

Programmation Web

Coté serveur : *PHP, PDO, MVC, DAL, Front Controller*

Rémy Malgouyres
LIMOS UMR 6158, IUT, département info
Université Clermont 1
B.P. 86
63172 AUBIERE cedex
<http://malgouyres.org/>

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

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

<http://malgouyres.org/programmation-html-css>

Tutoriel sur le *CMS Drupal* :

<http://malgouyres.org/tutoriel-drupal>

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

<http://malgouyres.org/programmation-php>

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

<http://malgouyres.org/programmation-javascript>

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

<http://malgouyres.org/administration-reseau>

Table des matières

Table des matières	1
I Bases du langage <i>PHP</i>	4
1 PHP procédural	7
1.1 Notion de <i>CGI</i>	7
1.2 Générer du code <i>HTML</i> avec un <i>CGI</i> en <i>PHP</i>	8
1.3 Exemple de fonction en <i>PHP</i>	9
1.4 Inclure un fichier <i>PHP</i> dans un autre	10
1.5 Arithmétique : types <code>int</code> et <code>float</code>	11
1.6 Tableaux indexés : avec une clé de type <code>int</code>	12
1.7 Tableaux associatifs : avec une clé de type <code>String</code>	13
1.8 Passage de paramètre à un script <i>PHP</i>	15
1.9 Variables Locales ou Globales, Références	18
2 Les classes en <i>PHP</i>	21
2.1 Conception Objet, Modularité et Interopérabilité	21
2.2 Exemples de classes <i>PHP</i>	23
2.3 Validation en entrée et gestion d'une exception	31
2.4 Classe <code>Employe</code> héritant de la classe <code>Personne</code>	41
II Formulaires et Filtrage des Données Utilisateur	46
3 Formulaires <i>HTML/PHP</i>	49
3.1 Formulaires <i>HTML</i>	49
3.2 Exemple de style <i>CSS</i> pour formulaire	50
3.3 Validation pour la sécurité : Appel de <code>filter_var</code>	55
3.4 Appel des vues	57
3.5 Tableaux <code>\$_POST</code> <code>\$_GET</code> <code>\$_REQUEST</code>	58
3.6 Formulaires dynamiques an javascript	60
4 Injection, Filtrage, Expressions Régulières	63
4.1 Injections <i>HTML</i> et échappement	63
4.2 Injections <i>SQL</i>	70
4.3 La fonction <code>filter_var</code>	76
4.4 Expressions régulières	80

5	Formulaires <i>PHP/HTML</i>, filtrage, exceptions	82
5.1	Modélisation : Diagrammes de Classes	82
5.2	La Classe <i>Adresse</i>	84
5.3	Filtrage des attributs	85
5.4	Fabrique d' <i>Adresse</i>	90
5.5	Génération de formulaires et classe <i>AdresseFormView</i>	91
5.6	Enchaînement de la saisie à la vue	97
III	Persistence	102
6	Cookies	107
6.1	Création d'un <i>cookie</i>	107
6.2	Récupération d'un <i>cookie</i>	109
6.3	Suppression d'un <i>cookie</i>	110
6.4	Mise à jour d'un <i>cookie</i>	111
7	Sessions	112
7.1	Concept de Session et Problèmes de Sécurité	112
7.2	Créer une session	113
7.3	Création d'une session commune à tous les utilisateurs	113
7.4	Durée et <i>SID</i> d'une session	114
7.5	Destruction d'une Session	115
7.6	Exemple de Session avec <i>SID</i> aléatoire transmis par <i>GET</i>	116
7.7	Exemple de Session avec <i>SID</i> aléatoire transmis par <i>COOKIE</i>	119
7.8	<i>Login/Password</i> : Exemple de Politique de Sécurité	120
8	Bases de Données et <i>PHP Data Objects</i>	130
8.1	Créer un Base de Données dans <i>phpmyadmin</i>	130
8.2	Initiation à <i>PDO</i> : connexion, <i>query</i> , destruction	134
8.3	Requêtes Préparées	141
9	Couche d'Accès aux données	146
9.1	Diagrammes de Conception	146
9.2	Classe de Connexion à une Base de Données	146
9.3	Classes <i>Gateway</i> : Persistence des Objets Métiers	151
IV	Conception d'Architectures Avancées	159
10	Analyse Fonctionnelle	163
10.1	<i>Storyboards</i>	163
10.2	Diagrammes de Cas d'Utilisations	164
11	Organisation des Répertoires et Configuration	165
11.1	Organisation des Répertoires	165
11.2	<i>Autoload</i>	166
11.3	La classe <i>Config</i> : éviter les <i>URL</i> en dût	168

12 Architectures <i>MVC</i>	172
12.1 Le Contrôleur	172
12.2 Le Modèle	176
12.3 Les Vues	180
13 Utilisateurs et <i>Front Controller</i>	182
13.1 <i>Storyboards</i>	182
13.2 Diagramme de Cas d'Utilisation	183
13.3 Le <i>Front-Controller</i>	183
13.4 Gestion de l'Authentification	190
13.5 Gestion de plusieurs classes métier	193

Première partie
Bases du langage *PHP*

Table of Contents

1	PHP procédural	7
1.1	Notion de <i>CGI</i>	7
1.2	Générer du code <i>HTML</i> avec un <i>CGI</i> en <i>PHP</i>	8
1.3	Exemple de fonction en <i>PHP</i>	9
1.4	Inclure un fichier <i>PHP</i> dans un autre	10
1.5	Arithmétique : types <code>int</code> et <code>float</code>	11
1.6	Tableaux indexés : avec une clé de type <code>int</code>	12
1.7	Tableaux associatifs : avec une clé de type <code>String</code>	13
1.8	Passage de paramètre à un script <i>PHP</i>	15
1.9	Variables Locales ou Globales, Références	18
2	Les classes en <i>PHP</i>	21
2.1	Conception Objet, Modularité et Interopérabilité	21
2.1.1	Notion de Programmation Objet	21
2.1.2	Standard de Codage pour l'Interopérabilité <i>PSR</i>	22
2.2	Exemples de classes <i>PHP</i>	23
2.2.1	Classes de Base	23
2.2.2	Structuration des Objets, Vues	25
2.2.3	Utilisation des Classes et Vue <i>HTML</i>	30
2.3	Validation en entrée et gestion d'une exception	31
2.3.1	Qu'est-ce que le filtrage ?	31
2.3.2	Les classes <code>Personne</code> et <code>Employe</code>	32
2.3.3	Classe <code>Personne</code> avec filtrage dans les <i>setters</i>	32
2.3.4	Test de construction de Personnes et récupération des exceptions	36
2.4	Classe <code>Employe</code> héritant de la classe <code>Personne</code>	41

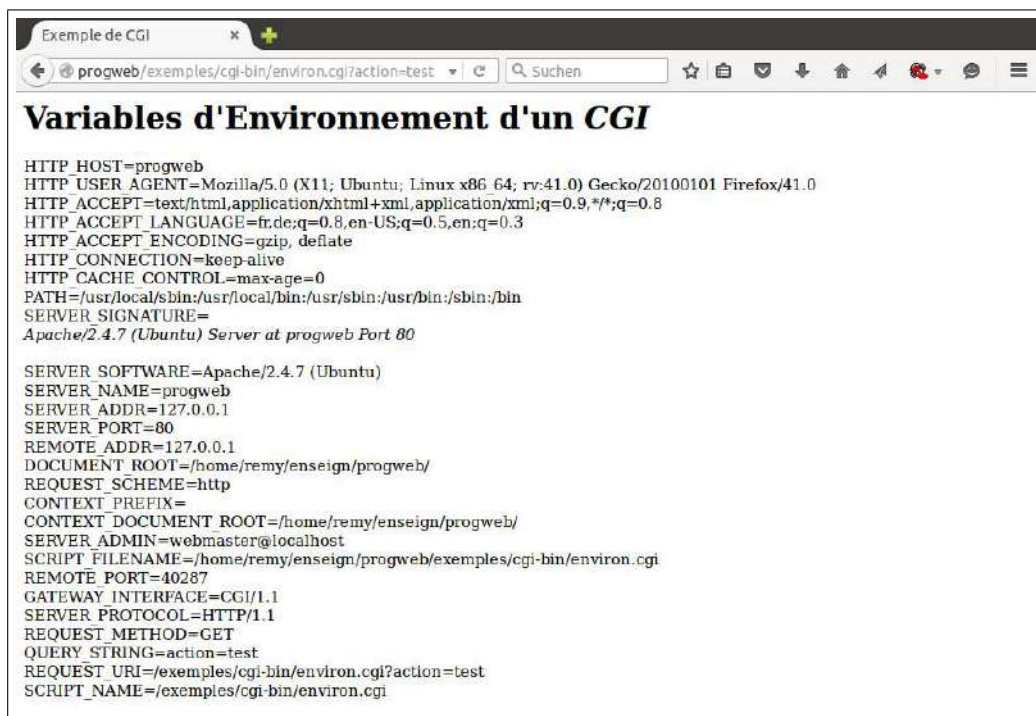
Chapitre 1

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* :



exemples/cgi-bin/envIRON.c

```
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 G n rer du code *HTML* avec un *CGI* en *PHP*

Le PHP est un langage de programmation (ou langage de scripts) qui permet de g n rer et d'afficher des pages webs dynamiques, c'est   dire des pages dont le contenu d pend des actions de l'utilisateur ou de l' tat, par exemple, d'une base de donn es. En fin de compte, le code affich  est toujours du code HTML. Ce code HTML est g n r  par le programme PHP via la commande `echo`. La protection des caract res sp ciaux du HTML (comme les guillemets) et le m lange du code PHP et du code HTML rend souvent le code d'un script PHP. Nous verrons plus loin comment att nuer ce probl me par une approche modulaire fond e sur la programmation objet.

Le script PHP est ins r    l'int rieur d'une balise `<?php >` qui peut s'ins rer au sein du code HTML.



exemples/php1/ex01_helloWorld.php

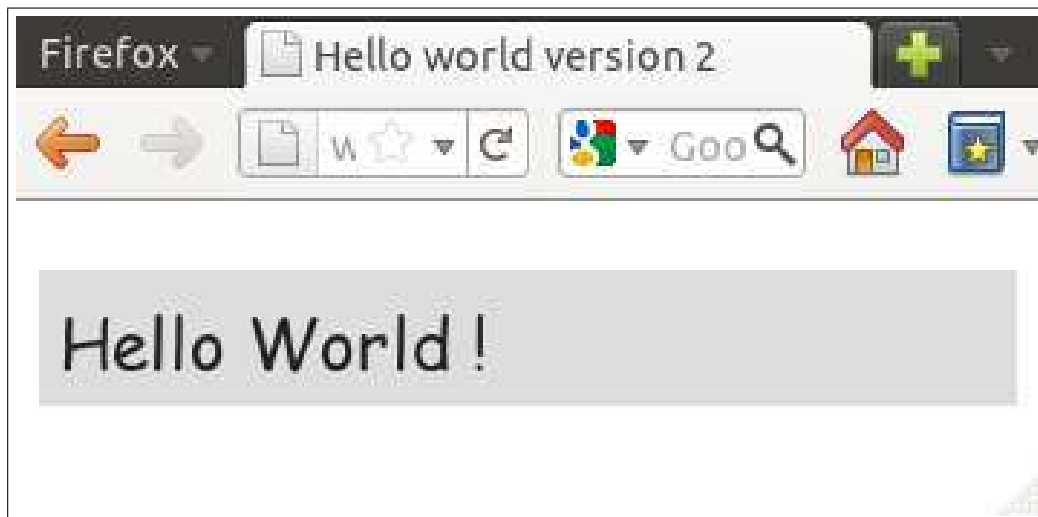
```

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 disparaît pour être remplacé par une seule ligne de code, ce qui en fin de compte allégera de beaucoup le code source PHP.



exemples/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  <?php
19  function outputFinFichierHTML5()
20  {
21  echo "</body>|n</html>|n\"";
22  }
23  ?>
24
25  <?php
26  outputEnTeteHTML5( 'Hello world version 2', 'UTF-8', 'myStyle.css' );
27  ?>
28  <?php // début du script PHP
29  echo "<p>Hello World !</p>"; // On affiche du code HTML si la sortie
30  // fin du script PHP
31  ?>
32  <?php
33  outputFinFichierHTML5();
34  ?>

```

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.)

exemples/php1/commonFunctions.php

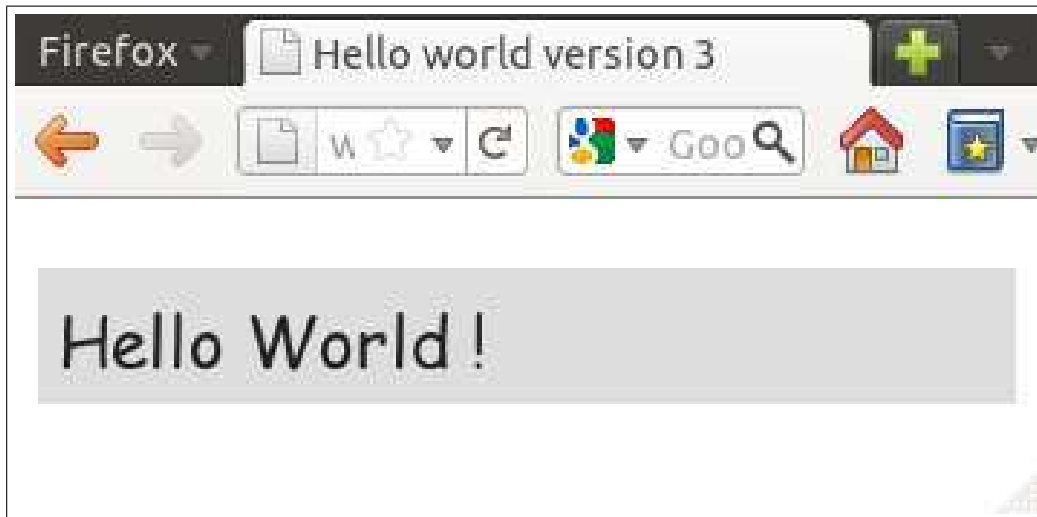
```

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
19 <?php
20 function outputFinFichierHTML5()
21 {
22     echo "</body>\n</html>\n";
23 }
24 ?>
    
```



exemples/php1/ex03_include.php

```

1 <?php require( './commonFunctions.php' );
2
3     outputEnTeteHTML5( 'Hello world version 3', 'UTF-8', 'myStyle.css' );
4 ?>
5 <p>
6 <?php // début du script PHP
7     echo "Hello World !"; // On affiche du code HTML si la sortie
8     // fin du script PHP
9 ?>
10 </p>
11 <?php
12     outputFinFichierHTML5();
13 ?>
    
```

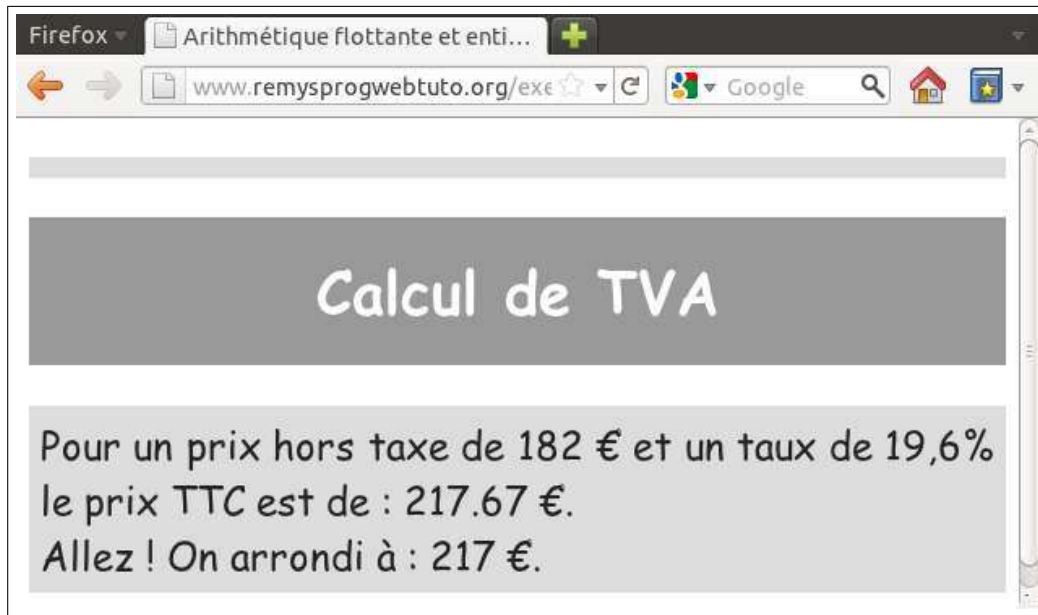
1.5 Arithmétique : types int et float

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.

exemples/php1/ex04_arithmetique_types.php

```

1 <?php require_once './commonFunctions.php'; ?>
2
3 <?php
4 outputEnTeteHTML5( 'Arithmétique flottante et entière', 'UTF-8', 'myStyle.css' );
    
```



```

5  ?>
6  <p>
7  <?php // début du script PHP
8  fonction appliqueTVA($prixHT, $taux) {
9      $prixTTC = $prixHT*(1.0+$taux/100.0);
10     return $prixTTC;
11 }
12 ?>
13 <h1>Calcul de TVA</h1>
14 <p>
15 <?php
16     $prix = 182.0;
17     echo "Pour un prix hors taxe de ".$prix." €euro; et un taux de 19,6%\n";
18     echo "le prix TTC est de : "
19     .round(appliqueTVA($prix, 19.6),2). " €euro;.\n";
20     echo "<br/>\nAllez ! On arrondi à : ".intval(appliqueTVA($prix, 19.6)). " €euro
21     ;.\n";
22 ?>
23 </p>
24 <?php
25     outputFinFichierHTML5();
26 ?>

```

1.6 Tableaux indexés : avec une clé de type int

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

exemples/php1/ex05_tableaux_keyInt.php

```

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>
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`.

exemples/php1/ex06_tableaux_keyString.php

```

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

```



```
10 $tableau = array( 'nom' => 'Caesar', 'pr nom' => 'Jules' );
11 echo "<ul>\n";
12 // acc s aux  l ments :
13 echo "<li>Acc s aux  l ments du tableau <br/>";
14 echo "Nom : ".$tableau[ 'nom' ]. "<br/>\n";
15 echo "Pr nom : ".$tableau[ 'pr nom' ]. "<br/></li>\n";
16
17 // affichage de l'ensemble des valeurs du tableau par foreach :
18 echo "<li>Les valeurs du tableau sont <br/></li>\n";
19 foreach ( $tableau as $chaine ) {
20     echo $chaine. " ";
21 }
22 echo "<br/>\n";
23
24 // affichage des cl s et des valeurs du tableau
25 echo "<li>Les donn es du tableau sont <br/>";
26 foreach ( $tableau as $cle => $chaine ) {
27     echo $cle. " : ".$chaine. "<br/></li>\n";
28 }
29 echo "</ul>\n";
30 ?>
31 </p>
32 <?php
33     outputFinFichierHTML5 ( ) ;
34 ?>
```

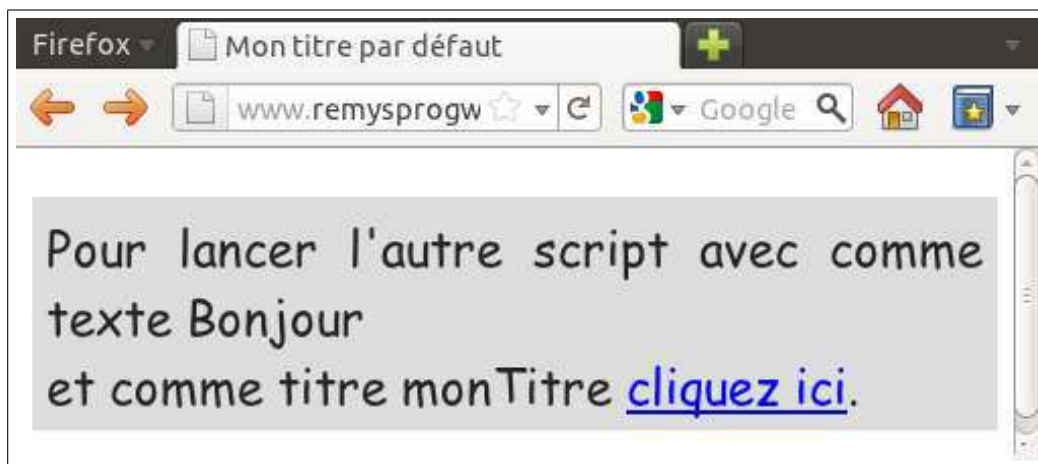
1.8 Passage de paramètre à un script PHP

Dans l'exemple suivant, le premier script passe deux paramètres 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
```



exemples/php1/ex07_passages_parametres1.php

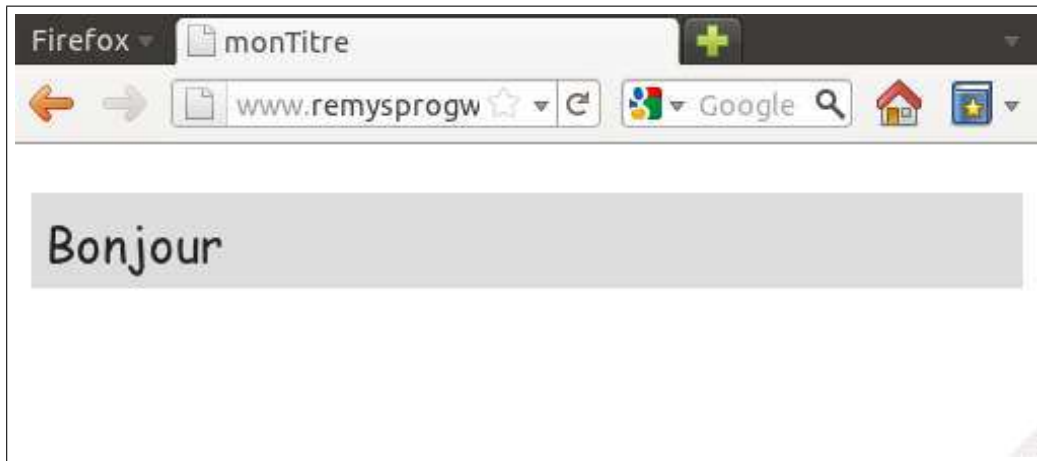
```

1 <?php require_once './commonFunctions.php'; ?>
2
3 <?php
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`.



