...)
68        $method = strtolower($_SERVER['REQUEST_METHOD']);
69        switch ($method){
70          case "head":
71            $this->headVerb = true;
72            // let fall through to the "get" case:
73          case "get":
74            $this->action .= "get";
75            // Si aucun ID n'a été spécifié et (par exemple) entity="adresse",
76            // alors action="adresse-get-all"
77            if (empty($this->parsedBodyData)){
78              $this->action .= "-all";
79            }
80            break;
81          case "post": // Doit pouvoir créer une entité
```

```
82          $this->action .= "create";
83          break;
84        case "put": // Doit être idempotent
85          $this->action .= "update";
86          break;
87        case "delete":
88          $this->action .= "delete";
89          break;
90        case "options":
91            $this->action .= "options";
92          break;
93        default:
94          $httpStatusCode = 405; // "Method Not Allowed"
95      }
96
97    }
98  }
99  ?>
```

14.2.3.c Fichier d'Index Construisant le Contrôleur

Le fichier source *PHP* d'index permet de construire l'instance de `ParseHttpRequest` et de construire l'instance du *Front Controller* (partie 14.4.1). Le fichier d'index définit aussi une variable globale `$httpStatusCode`, par défaut égalà 200, destinée à contenir un éventuel code d'erreur *HTTP* (voir la partie 14.3.2).

Code Source 14.3 : /apiRestful/index.php

```
1  <?php
2    // Répertoire racine du MVC
3    $rootDirectory = dirname(__FILE__)."/";
4
5    // Calcul portable de l'URI de la racine du MVC (sans la query string)
6    // 1) On enlève la "query string" : ?action=blabla&id=03456789
7    $urlWithoutQueryString = explode("?", $_SERVER['REQUEST_URI'])[0];
8    // 2) On coupe l'URL du script au niveau de l'extension ".php"
9    $scriptWithoutExtention = explode(".php", $urlWithoutQueryString)[0];
10   // 3) puis on s'arrête au dernier slash (pour enlever la basename du script)
11   $longueurRootURI = strrpos($scriptWithoutExtention, '/');
12   // 4) On prend le début de l'URL en coupant à la bonne longueur
13   $rootURI = substr($_SERVER['REQUEST_URI'], 0, $longueurRootURI);
14
15   // chargement de l'autoload pour autochargement des classes
16   require_once($rootDirectory.'/Config/Autoload.php');
17   CoursPHP\Config\Autoload::load_PSR_4('CoursPHP\\');
18
19   $requestURI = (isset($_GET['request_uri']) ? $_GET['request_uri'] : "");
20   $httpStatusCode = 200; // "OK" : Status Code par défaut
21   // On parse l'URI de la requête HTTP et le "Body" du message HTTP
22   // on définit l'action du contrôleur et les données en entrée
23   // Status Code = 200 (OK) ou 400 (Bad Request) ou 405 (Method Not Allowed)
24   $httpRequestData = new \CoursPHP\Config\ParseHttpRequest($requestURI,
25                                                            $httpStatusCode);
26   // Création de l'instance du contrôleur (voir ControleurFront.php)
27   $ctrl = new \CoursPHP\Controleur\ControleurFront($httpRequestData->getAction()
         ,
```

```
28                                                  $httpStatusCode);
29 ?>
```

14.3 La Sortie de l'*API* (Cas du format *JSON*) et *Status Codes*

14.3.1 Génération des données *JSON* représentant le modèle

La classe `Config` définit les *URL* des fichiers de génération de *JSON*, pour éviter les *URL* en dûr, comme pour les vues d'un *CGI*.

Code Source 14.4 : /apiRestful/Config/Config.php