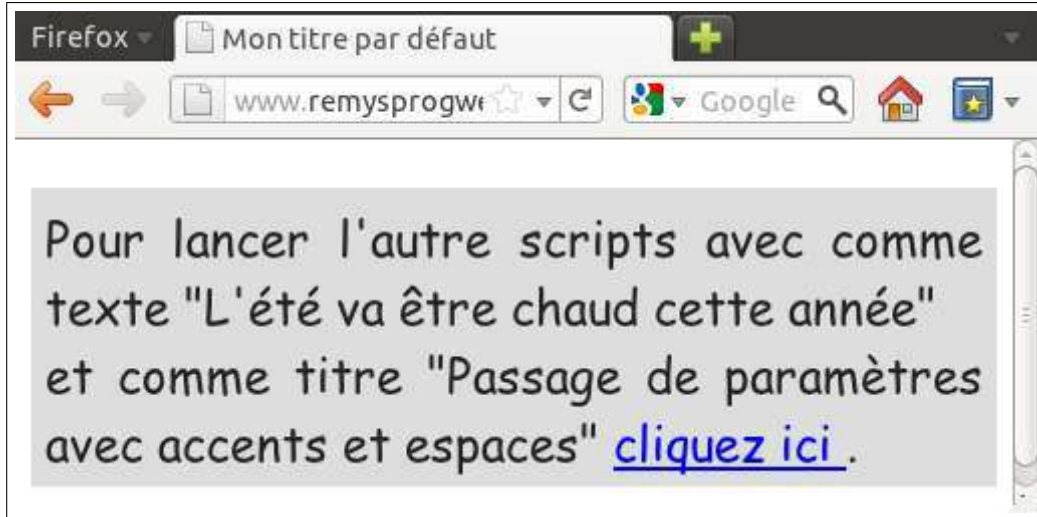
exemples/php1/ex08_passages_parametres2.php

```
1 <?php require_once './commonFunctions.php'; ?>
2
3 <?php
4     $titre = 'Mon titre par défaut';
5     if (isset($_GET['titre'])){
6         $titre = $_GET['titre'];
7     }
8
9     outputEnTeteHTML5($titre, 'UTF-8', 'myStyle.css');
10
11 ?>
12 <p>
13 <?php // début du script PHP
14     if (isset($_GET['texte'])){
15         echo $_GET['texte'];
16     }else{
17         echo "Hello World !"; // On affiche du code HTML si la sortie
18     }
19     // fin du script PHP
20 ?>
21 </p>
22 <?php
23     outputFinFichierHTML5();
24 ?>
```

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

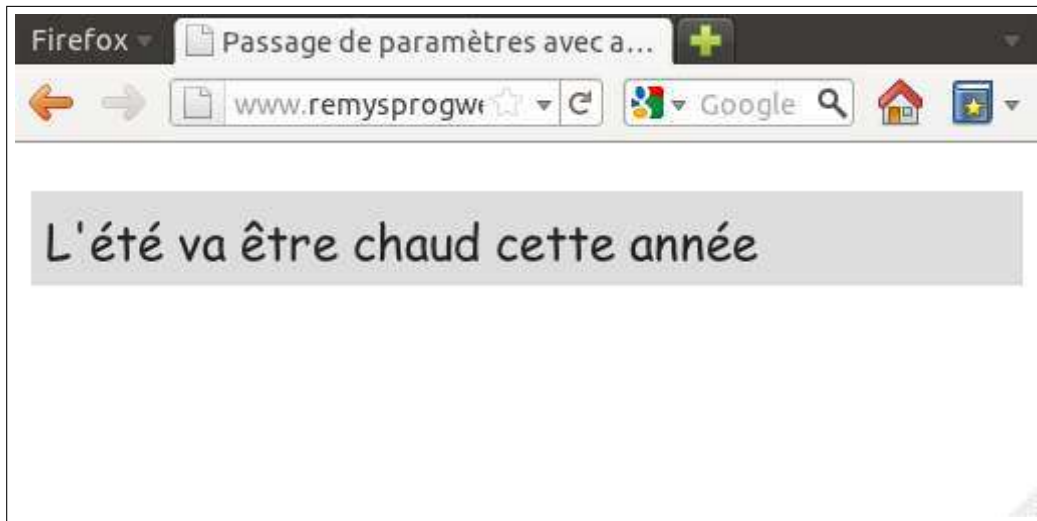


exemples/php1/ex09_passages_parametres3.php

```

1  <?php require_once './commonFunctions.php'; ?>
2
3  <?php
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 scripts avec comme texte
12 <?php
13     $texte = "L'été va être chaud cette année";
14     echo "'".$texte."'";
15     echo "<br/> et comme titre ";
16     $titre = "Passage de paramètres avec accents et espaces";
17     echo "'".$titre."'";
18     echo "\n<a href= \"";
19     echo './ex10_passages_parametres4.php?texte='
20         . htmlentities($texte, ENT_COMPAT, "UTF-8")
21         . "&titre="
22         . htmlentities($titre, ENT_COMPAT, "UTF-8")
23         . "\">\n|tcliquez ici|\n</a>";
24 ?>.
25 </p>
26 <?php
27     outputFinFichierHTML5();
28 ?>
    
```

exemples/php1/ex10_passages_parametres4.php



```

1 <?php require_once './commonFunctions.php'; ?>
2
3 <?php
4     $titre = 'Mon titre par défaut';
5     if (isset($_GET['titre'])){
6         $titre = html_entity_decode($_GET['titre']);
7     }
8
9     outputEnTeteHTML5($titre, 'UTF-8', 'myStyle.css');
10
11 ?>
12 <p>
13 <?php // début du script PHP
14     if (isset($_GET['texte'])){
15         echo html_entity_decode($_GET['texte']);
16     }else{
17         echo "Hello World !"; // On affiche du code HTML si la sortie
18     }
19     // fin du script PHP
20 ?>
21 </p>
22 <?php
23     outputFinFichierHTML5();
24 ?>

```

1.9 Variables Locales ou Globales, Références

exemples/php1/ex11_porteeVariables.php

```

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
13      $a
14  }
15
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>
23  myFunctionWithLocalVariable</code>&nbsp;:<br/>". $a. "<br/>";
24  myFunctionWithGlobalVariableAccess();
25  echo "Contenu de la variable <code>a</code> après la fonction <code>
26  myFunctionWithGlobalVariableAccess</code>&nbsp;:<br/>". $a. "<br/>";
27  ?>
<?php
    outputFinFichierHTML5();
?>

```



exemples/php1/ex12_passageParReference.php

```
1 <?php require_once './commonFunctions.php';
2
3 outputEnTeteHTML5("Port e des Variables", 'UTF-8', 'myStyle.css');
4 ?>
5 <h1>Passage par R f rence et par Valeur</h1>
6 <?php
7     // D claration d'une variable globale
8     $a = "Contenu initial de la variable globale";
9
10    // Fonction avec une variable locale "homonyme"
11    function myFunctionWithLocalVariable($myParam){
12        $myParam = "Contenu de la variable affect  dans la fonction";
13    }
14
15    function myFunctionWithGlobalVariableAccess(&$myParam){
16
17        $myParam = "Contenu de la variable affect  dans la fonction";
18    }
19
20    myFunctionWithLocalVariable($a);
21    echo "Contenu de la variable <code>a</code> apr s la fonction <code>
22        myFunctionWithLocalVariable</code>&nbsp;:<br/>". $a. "<br/>";
23    myFunctionWithGlobalVariableAccess($a);
24    echo "Contenu de la variable <code>a</code> apr s la fonction <code>
25        myFunctionWithGlobalVariableAccess</code>&nbsp;:<br/>". $a. "<br/>";
26 ?>
27 <?php
28     outputFinFichierHTML5();
29 ?>
```

Chapitre 2

Les classes en PHP

2.1 Conception Objet, Modularité et Interopérabilité

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 développeur 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 grands outils (bibliothèques, *frameworks* externes, etc.) utilisés par le *framework* doivent être 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 le 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 standard *PSR-1 : Basic Coding Standard*.

Ce standard impose de suivre une organisation d'auto-chargement des classes qui impose une organisation où les répertoires contenant du code source correspondent à des *namespaces PHP*, ou autrement dit, des modules, qui correspondront à 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, qui permet 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

Une 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 opérations nécessaires sur ces objets, à travers 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, ce 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 cela, 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'un 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 *views*.

exemples/php2/ex00_vueHtmlUtils.php

```

1 <?php
2 namespace CoursPHP\Vue;
3
4 class VueHtmlUtils {
5     public static function enTeteHTML5($title , $charset , $css_sheet){
6         // sortie du doctype. Les guillemets HTML sont protégés par |
7         $htmlCode = "<!doctype html>\n<html lang=|\"fr|\">\n<head>\n";
8         $htmlCode .= "<meta charset=|\"\". $charset. \"|\"/>\n";
9         $htmlCode .= "<link rel=|\"stylesheet|\" href=|\"\". $css_sheet. \"|\" />\n";
10        $htmlCode .= "<title>\". $title. "</title >\n";
11        $htmlCode .= "</head>\n<body>\n";
12        return $htmlCode;
13    }
14
15    public static function finFichierHTML5()
16    {
17        return "</body>\n</html>\n";
18    }
19 }
20 ?>
```

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 spécialement conçues qui testent les conditions que doivent satisfaire les attributs (ici être non `null`) avant de 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.

exemples/php2/ex01_classeTelephone.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 numéro 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 $numero le numéro de téléphone à utiliser. peut être null.
24    * */
25    public function setNumero($numero){
26        if (empty($numero))
27            $this->numero = "";
28        else
29            $this->numero = $numero;
30    }
31
32    /** @brief Setter : Initialiser ou de modifie le libellé de téléphone
33    * @param $numero le libellé de téléphone à utiliser. peut être null.
34    * */
35    public function setLibelle($libelle){
36        if (empty($libelle))
37            $this->libelle = "";
38        else
39            $this->libelle = $libelle;
40    }
41
42    /** @brief Constructeur : Construire et initialiser un Objet Telephone
43    * Appelle systématiquement les setters.
44    */
45    public function __construct($libelle, $numero){
46        $this->setLibelle($libelle);
47        $this->setNumero($numero);
48    }
49
50    /**
51    * @brief Méthode de génération d'HTML. Permet d'afficher un téléphone.
52    * Les attributs doivent être non null. (mais normalement ça ne risque pas
53    * d'arriver car les attributs sont privés donc l'utilisateur de la classe
54    * n'a pas pu les mettre à null. Les setters et le constructeur est ainsi
55    * conçu que les attributs ne peuvent pas être null.)
56    * @return le code HTML du téléphone
```

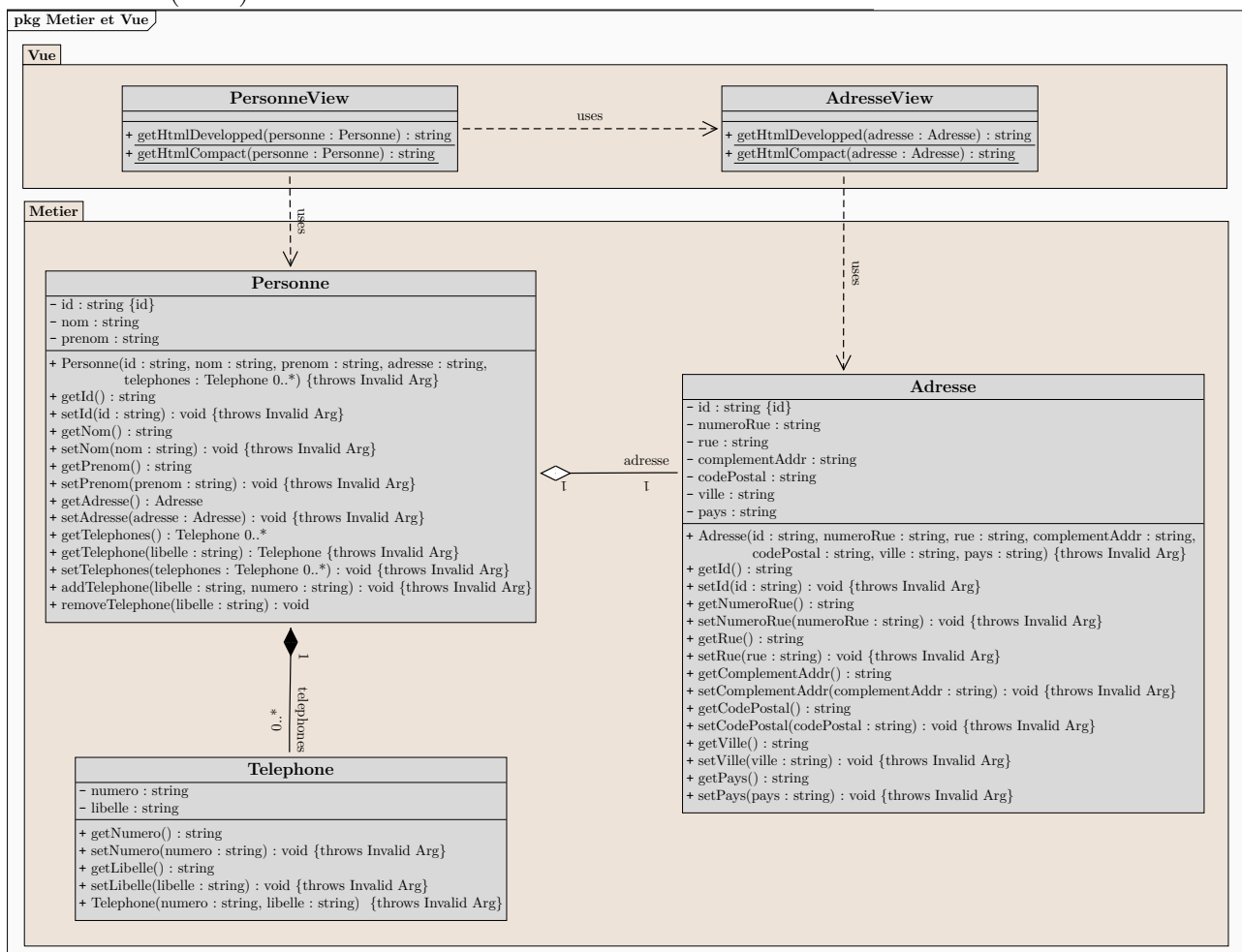
```

57 */
58 public function toHTML() {
59     return $this->libelle."&nbsp;"; ". $this->numero;
60 }
61 }
62 ?>
    
```

Comme on le voit, la classe `Telephone` fait partie d'un sous-namespace `People\Contact` du namespace `People`. Les namespace sous 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).



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

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 dévelop-

p e et l'autre compact. Comme toujours, notre mod lisation n'est pas canonique 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 s par  un ensemble de m thodes qui font partie d'une classe. Un trait peut meme d finir une parte de plusieurs classes, mais les m thodes de ces classes doivent avoir exactement le meme code. (c'est une mani re un peu "bricole" de faire de la programmation g n rique en *PHP*. Dans notre exemple, le **trait** `AdresseProperties` contient tous les *getters* et *setters* de la classe `Adresse`. Le **trait** et ses m thodes sont ins r s dans la classe `Adresse` avec le mot cl  `use`.

Nous d veloppons maintenant le code *PHP* de la classe `Adresse`.

exemples/php2/ex02_classeAdresse.php

```
1 <?php
2 namespace CoursPHP\Metier;
3
4 require_once(dirname(__FILE__).' /AdressePropertiesTrait.php ');
5
6 /** @brief La classe adresse contient l'adresse d'une personne
7  (qui peut  tre un client, un employ , un fournisseur, etc...)
8  */
9 class Adresse {
10  /** Identifiant unique de l'adresse */
11  private $id;
12  /** Num ro dans la rue, ne doit pas  tre null mais peut  tre vide */
13  private $numeroRue;
14  /** Nom de la rue, ne doit pas  tre null mais peut  tre vide */
15  private $rue;
16  /** Compl ment (lieu dit, etc. ne doit pas  tre null mais peut  tre vide */
17  private $complementAddr;
18  /** code postal */
19  private $codePostal;
20  /** nom de la ville. ne doit pas  tre null mais peut  tre vide*/
21  private $ville;
22  /** nom du pays. ne doit pas  tre null mais peut  tre vide*/
23  private $pays;
24
25  // Inclusion du trait AdresseProperties d finissant les accesseurs et setters
26  use AdresseProperties;
27
28  /** @brief Constructeur : initialise les attributs   partir des param tres.
29   * Les param tres correspondent aux valeurs   mettre dans les attributs.
30   * Tout objet doit  tre initialis  avec le constructeur (appel   new).
31   * Ici, les param tres peuvent  tre null. Les attributs sont alors initialis s
32   *   une cha ne vide, permettant la coh rence de la classe.
33   */
34  public function __construct($id, $numeroRue, $rue, $complementAddr,
35  $codePostal, $ville, $pays) {
36  $this->setId($id);
37  $this->setNumeroRue($numeroRue);
38  $this->setRue($rue);
39  $this->setComplementAddr($complementAddr);
40  $this->setCodePostal($codePostal);
41  $this->setVille($ville);
42  $this->setPays($pays);
43  }
```

```
44 } // end of class Adresse
45 ?>
```

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

exemples/php2/ex03_classeAdressePropertiesTrait.php

```
1 <?php
2 namespace CoursPHP\Metier;
3
4 /**
5  @brief La classe adresse contient l'adresse d'une personne
6  (qui peut être un client, un employé, un fournisseur, etc...)
7  */
8 trait AdresseProperties {
9
10  /** @brief Accesseur : permet d'obtenir l'identifiant de l'instance. */
11  public function getId() {
12      return $this->id;
13  }
14
15  /** @brief Accesseur : permet d'obtenir le numéro dans la rue. */
16  public function getNumeroRue() {
17      return $this->numeroRue;
18  }
19
20  /** @brief Accesseur : permet d'obtenir le nom la rue. */
21  public function getRue() {
22      return $this->rue;
23  }
24
25  /** @brief Accesseur : permet d'obtenir le nom le complément d'adresse. */
26  public function getComplementAddr() {
27      return $this->complementAddr;
28  }
29
30  /** @brief Accesseur : permet d'obtenir le nom le code postal. */
31  public function getCodePostal() {
32      return $this->codePostal;
33  }
34
35  /** @brief Accesseur : permet d'obtenir le nom la ville. */
36  public function getVille() {
37      return $this->ville;
38  }
39
40
41  /** @brief Accesseur : permet d'obtenir le pays. */
42  public function getPays() {
43      return $this->pays;
44  }
45
46  /** @brief setter : permet d'initialiser ou de modifier le nom de la rue.
47  @param $NumeroRue le numéro à utiliser. peut être null.
48  */
49  public function setId($id) {
50      $this->id = empty($id) ? "" : $id;
```

```
51 }
52
53 /** @brief setter : permet d'initialiser ou de modifier le nom de la rue.
54  * @param $NumeroRue le numéro à utiliser. peut être null.
55  */
56 public function setNumeroRue($numeroRue) {
57     $this->numeroRue = ($numeroRue == null) ? "" : $numeroRue;
58 }
59
60 /** @brief setter : permet d'initialiser ou de modifier le numéro dans la rue.
61  * @param $Rue le nom de la rue ou de la place à utiliser. peut être null.
62  */
63 public function setRue($rue) {
64     $this->rue = ($rue == null) ? "" : $rue;
65 }
66
67 /** @brief setter : permet d'initialiser ou de modifier le complément d'
68  * adresse.
69  * @param $ComplementAddr le complément d'adresse à utiliser. peut être null.
70  */
71 public function setComplementAddr($complementAddr) {
72     $this->complementAddr = ($complementAddr == null) ? "" : $complementAddr;
73 }
74
75 /** @brief setter : permet d'initialiser ou de modifier le code postal.
76  * @param $CodePostal le numéro à utiliser. peut être null
77  */
78 public function setCodePostal($codePostal) {
79     $this->codePostal = ($codePostal == null) ? "" : $codePostal;
80 }
81
82 /** @brief setter : permet d'initialiser ou de modifier le nom de la ville.
83  * @param $Ville le nom de la ville à utiliser. peut être null
84  */
85 public function setVille($ville) {
86     $this->ville = ($ville == null) ? "" : $ville;
87 }
88
89 /** @brief setter : permet d'initialiser ou de modifier le nom du Pays
90  * @param $pays le nom du Pays à utiliser. peut être null
91  */
92 public function setPays($pays) {
93     $this->pays = ($pays == null) ? "" : $pays;
94 }
95 }
96 ?>
```

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

exemples/php2/ex04_classeAdresseView.php

```
1 <?php
2 namespace CoursPHP\Vue;
3 /**
4  * @brief La classe AdresseView implémente la génération d'HTML pour afficher
5  * une adresse dans une vue dans un navigateur.
6  * Implémente aussi des utilitaires de conversion à partir d'une Adresse
```