```
1  <?php
2  namespace CoursPHP\Config;
3  /** @brief Classe de configuration
4   * Donne accès aux paramères spécifiques concernant l'application
5   * telles que les chemins vers les vues, les vues d'erreur,
6   * les hash pour les ID de sessions, etc. */
7  class Config
8  {
9    /** @brief Données nécessaires à la connexion à la base de données.
10    * Les valeurs pourraient être initialisées à partir d'un
11    * fichier de configuration séparé (require('configuration.php'))
12    * pour faciliter la maintenance par le webmaster.
13    */
14   public static function getAuthData(&$db_host, &$db_name,
15                                       &$db_user, &$db_password){
16     $db_host="mysql:host=localhost;";
17     $db_name="dbname=ExempleCompositionBD_js";
18     $db_user="remy";
19     $db_password="my_password";
20   }
21
22   /** @return Le préfixe commun aux tables de la BD de l'application */
23   public static function getTablesPrefix(){
24     return "web_";
25   }
26
27   /**
28    * @brief retourne le tableau des (chemins vers les) fichiers
29    * de génération de JSON.
30    */
31   public static function getJsonOutput(){
32       // Racine du site
33       global $rootDirectory;
34       // Répertoire contenant les fichiers de génération de JSON
35       $jsonDirectory = $rootDirectory."Json/jsonModels/";
36       return array(
37             "collectionPersonne" => $jsonDirectory."jsonCollectionPersonne.php",
38             "instancePersonne" => $jsonDirectory."jsonInstancePersonne.php",
39             "collectionAdresse" => $jsonDirectory."jsonCollectionAdresse.php",
40             "instanceAdresse" => $jsonDirectory."jsonInstanceAdresse.php",
41             "success" => $jsonDirectory."jsonSuccess.php",
```

```
42                "errorHandled" => $jsonDirectory."jsonErrorHandled.php",
43                "errorDebug" => $jsonDirectory."jsonErrorDebug.php",
44                "documentationPersonne" => $jsonDirectory."jsonDocumentationPersonne
                      .php",
45                "documentationAdresse" => $jsonDirectory."jsonDocumentationAdresse.
                      php"
46            );
47    }
48 }
49 ?>
```

Pour chaque classe métier, un utilitaire permet de convertir les instances, ou les collections d'instances, en tableaux associatifs.

Code Source 14.5 : /apiRestful/Json/AdresseJsonUtils.php

```
1  <?php
2  namespace CoursPHP\Json;
3  /** @brief Implémente la conversion d'instances ( et de collections) d'Adresse
4    * vers des données sous la forme de tableaux associatifs
5    * dans le but de générer un codage JSON de ces données
6    * (par exemple avec la fonction json_encode()) */
7  class AdresseJsonUtils {
8    /** @brief retourne une représentation des attributs d'une instance
9     *        sous forme de tableau associatif.
10    * @param adresse un instance d'Adresse à convertir
11    * @return la représentation des données sous forme d'array.    */
12   public static function instanceToArray($adresse){
13     $arrayData = array(
14       "id" => $adresse->idAdresse,
15       "numeroRue" => $adresse->numeroRue,
16       "rue" => $adresse->rue,
17       "complementAddr" => $adresse->complementAddr,
18       "codePostal" => $adresse->codePostal,
19       "ville" => $adresse->ville,
20       "pays" => $adresse->pays
21     );
22     return $arrayData;
23   }
24
25   /** @brief retourne une représentation d'une collection d'instances
26    *        sous forme de tableau associatif.
27    * @param collectionAdresse un collection d'Adresse(s) à convertir
28    * @return la représentation des données sous forme d'array. */
29   public static function collectionToArray($collectionAdresses){
30     $arrayData = array();
31     foreach ($collectionAdresses as $adresse){
32       // Ajout d'un élément au tableau
33       $arrayData[] = self::instanceToArray($adresse);
34     }
35     return $arrayData;
36   }
37 } // end of class AdresseJsonUtils
38 ?>
```

Code Source 14.6 : /apiRestful/Json/PersonneJsonUtils.php

```php
<?php
namespace CoursPHP\Json;
/** @brief Implémente la conversion d'instances ( et de collections) de Personne
  * vers des données sous la forme de tableaux associatifs
  * dans le but de générer un codage JSON de ces données
  * (par exemple avec la fonction json_encode()) */
class PersonneJsonUtils {
  /** @brief retourne une représentation des attributs d'une instance
   *        sous forme de tableau associatif.
   * @param adresse un instance de Personne à convertir
   * @return la représentation des données sous forme d'array. */
  public static function instanceToArray($personne){
    $arrayData = array(
      "id" => $personne->idPersonne,
      "nom" => $personne->nom,
      "adresses" => AdresseJsonUtils
                    ::collectionToArray($personne->getAdresses())
    );
    return $arrayData;
  }