```

7  * pour obtenir facilement le code HTML pour afficher une Adresse.
8  */
9  class AdresseView {
10
11  /**
12  * @brief Méthode de génération de code HTML. Permet d'afficher une adresse.
13  * Les attributs doivent être non null.
14  * (mais normalement ça ne risque pas d'arriver car les attributs sont privés
15  * donc l'utilisateur de la classe n'a pas pu les mettre à null.
16  * Les setters et le constructeur est ainsi conçu que les attributs
17  * ne peuvent pas être null.)
18  */
19  public static function getHtmlDevelopped($adresse){
20  $htmlCode = "";
21  $htmlCode .= "<strong>Adresse : </strong><br/>\n";
22  $htmlCode .= $adresse->getNumeroRue();
23  if (!empty($adresse->getNumeroRue()))
24  $htmlCode .= ", ";
25  $htmlCode .= $adresse->getRue();
26  if (!empty($adresse->getRue()))
27  $htmlCode .= "<br/>";
28  $htmlCode .= $adresse->getComplementAddr();
29  if (!empty($adresse->getComplementAddr()))
30  $htmlCode .= "<br/>";
31  $htmlCode .= $adresse->getCodePostal(). " ";
32  $htmlCode .= $adresse->getVille();
33  if (!empty($adresse->getVille()))
34  $htmlCode .= "<br/>";
35  $htmlCode .= $adresse->getPays(). "<br/>";
36
37  return $htmlCode;
38  }
39
40  /**
41  * @brief Méthode de génération d'HTML. Permet d'afficher une adresse en
42  * HTML.
43  * Les attributs doivent être non null. (mais normalement ça ne risque pas d'
44  * arriver
45  * car les attributs sont privés donc l'utilisateur de la classe n'a pas pu
46  * les mettre à null.
47  * Les setters et le constructeur est ainsi conçu que les attributs ne
48  * peuvent pas être null.)
49  * La méthode retourne le code HTML pour un affichage compact.
50  */
51  public static function getHtmlCompact($adresse){
52  $htmlCode = "";
53  $htmlCode .= $adresse->getNumeroRue();
54  if (!empty($adresse->getNumeroRue()))
55  $htmlCode .= ", ";
56  $htmlCode .= $adresse->getRue();
57  if (!empty($adresse->getRue()))
58  $htmlCode .= ", ";
59  $htmlCode .= $adresse->getComplementAddr();
60  if (!empty($adresse->getComplementAddr()))
61  $htmlCode .= ", ";
62  $htmlCode .= $adresse->getCodePostal(). " ";

```

```

59     $htmlCode .= $adresse->getVille ();
60     if (!empty($adresse->getVille ()))
61         $htmlCode .= ", ";
62     $htmlCode .= $adresse->getPays ();
63
64     return $htmlCode;
65 }
66 } // end of class AdresseView
67 ?>

```

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 préférera une structure dans laquelle la génération du code *HTML* se trouve dans un script séparé, appelé *vue*. Pour cela, le script de test prépare 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 préparées par le script de test pour les afficher.



exemples/php2/ex05_testExampleImportNamespace.php

```

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
7 // Adresse Complète :
8 $adresse1 = new CoursPHP\Metier\Adresse("0af46d3bd9", '10', 'allée du net',
9     'Quartier de l\'avenir', '63000', 'Clermont-Ferrand', 'France');

```

```

10 // Adresse sans code postal ni complément d'adresse
11 $adresse2 = new CoursPHP\Metier\Adresse("2bf46d3ba32", '10', 'Downing Street',
12                                         null, null, 'London', 'United Kingdom');
13
14 // Appel de la vue (Génération du code HTML)
15 require('ex05_vueExampleImportNamespace.php');
16 ?>

```

Voici maintenant le code de la vue :

exemples/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>". $telephone->toHTML(). "<br/>";
11
12  echo "<strong>Adresse au format compact&nbsp;:</strong><br/>".
13      CoursPHP\Vue\AdresseView :getHtmlCompact($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 mécanisme de gestion des erreurs par *exceptions*.

2.3.2 Les classes *Personne* et *Employe*

2.3.3 Classe *Personne* avec filtrage dans les *setters*

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

exemples/php2/ex07_classePersonne.php

```
1 <?php
2 namespace CoursPHP\Metier;
3 require_once(dirname(__FILE__).'PersonnePropertiesTrait.php');
4 /**
5  @brief La classe Personne représente une personne (client, employé, fournisseur
6  , contact...).
7  Elle contient l'identité (nom prénom), l'adresse, le numéro de téléphone
8  et le salaire mensuel de l'employé.
9  */
9 class Personne {
10  /** Identifiant unique de la personne */
11  protected $id;
12  /** nom de l'employé : obligatoire. Le nom de l'employé ne peut pas être null
13  ou vide. */
13  protected $nom;
14  /** prénom de l'employé */
15  protected $prenom;
16  /** adresse de l'employé */
17  protected $adresse;
18  /** Tableau des numéros de téléphone */
19  protected $telephones;
20
21  use PersonneProperties;
22
23  /**
24  @brief Constructeur : initialise les attributs à partir des paramètres.
25  Les paramètres correspondent aux valeurs à mettre dans les attributs.
26  Tout objet doit être initialisé avec le constructeur (appel à new).
27  Des exceptions sont rejetées en cas de paramètres invalide.
28  */
29  public function __construct($id, $nom, $prenom, $adresse, $telephones){
30
31      $this->setId($id);
32      $this->setNom($nom);
33      $this->setPrenom($prenom);
34      $this->setAdresse($adresse);
35      $this->setTelephones($telephones);
```

```
36 }
37 }
38 ?>
```

exemples/php2/ex08_classePersonnePropertiesTrait.php

```
1 <?php
2 namespace CoursPHP\Metier;
3
4 trait PersonneProperties {
5     /** @brief accesseur : permet d'obtenir le nom de l'employé */
6     public function getId() {
7         return $this->id;
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 de l'employé
26        */
27    public function getTelephones() {
28
29        return $this->telephones;
30    }
31
32    /** @brief accesseur : permet d'obtenir un numéro de téléphone de l'employé */
33    public function getTelephone($libelle) {
34        if (empty($this->telephones[$libelle])){
35            throw new \Exception('Désolé, Le téléphone "'. $libelle. '" n'existe pas.
36                Have a try in the phonebook... ');
37        }
38        return $this->telephones[$libelle];
39    }
40
41    /** setter : permet d'initialiser ou de modifier l'identifiant de la personne
42        @param $id l'identifiant de la personne. Doit être non null et non vide
43        */
44    public function setId($id) {
45        if (empty($id) || strlen($id) != 10){
46            throw new \Exception('Désolé, toute personne doit avoir un identifiant de
47                10 caractères !');
48        }else{
49            $this->id= $id;
50        }
51    }
52 }
```

```
48 }
49
50 /** setter : permet d'initialiser ou de modifier le nom de la personne
51     @param $Nom le nom de la personne. Doit être non null et comporter
52     au moins 1 caractère.
53 */
54 public function setNom($nom) {
55     if (empty($nom) || strlen($nom) > 100){
56         throw new \Exception('Désolé, toute personne doit avoir un nom et le nom a
57             au plus 100 caractères !');
58     }else{
59         $this->nom = $nom;
60     }
61 }
62
63 /** setter : permet d'initialiser ou de modifier le nom de la personne */
64 public function setPrenom($prenom) {
65     if (empty($prenom) || strlen($prenom) > 50){
66         throw new \Exception('Désolé, toute personne doit avoir un prenom et le
67             prenom a au plus 50 caractères !');
68     }else{
69         $this->prenom = $prenom;
70     }
71 }
72
73 /** setter : permet d'initialiser ou de modifier l'adresse de la personne */
74 public function setAdresse($adresse) {
75     if ($adresse == null || get_class($adresse) != 'CoursPHP\Metier\Adresse'){
76         throw new \Exception('Erreur : Adresse Invalide');
77     }else{
78         $this->adresse = $adresse;
79     }
80 }
81
82 /** setter : permet d'initialiser ou de modifier l'adresse de la personne */
83 public function setTelephones($telephones) {
84     if (!is_array($telephones)){
85         throw new \Exception('Erreur : Téléphones Invalide');
86     }else{
87         $this->telephones = $telephones;
88     }
89 }
90
91 /** setter : permet d'ajouter un numéro de téléphone de la personne */
92 public function addTelephone($libelle, $numero) {
93     if (!empty($numero) && strlen($numero) <= 15) {
94         if (!is_array($this->telephones)){
95             $this->telephones = array();
96         }
97         $this->telephones[$libelle] = new Telephone($libelle, $numero);
98     }else{
99         throw new \Exception('Erreur : Téléphone Invalide');
100     }
101 }
```

```

102  /** setter : permet d'ajouter un numéro de téléphone de la personne */
103  public function removeTelephone($libelle) {
104      if (!empty($this->telephone[$libelle])){
105          unset($this->telephone[$libelle]);
106      }
107  }
108
109  }
110  ?>

```

exemples/php2/ex09_classePersonneView.php

```

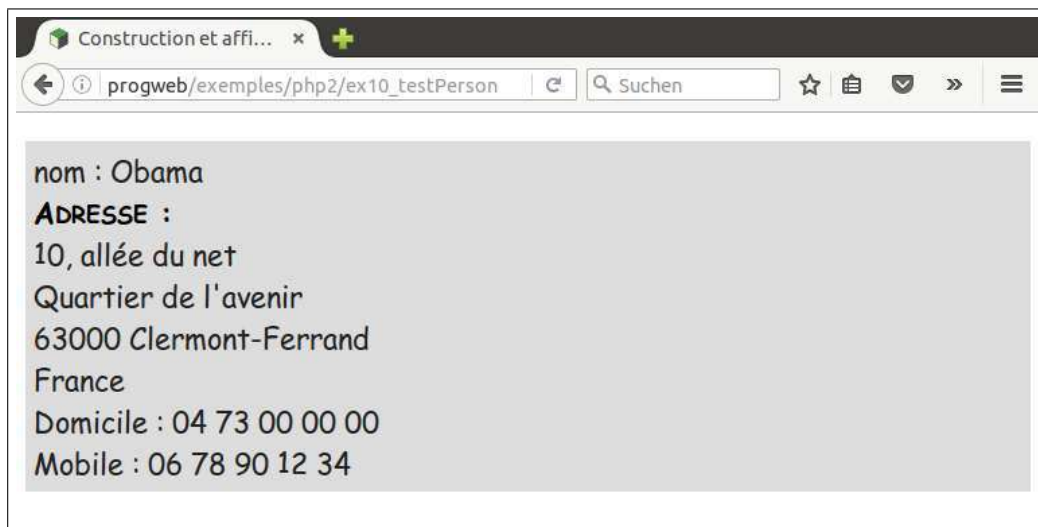
1  <?php
2  namespace CoursPHP\Vue;
3
4  class PersonneView {
5      /**
6       * @brief Méthode de génération de code HTML. Permet d'afficher une personne.
7       * Les attributs doivent être non null.
8       * (mais normalement ça ne risque pas d'arriver car les attributs sont privés
9       * donc 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     */
13     public static function getHtmlDevelopped($personne){
14         $htmlCode = "";
15         $htmlCode .= "nom : ".$personne->getNom(). "<br/>\n";
16         if (strlen($personne->getPrenom())>=1)
17             $htmlCode .= "Prénom : ".$personne->getPrenom(). "<br/>\n";
18         $htmlCode .= AdresseView::getHtmlDevelopped($personne->getAdresse());
19         $count = 0;
20         foreach ($personne->getTelephones() as $telephone) {
21             if ($count != 0){
22                 $htmlCode .= "<br/>";
23             }
24             $count++;
25             $htmlCode .= $telephone->toHTML();
26         }
27         $htmlCode .= "<br/>\n";
28         return $htmlCode;
29     }
30
31     /**
32      * @brief Méthode de génération d'une ligne de tableHTML.
33      * Permet d'afficher des Personnes dans une table HTML.
34     */
35     public static function getHtmlTableRow($personne){
36         $htmlCode = "<tr>";
37         $htmlCode .= "<td>".$personne->getNom(). "</td>";
38         $htmlCode .= "<td>".$personne->getPrenom(). "</td>";
39         $htmlCode .= "<td>". AdresseView::getHtmlCompact($personne->getAdresse()). "</td>";
40         $htmlCode .= "<td>";
41         $count = 0;
42         foreach ($personne->getTelephones() as $telephone) {
43             if ($count != 0){

```

```
44     $htmlCode .= "<br/>";
45     }
46     $count++;
47     $htmlCode .= $telephone->toHTML();
48     }
49     $htmlCode .= "</td>";
50     $htmlCode .= "</tr>";
51
52     return $htmlCode;
53 }
54
55
56
57 /**
58  * Permet d'obtenir une ligne de table HTML avec les en-têtes de colonnes
59  * pour affichage d'une table de Personnes.
60  */
61 public static function getHtmlTableHeads() {
62     $htmlCode = "<tr>";
63     $htmlCode .= "<th>Nom</th>";
64     $htmlCode .= "<th>Prénom</th>";
65     $htmlCode .= "<th>Adresse</th>";
66     $htmlCode .= "<th>Téléphone(s)</th>";
67     $htmlCode .= "</tr>";
68     return $htmlCode;
69 }
70 } // end of class PersonneView
71 ?>
```

2.3.4 Test de construction de Personnes et récupération des exceptions

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



```

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 ?>

```

exemples/php2/ex10_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(
9     'Construction et affichage d\'une Personne', 'UTF-8', 'myStyle.css')
10 ;
11 echo "<p>";
12 echo PersonneView::getHtmlDevelopped($personne);
13 echo "</p>";
14
15 echo CoursPHP\Vue\VueHtmlUtils::finFichierHTML5();
16 ?>

```

exemples/php2/ex10_vueErreur.php

```

1 <?php
2 require_once('ex00_vueHtmlUtils.php ');
3
4 echo CoursPHP\Vue\VueHtmlUtils::enTeteHTML5('Gestion d\'une exception',
5     'UTF-8', 'myStyle.css');
6
7 echo "<h1>Une Erreur c'est produite</h1>";
8 echo "<p>Exception reçue : ".$e->getMessage(). "</p>";
9

```

```
10 echo CoursPHP\Vue\VueHtmlUtils::finFichierHTML5();
11 ?>
```

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

Nom	Prénom	Adresse	Téléphone(s)
Obama	Barack	10, allée du net, Quartier de l'avenir, 63000 Clermont-Ferrand, France	Domicile : 04 73 00 00 00
Modèle	Jean	12, Georgy, 63000 Trench Town, Jamaica	Emergency : 911
Génération	iGrec	10, Rock'n Roll Street, Bronx, 63000 Rackamadour, France	Travail : 01 23 45 67 89

exemples/php2/ex11_testTableViewPersonnes.php

```
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 echo "<p>";
11 try {
12     $adresse1 = new Adresse("0af46d3bd9", '10', 'allée du net', 'Quartier de l\'
13         avenir',
14         '63000', 'Clermont-Ferrand', 'France');
15     $telephones1 = array(new Telephone("Domicile", "04 73 00 00 00"));
16     $personne1 = new Personne("043f46d3a3", "Obama", "Barack", $adresse1,
17         $telephones1);
18     $adresse2 = new Adresse("0af46d3be0", '12', 'Georgy', null, '63000', 'Trench
19         Town', 'Jamaica');
20     $telephones2 = array(new Telephone("Emergency", "911"));
21     $personne2 = new Personne("af3f46d27f", "Modèle", "Jean", $adresse2,
22         $telephones2);
23     $adresse3 = new Adresse("0af46d3be1", '10', 'Rock\'n Roll Street', 'Bronx',
24         '63000', 'Rackamadour', 'France');
25     $personne3 = new Personne("3aef46d7fe", "Génération", "iGrec", $adresse3,
26         array(new Telephone("Travail", "01 23 45 67 89")));
27     $personnes = array(1 => $personne1, 2 => $personne2, 3 => $personne3);
28
29     require("ex11_vueNormale.php");
30 } catch (Exception $e) {
31     require("ex10_vueErreur.php");
32 }
```

28 | ?>

exemples/php2/ex11_vueNormale.php

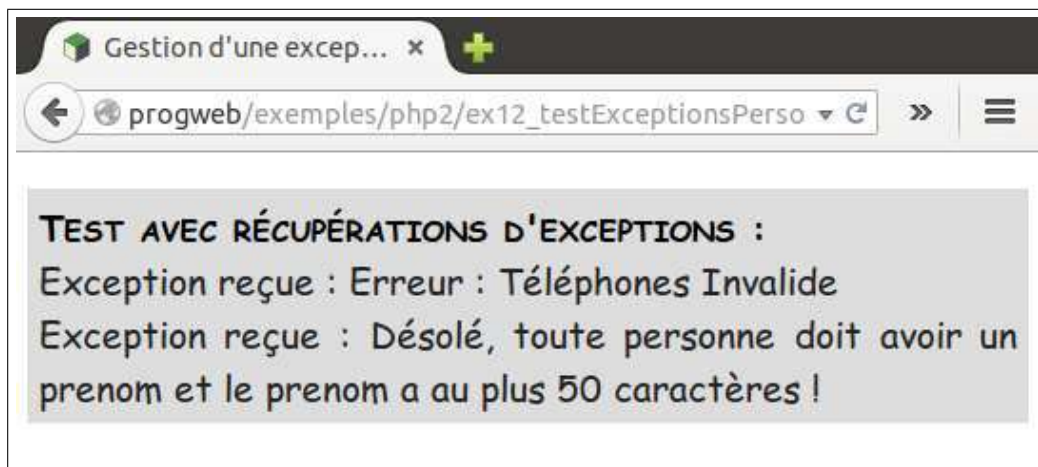
```

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&nbsp;&nbsp;&nbsp;</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).



exemples/php2/ex12_testExceptionsPersonnes.php

```

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 $dataError = array();
11
12 try {
13     $adresse1 = new Adresse("0af46d3bd9", '10', 'Downing Street', null,
14                             null, 'London', 'United Kingdom');
15     $personne1 = new Personne("e2f46d3ba6", "Thatcher", "Marggy", $adresse1,
16                               "01 23 45 67 89"); // phony phone number...
17 } catch (Exception $e) {
18     $dataError["personne1"] = $e->getMessage();
19 }
20
21 try {
22     $adresse2 = new Adresse("b3f46d3a5d", '10', 'allée du net',
23                             'Quartier de l\'avenir', '63000', 'Clermont-Ferrand',
24                             'Technique');
25     $personne2 = new Personne("a4b46d3a5c", "Urluberlu", null, $adresse2, null);
26     // Given name ??
27 } catch (Exception $e) {
28     $dataError["personne2"] = $e->getMessage();
29 }
30 // Appel de la vue :
31 require('ex12_vueExceptionsPersonnes.php');
32 ?>

```

exemples/php2/ex12_vueExceptionsPersonnes.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\AdresseView;
7 use CoursPHP\Vue\PersonneView;
8
9 echo CoursPHP\Vue\VueHtmlUtils::enTeteHTML5('Gestion d\'une exception',
10                                             'UTF-8', 'myStyle.css');
11
12 echo "<p>";
13     echo "<strong>Test avec récupérations d'exceptions&nbsp;   </strong><br/>";
14     if (empty($dataError["personne1"])){
15         echo PersonneView::getHtmlDeveloppe($personne1);
16     }else{
17         echo $dataError["personne1"];
18     }
19 echo "</p>";
20
21 echo "<p>";
22     if (empty($dataError["personne2"])){
23         echo PersonneView::getHtmlDeveloppe($personne2);
24     }else{

```

```

25     echo $dataError [ "personne2" ];
26     }
27     echo "</p>";
28
29     echo CoursPHP\Vue\VueHtmlUtils : :finFichierHTML5 ( ) ;
30     ?>

```

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 chaîne de caractères 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 méthode 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.

exemples/php2/ex13_classeEmploye.php

```

1 <?php
2 namespace CoursPHP\Metier;
3
4 require_once(dirname(__FILE__).' /EmployeProperties.php ');
5
6 class Employe extends Personne {
7
8     /** salaire mensuel de la personne en euros/mois */
9     protected $salaireMensuel;
10
11    /** catégorie de l'employé : "secrétaire", "commercial", "technique" ou "pdg"
12        */
13    protected $categorie;
14
15    /** @brief tableau de toutes les catégories d'employés possibles.
16        un attribut statique est un attribut qui existe en un seul exemplaire
17        commun à tous les objets de la classe.
18        Cela évite d'avoir autant de copies du tableau $categoriesEmployes
19        qu'il y a d'instance de la classe Employe en mémoire.
20    */
21    private static $categoriesEmployes = array("secrétaire", "commercial", "
22        technique", "boss");
23
24    use EmployeProperties;
25
26    public function __construct($id, $nom, $prenom, $adresse,
27        $telephones, $salaire, $categorie) {
28
29        // Appel du constructeur de la classe mère :
30        parent::__construct($id, $nom, $prenom, $adresse, $telephones);
31
32        $this->setSalaireMensuel($salaire);
33        $this->setCategorie($categorie);
34    }