  /** @brief retourne une représentation d'une collection d'instances
   *        sous forme de tableau associatif.
   * @param collectionAdresse un collection de Personne(s) à convertir
   * @return la représentation des données sous forme d'array. */
  public static function collectionToArray($collectionPersonnes){
    $arrayData = array();
    foreach ($collectionPersonnes as $personne){
      // Ajout d'un élément au tableau
      $arrayData[] = self::instanceToArray($personne);
    }
    return $arrayData;
  }
}
?>
```

À la place des vues dans un *CGI*, un fichier génère les données *JSON* correspondant au modèle de la réponse (ici une collection de personnes).

Code Source 14.7 : /apiRestful/Json/jsonModels/jsonCollectionPersonne.php

```php
<?php
  header('content-type : application/json; charset=utf-8');
  http_response_code(200); // 200 OK
  // Output body si le verbe est différent de HEAD
  if (!$httpRequestData->isHeadVerb()) {
    $arrayData = array("error" => null,
                       "data" => \CoursPHP\Json\PersonneJsonUtils
                                 ::collectionToArray($modele->getData()));
    echo json_encode($arrayData);
  }
?>
```

14.3.2 Gestion des Erreurs et *Status Codes*

Le protocole *HTTP* spécifie un certain nombre de codes sur le statut de la requête (*Status Code*, qui sont retournés au client avec la réponse. Ce sont des codes d'erreurs dont la signification, au moins dans notre implémentation, est assez transparente :

- **200 OK** : Comportement par défaut en cas de succès de la requête et réponse normale.
- **400 Bad Request** : si le *Body* est mal formé et ne peut être lu.
- **405 Method Not Allowed** : Si la méthode spécifiée n'est pas disponible sur le point d'entrée considéré (ne pas confondre avec *401 Unauthorized*, qui n'est pas gérée dans notre exemple et concerne les droits d'accès).
- **404 Not Found** : si l'*ID* d'une ressource spécifiée dans les paramètres et qui est requise (méthode `GET` ou, dans notre implémentation, méthode *PUT* correspondant à une opération *Update*) n'existe pas.
- **422 Unprocessable Entity** : Si erreur sur la forme des attributs est détectée (non respect de la logique métier ; voir la partie 14.3.3).
- **500 Internal Server Error** : Si erreur inprévue se produit (par exemple, le serveur de base de données est inaccessible).

Un fichier spécifique permet de renvoyer vers le client les messages correspondant aux erreurs détectées par le serveur (erreurs d'accès au serveur de base de données, données de forme incorrecte, etc.)

Code Source 14.8 : /apiRestful/Json/jsonModels/jsonErrorHandled.php

```php
<?php
  // On retourne le tableau associatif des erreurs
  header('content-type : application/json ; charset=utf-8');
  http_response_code($httpStatusCode);
  // Output body si le verbe est différent de HEAD
  if (!$httpRequestData->isHeadVerb()) {
    echo json_encode(array("error" => $modele->getError(),
                           "data" => array()));
  }
?>
```

Un autre fichier permet, dans le cas où aucune donnée n'est attendue du client (comme par exemple la suppression d'une personne) d'indiquer qu'aucune erreur n'a été détectée.

Code Source 14.9 : /apiRestful/Json/jsonModels/jsonSuccess.php

```php
<?php
  // On retourne une erreur null et un objet dada vide
  header('content-type : application/json ; charset=utf-8');
  http_response_code(200); // 200 OK
  // Output body si le verbe est différent de HEAD
  if (!$httpRequestData->isHeadVerb()) {
    echo json_encode(array("error" => null, "data" => array()));
  }
?>
```

14.3.3 Documentation via la Méthode `OPTIONS`

Voici par exemple la sortie *JSON* d'une requête avec la méthode `OPTIONS` sur notre point d'entrée `personne` :

Code Source 14.10 :