```

```
33 }  
34 ?>
```

exemples/php2/ex14_classeEmployeProperties.php

```
1 <?php  
2 namespace CoursPHP\Metier;  
3  
4 trait EmployeProperties {  
5  
6     /** Méthode statique de validation d'un paramètre de catégorie.  
7         la valeur doit se trouver dans les tableau $categoriesEmployes.  
8         Une méthode statique est une méthode qui ne s'applique pas à un objet  
9         particulier.  
10        On l'utilise avec self:: ou à l'extérieur de la classe avec Employe::  
11 */  
12 public static function isValidCategorie($categorie) {  
13     if ($categorie == null || !is_string($categorie) ||  
14         !in_array($categorie, self::$categoriesEmployes)){  
15         return false;  
16     }  
17     return true;  
18 }  
19 /** @brief accesseur : permet d'obtenir la catégorie de l'employé */  
20 public function getCategorie() {  
21     return $this->categorie;  
22 }  
23  
24 /** @brief accesseur : permet d'obtenir le téléphone 1 de l'employé */  
25 public function getSalaireMensuel() {  
26     return $this->salaireMensuel;  
27 }  
28  
29 /** setter : permet d'initialiser ou de modifier la catégorie de la personne  
30         @param $Categorie doit correspondre à une catégorie d'employé ré  
31         pertoriée.  
32 */  
33 public function setCategorie($categorie) {  
34     if (!self::isValidCategorie($categorie)){  
35         throw new \Exception("Erreur, catégorie d'employé \"\".$categorie  
36             .\"\" invalide.");  
37     }else{  
38         $this->categorie = $categorie;  
39     }  
40 }  
41 /** setter : permet d'initialiser ou de modifier le salaire mensuel de la  
42         personne  
43         @param $Salaire salaire mensuel en euros/mois  
44 */  
45 public function setSalaireMensuel($salaire) {  
46     if ($salaire == null || !is_numeric($salaire)){  
47         $this->salaireMensuel = 0.0;  
48     }else {  
49         $this->salaireMensuel = $salaire;  
50     }  
51 }
```

```

49     }
50   }
51 }
52 ?>

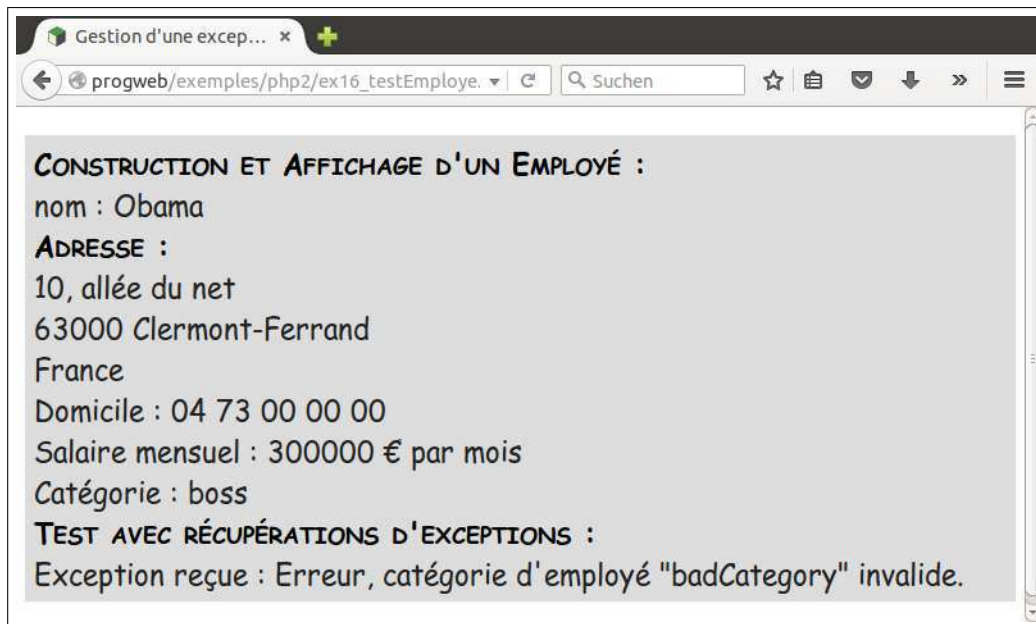
```

exemples/php2/ex15_classeEmployeView.php

```

1 <?php
2 namespace CoursPHP\Vue;
3
4 class EmployeView {
5     /**
6      * @brief Méthode de génération de code HTML. Permet d'afficher un Employe.
7      */
8     public static function getHtmlDevelopped($employe){
9         $htmlCode = PersonneView::getHtmlDevelopped($employe);
10        $htmlCode .= "Salaire mensuel : ".$employe->getSalaireMensuel()." €euro; par
11           mois<br/>\n";
12        $htmlCode .= "Catégorie : ".$employe->getCategorie()."<br/>\n";
13        return $htmlCode;
14    }
15 } // end of class EmployeView
16 ?>

```



exemples/php2/ex16_testEmploye.php

```

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 require_once(dirname(__FILE__). '/classes/Employe.php ');
6
7 use CoursPHP\Metier\Adresse;
8 use CoursPHP\Metier\Telephone;

```

```

9  use CoursPHP\Metier\Employe;
10
11  $dataError = array();
12
13  try {
14      $adresse1 = new Adresse("0af46d3bd9", '10', 'allée du net', 'Quartier de l\'
          avenir ',
15                          '63000', 'Clermont-Ferrand', 'France');
16      $telephones1 = array(new Telephone("Domicile", "04 73 00 00 00"));
17      $employe1 = new Employe("0af46d3bd9", "Obama", "Barack", $adresse1,
          $telephones1, 300000.0, 'boss');
18  } catch (Exception $e) {
19      $dataError["employe1"] = $e->getMessage();
20  }
21
22  try {
23      $adresse2 = new Adresse("5b246d3da2", '10', 'Downing Street', null,
          null, 'London', 'United Kingdom');
24      $employe2 = new Employe("d7c46d3a3b", "Thatcher", "Margaret", $adresse2,
          array(new Telephone("Emergency", "911")), null, 'badCategory
          ');
25  } catch (Exception $e) {
26      $dataError["employe2"] = $e->getMessage();
27  }
28  }
29
30  // Appel de la vue :
31  require('ex16_vueEmploye.php');
32
33  ?>

```

exemples/php2/ex16_vueEmploye.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  require_once(dirname(__FILE__).'/classes/EmployeView.php');
6
7  use CoursPHP\Vue\EmployeView as EmployeView;
8
9  echo CoursPHP\Vue\VueHtmlUtils::enTeteHTML5('Gestion d\'une exception', 'UTF-8
          ', 'myStyle.css');
10
11  echo "<h2>Construction et Affichage d'un Employé&nbsp;:</h2>";
12  echo "<p>";
13  echo "<strong>Test avec récupérations d'exceptions&nbsp;:</strong><br/>";
14  echo "</p>";
15
16  echo "<p>";
17  if (empty($dataError["employe1"])){
18      echo EmployeView::getHtmlDeveloppe($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

3	Formulaires HTML/PHP	49
3.1	Formulaires <i>HTML</i>	49
3.1.1	Premier Formulaire <i>HTML</i>	49
3.2	Exemple de style <i>CSS</i> pour formulaire	50
3.2.1	Réception des données en <i>PHP</i>	55
3.3	Validation pour la sécurité : Appel de <code>filter_var</code>	55
3.4	Appel des vues	57
3.5	Tableaux <code>\$_POST</code> <code>\$_GET</code> <code>\$_REQUEST</code>	58
3.6	Formulaires dynamiques an javascript	60
4	Injection, Filtrage, Expressions Régulières	63
4.1	Injections <i>HTML</i> et échappement	63
4.1.1	Injections <i>HTML</i>	63
4.1.2	Prévention des injections <i>HTML</i> par échappement	65
4.2	Injections <i>SQL</i>	70
4.3	La fonction <code>filter_var</code>	76
4.3.1	Principe de la fonction <i>PHP</i> <code>filter_var</code>	76
4.3.2	Les filtres de Validation	77
4.3.3	Les filtres de Nettoyage	79
4.3.4	Le filtre personnalisé <code>FILTER_CALLBACK</code>	79
4.4	Expressions régulières	80
5	Formulaires <i>PHP/HTML</i>, filtrage, exceptions	82
5.1	Modélisation : Diagrammes de Classes	82
5.2	La Classe <code>Adresse</code>	84
5.3	Filtrage des attributs	85
5.4	Fabrique d' <code>Adresse</code>	90

5.5	Génération de formulaires et classe AdresseFormView	91
5.6	Enchaînement de la saisie à la vue	97
5.6.1	Diagramme de Séquence	97
5.6.2	Saisie et Soumission du Formulaire	97
5.6.3	Les Vues	99
5.6.4	Modification d'une Adresse	101

Chapitre 3

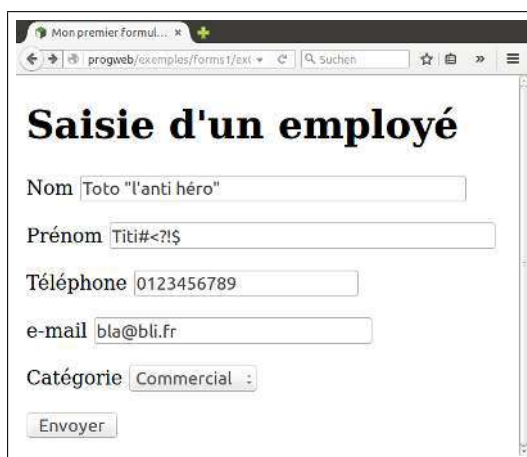
Formulaires HTML/PHP

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 les 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*



Mon premier Formul...
progweb/exemples/forms1/ex1
Suchen

Saisie d'un employé

Nom

Prénom

Téléphone

e-mail

Catégorie

exemples/forms1/ex01_form_html.html

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>
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="secrtaire" 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"/>
37    </p>
38  </form>
39 </body>
40 </html>

```

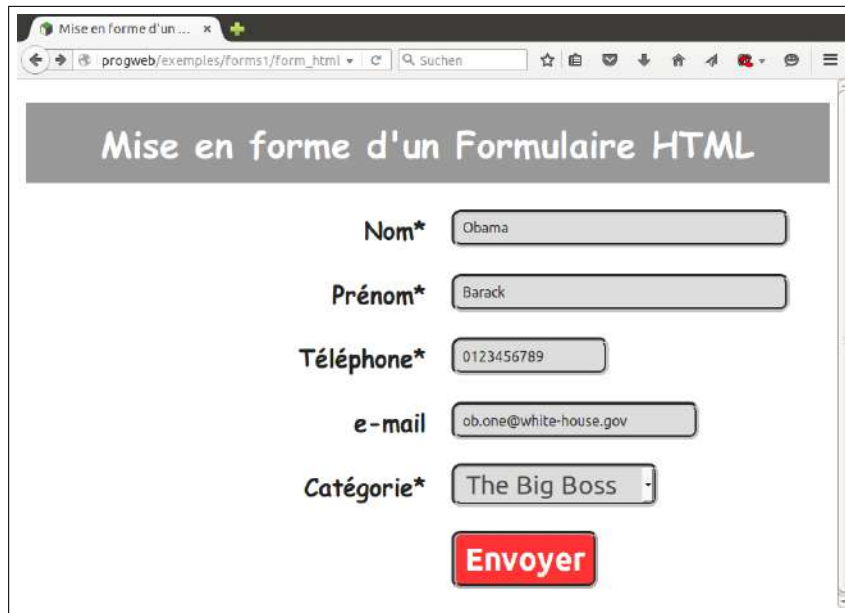
3.2 Exemple de style CSS pour formulaire

exemples/forms1/myStyle.css

```

1 /* style par défaut du texte */
2 body {
3   font-family : "Comic Sans MS";
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    min-width : 800px;
26  }
27
28  /* tous les labels ont la même largeur pour aligner les inputs */
29  form p label {
30    float : left;
31    width : 400px;
32    text-align : right;
33    padding : 6px;
34    font-weight : bold;
35  }
36
37  form p input {
38    padding : 6px;
39    margin-left : 20px;
40    background-color : #ddd;
41    border-style : groove;
42    border-width : 5px;
43    border-color : #444;
44    border-radius :10px;
45  }
46

```

```
47 form p select {
48   padding : 1px 6px;
49   margin-left : 20px;
50   background-color : #ddd;
51   border-style : groove;
52   border-width : 5px;
53   border-color : #444;
54   border-radius :10px;
55   font-size : 110%;
56 }
57
58 /* input spécial pour le bouton submit */
59 form p input.sansLabel {
60   margin-left : 432px; /* 400+6+6+20 : aligné sur les autres inputs */
61   font-size : 130%;
62   font-weight : bolder;
63   background-color :#ff3333;
64   color :white;
65 }
66
67 form p {
68   background-color : #fff;
69   padding : 0px;
70 }
71
72 form p span.errorMsg {
73   color :red;
74   font-weight :bolder;
75 }
76
77 /******
78
79
80 /* style par défaut des liens */
81 a :link {
82   text-decoration : underline; /* souligné */
83   color : #00e;
84 }
85
86 /* style des liens visités */
87 a :visited {
88   text-decoration : underline; /* souligné */
89   color : #00c; /* bleu clair */
90 }
91
92 /* style des liens visités */
93 a :hover {
94   text-decoration : underline; /* souligné */
95   color : #e40; /* rouge vif */
96 }
97
98 /* style des éléments importants */
99 strong {
100   font-variant : small-caps;
101   font-weight : bolder;
102   color : black;
```

```
103 }
104
105 /* style des éléments mis en évidence */
106 em {
107     font-style : italic;
108     color : black;
109 }
110
111 p {
112     background-color : #ddd;
113     text-align : justify;
114     padding : 5pt;
115 }
```

exemples/forms1/myStyle.css

```
1 /* style par défaut du texte */
2 body {
3     font-family : "Comic Sans MS";
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     min-width : 800px;
26 }
27
28 /* tous les labels ont la même largeur pour aligner les inputs */
29 form p label {
30     float : left;
31     width : 400px;
32     text-align : right;
33     padding : 6px;
34     font-weight : bold;
35 }
36
37 form p input {
38     padding : 6px;
39     margin-left : 20px;
40     background-color : #ddd;
```

```
41 border-style : groove;
42 border-width : 5px;
43 border-color : #444;
44 border-radius :10px;
45 }
46
47 form p select {
48 padding : 1px 6px;
49 margin-left : 20px;
50 background-color : #ddd;
51 border-style : groove;
52 border-width : 5px;
53 border-color : #444;
54 border-radius :10px;
55 font-size : 110%;
56 }
57
58 /* input spécial pour le bouton submit */
59 form p input.sansLabel {
60 margin-left : 432px; /* 400+6+6+20 : aligné sur les autres inputs */
61 font-size : 130%;
62 font-weight : bolder;
63 background-color :#ff3333;
64 color :white;
65 }
66
67 form p {
68 background-color : #fff;
69 padding : 0px;
70 }
71
72 form p span.errorMsg {
73 color :red;
74 font-weight :bolder;
75 }
76
77 /******
78
79
80 /* style par défaut des liens */
81 a:link {
82 text-decoration : underline; /* souligné */
83 color : #00e;
84 }
85
86 /* style des liens visités */
87 a:visited {
88 text-decoration : underline; /* souligné */
89 color : #00c; /* bleu clair */
90 }
91
92 /* style des liens visités */
93 a:hover {
94 text-decoration : underline; /* souligné */
95 color : #e40; /* rouge vif */
96 }
```

```

97
98 /* style des éléments importants */
99 strong {
100     font-variant : small-caps;
101     font-weight : bolder;
102     color : black;
103 }
104
105 /* style des éléments mis en évidence */
106 em {
107     font-style : italic;
108     color : black;
109 }
110
111 p {
112     background-color : #ddd;
113     text-align : justify;
114     padding : 5pt;
115 }

```

3.2.1 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**.

exemples/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 ?>

```

3.3 Validation pour la sécurité : Appel de `filter_var`

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

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

exemples/forms1/ex04_validation.php

```
1 <?php
2   require_once ("ex03_validUtils.php");
3
4   $dataErrors = array();
5
6   // validation du nom
7   $nom = filter_var($nom, getSanitizeFilter('string'));
8
9   // validation du prénom
10  $prenom = filter_var($prenom, getSanitizeFilter('string'));
11
12  // validation du téléphone
13  $telephone = filter_var($telephone, getSanitizeFilter('string'));
14
15  // validation de l'adresse e-mail
16
17  if (filter_var($email, getValidateFilter('email'))===false){
18    $dataErrors['email'] = "Erreur : l'adresse e-mail est invalide.";
19  }
20
21  // validation de la catégorie
22  $categorie = filter_var($categorie, getSanitizeFilter('string'));
23 >
```

exemples/forms1/ex03_validUtils.php

```
1 <?php
2 // Méthode retournant le filtre de validation à utiliser
3 // dans la fonction filter_var
4 function getValidateFilter($type)
5 {
6   switch($type){
7     case "email":
8       $filter = FILTER_VALIDATE_EMAIL;
9       break;
10    case "int":
11      $filter = FILTER_VALIDATE_INT;
12      break;
13    case "boolean":
14      $filter = FILTER_VALIDATE_BOOLEAN;
15      break;
16    case "ip":
17      $filter = FILTER_VALIDATE_IP;
18      break;
19    case "url":
20      $filter = FILTER_VALIDATE_URL;
21      break;
22    default : // important !!!
23      $filter = false; // Si type est faux, la valid. échoue.
24  }
25  return $filter;
26 }
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
48 ?>

```

3.4 Appel des vues

Comme nous l'avons vu dans la partie 3.2.1, un test permet, à la suite de 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.



exemples/forms1/ex05_vueSuccess.php

```

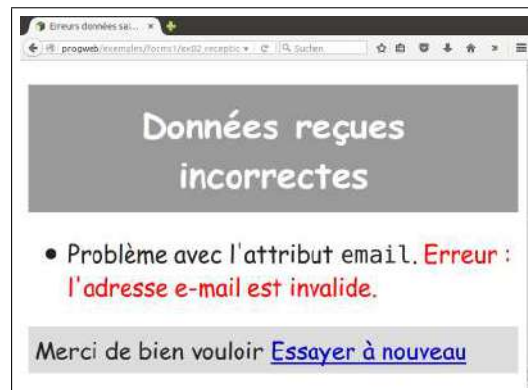
1 <?php
2     require_once(dirname(__FILE__) . '/commonFunctions.php');
3
4     outputEnTeteHTML5('Affichage des données saisies', 'UTF-8', 'myStyle.css');
5
6     echo "<h1>Données reçues</h1>\n";

```

```

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 ?>

```



exemples/forms1/ex06_vueError.php

```

1 <?php
2     require_once(dirname(__FILE__). '/commonFunctions.php ');
3     outputEnTeteHTML5('Erreurs données saisies', 'UTF-8', 'myStyle.css ');
4
5     echo "<h1>Données reçues incorrectes</h1>\n";
6
7     echo "<ul>";
8     foreach ($dataErrors as $field => $message){
9         echo '<li>Problème avec l\'attribut <code>'. $field
10            . '</code>. <span style="color : red;">'. $message. '</span></li>';
11     }
12     echo "</ul>";
13
14     echo '<p>Merci de bien vouloir <a href="ex01_form_html.html">Essayer à nouveau
15        </a></p>';
16
17     outputFinFichierHTML5();
18 ?>

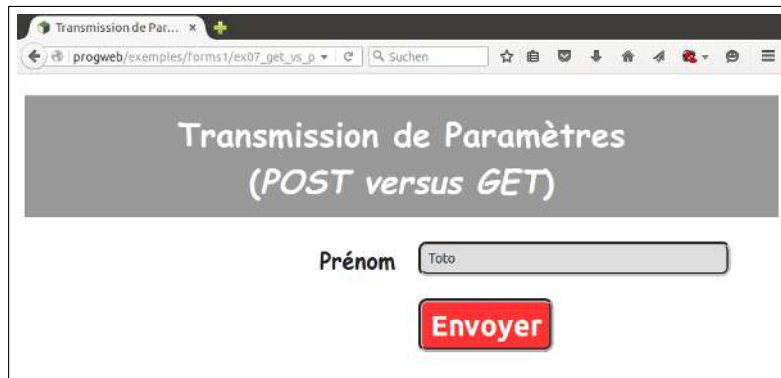
```

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

3.5 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`.



exemples/forms1/ex07_get_vs_postHidden.php

```

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>

```

exemples/forms1/ex08_get_post_request_param.php

```

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     }
15     foreach ($_POST as $key => $val){
16         echo htmlentities ("\"$_POST['\".$key.\"'] = \".$val , ENT_COMPAT, \"UTF-8\")." <br
17         />";
18     }
19     foreach ($_REQUEST as $key => $val){
20         echo htmlentities ("\"$_REQUEST['\".$key.\"'] = \".$val , ENT_COMPAT, \"UTF-8\")."
21         <br/>";
22     }
23     ?>
24 </body>
25 </html>

```

3.6 Formulaires dynamiques an javascript

Nous voyons ici un exemple d'utilisation du *Javascript* pour cr er un formulaire dont les attributs d pendent de la valeur d'un premier champ. Lorsqu'on s lectionne "deuxi me ann e", un nouveau champ appara t. Pour cel , on utilise l' v nement `onchange` sur l'`input` de l'ann e, qui est g r  par la fonction `anneeChange`. On teste alors la valeur de l'attribut, puis le cas  ch ant on g n re 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.