```
{
  "$schema": "http://progjs/exemples/clientAndAPI/api/personne#",
  "POST": {
    "description": "Create a person",
    "parameters": {
      "idPersonne": {
        "description": "Person's unique ID.",
        "type": "string",
        "pattern": "^[0-9a-f]{10}$",
        "required": true
      },
      "nom": {
        "description": "Person's name.",
        "type": "string",
        "pattern": "^[0-9a-zA-ZÀÁÂÃÄÅÆÇÈÉÊËÌÍÎÏÐÑÒÓÔÕÖ×ØÙÚÛÜÝÞßàáâãäåæçèéêëìíî
            ïðñòóôõö÷øùúûÃãüýþÿ \"\\'\\-\\ ]{1-50}$",
        "required": true
      }
    },
    "output": "{\"error\":null, \"data\":[]}",
    "idempotent": "yes (Cause I like it...)"
  },
  "PUT": {
    "parameters": {
      "description": "Update an EXISTING person (ID must be present)",
      "idPersonne": {
        "description": "Person's unique ID.",
        "type": "string",
        "pattern": "^[0-9a-f]{10}$",
        "required": true
      },
      "nom": {
        "description": "Person's name.",
        "type": "string",
        "required": true
      }
    },
    "output": "{\"error\":null, \"data\":[]}",
    "idempotent": "yes (HTTP Mandate)"
  },
  "GET": {
    "description": "Retrieve people",
    "parameters": {
      "idPersonne": {
        "description": "Person's unique ID.",
        "type": "string",
        "pattern": "^[0-9a-f]{10}$",
        "required": false
      },
    },
```

```
50      "output": "{\"error\":null, \"data\":{type:[Personne, array, items:{type:
            Personne}]}}",
51      "safe": "yes (HTTP Mandate)"
52    },
53    "DELETE": {
54      "description": "Delete a Person (if exists)",
55      "parameters": {
56        "idPersonne": {
57          "description": "Person's unique ID.",
58          "type": "string",
59          "pattern": "^[0-9a-f]{10}$",
60          "required": true
61        }
62      },
63      "output": "{\"error\":null, \"data\":[]}",
64      "idempotent": "yes (HTTP Mandate)"
65    }
66  }
```

14.4 L'implémentation des Actions

14.4.1 Le *Front Controller*

Le *Front Controller* nous permet d'identifier l'action, de déterminer si l'utilisateur a des droits suffisants pour exécuter l'action, et d'appeler, en tenant compte du rôle de l'utilisateur et de l'action, le contrôleur dédié qui va implémenter l'action. La gestion des erreurs (comme l'action non définie) sera vue dans la partie 14.3.1.

Code Source 14.11 : /apiRestful/Controleur/ControleurFront.php

```
1  <?php
2  namespace CoursPHP\Controleur;
3  /**
4   * @brief Identifie l'action et le rôle de l'utilisateur.
5   * Dans le cas où l'utilisateur a des droits insuffisants pour l'action,
6   * le ControleurFront affiche une vue d'autentification ou un vue d'erreur.
7   * Sinon, ControleurFront instancie le contrôleur adapté pour les rôle et action
8   * Il gère aussi les exceptions et appelle le cas échéant une vue d'erreur.
9   */
10 class ControleurFront {
11   /**
12    * @brief C'est dans le contructeur que le contrôleur fait son travail.
13    * @param action action to be performed
14    */
15   function __construct($action, &$httpStatusCode) {
16     try{
17       // Si une erreur a déjà été détectée sur le Verbe ou le Body HTTP
18       if ($httpStatusCode != 200) {
19         $modele = new \CoursPHP\Modele\Model(
20                   array('input data' => "Verbe ou Body HTTP incorrect"));
21         require (\CoursPHP\Config\Config::getJsonOutput()["errorHandled"]);
22         return;
23       }
24       // On distingue des cas d'utilisation, suivant l'action
```