exemples/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)"
14 <option value="choisissez" selected disabled>— choisissez --</option>
15 <option value="premiere">Première année</option>
16 <option value="deuxieme">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 == "deuxieme"){
31 paragraphe.innerHTML+="<label>Orientation prévue pour l'année prochaine
32 :</label>"
33 + '<select name="orientation" id="orientation">'
34 + '<option value="LP">LP</option>'
35 + '<option value="master">master</option>'
36 + "<option value=\"\tinge\">Ecole d'ingé</option>"
37 + '<option value="boulot">Boulot</option>'
38 + '<option value="autre">Autre</option>'
39 + '</select>';
40 }
41 }
42 anneeChange();
43 </script>
44 </body>
45 </html>

```

exemples/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"])) ? $_POST["annee"] : "année indéterminée";
11      echo "Nom : ".$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

Injection, 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 "description", 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

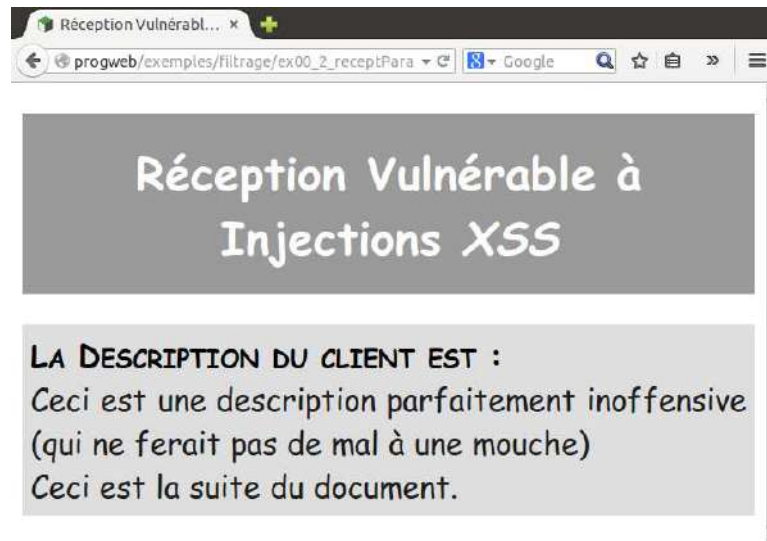


FIGURE 4.2 : Le site affiche en *HTML* les données saisies dans le gentil formulaire

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

exemples/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;">
12      Description :</label>
13    <textarea name="description" id="description" cols="50" row="6" style="
14      vertical-align: top;">
15    </textarea>
16    <input type="submit" value="Envoyer"/>
17  </form>
18 </body>
19 </html>

```

exemples/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 &nbsp;&nbsp;&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 :

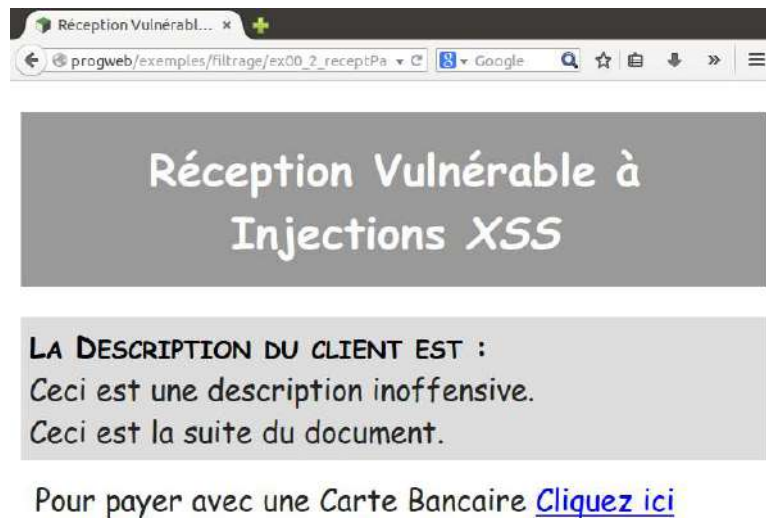


FIGURE 4.4 : L'affichage des données du formulaire sort le code *HTML* entré par le pirate

4.1.2 Prévention des injections *HTML* par échappement

4.1.2.a Échappement par `htmlspecialchars`

Différents outils de filtrage sont disponibles en PHP. Le plus simple pour la sécurité consiste à utiliser la méthode `htmlspecialchars` qui transforme 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 caractères 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 `htmlspecialchars` ne fonctionnera pas bien, laissant la porte ouverte à des attaques. On peut spécifier l'encodage de sortie de `htmlspecialchars` dans son troisième paramètre.

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

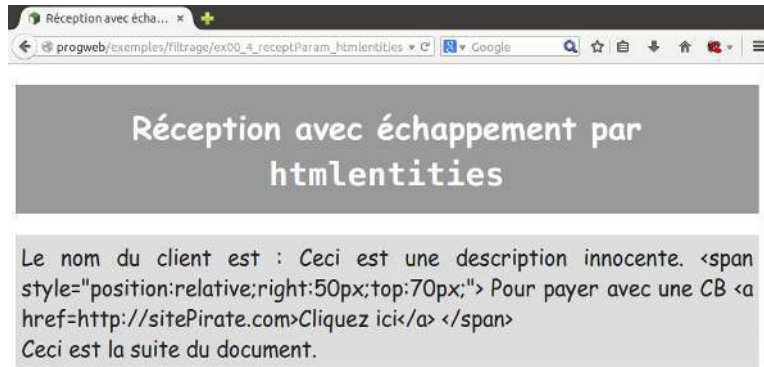


FIGURE 4.5 : Un gentil formulaire

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

exemples/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>htmlspecialchars</code></h1>
9   <p>
10     Le nom du client est :
11     Ceci est une description innocente.
12     &lt;span style="position: relative; right: 50px; top: 70px;"&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 verrons 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 chaînes positées sont les suivantes :

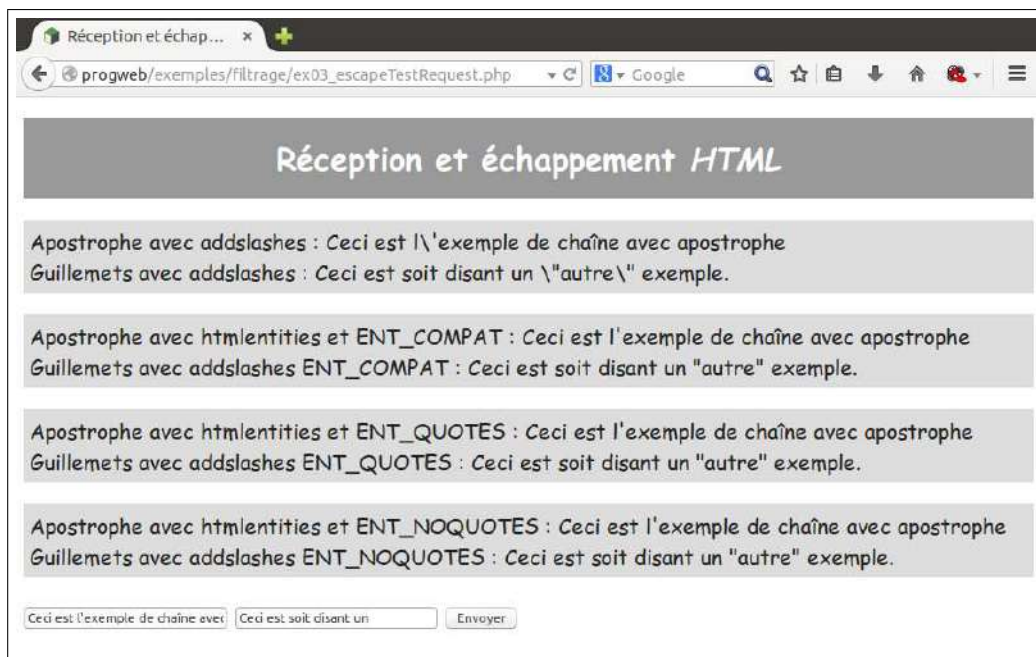
exemples/filtrage/ex02_postParam.php

```

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="chaîne1" value="Ceci est l'exemple de chaîne avec
        apostrophe"/>
12    <input type="hidden" name="chaîne2" value="Ceci est soit disant un 'autre';
        autre' 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` :



exemples/filtrage/ex03_escapeTestRequest.php

```

1 <?php require( './commonFunctions.php' );
2
3     outputEnTeteHTML5( 'Réception et échappement HTML', 'UTF-8', 'myStyle.css' );
4
5     echo "<h1>Réception et échappement <i>HTML</i></h1>";
6
7     $chaîne1 = $_REQUEST[ 'chaîne1' ];
8     $chaîne2 = $_REQUEST[ 'chaîne2' ];
9

```

```

10  echo "<p>\n";
11  echo "Apostrophe avec addslashes : ".addslashes($chaine1). "<br>\n";
12  echo "Guillemets avec addslashes : ".addslashes($chaine2). "<br>\n";
13  echo "</p>\n";
14
15  echo "<p>\n";
16  echo "Apostrophe avec htmlentities et ENT_COMPAT : ".htmlentities($chaine1,
    ENT_COMPAT, 'UTF-8', false). "<br>\n";
17  echo "Guillemets avec addslashes ENT_COMPAT : ".htmlentities($chaine2,
    ENT_COMPAT, 'UTF-8', false). "<br>\n";
18  echo "</p>\n";
19
20  echo "<p>\n";
21  echo "Apostrophe avec htmlentities et ENT_QUOTES : ".htmlentities($chaine1,
    ENT_QUOTES, 'UTF-8', false). "<br>\n";
22  echo "Guillemets avec addslashes ENT_QUOTES : ".htmlentities($chaine2,
    ENT_QUOTES, 'UTF-8', false). "<br>\n";
23  echo "</p>\n";
24
25  echo "<p>\n";
26  echo "Apostrophe avec htmlentities et ENT_NOQUOTES : ".htmlentities($chaine1,
    ENT_NOQUOTES, 'UTF-8', false). "<br>\n";
27  echo "Guillemets avec addslashes ENT_NOQUOTES : ".htmlentities($chaine2,
    ENT_NOQUOTES, 'UTF-8', false). "<br>\n";
28  echo "</p>";
29
30  ?>
31  <form method="post" action="ex03_escapeTestRequest.php">
32  <input type="text" name="chaine1" value="<?php echo $chaine1;?>">
33  <input type="text" name="chaine2" value="<?php echo $chaine2;?>">
34  <input type="submit" value="Envoyer" class="sansLabel"></input>
35  </form>
36  <?php
37  outputFinFichierHTML5();
38  ?>

```

Le code source *HTML* g n r  par le *CGI* est le suivant :

exemples/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>R ception et  chappement HTML</title>
7  </head>
8  <body>
9  <h1>R ception et  chappement <i>HTML</i></h1><p>
10  Apostrophe avec addslashes : Ceci est l' exemple de cha ne 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 ne
    avec apostrophe<br>
15  Guillemets avec addslashes ENT_COMPAT : Ceci est soit disant un "autre"
    ; exemple.<br>

```

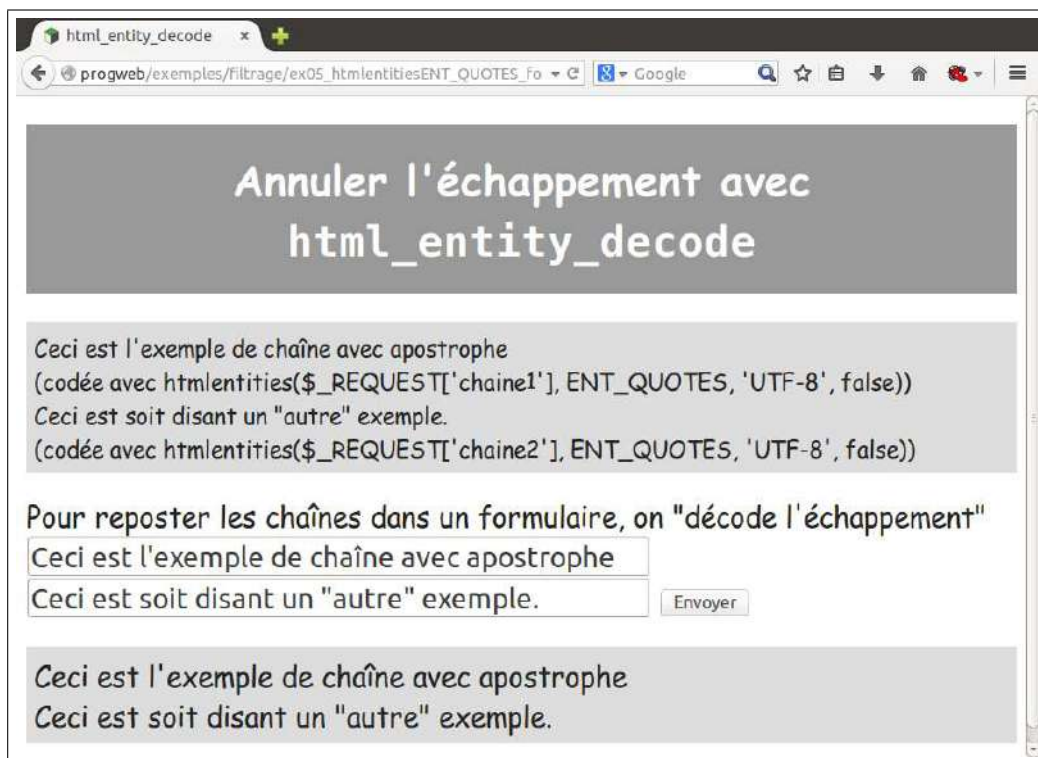
```

16 </p>
17 <p>
18 Apostrophe avec htmlspecialchars et ENT_QUOTES : Ceci est l'&#039;exemple de chaîne
    icirc;ne avec apostrophe<br>
19 Guillemets avec htmlspecialchars ENT_QUOTES : Ceci est soit disant un &quot;autre&quot
    ; exemple.<br>
20 </p>
21 <p>
22 Apostrophe avec htmlspecialchars et ENT_QUOTES : Ceci est l'exemple de chaîne;
    ne avec apostrophe<br>
23 Guillemets avec htmlspecialchars ENT_QUOTES : 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 brutes d'origine, par exemple pour renvoyer ces données dans les inputs d'un formulaire :



exemples/filtrage/ex05_htmlentitiesENT_QUOTES_formInput.php

```

1 <?php require( './commonFunctions.php' );
2
3     outputEnTeteHTML5( 'html_entity_decode', 'UTF-8', 'myStyle.css' );
4
5     echo "<h1>Annuler l'échappement avec <code>html_entity_decode</code></h1>";
6
7     $chaine1 = htmlentities( $REQUEST[ 'chaine1' ], ENT_QUOTES, 'UTF-8', false );
8     $chaine2 = htmlentities( $REQUEST[ 'chaine2' ], ENT_QUOTES, 'UTF-8', false );
9
10    echo "<p style=|\"font-size :80%;|\">|n\";
11    echo $chaine1. "<br/>(codée avec ". htmlentities( "htmlentities( |$REQUEST[ '
12    chaine1 ' ], ENT_QUOTES, 'UTF-8', false )", ENT_QUOTES, 'UTF-8', false ). )" <br
13    >|n\";
14    echo $chaine2. "<br/>(codée avec ". htmlentities( "htmlentities( |$REQUEST[ '
15    chaine2 ' ], ENT_QUOTES, 'UTF-8', false )", ENT_QUOTES, 'UTF-8', false ). )" <br
16    >|n\";
17    echo "</p>Pour reposer les chaînes dans un formulaire, on |\"décode l'é
18    chappement|\"<br/>|n\";
19    ?>
20    <form method="post" action="ex05_htmlentitiesENT_QUOTES_formInput.php">
21    <input style="font-size :100%;" type="text" name="chaine1" value="<?php echo
22    $chaine1;?>" size="30"/>
23    <input style="font-size :100%;" type="text" name="chaine2" value="<?php echo
24    $chaine2;?>" size="30"/>
25    <input type="submit" value="Envoyer" class="sansLabel"/>
26    </form>
27    <?php
28    echo "<p>|n\";
29    echo html_entity_decode( $chaine1, ENT_QUOTES, 'UTF-8' ). " <br>|n\";
30    echo html_entity_decode( $chaine2, ENT_QUOTES, 'UTF-8' ). " <br>|n\";
31    echo "</p>|n\";
32
33    outputFinFichierHTML5( );
34    ?>

```

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 chaîne (`varchar`) dans une table `Table1`.

exemples/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>

```

exemples/filtrage/ex07_exInjectMySql.php

```

1 <?php
2 include( './commonFunctions.php ' );
3 outputEnTeteHTML5( 'Exemple d\'injection SQL', 'UTF-8', 'myStyle.css' );
4
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'hôte 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 : ' . mysqli_error
32     ( ). "<br/>" );
33
34 // On ferme la connexion
35 $mysqli->close ( );
36
37 outputFinFichierHTML5 ( );
38 ?>

```


Voici deux exemple d'injections exploitant l'absence de filtrage dans ce code. Le premier exemple, gentillet, consiste juste   ins rer deux donn es au lieu d'une dans la table :



FIGURE 4.6 : L' tat des donn es avant l'injection SQL par le pirate



FIGURE 4.7 : Donn es saisies par le pirate pour l'injection SQL

Le deuxi me exemple, plus m chant, consiste pour le pirate   supprimer toutes les donn es contenues dans la table :

exemples/filtrage/ex09_mySqlEscapeString.php

```
1 <?php
2 include( './commonFunctions.php ' );
3 outputEnTeteHTML5( 'Echappement mysql: :real_escape_string', 'UTF-8', 'myStyle.
4 css ' );
5 echo "<h1>R ception avec  chappement <code>mysql: :real_escape_string</code></
6 h1>";
7 // On se connecte au serveur de bases de donn es.
8 // Adapter le nom d'hote o  se trouve le serveur mysql
9 // mysql($sql_server_url, $sql_user, $sql_user_password, $database_name)
10 $mysqli = new mysqli( 'progweb', "testUser", "motdepasse", 'baseTest' );
11
12 // V rification de la connexion :
13 if ( mysqli_connect_errno() ) {
```



FIGURE 4.8 : Requête exécutée lors de l'injection SQL

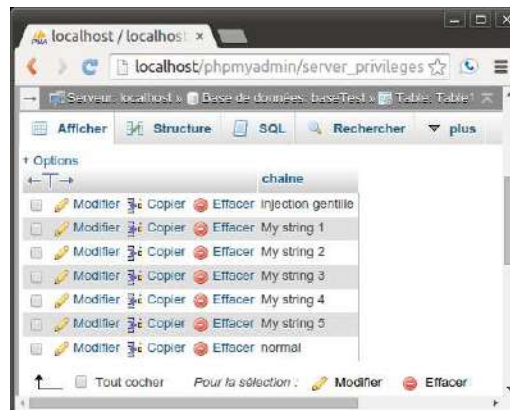


FIGURE 4.9 : L'état des données après l'injection SQL par le pirate



FIGURE 4.10 : Données saisies par le pirate pour l'injection SQL



FIGURE 4.11 : Requête exécutée lors de l'injection SQL



FIGURE 4.12 : L' tat des donn es pr s l'injection SQL par le pirate



```

14     printf(" chec de la connexion : %s\n", mysqli_connect_error());
15     exit();
16 }
17 echo 'Connect  au serveur de bases de donn es mysql.<br/>';
18
19 $chaine1="";
20 if (isset($_POST['chaine1'])){
21     $chaine1 = $mysqli->real_escape_string($_POST['chaine1']);
22 }
23 echo "Cha ne entr e par l'utilisateur : <code>".$_POST['chaine1']. "</code><br
    />";
24 echo "Cha ne entr e par l'utilisateur  chap e : <code>".$chaine1."</code><br/>
    ";
25
26 // Insertion de la cha ne dans la table Table1 (requ te SQL) :
27 $requete = 'INSERT INTO Table1(chaine) VALUES ("'.$chaine1.'")';
28
29 echo "Requ te ex cut e :<br/><code>".$requete."</code><br/>";
30
31 $result = $mysqli->multi_query($requete) or die('Query failed : ' . mysql_error
    (')." <br/>");
32
33 // On ferme la connexion
34 $mysqli->close();

```

```

35
36 outputFinFichierHTML5();
37 ?>

```

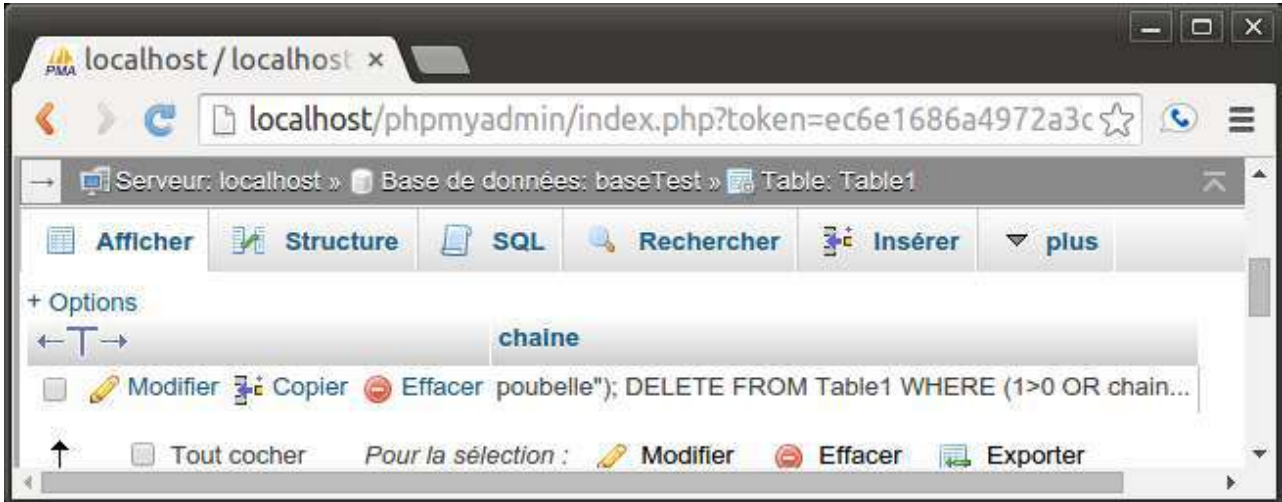


FIGURE 4.13 : Données dans la base après tentative d'injection SQL avec échappement



exemples/filtrage/ex11_mySqlEscapeStringOutput.php

```

1 <?php
2 include( './commonFunctions.php ');
3 outputEnTeteHTML5( 'En sortie de BD', 'UTF-8', 'myStyle.css ');
4
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'hôte 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[ '
26         chaine '], ENT_QUOTES, 'UTF-8'). "</code><br/>";
27 }
28 // On ferme la connexion
29 $mysqli->close ();
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 chaîne de caractères attendue suivant son usage (exemple : adresse e-mail, URL, nombre réel, adresse IP, etc.).
2. de nettoyer une chaîne 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 chaque'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 cela 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 conforme 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 contre-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 chaîne 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 chaînes de caractères représentant un booléen)!

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

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).

exemples/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 être 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 correctes 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'empêche 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 différents 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).

exemples/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       )){
14       return $tel;
15     }else{
16       return false;
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 PCRE 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-ZàâéêèèôùûçÀÃÊËÏÏÛÇäöëüÄÖËÜ] (\-| ( )*))){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 chaîne de caractères accentués (au moins en français) peut se faire un plus systématiquement 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 :



exemples/filtrage/ex14_regexAccents.php

```

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     $chaine = htmlentities("à à è è À È É Ê ed atJJê ÊËä æ œ | " ' ' .", ENT_QUOTES);
12     echo "Chaîne ". htmlentities($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]
15             ]uml \; ) ' // caractères accentués
16         . ' / ( [a-zA-Z] cedil \; ) / ( [a-zA-Z] [a-zA-Z] lig \; ) / ( [a-zA-Z] szlig \; ) / ( [a-zA-Z]
17             tilde \; ) ' // caractères accentués
18         . ' / ( \- ) / ( ) / ( \#039 \; ) / ( \ quot \; ) / ( \. ) + $ / ', // Espaces, trait d'
19             union, simples et doubles quotes
20         $chaine)) {
21         echo " <strong>valide </strong>";
22     } else {
23         echo " <strong>invalide </strong>";
24     }
25 }
26 ?>
27 </body>
28 </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

Formulaires *PHP/HTML*, filtrage, exceptions

Dans ce chapitre, nous proposons une conception de classes métiers et classes d'utilitaires pour générer des vues *HTML*. La gestion des données saisies par l'utilisateur nous conduit, au vu du chapitre 4, à gérer plus rigoureusement le filtrage.

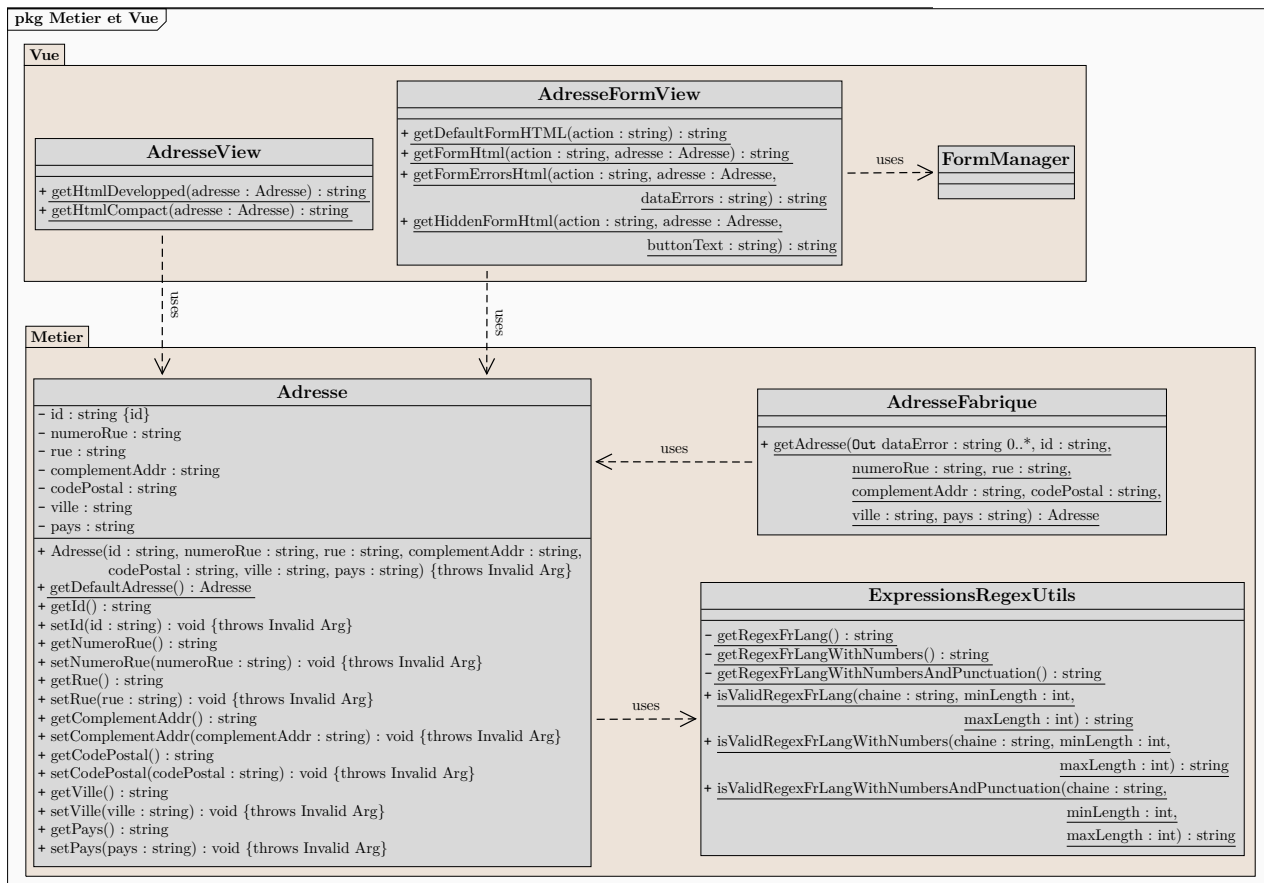
Nous distinguons **deux types de filtrage des données** :

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 une connaissance de la sémantique des objets métiers et de leurs attributs, est le plus souvent basé sur des tests d'expressions régulières. Dans notre implémentation, ce filtrage sera réalisé dans les *setters* des classes de représentation des données métier.

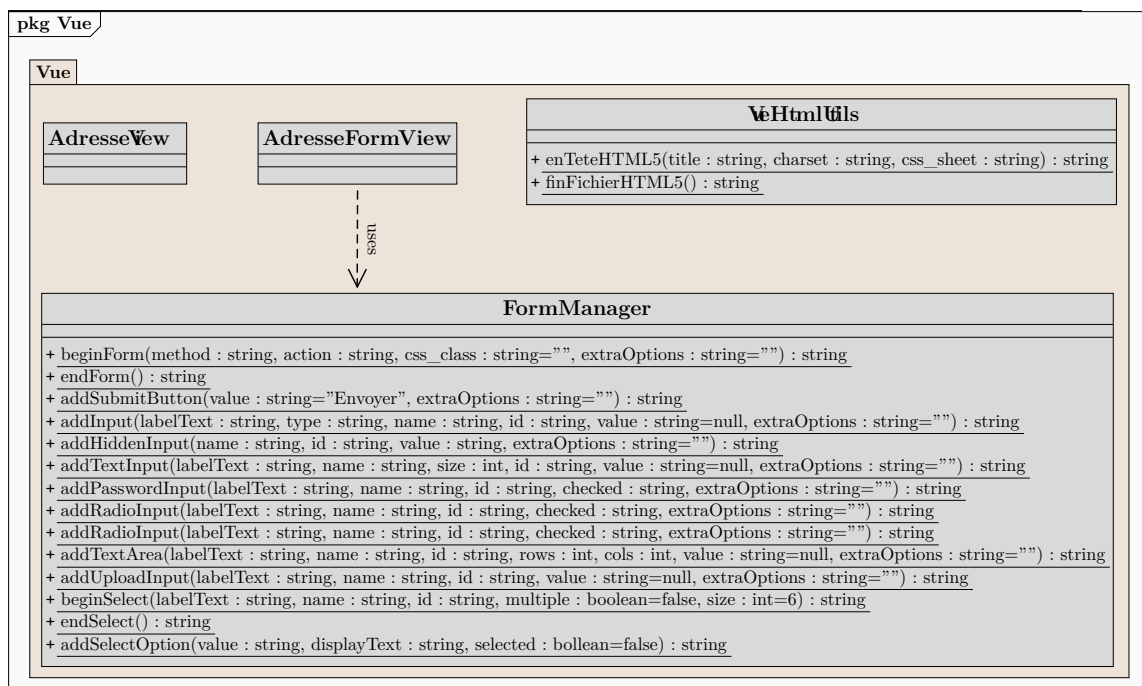
5.1 Modélisation : Diagrammes de Classes

Nous reprenons à peu près la classe adresse de la partie 2.2.2, 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



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

5.2 La Classe Adresse

Nous ajoutons :

1. Un filtrage pour la cohérence des données un peu plus réaliste au niveau des setters ;
2. Une méthode statique permettant d'avoir une adresse par défaut (exemple : les champs vides pour initialiser un formulaire en utilisant la classe `AdresseFormView` ci-dessous).

exemples/forms2/ex01_classeAdresse.php

```
1 <?php
2 namespace CoursPHP\Metier;
3
4 ///////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
5 // Classe Adresse :
6
7 /**
8  * @brief La classe adresse contient l'adresse d'une personne
9  * (qui peut être un client, un employé, un fournisseur, etc...)
10  */
11 class Adresse {
12     /** id unique de l'adresse */
13     private $id;
14     /** Numéro dans la rue, ne doit pas être null mais peut être vide */
15     private $numeroRue;
16     /** Nom de la rue, ne doit pas être null mais peut être vide */
17     private $rue;
18     /** Complément (lieu dit, etc. ne doit pas être null mais peut être vide */
19     private $complementAddr;
20     /** code postal */
21     private $codePostal;
22     /** nom de la ville. ne doit pas être null mais peut être vide*/
23     private $ville;
24     /** nom du pays. ne doit pas être null mais peut être vide*/
25     private $pays;
26
27     // Inclusion du trait AdresseProperties définissant les accesseurs et setters
28     use AdresseProperties;
29
30     /**
31      * @brief Génère 10 chiffres hexa aléatoires (soit 5 octets) :
32      */
33     private static function generateRandomId()
34     {
35         // Génération de 5 octets (pseudo-)aléatoires codés en hexa
36         $cryptoStrong = false; // Variable pour passage par référence
37         $octets = openssl_random_pseudo_bytes(5, $cryptoStrong);
38         return bin2hex($octets);
39     }
40
41     /**
42      * @brief Constructeur : initialise les attributs à partir des paramètres.
43      * Les paramètres correspondent aux valeurs à mettre dans les attributs.
44      * Tout objet doit être initialisé avec le constructeur (appel à new).
45      * Des exceptions sont rejetées en cas de paramètre invalide.
```

```

46  */
47  public function __construct($id, $numeroRue, $rue, $complementAddr,
48      $codePostal, $ville, $pays) {
49      $this->setId($id);
50      $this->setNumeroRue($numeroRue);
51      $this->setRue($rue);
52      $this->setComplementAddr($complementAddr);
53      $this->setCodePostal($codePostal);
54      $this->setVille($ville);
55      $this->setPays($pays);
56  }
57  /**
58   * @brief construit une Adresse par défaut
59   * Remarque : on ne peut pas surcharger les constructeurs en PHP 5...
60   */
61  public static function getDefaultAdresse(){
62      // On appelle le constructeur avec des arguments convenables
63      // en terme d'expressions régulières
64      $adresse = new Adresse(self::generateRandomId(), '2', 'Rue', "", "12345",
65          "Ville", "Pays");
66      $adresse->numeroRue = "";
67      $adresse->rue = "";
68      $adresse->complementAddr = "";
69      $adresse->codePostal = "";
70      $adresse->ville = "";
71      $adresse->pays = "France";
72      return $adresse;
73  }
74  ?>

```

5.3 Filtrage des attributs

Nous proposons tout d'abord des méthodes génériques de test d'expressions régulières pour la langue française, avec ou sans chiffres. Ces expressions régulières doivent être testées après échappement par `htmlentities`, ce qui permet une approche générique du traitement des accents (exemple : `é`), des apostrophes, guillemets (simple ou doubles quotes), etc. qui peuvent apparaître en français.

exemples/forms2/ex02_expressionsRegexUtils.php

```

1  <?php
2  namespace CoursPHP\Metier;
3
4  //////////////////////////////////////
5  // Expressions régulières utiles.
6  class ExpressionsRegexUtils{
7      /**
8       * @brief : expression régulière pour la langue Française avec accents
9       * @warning La chaîne doit être échappée par htmlentities
10     */
11     private static function getRegexFrLang(){
12         return '/^[a-zA-Z]';

```

```

13     . '(\&[a-zA-Z]grave\;)/(\&[a-zA-Z]acute\;)/(\&[a-zA-Z]circ\;)'
14     . '(\&[a-zA-Z]uml\;)/(\&[a-zA-Z]cedil\;)/(\&[a-zA-Z][a-zA-Z]lig\;)'
15     . '(\&szlig\;)/(\&[a-zA-Z]tilde\;)/(\-)/(\ )/(\&amp\;|#39\;)'
16     . '(\&#039\;)/(\&amp\;|#34\;)/(\&#034\;)/(\&quot\;)/(\.))*$/';
17 }
18
19 /**
20  * @brief : expression régulière pour la langue Française avec accents,
21  * et chiffres
22  * @warning La chaîne doit être échappée par htmlentities
23  */
24 private static function getRegexFrLangWithNumbers(){
25     return '/^[a-zA-Z0-9]'
26         . '(\&[a-zA-Z]grave\;)/(\&[a-zA-Z]acute\;)/(\&[a-zA-Z]circ\;)'
27         . '(\&[a-zA-Z]uml\;)/(\&[a-zA-Z]cedil\;)/(\&[a-zA-Z][a-zA-Z]lig\;)'
28         . '(\&szlig\;)/(\&[a-zA-Z]tilde\;)/(\-)/(\ )/(\&amp\;|#39\;)'
29         . '(\&#039\;)/(\&amp\;|#34\;)/(\&#034\;)/(\&quot\;)/(\.))*$/';
30 }
31
32 /**
33  * @brief : expression régulière pour la langue Française avec accents,
34  * chiffres et ponctuation
35  * @warning La chaîne doit être échappée par htmlentities
36  */
37 private static function getRegexFrLangWithNumbersAndPunctuation(){
38     return '/^[a-zA-Z0-9]'
39         . '(\&[a-zA-Z]grave\;)/(\&[a-zA-Z]acute\;)/(\&[a-zA-Z]circ\;)'
40         . '(\&[a-zA-Z]uml\;)/(\&[a-zA-Z]cedil\;)/(\&[a-zA-Z][a-zA-Z]lig\;)'
41         . '(\&szlig\;)/(\&[a-zA-Z]tilde\;)/(\-)/(\ )/(\&amp\;|#39\;)'
42         . '(\&#039\;)/(\&amp\;|#34\;)/(\&#034\;)/(\&quot\;)/(\.)'
43         . '(!)|(|?)|(|:)|(|;)|(|,)|(|()|(|)))*/';
44 }
45
46 /**
47  * @brief : Test expression régulière pour la langue Française avec accents
48  * avec conditions de longueur (par exemple pour un champ obligatoire)
49  */
50 public static function isValidRegexFrLang($chaine, $minLength, $maxLength){
51     return (isset($chaine) &&
52         strlen($chaine) >= $minLength && strlen($chaine) <= $maxLength
53         && preg_match(self::getRegexFrLang(), $chaine));
54 }
55
56 /**
57  * @brief : Test expression régulière pour la langue Française avec accents
58  * et chiffres
59  * avec conditions de longueur (par exemple pour un champ obligatoire)
60  */
61 public static function isValidRegexFrLangWithNumbers($chaine,
62     $minLength, $maxLength){
63     return (isset($chaine) &&
64         strlen($chaine) >= $minLength && strlen($chaine) <= $maxLength
65         && preg_match(self::getRegexFrLangWithNumbers(), $chaine));
66 }
67
68 /**

```

```

69  * @brief : Test expression régulière pour la langue Française avec accents ,
70  * chiffres et ponctuation
71  * avec conditions de longueur (par exemple pour un champ obligatoire)
72  */
73  public static function isValidRegexFrLangWithNumbersAndPunctuation($chaine ,
74                                                                     $minLength, $maxLength){
75      return (isset($chaine) &&
76              strlen($chaine) >= $minLength && strlen($chaine) <= $maxLength
77              && preg_match(self::getRegexFrLangWithNumbersAndPunctuation(), $chaine));
78  }
79
80  /**
81  * @brief : Test expression régulière passée en paramètre
82  * avec conditions de longueur (par exemple pour un champ obligatoire)
83  */
84  public static function isValidString($chaine, $regex, $minLength, $maxLength)
85  {
86      return (isset($chaine) &&
87              strlen($chaine) >= $minLength && strlen($chaine) <= $maxLength
88              && preg_match($regex, $chaine));
89  }
90  ?>

```

Dans notre implémentation, nous réalisons les tests de cohérence des données par expressions régulières au niveau des *setters* des classes métier. Nous réalisons au préalable, dans le setter, un échappement par `htmlspecialchars`, de manière à pouvoir gérer les apostrophes ou guillemets, qui seraient sinon rejetées lors du filtrage destiné à éviter les injections *SQL* (par exemple dans les requêtes préparées *PDO*). Cet échappement par `htmlspecialchars` nous permet aussi d'utiliser les expressions régulières génériques ci-dessus.

exemples/forms2/ex03_AdresseProperties.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   */
7  trait AdresseProperties {
8
9   * @brief Accesseur : permet d'obtenir l'id de l'adresse. */
10 public function getId() {
11     return $this->id;
12 }
13
14 * @brief Accesseur : permet d'obtenir le numéro dans la rue. */
15 public function getNumeroRue() {
16     return htmlspecialchars($this->numeroRue, ENT_QUOTES, "UTF-8");
17 }
18
19 * @brief Accesseur : permet d'obtenir le nom la rue. */
20 public function getRue() {
21     return htmlspecialchars($this->rue, ENT_QUOTES, "UTF-8");
22 }
23

```



```
24  /** @brief Accesseur : permet d'obtenir le nom le complément d'adresse. */
25  public function getComplementAddr() {
26      return html_entity_decode($this->complementAddr, ENT_QUOTES, "UTF-8");
27  }
28
29  /** @brief Accesseur : permet d'obtenir le nom le code postal. */
30  public function getCodePostal() {
31      return html_entity_decode($this->codePostal, ENT_QUOTES, "UTF-8");
32  }
33
34  /** @brief Accesseur : permet d'obtenir le nom la ville. */
35  public function getVille() {
36      return html_entity_decode($this->ville, ENT_QUOTES, "UTF-8");
37  }
38
39
40  /** @brief Accesseur : permet d'obtenir le pays. */
41  public function getPays() {
42      return html_entity_decode($this->pays, ENT_QUOTES, "UTF-8");
43  }
44
45  /**
46   * @brief setter : permet d'initialiser ou de modifier l'ID.
47   * @param $id l'identifiant de l'adresse (10 chiffres hexa).
48   */
49  public function setId($id) {
50      if (!isset($id) || !preg_match("/^[0-9a-f]{10}$/", $id)){
51          throw new \Exception("Erreur, identifiant \"\".$id.\"\" incorrect");
52      }
53      $this->id = $id;
54  }
55
56  /** @brief setter : permet d'initialiser ou de modifier le nom de la rue.
57   * @param $NumeroRue le numéro à utiliser. peut être null.
58   */
59  public function setNumeroRue($numeroRue) {
60      $escaped = htmlentities($numeroRue, ENT_QUOTES, 'UTF-8');
61      if (!ExpressionsRegexUtils::isValidRegexFrLangWithNumbers($escaped, 0, 50)){
62          throw new \Exception("Erreur, le numéro de la rue doit comporter "
63              . "au plus 50 caractères"
64              . "(alphabétiques, chiffres ou ponctuation)");
65      }
66
67      $this->numeroRue = empty($escaped) ? "" : $escaped;
68  }
69
70  /** @brief setter : permet d'initialiser ou de modifier le numéro dans la rue.
71   * @param $Rue le nom de la rue ou de la place à utiliser. peut être null.
72   */
73  public function setRue($rue) {
74      $escaped = htmlentities($rue, ENT_QUOTES, 'UTF-8');
75      if (!ExpressionsRegexUtils::isValidRegexFrLangWithNumbers($escaped, 1, 150))
76          {
77          throw new \Exception("Erreur, le nom de la rue, obligatoire "
78              . "doit comporter au plus 150 caractères"
79              . "(alphabétiques, chiffres ou ponctuation)");
```