```
25          switch($action){
26            case "adresse-update": // Mise à jour d'une Adresse dans la BD
27            case "adresse-create": // Création d'une nouvelle Adresse dans la BD
28            case "adresse-delete": // Supression d'une Adresse à partir de son ID
29              $adminCtrl = new \CoursPHP\Controleur\ControleurAdminAdresse(
30                                                                    $action);
31              break;
32            case "personne-update": // Mise à jour d'une Personne dans la BD
33            case "personne-create": // Creation d'une nouvelle Personne dans la BD
34            case "personne-delete": // Supression d'une Personne à partir de son ID
35              $adminCtrl = new \CoursPHP\Controleur\ControleurAdminPersonne(
36                                                                    $action);
37              break;
38            case "personne-get-all": // Accès à toutes les Personne's
39            case "personne-get": // Accès à une Personne à partir de son ID
40              $publicCtrl = new \CoursPHP\Controleur\ControleurVisitorPersonne(
41                                                                      $action);
42              break;
43            case "adresse-get-all": // Accès à toutes les Adresses
44            case "adresse-get": // Accès à une Adresse à partir de son ID
45              $publicCtrl = new \CoursPHP\Controleur\ControleurVisitorAdresse(
46                                                                     $action);
47              break;
48            // Documentation des Points d'Entrée (end-points)
49            case "personne-options": // Documentation du type Personne (Schéma JSON)
50              require (\CoursPHP\Config\Config
51                                  ::getJsonOutput()["documentationPersonne"]);
52              break;
53            case "adresse-options": // Documentation du type Adresse (Schéma JSON)
54              require (\CoursPHP\Config\Config
55                                  ::getJsonOutput()["documentationAdresse"]);
56              break;
57            default:
58              $httpStatusCode = 422;  // "Unprocessable Entity"
59              $modele = new \CoursPHP\Modele\Model(array(
60                    'action' => "Action \"".$action
61                                ."\" non définie (ressource(s) introuvables)"));
62              require (\CoursPHP\Config\Config::getJsonOutput()["errorHandled"]);
63          }
64        }catch (\Exception $e){ // Page d'erreur par défaut
65            // SI l'exception ne renvoie pas un code d'erreur standard :
66            if (!preg_match("/^[1-5]{1}[0-9]{2}$/", $e->getMessage())){
67              $httpStatusCode = 500; // "Internal Server Error"
68            }else{
69              $httpStatusCode = intval($e->getMessage());
70            }
71            $modele = new \CoursPHP\Modele\Model(
72                                              array('persistance' => $e->getMessage())
                                                  );
73            require (\CoursPHP\Config\Config::getJsonOutput()["errorHandled"]);
74        }
75      }
76  }
77  ?>
```

14.4.2 Implémentation des actions des contrôleurs

Code Source 14.12 : /apiRestful/Controleur/ControleurVisitorPersonne.php

```php
<?php
namespace CoursPHP\Controleur;
/**
 * @brief Identifie l'action concernant des Personne avec le rôle admin
 * et appelle la méthode pour construire le modèle correspondant à l'action
 * avec le rôle "admin". Le controleur appelle aussi la vue correspondante.
 * Il ne gère pas les exceptions, qui remontent au Front Controller.
 */
class ControleurVisitorPersonne {
  /**
   * @brief C'est dans le contructeur que le contrôleur fait son travail.
   */
  function __construct($action) {
    // On distingue des cas d'utilisation, suivant l'action
    switch($action){
      case "personne-get": // Affichage d'une Personne à partir de son ID
        $this->actionGet();
        break;
      case "personne-get-all": // Affichage de toutes les Personne's
        $this->actionGetAll();
        break;
      default: // L'action indéfinie (page par défaut, ici accueil)
        require (\CoursPHP\Config\Config::getJsonModel()["default"]);
        break;
    }
  }

  /** @brief Implemente l'action "get-all" (récupère toutes les instances) */
  private function actionGetAll(){
    global $httpRequestData; // Pour utilisation dans la sortie JSON
    $modele = \CoursPHP\Modele\ModelCollectionPersonne::getModelPersonneAll();
    if ($modele->getError() === false){
      require (\CoursPHP\Config\Config::getJsonOutput()["collectionPersonne"]);
    }else{
      require (\CoursPHP\Config\Config::getJsonOutput()["errorHandled"]);
    }
  }

  /** @brief Implemente l'action "get" (récupère une instance à partir de ID) */
  private function actionGet(){
    // ID de l'instance à récupérer
    global $httpRequestData;
    $idPersonne = isset($httpRequestData->getParsedBodyData()['personne']['
        idPersonne']) ?
                  $httpRequestData->getParsedBodyData()['personne']['
                      idPersonne'] : "";

    $modele = \CoursPHP\Modele\ModelPersonne::getModelPersonne($idPersonne);
    if ($modele->getError() === false){
      require (\CoursPHP\Config\Config::getJsonOutput()["instancePersonne"]);
    }else{
      require (\CoursPHP\Config\Config::getJsonOutput()["errorHandled"]);
    }
```

```
52    }
53 }
54 ?>
```

Code Source 14.13 : /apiRestful/Controleur/ControleurAdminPersonne.php

```
1  <?php
2  namespace CoursPHP\Controleur;
3  /** @brief Identifie l'action concernant des Personne avec le rôle admin
4   * et appelle la méthode pour construire le modèle correspondant à l'action
5   * avec le rôle "admin". Le controleur appelle aussi la vue correspondante.
6   * Il ne gère pas les exceptions, qui remontent au Front Controller */
7  class ControleurAdminPersonne {
8    /**
9     * @brief C'est dans le contructeur que le contrôleur fait son travail.
10    */
11   function __construct($action) {
12     // On distingue des cas d'utilisation, suivant l'action
13     switch($action){
14       case "personne-update": // Met à jour une Adresse dans la BD
15         $this->actionUpdate();
16         break;
17       case "personne-create": // Cration d'une nouvelle Adresse dans la BD
18         $this->actionCreate();
19         break;
20       case "personne-delete": // Supression d'une Adresse à partir de son ID
21       $this->actionDelete();
22         break;
23       default: // L'action indéfinie
24         $modele = new \CoursPHP\ModeleModel(array(
25                 'action' => "Action non définie (ressource(s) introuvables)"));
26         require (\CoursPHP\Config\Config::getJsonOutput()["errorHandled"]);
27       break;
28     }
29   }
30
31   /** @brief Implemente l'action "update" (met à jour l'instance dans la BD) */
32   private function actionUpdate(){
33     // Construire le modèle de Personne mise à jour à partir du JSON
34     global $httpRequestData;
35     $modele = \CoursPHP\Modele\ModelPersonne
36                                 ::getModelPersonneUpdate(
37                                     $httpRequestData->getParsedBodyData()['
                                         personne']
38                                   );
39     if ($modele->getError() === false){
40       require (\CoursPHP\Config\Config::getJsonOutput()["success"]);
41     }else{ // Problème de forme des attributs
42       $httpStatusCode = 422; // "Unprocessable Entity"
43       require (\CoursPHP\Config\Config::getJsonOutput()["errorHandled"]);
44     }
45   }
46
47   /** @brief Implemente l'action "create" (crée une instance dans la BD) */
48   private function actionCreate(){
49     // Construire le modèle de Personne mise à jour à partir du JSON
```

```
50      global $httpRequestData;
51
52      $modele = \CoursPHP\Modele\ModelPersonne
53                      ::getModelPersonneCreate(
54                          $httpRequestData->getParsedBodyData()['personne']
55                      );
56      if ($modele->getError() === false){
57        require (\CoursPHP\Config\Config::getJsonOutput()["success"]);
58      }else{ // Problème de forme des attributs
59        $httpStatusCode = 422; // "Unprocessable Entity"
60        require (\CoursPHP\Config\Config::getJsonOutput()["errorHandled"]);
61      }
62    }
63
64    /** @brief Implemente l'action "delete" (supprime une instance via son ID) */
65    private function actionDelete(){
66      // ID de l'instance à supprimer
67      global $httpRequestData;
68      $idPersonne = isset($httpRequestData->getParsedBodyData()['personne']['
            idPersonne']) ?
69                      $httpRequestData->getParsedBodyData()['personne']['
                          idPersonne'] : "";
70
71      $modele = \CoursPHP\Modele\ModelPersonne::deletePersonne($idPersonne);
72
73      require (\CoursPHP\Config\Config::getJsonOutput()["success"]);
74    }
75
76  }
77  ?>
```

Code Source 14.14 : /apiRestful/Controleur/ControleurAdminAdresse.php