```

79     }
80     $this->rue = empty($escaped) ? "" : $escaped;
81 }
82
83 /** @brief setter : permet d'initialiser ou de modifier le complément d'
84     adresse.
85  * @param $complementAddr le complément d'adresse à utiliser. peut être null.
86  */
87 public function setComplementAddr($complementAddr) {
88     $escaped = htmlentities($complementAddr, ENT_QUOTES, 'UTF-8');
89     if (!ExpressionsRegexUtils::isValidRegexFrLangWithNumbersAndPunctuation(
90         $escaped, 0, 150)){
91         throw new \Exception("Erreur, le Complément d'adresse \"$.complementAddr
92             .\"\" doit comporter au plus 150 caractères \"
93             .\" (alphabétiques, chiffres ou ponctuation)");
94     }
95     $this->complementAddr = empty($escaped) ? "" : $escaped;
96 }
97
98 /** @brief setter : permet d'initialiser ou de modifier le code postal.
99  * @param $CodePostal le numéro à utiliser. peut être null
100  */
101 public function setCodePostal($codePostal) {
102     if (!ExpressionsRegexUtils::isValidString($codePostal, "/^[0-9]{5}$/", 5, 5)
103         ){
104         throw new \Exception("Erreur, le code postal n'est pas valide \"
105             .\"(code postal france métropolitaine sans cedex ni B.P.)");
106     }
107     $this->codePostal = $codePostal;
108 }
109
110 /** @brief setter : permet d'initialiser ou de modifier le nom de la ville.
111  * @param $Ville le nom de la ville à utiliser. peut être null
112  */
113 public function setVille($ville) {
114     $escaped = htmlentities($ville, ENT_QUOTES, 'UTF-8');
115     if (!ExpressionsRegexUtils::isValidRegexFrLang($escaped, 1, 150)){
116         throw new \Exception("Erreur, le nom de la ville \"$.ville.\"\", \"
117             .\"obligatoire doit comporter au plus 150 caractères \"
118             .\" (alphabétiques ou ponctuation)");
119     }
120     $this->ville = empty($escaped) ? "" : $escaped;
121 }
122
123 /** @brief setter : permet d'initialiser ou de modifier le nom du Pays
124  * @param $pays le nom du Pays à utiliser. peut être null
125  */
126 public function setPays($pays) {
127     $escaped = htmlentities($pays, ENT_QUOTES, 'UTF-8');
128     if (!ExpressionsRegexUtils::isValidRegexFrLang($escaped, 1, 100)){
129         throw new \Exception("Erreur, le nom du Pays, obligatoire doit \"
130             .\"comporter au plus 100 caractères \"
131             .\" (alphabétiques ou ponctuation)");
132     }
133     $this->pays = empty($escaped) ? "" : $escaped;
134 }

```

```
132 }
133 ?>
```

5.4 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.

exemples/forms2/ex06_classeAdresseFabrique.php

```
1 <?php
2 namespace CoursPHP\Metier;
3 /**
4  * @brief La classe AdresseFabrique implémente la construction d'un objet Adresse
5  * à partir des données saisies dans un formulaire
6  * Les erreurs générées dans les setters ( reçues via des exceptions) sont
7  * accumulées
8  * dans un tableau associatif d'erreurs, pour générer le cas échéant une vue d'
9  * erreur.
10 */
11 class AdresseFabrique {
12     /**
13      * @brief Obtention d'un objet de classe Adresse à partir des
14      * données saisies dans un formulaire.
15      * Cette méthode prend les mêmes paramètres que le constructeur
16      * plus un paramètre passé par référence qui retourne les messages d'exception.
17      * Initialise les attributs à partir des paramètre.
18      * Pour chaque attribute de la classe, si une exception est générée
19      * au niveau du setter, le message d'exception est ajouté dans un tableau
20      * associatif d'erreurs.
21      * Ces messages d'exception sont
22      * ainsi renvoyés au contrôleur.
23      */
24     public static function getAdresse(&$dataErrors, $id, $numeroRue, $rue,
25         $complementAddr, $codePostal, $ville, $pays) {
26         $adr = Adresse::getDefaultAdresse();
27         $dataErrors = array();
28         try {
29             $adr -> setId($id);
30         } catch (\Exception $e) {
31             $dataErrors['id'] = $e->getMessage();
32         }
33         try {
34             $adr -> setNumeroRue($numeroRue);
35         } catch (\Exception $e) {
36             $dataErrors['numeroRue'] = $e->getMessage();
37         }
38     }
39 }
```

```

38     try {
39         $adr -> setRue($rue);
40     } catch (\Exception $e) {
41         $dataErrors['rue'] = $e->getMessage();
42     }
43     try {
44         $adr -> setComplementAddr($complementAddr);
45     } catch (\Exception $e) {
46         $dataErrors['complementAddr'] = $e->getMessage();
47     }
48     try {
49         $adr -> setCodePostal($codePostal);
50     } catch (\Exception $e) {
51         $dataErrors['codePostal'] = $e->getMessage();
52     }
53     try {
54         $adr -> setVille($ville);
55     } catch (\Exception $e) {
56         $dataErrors['ville'] = $e->getMessage();
57     }
58     try {
59         $adr -> setPays($pays);
60     } catch (\Exception $e) {
61         $dataErrors['pays'] = $e->getMessage();
62     }
63     return $adr;
64 }
65 }
66 ?>

```

5.5 Génération de formulaires et classe AdresseFormView

Nous définissons ensuite la classe `AdresseFormView` qui présente (essentiellement) deux méthodes pour générer des formulaires, avec ou sans gestion d'erreur dans des vues. Une 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 5.4).

exemples/forms2/ex04_classeAdresseFormView.php

```

1 <?php
2 namespace CoursPHP\Vue;
3
4 /**
5  * @brief Implémente la génération de formulairee HTML de saisie d'adresses
6  * Les formulaires peuvent être vierges ou pré-remplis evc d'éventuels
7  * messages d'erreur sur la forme des attributs.
8  */
9 class AdresseFormView {

```

```
10
11  /** @brief Méthode de génération d'un formulaire HTML vierge.
12  */
13  public static function getDefaultFormHTML($action){
14      return self::getFormHtml($action, \CoursPHP\Metier\Adresse::
15          getDefaultAdresse());
16  }
17
18  /**
19   * @brief Méthode de génération d'un formulaire HTML prérempli.
20  */
21  public static function getFormHtml($action, $adresse){
22      $htmlCode = FormManager::beginForm("post", $action);
23      $htmlCode .= FormManager::addHiddenInput("id", "id", $adresse->getId());
24      $htmlCode .= FormManager::addTextInput("Numéro", "numeroRue", "numeroRue",
25          "4",
26          $adresse->getNumeroRue())."<br/>";
27      $htmlCode .= FormManager::addTextInput("Rue/place", "rue", "rue", "30",
28          $adresse->getRue())."<br/>";
29      $htmlCode .= FormManager::addTextInput("Complément d'adresse",
30          "complementAddr", "complementAddr",
31          "30", $adresse->getComplementAddr())."<br/>";
32      $htmlCode .= FormManager::addTextInput("Code postal", "codePostal", "
33          codePostal", "10",
34          $adresse->
35          getCodePostal())."<br/>";
36      $htmlCode .= FormManager::addTextInput("Ville", "ville", "ville", "20",
37          $adresse->getVille(), ENT_QUOTES, "UTF-8")."<br
38          />";
39      $htmlCode .= FormManager::addTextInput("Pays", "pays", "pays", "15",
40          $adresse->getPays())."<br/>";
41      $htmlCode .= FormManager::addSubmitButton("Envoyer", "class=\"sansLabel\"")
42          )."<br/>";
43      $htmlCode .= FormManager::endForm();
44  }
45
46  return $htmlCode;
47 }
48
49 /** Génère un message d'erreur associé à un attribut, s'il en existe un.
50  * Le texte du message est formé du message d'erreur contenu dans dataError.
51 */
52 private static function addErrorMsg($dataErrors, $fieldName){
53     if (!empty($dataErrors[$fieldName])){
54         $htmlCode .= "<span class=\"errorMsg\">". $dataErrors[$fieldName]. "</span
55             ><br/>";
56     }
57     return $htmlCode;
58 }
59
60 /** @brief Méthode de génération de formulaire HTML prérempli avec erreurs
61  * Génère des messages d'erreur associés aux attribut, s'il en existe.
62 */
63 public static function getFormErrorsHtml($action, $adresse, $dataErrors){
64     $htmlCode = FormManager::beginForm("post", $action);
65     $htmlCode .= FormManager::addHiddenInput("id", "id", $adresse->getId());
66     $htmlCode .= self::addErrorMsg($dataErrors, "numeroRue");
67     $htmlCode .= FormManager::addTextInput("Numéro", "numeroRue", "numeroRue",
```

```

        "4",
59         $adresse->getNumeroRue()). "<br/>";
60 $htmlCode .= self::addErrorMsg($dataErrors, "rue");
61 $htmlCode .= FormManager::addTextInput("Rue/place", "rue", "rue", "30",
62     $adresse->getRue()). "<br/>";
63 $htmlCode .= self::addErrorMsg($dataErrors, "complementAddr");
64 $htmlCode .= FormManager::addTextInput("Complément d'adresse",
65     "complementAddr", "complementAddr",
66     "30", $adresse->getComplementAddr()). "<br/>";
67 $htmlCode .= self::addErrorMsg($dataErrors, "codePostal");
68 $htmlCode .= FormManager::addTextInput("Code postal", "codePostal", "
69     codePostal", "10", $adresse->
70     getCodePostal()). "<br/>";
71 $htmlCode .= self::addErrorMsg($dataErrors, "ville");
72 $htmlCode .= FormManager::addTextInput("Ville", "ville", "ville", "20",
73     $adresse->getVille(), ENT_QUOTES, "UTF-8"). "<br
74     />";
75 $htmlCode .= self::addErrorMsg($dataErrors, "pays");
76 $htmlCode .= FormManager::addTextInput("Pays", "pays", "pays", "15",
77     $adresse->getPays()). "<br/>";
78 $htmlCode .= FormManager::addSubmitButton("Envoyer", "class=\"sansLabel\"")
79     ). "<br/>";
80 $htmlCode .= FormManager::endForm();
81
82 return $htmlCode;
83 }
84
85 /** @brief Méthode de génération d'un formulaire HTML caché prérempli.
86  * Permet de transmettre les données d'une instance via $_POST.
87  * Tous les attributs du formulaire sont de type "hidden".
88  */
89 public static function getHiddenFormHtml($action, $adresse, $buttonText){
90     $htmlCode = FormManager::beginForm("post", $action);
91     $htmlCode .= FormManager::addHiddenInput("id", "id", $adresse->getId());
92     $htmlCode .= FormManager::addHiddenInput("numeroRue", "numeroRue",
93         $adresse->getnumeroRue());
94     $htmlCode .= FormManager::addHiddenInput("rue", "rue", $adresse->getRue())
95     ;
96     $htmlCode .= FormManager::addHiddenInput("complementAddr", "complementAddr",
97         $adresse->getComplementAddr());
98     $htmlCode .= FormManager::addHiddenInput("codePostal", "codePostal",
99         $adresse->getCodePostal());
100    $htmlCode .= FormManager::addHiddenInput("ville", "ville", $adresse->
101        getVille());
102    $htmlCode .= FormManager::addHiddenInput("pays", "pays", $adresse->getPays
103        ());
104    $htmlCode .= FormManager::addSubmitButton($buttonText, "class=\"sansLabel
105        \");
106    $htmlCode .= FormManager::endForm();
107
108    return $htmlCode;
109 }
110 }
111 ?>

```

La classe *template* de g n ration de formulaires utilis e ci-dessous est la suivante :

exemples/forms2/ex05_classeFormManager.php

```

1 <?php
2 namespace CoursPHP\Vue;
3 /**
4  @brief Cette classe sert   facilit  la g n ration de formulaires HTML en PHP.
5  * Elle fournit de m thodes pour g n rer le d but, la fin du formulaire,
6  * ainsi que les inputs avec les options de base.
7  * Les options compl mentaires des inputs peuvent  tre s lectionn es
8  * via une variable $extraOptions des m thodes.
9  */
10 class FormManager {
11 /**
12  * @brief g n re la balise <form> avec m thode (Post, Get) et action (url)
13  */
14  public static function beginForm($method, $action, $css_class="",
15                                  $extraOptions=""){
16      if (!empty($css_class)){
17          $css_class_option = "class=\"". $css_class . "\" ";
18      }
19      return "<form method=\"". $method . "\" action=\"". $action . "\" "
20             . $css_class_option . $extraOptions . ">\n";
21  }
22
23 /**
24  @brief ferme le formulaire
25  */
26  public static function endForm() {
27      return "</form>";
28  }
29
30 /**
31  m thode g n rique de g n ration d'un input
32  @param $labelText texte du label correspondant   l'input
33  @param $type type d'input : texte, date, checkbox, radio, submit...
34  @param $id ID de l'input pour la correspondance label/input
35  @param $value valeur initiale du champs de l'input
36  @param $extraOptions chaine de caract res contenant les options
37  *      suppl mentaires de l'input suivant la syntaxe HTML.
38  */
39  public static function addInput($labelText, $type, $name, $id, $value=null,
40                                  $extraOptions="", $noBR=false){
41      $valueOption = ($value == null) ? "" : " value=\"". $value . "\" ";
42      if ($extraOptions == null) {
43          $extraOptions="";
44      }
45
46      if ($labelText!=null && $labelText!=""){
47          $returnText .= "<label for=\"". $id . "\">". $labelText . "</label>\n";
48      }
49      $returnText .= "<input type=\"". $type . "\" name=\"". $name . "\" id=\"".
50                     . $id . "\" ". $valueOption . " ". $extraOptions . " />\n";
51
52      if (!$noBR){
53          $returnText .= "<br/>\n";

```

```

54     }
55     return $returnText;
56 }
57
58 /** @brief méthode pour générer un input de type text */
59 public static function addTextInput($labelText, $name, $id, $size,
60     $value=null, $extraOptions=""){
61     return self::addInput($labelText, "text", $name, $id, $value,
62         "size =\"". $size. "\" " . $extraOptions
63         );
64 }
65 /** @brief méthode pour générer un input de type password */
66 public static function addPasswordInput($labelText, $name, $id, $size,
67     $value=null, $extraOptions=""){
68     return self::addInput($labelText, "password", $name, $id, $value,
69         "size =\"". $size. "\" " . $extraOptions);
70 }
71
72 /** @brief méthode pour générer un input de type radio */
73 public static function addRadioInput($labelText, $name, $id, $checked,
74     $value=null, $extraOptions=""){
75     return self::addInput($labelText, "radio", $name, $id, $value,
76         (strcmp($checked, 'checked')==0)? "checked =\`checked\` " "
77         : " " . $extraOptions);
78 }
79
80 /** @brief méthode pour générer un input de type checkbox */
81 public static function addCheckboxInput($labelText, $name, $id, $checked,
82     $value, $extraOptions=""){
83     return self::addInput($labelText, "checkbox", $name, $id, $value,
84         (strcmp($checked, 'checked')==0)? "checked =\`checked\` " "
85         : " " . $extraOptions);
86 }
87
88 /** @brief méthode pour générer une zone de saisie <textarea> */
89 public static function addTextArea($labelText, $name, $id, $rows, $cols,
90     $value=null, $extraOptions=""){
91     $valueOption = ($value == null) ? "" : $value;
92     if ($extraOptions == null) {
93         $extraOptions="";
94     }
95     $returnText .= "<p>\n";
96     if ($labelText!=null && $labelText!=""){
97         $returnText .= "<label for=\"". $id. "\">". $labelText. "</label>\n";
98     }
99     $returnText .= "<textarea name=\"". $name. "\" id=\"". $id. "\" rows=\"". $rows
100         . "\" cols=\"". $cols. "\" " . $extraOptions. ">". $valueOption
101         . "</textarea>\n";
102     $returnText .= "</p>\n";
103     return $returnText;
104 }
105
106 /** @brief méthode pour générer un input de type file (upload) */
107 public static function addUploadInput($labelText, $name, $id, $size,
108     $value="", $extraOptions=""){

```



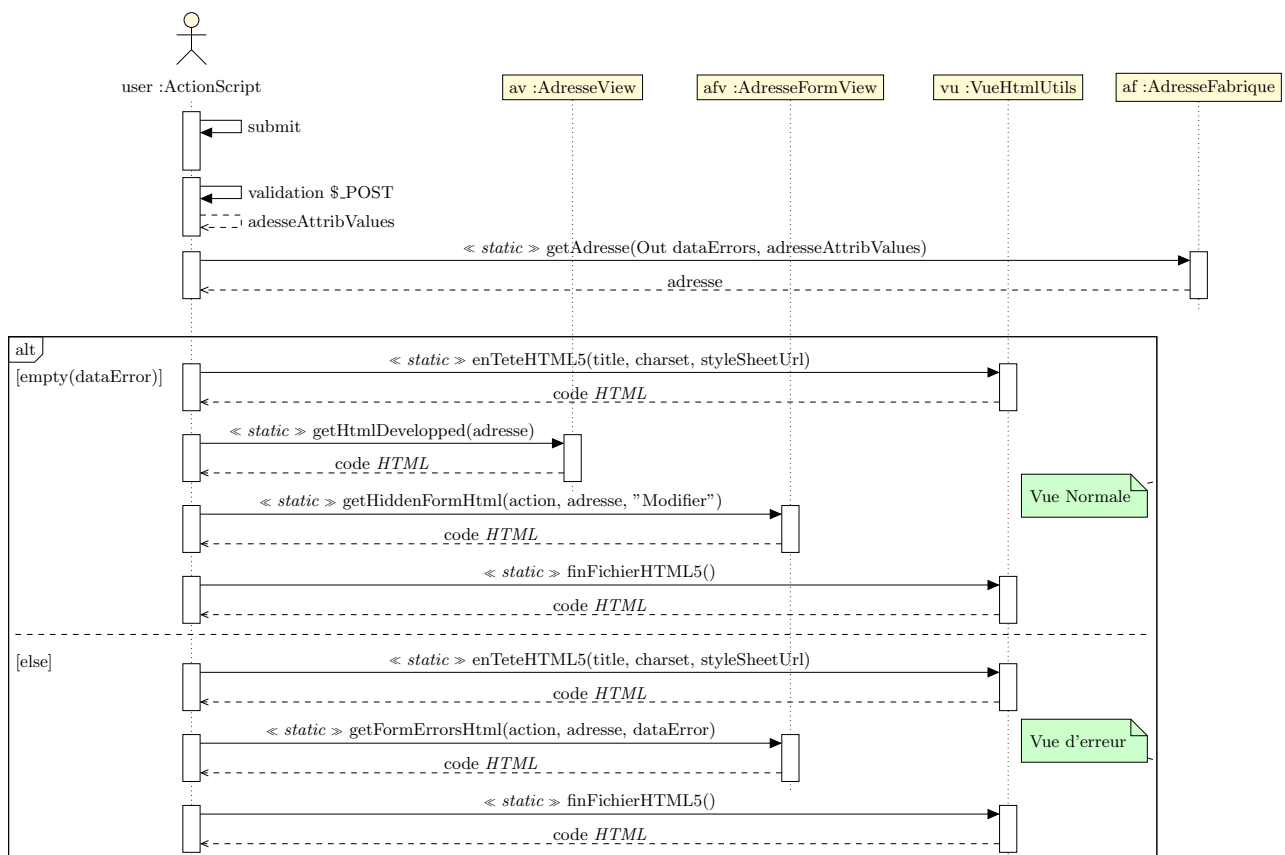
```
109     $valueOption = ($value == null) ? "value=|"|" : " value=|" . $value . "|" ";
110     if ($extraOptions == null) {
111         $extraOptions="";
112     }
113     return self::addInput($labelText, "file", $name, $id, $value,
114                         "size =|" . $size . "|" . $valueOption . " " . $extraOptions)
115         ;
116 }
117 /** @brief m thode pour commencer une liste d'options <select> */
118 public static function beginSelect($labelText, $name, $id, $multiple=false,
119                                 $size=6){
120     $returnText = "";
121     if ($multiple){
122         $multipleOption="multiple=|"multiple|" size=|" . $size . "|" ";
123     }else{
124         $multipleOption="";
125     }
126     if ($labelText != null && $labelText != ""){
127         $returnText .= "<p><label for=|" . $id . "|" >". $labelText . "</label>\n";
128     }
129     $returnText .= "<select name=|" . $name . "|" id=|" . $id . "|" "
130                 . $multipleOption . ">\n";
131     return $returnText;
132 }
133
134 /** @brief m thode simplifi e pour terminer une liste d'options <select> */
135 public static function endSelect(){
136     $returnText = "</select></p>\n";
137     return $returnText;
138 }
139
140 /** @brief m thode simplifi e pour ajouter une <option> de liste <select> */
141 public static function addSelectOption($value, $displayText, $selected=false){
142     $returnText = "";
143     if ($selected){
144         $selectedOption="selected=|"selected|" ";
145     }else{
146         $selectedOption="";
147     }
148     $returnText .= "<option value=|" . $value . "|" " . $selectedOption . ">"
149                 . $displayText . "</option>\n";
150     return $returnText;
151 }
152
153 /** @brief m thode simplifi e pour g n rer un input de type radio */
154 public static function addHiddenInput($name, $id, $value, $extraOptions=""){
155     return self::addInput("", "hidden", $name, $id, "" . $value, $extraOptions,
156                         true);
157 }
158
159 /** @brief m thode simplifi e pour g n rer un bouton submit */
160 public static function addSubmitButton($value="Envoyer", $extraOptions=""){
161     return self::addInput(null, "submit", "", "", $value, "" . $extraOptions);
162 }
163 }
```

5.6 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.

5.6.1 Diagramme de Séquence



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

5.6.2 Saisie et Soumission du Formulaire

Voici tout d'abord la vue permettant de saisir une adresse :

exemples/forms2/ex07_vueDefaultFormAdresse.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/ExpressionsRegexUtils.php ');
5  require_once(dirname(__FILE__). '/classes/AdresseProperties.php ');
6  require_once(dirname(__FILE__). '/classes/Adresse.php ');
7  require_once(dirname(__FILE__). '/classes/AdresseFormView.php ');
8
9  echo CoursPHP\Vue\VueHtmlUtils : :enTeteHTML5( 'Saisie d\'une Adresse ',
10                                     'UTF-8', 'myStyle.css ');
11  echo CoursPHP\Vue\AdresseFormView : :getDefaultFormHTML(
12                                     'ex09_receptionAdresse.php ');
13  echo CoursPHP\Vue\VueHtmlUtils : :finFichierHTML5 ();
14  ?>

```

Le script qui re oit les donn es saisies dans le formulaire, effectue le filtrage pour la s curit  (ici un nettoyage syst matique par `filter_var`), puis construit l'instance d'adresse et appelle une vue (vue normale ou vue d'erreur, selon le cas).