```
1   <?php
2   namespace CoursPHP\Controleur;
3   /** @brief Identifie l'action concernant des Adresse avec le rôle admin
4    * et appelle la méthode pour construire le modèle correspondant à l'action
5    * avec le rôle "admin". Le controleur appelle aussi la vue correspondante.
6    * Il ne gère pas les exceptions, qui remontent au Front Controller. */
7   class ControleurAdminAdresse {
8     /**
9      * @brief C'est dans le contructeur que le contrôleur fait son travail.
10     */
11    function __construct($action) {
12      // On distingue des cas d'utilisation, suivant l'action
13      switch($action){
14        case "adresse-update": // Met à jour une Adresse dans la BD
15          $this->actionUpdate();
16          break;
17        case "adresse-create": // Cration d'une nouvelle Adresse dans la BD
18          $this->actionCreate();
19          break;
20        case "adresse-delete": // Supression d'une Adresse à partir de son ID
21        $this->actionDelete();
22          break;
23        default: // L'action indéfinie
```

```
24            $modele = new \CoursPHP\ModeleModel(array(
25              'action' => "Action non définie (ressource(s) introuvables)"));
26            require (\CoursPHP\Config\Config::getJsonOutput()["errorHandled"]);
27          break;
28        }
29      }
30
31      /** @brief Implemente l'action "update" (met à jour l'instance dans la BD) */
32      private function actionUpdate(){
33        // Construire le modèle d'Adresse mise à jour à partir du JSON
34        global $httpRequestData;
35
36        $modele = \CoursPHP\Modele\ModelAdresse
37                                  ::getModelAdresseUpdate(
38                                      $httpRequestData->getParsedBodyData()['adresse'
                                          ']
39                                    );
40        if ($modele->getError() === false){
41          require \CoursPHP\Config\Config::getJsonOutput()["success"];
42        }else{ // Problème de forme des attributs
43          $httpStatusCode = 422; // "Unprocessable Entity"
44          require (\CoursPHP\Config\Config::getJsonOutput()["errorHandled"]);
45        }
46      }
47
48      /** @brief Implemente l'action "create" (crée une instance dans la BD) */
49      private function actionCreate(){
50        // Construire le modèle d'Adresse créée à partir du JSON
51        global $httpRequestData;
52        $modele = \CoursPHP\Modele\ModelAdresse
53                              ::getModelAdresseCreate(
54                                    $httpRequestData->getParsedBodyData()['adresse']
55                                );
56        if ($modele->getError() === false){
57          require (\CoursPHP\Config\Config::getJsonOutput()["success"]);
58        }else{ // Problème de forme des attributs
59          $httpStatusCode = 422; // "Unprocessable Entity"
60          require (\CoursPHP\Config\Config::getJsonOutput()["errorHandled"]);
61        }
62      }
63
64      /** @brief Implemente l'action "delete" (supprime une instance via son ID) */
65      private function actionDelete(){
66        // ID de l'instance à supprimer créée à partir du JSON
67        global $httpRequestData;
68        $idAdresse = isset($httpRequestData->getParsedBodyData()['adresse']['
            idAdresse']) ?
69                    $httpRequestData->getParsedBodyData()['adresse']['idAdresse'] :
                          "";
70
71        $modele = \CoursPHP\Modele\ModelAdresse::deleteAdresse($idAdresse);
72        require (\CoursPHP\Config\Config::getJsonOutput()["success"]);
73      }
74    }
75    ?>
```