exemples/forms2/ex09_receptionAdresse.php

```

1  <?php
2  require_once(dirname(__FILE__). '/classes/ExpressionsRegexUtils.php ');
3  require_once(dirname(__FILE__). '/classes/AdresseProperties.php ');
4  require_once(dirname(__FILE__). '/classes/Adresse.php ');
5  require_once(dirname(__FILE__). '/classes/AdresseFabrique.php ');
6  require(dirname(__FILE__). '/classes/ValidationRequest.php ');
7
8  \CoursPHP\Controleur\ValidationRequest : :validationAdresse($id , $numeroRue ,
9                                     $rue , $complementAddr ,
10                                     $codePostal , $ville , $pays);
11  $adresse = \CoursPHP\Metier\AdresseFabrique : :getAdresse($dataErrors , $id ,
12                                     $numeroRue , $rue , $complementAddr , $codePostal , $ville , $pays);
13  if (empty($dataErrors)){
14      require (dirname(__FILE__). "/ex11_vueNormaleAdresse.php ");
15  }else{
16      require (dirname(__FILE__). "/ex12_vueErreurAdresse.php ");
17  }
18  ?>

```

exemples/forms2/ex10_validation.php

```

1 <?php
2 namespace CoursPHP\Controleur;
3 /**
4  * @brief Permet la validation initiale des données reçues via $_REQUEST.
5  * Nettoyage de toutes les chaînes. Initialisation à vide des inputs inexistants
6  */
7 class ValidationRequest {
8     /** @brief Nettoie une chaîne avec filter_var et FILTER_SANITIZE_STRING
9     */
10    private static function sanitizeString($chaine){
11        return isset($chaine) ? filter_var($chaine, FILTER_SANITIZE_STRING) : "";
12    }
13
14    /** @brief Validation et initialisation des données d'une adresse
15     * à partir des données reçues dans les tableau superglobal $_REQUEST.
16     */
17    public static function validationAdresse(&$id, &$numeroRue, &$rue,
18        &$complementAddr, &$codePostal, &$ville, &$pays){
19        $id = self::sanitizeString($_POST[ 'id ']);
20        $numeroRue = self::sanitizeString($_POST[ 'numeroRue ']);
21        $rue = self::sanitizeString($_POST[ 'rue ']);
22        $complementAddr = self::sanitizeString($_POST[ 'complementAddr ']);
23        $codePostal = self::sanitizeString($_POST[ 'codePostal ']);
24        $ville = self::sanitizeString($_POST[ 'ville ']);
25        $pays = self::sanitizeString($_POST[ 'pays ']);
26    }
27 }
28 ?>

```

5.6.3 Les Vues

La vue normale affiche simplement l'instance d'adresse saisie, qui est en principe sans erreurs.



exemples/forms2/ex11_vueNormaleAdresse.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/AdresseView.php ');
5 require_once(dirname(__FILE__). '/classes/AdresseFormView.php ');
6
7 echo CoursPHP\Vue\VueHtmlUtils::enTeteHTML5( 'L\'adresse a bien été saisie ',

```

```

8                                     'UTF-8', 'myStyle.css ');
9 echo CoursPHP\Vue\AdresseView : :getHtmlDevelopped ($adresse );
10 // Formulaire caché pour permettre de modifier l'adresse :
11 echo CoursPHP\Vue\AdresseFormView : :getHiddenFormHtml (
12                                     'ex13_receptionModifAdresse.php ', $adresse , "Modifier");
13 echo CoursPHP\Vue\VueHtmlUtils : :finFichierHTML5 ();
14 ?>

```

La vue d'erreur affiche un formulaire partiellement rempli avec les données correctes, et affiche le message d'erreur (initialement généré dans le *setter* de l'attribut) pour les données incorrectes.

exemples/forms2/ex12_vueErreurAdresse.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/AdresseFormView.php ');
5
6 echo CoursPHP\Vue\VueHtmlUtils : :enTeteHTML5 ( 'Saisie d\'une adresse ',
7                                               'UTF-8', 'myStyle.css ');
8 // Affichage de formulaire avec messages d'erreur sur les attributs
9 echo CoursPHP\Vue\AdresseFormView : :getFormErrorsHtml (
10                                     'ex09_receptionAdresse.php ', $adresse , $dataErrors);
11 echo CoursPHP\Vue\VueHtmlUtils : :finFichierHTML5 ();
12 ?>

```

5.6.4 Modification d'une Adresse

Lors de l'affichage de la vue normale de l'adresse ci-dessus, 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 normale avec un formulaire pré-rempli pour modifier l'adresse.

exemples/forms2/ex13_receptionModifAdresse.php

```
1 <?php
2 require_once(dirname(__FILE__).'/classes/ExpressionsRegexUtils.php');
3 require_once(dirname(__FILE__).'/classes/AdresseProperties.php');
4 require_once(dirname(__FILE__).'/classes/Adresse.php');
5 require_once(dirname(__FILE__).'/classes/AdresseFabrique.php');
6 require(dirname(__FILE__).'/classes/ValidationRequest.php');
7
8 \CoursPHP\Controleur\ValidationRequest::validationAdresse($id, $numeroRue,
9     $rue,
10     $complementAddr, $codePostal, $ville,
11     $pays);
12 // Construction d'une instance
13 $adresse = CoursPHP\Metier\AdresseFabrique::getAdresse($dataErrors, $id,
14     $numeroRue, $rue, $complementAddr, $codePostal, $ville, $pays);
15 // Appel de la vue avec un formulaire pré-rempli
16 require(dirname(__FILE__).'/ex12_vueErreurAdresse.php');
17 ?>
```

Troisième partie

Persistance

Table of Contents

6	Cookies	107
6.1	Création d'un <i>cookie</i>	107
6.2	Récupération d'un <i>cookie</i>	109
6.3	Suppression d'un <i>cookie</i>	110
6.4	Mise à jour d'un <i>cookie</i>	111
7	Sessions	112
7.1	Concept de Session et Problèmes de Sécurité	112
7.2	Créer une session	113
7.3	Création d'une session commune à tous les utilisateurs	113
7.4	Durée et <i>SID</i> d'une session	114
7.5	Destruction d'une Session	115
7.6	Exemple de Session avec <i>SID</i> aléatoire transmis par <i>GET</i>	116
7.7	Exemple de Session avec <i>SID</i> aléatoire transmis par <i>COOKIE</i>	119
7.8	<i>Login/Password</i> : Exemple de Politique de Sécurité	120
8	Bases de Données et <i>PHP Data Objects</i>	130
8.1	Créer un Base de Données dans <i>phpmyadmin</i>	130
8.1.1	Création d'une Base de Données Relationnelle	130
8.1.2	Créer un Utilisateur <i>MySQL</i> Responsable d'une <i>BD</i>	131
8.2	Initiation à <i>PDO</i> : connexion, <i>query</i> , destruction	134
8.2.1	Établir la Connexion à une Base de Données	134
8.2.2	Parcourir les Résultats d'une Requête	137
8.3	Requêtes Préparées	141

9	Couche d'Accès aux données	146
9.1	Diagrammes de Conception	146
9.2	Classe de Connexion à une Base de Données	146
9.3	Classes <i>Gateway</i> : Persistance des Objets Métiers	151

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 transactions. Théoriquement, suivant un tel protocole, il faudrait qu'un client qui se reconnecte reprenne tout depuis le début. Ce comportement peut être problématique pour certaines applications, dans lesquelles le serveur doit se souvenir des données personnelles des clients, comme son profil, son adresse, etc.

Pour cette raison, les développeurs *Web* doivent implémenter eux-mêmes la *persistence*, qui permet aux programmes de conserver des données d'une 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 tableau 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 connexions. Un pirate a des fois la possibilité de subtiliser un cookie. En général, on ne peut pas se fier de manière sûre au mécanisme des cookies, ce qui n'empêche pas d'exploiter les cookies. Les cookies sont surtout utilisés pour identifier le 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 validité du cookie, qui permet de limiter la fenêtre d'opportunité pour un pirate.
- Le *hashage* ou le *chiffrement*, qui permet de ne pas révéler en clair les données stockées 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 être 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 "bateau", 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 être 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 être 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) :

```
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* (voir documentation de la fonction `urlencode`) aux données du *cookie*, contrairement à `setcookie`.

```
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 sûr 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 la 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 :



exemples/cookies/ex01_setCookie.php

```

1 <?php
2 // Fonction qui retourne l'heure locale sous fomr de string :
3 fonction getLocalTimeFrenchFormat(){
4     $heureLocaleArray = localtime(time(),true);
5     $dayOfMonth = str_pad(intval($heureLocaleArray['tm_mday'], 10), 2,"0",
6         STR_PAD_LEFT);
7     $monthOfYear = str_pad(intval($heureLocaleArray['tm_mon'], 10)+1, 2,"0",
8         STR_PAD_LEFT);
9     $yearSinceEra = str_pad(intval($heureLocaleArray['tm_year'], 10)+1900, 4,"0"
10        ,STR_PAD_LEFT);
11     $hourOfDay = str_pad(intval($heureLocaleArray['tm_hour'], 10), 2,"0",
12         STR_PAD_LEFT);
13     $minOfHour = str_pad(intval($heureLocaleArray['tm_min'], 10), 2,"0",
14         STR_PAD_LEFT);
15     $secOfMin = str_pad(intval($heureLocaleArray['tm_sec'], 10), 2,"0",
16         STR_PAD_LEFT);
17     $heureLocaleFormatee = $dayOfMonth."/". $monthOfYear."/". $yearSinceEra
18         . " à ". $hourOfDay." :". $minOfHour." :".
19         $secOfMin;
20     return $heureLocaleFormatee;
21 }
22 // trois chaîne séparées par des virgules (c'est juste un exemple avec explode
23 ...)
```

```

18 $valeur = "ma chaîne 1, ma chaîne 2, ma chaîne 3, ".getLocalTimeFrenchFormat()
19 ;
20 setcookie("essaiCookie", $valeur, time()+3600);
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*.



exemples/cookies/ex02_retrieveCookie.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     $dataError = array('cookie' => "essaiCookie introuvable&nbsp;! ");
9 }
10
11 // Appel des vues :
12 require_once('classes/VueHtmlUtils.php');
13 if (empty($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 ?>

```



exemples/cookies/ex02_vueErreur.php

```

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.

exemples/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'autres méthodes pour mettre à jour un *cookie* que d'en créer un 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* :



exemples/cookies/ex04_setAndProlongCookie.php

```

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 cookie est prolongé tant que l'utilisateur 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.
14      css');
15  echo "<h1>Prolongement de la Validité d'un <i>cookie</i></h1>";
16  echo "<p>Valeur courante du cookie : <i>". $valeur. "</i></p>";
17  // Lien sur ce même script (en enlevant la "query string" avec basename par sé
18  curité) :
19  echo "<p><a href=\"". basename($_SERVER[REQUEST_URI], ".php"). ".php\">Cliquez
20      ici</a> "
21      . "pour voir si le <i>cookie</i> a bien été stocké chez le client.</p>";
22  echo Vue\VueHtmlUtils::finFichierHTML5();
23  ?>
```


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 de 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 où le serveur lui-même est sécurisé).

7.2 Créer une session

7.3 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.



exemples/sessions/ex01_createSessionForDummies.php

```

1 <?php
2 // Création d'un identifiant de session valable pour tous les utilisateurs
3 session_id("all-users-session");
4
5 // Le démarrage de session doit avoir lieu avant toute sortie de code HTML via
6 // echo, print, etc.
7 session_start();
8
9 // Si la variable de session du jour existe (il y a déjà eu un script chargé
10 // aujourd'hui)
11 if (isset($_SESSION['counter'])){
12     $counter = intval($_SESSION['counter'], 10);
13     $counter++;
14     $_SESSION['counter'] = "$counter";
15 }else{
16     $_SESSION['counter'] = "1";
17 }
18
19 // Mémorisation de la donnée de session avant fermeture
20 $counterValue = $_SESSION['counter'];
21 // Flush des Données de Session, (sauvegarde immédiate sur le disque)
  
```

```

20 // 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 ?>

```

7.4 Durée et *SID* d'une session



exemples/sessions/ex02_createSessionBasicId.php

```

1 <?php
2 // On fixe à 24 heures la durée de persistance de la session sur le serveur
3 // à partir de chaque connexion
4 session_cache_expire(60*24);
5
6 // Création d'un identifiant de session valable pour tous les utilisateurs
7 session_id( "all-users-session" );
8
9 // Le démarrage de session doit avoir lieu avant toute sortie de code HTML via
    echo, print, etc.
10 session_start ();
11
12 // Chaîne qui code la date d'aujourd'hui
13 $heureLocaleArray = localtime( time(), true );
14 $dayOfMonth = str_pad( intval( $heureLocaleArray[ 'tm_mday' ], 10 ), 2, "0",
    STR_PAD_LEFT );
15 $monthOfYear = str_pad( intval( $heureLocaleArray[ 'tm_mon' ], 10 ) + 1, 2, "0",
    STR_PAD_LEFT );
16 $yearSinceEra = str_pad( intval( $heureLocaleArray[ 'tm_year' ], 10 ) + 1900, 4, "0",
    STR_PAD_LEFT );
17 $dateSting = $yearSinceEra . "-" . $monthOfYear . "-" . $dayOfMonth;
18

```

```

19 // Si la variable de session du jour existe (il y a déjà eu un script chargé
    aujourd'hui)
20 if (isset($_SESSION['counter-'].$dateSting)){
21     $counter = intval($_SESSION['counter-'].$dateSting), 10);
22     $counter++;
23     $_SESSION['counter-'].$dateSting = "$counter";
24 }else{
25     // Chaîne qui code la date d'hier
26     $yesterdayLocaleArray = localtime(time()-60*60*24,true);
27     $yestadayDayOfMonth = str_pad(intval($yesterdayLocaleArray['tm_mday'], 10),
    2, "0",STR_PAD_LEFT);
28     $yestadayMonthOfYear = str_pad(intval($yesterdayLocaleArray['tm_mon'], 10)
    +1, 2, "0",STR_PAD_LEFT);
29     $yestadayYearSinceEra = str_pad(intval($yesterdayLocaleArray['tm_year'], 10)
    +1900, 4, "0",STR_PAD_LEFT);
30     $yesterdayDateString = $yestadayYearSinceEra."-".$yestadayMonthOfYear."-".
    $yestadayDayOfMonth;
31     // On efface le compteur de la veille pour éviter que les compteurs s'
    accumulent...
32     unset($_SESSION['counter-'].$yesterdayDateString]);
33     // On initialise le compteur du jour
34     $_SESSION['counter-'].$dateSting = "1";
35 }
36 // Mémorisation de la donnée de session avant fermeture
37 $counterValue = $_SESSION['counter-'].$dateSting];
38 // Flush des Données de Session, (sauvegarde immédiate sur le disque)
39 // Libère instantanément le verrou pour l'accès à la session par d'autres
    scripts ou clients
40 session_write_close ();
41
42 // Code de la vue :
43 require_once('classes/VueHtmlUtils.php');
44 echo CoursPHP\Vue\VueHtmlUtils::enTeteHTML5("Utilisation basique d'une session
    ", 'UTF-8', 'myStyle.css');
45 echo "<h1>Utilisation d'une Session<br/>Commune à Tous les Clients</h1>";
46
47 echo "<p>Le script a été chargé ".$_SESSION['counter-'].$dateSting." fois aujourd'hui (le ".
    $dayOfMonth."/".$monthOfYear."/".$yearSinceEra.").</p>";
48 echo CoursPHP\Vue\VueHtmlUtils::finFichierHTML5();
49 ?>

```

7.5 Destruction d'une Session

exemples/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 démarrage 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 ?>
```

7.6 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). 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*.

exemples/sessions/ex04_sessionRandomIdGET.php

```
1 <?php
2 /** Fonction qui retourne un message de bienvenue dans la langue choisie */
3 fonction getGreeting($PREF_LANG){
4     $htmlGreeting = "";
5     switch ($PREF_LANG){
6         case "en" : $htmlGreeting .= "Hi, guys, Welcome to <code>mySite.com</
7             code>&nbsp;! ";
8         case "fr" : $htmlGreeting .= "Salut la compagnie, bienvenue sur <code>
9             monSite.fr</code>&nbsp;! ";
10        break;
11        default : $htmlGreeting .= "Wilkommen, Bienvenue, Welcomme !<br/>";
12        break;
13    }
14    return $htmlGreeting;
15 }
16
17 $dataError = array();
18 // Test pour voir si l'identifiant de session existe et si la donnée a la
19 // bonne forme
20 // (10 chiffres hexa entre 0 et f)
21 if (isset($_GET['session-id']) && preg_match("/^[0-9a-fA-F]{10}$/", $_GET['
22     session-id'])){
23     // On a bien vérifié la forme par expression régulière donc, pas d'autre pré
24     caution
25     $mySid = $_GET['session-id'];
26 }else{
27     if (isset($_GET['session-id'])){
28         $dataError['session-id'] = "Identifiant de session incorrect. Pirates s'
29             abstenir... ";
30     }
31     // Génération d'un SID par des octets (pseudo-)aléatoires codés en hexa
32     $cryptoStrong = false; // Variable pour passage par référence
33     $octets = openssl_random_pseudo_bytes(5, $cryptoStrong);
34     $mySid = bin2hex($octets);
35 }
```

```

32 session_id($mySid);
33
34 // Le démarrage de session doit avoir lieu avant toute sortie de code HTML via
    echo, print, etc.
35 session_start();
36
37 // Initialisation des paramètres régionaux
38
39 // Si un choix de langage est précisé dans l'URL, on modifie la variable de
    session
40 if (isset($_GET['pref_lang'])){
41     if ($_GET['pref_lang'] == "en" || $_GET['pref_lang'] == "fr"){
42         // Les données entrées en session doivent être filtrées,
43         // même si, dans ce cas, il n'y a pas de danger car on a testé avec ==
44         $_SESSION['preferred_language'] = filter_var($_GET['pref_lang'],
            FILTER_SANITIZE_STRING);
45     }else{
46         // Paramètre imprévu, on détruit la donnée de session, au cas où
47         unset($_SESSION['preferred_language']);
48     }
49 }
50 // Si une préférence de langage a été définie, soit dans l'URL, soit en
    session
51 if (isset($_SESSION['preferred_language'])){
52     $PREFERRED_LANG = $_SESSION['preferred_language'];
53 }else{
54     $PREFERRED_LANG = "undef";
55 }
56
57 // Flush des Données de Session, (sauvegarde immédiate sur le disque)
58 session_write_close();
59
60 require_once('classes/VueHtmlUtils.php');
61 // Si aucun SID incorrect dans l'URL
62 if (empty($dataError)){ // Code de la vue normale :
63     echo CoursPHP\Vue\VueHtmlUtils::enTeteHTML5("Session avec SID Aléatoire", '
        UTF-8', 'myStyle.css');
64     echo "<h1>Session avec <i>SID</i> Aléatoire<br/>Transmis par <code>GET</code>
        <</h1>";
65     echo "<p>";
66     // Message de bienvenue dans la langue sélectionnée ou en multilingue si
        undef
67     echo getGreeting($PREFERRED_LANG). "<br/>";
68     echo "<a href=\"".$_SERVER['SCRIPT_NAME']."?session-id=".$mySid."&pref_lang=
        fr\">Français</a> ou "
69     . "<a href=\"".$_SERVER['SCRIPT_NAME']."?session-id=".$mySid."&pref_lang=
        en\">English</a>";
70     echo "</p>";
71     echo CoursPHP\Vue\VueHtmlUtils::finFichierHTML5();
72 }else{ // Appel de la vue d'erreur :
73     require('ex04_vueErreur.php');
74 }
75
76
77 ?>

```



FIGURE 7.1 : Accueil d'un nouveau client (pas de session en cours)



FIGURE 7.2 : Accueil d'un client avec langue préférée en anglais



FIGURE 7.3 : Accueil d'un client avec *SID* incorrect

7.7 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*.

exemples/sessions/ex05_sessionRandomIdCOOKIE.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</
7              code>&nbsp;";
8          break;
9          case "fr" : $htmlGreeting .= "Salut la compagnie, bienvenue sur <code>
10             monSite.fr</code>&nbsp;";
11         break;
12         default : $htmlGreeting .= "Wilkommen, Bienvenue, Welcomme !<br/>";
13         break;
14     }
15     return $htmlGreeting;
16 }
17 $dataError = array();
18 // Test pour voir si l'identifiant de session existe et si la donnée a la
19 bonne forme
20 // (10 chiffres hexa entre 0 et f)
21 if (isset($_COOKIE['session-id']) && preg_match("/^[0-9a-fA-F]{10}$/",
22     $_COOKIE['session-id'])){
23     // On a bien vérifié la forme par expression régulière donc, pas d'autre pré
24     caution
25     $mySid = $_COOKIE['session-id'];
26 }else{
27     if (isset($_COOKIE['session-id'])){
28         $dataError['session-id'] = "Identifiant de session incorrect. Pirates s'
29             abstenir...";
30     }
31     // Génération d'un SID par des octets (pseudo-)aléatoires codés en hexa
32     $cryptoStrong = false; // Variable pour passage par référence
33     $octets = openssl_random_pseudo_bytes(5, $cryptoStrong);
34     $mySid = bin2hex($octets);
35 }
36 session_id($mySid);
37 // Création (ou mise à jour) du cookie. Nouvelle validité du cookie : 10 jours
38 setcookie("session-id", $mySid, time()+60*60*24*10);
39
40 // Le démarrage de session doit avoir lieu avant toute sortie de code HTML via
41 echo, print, etc.

```



```

37 session_start();
38
39 // Initialisation des paramètres régionaux
40
41 // Si un choix de langage est précisé dans l'URL, on modifie la variable de
    session
42 if (isset($_GET['pref_lang'])){
43     if ($_GET['pref_lang'] == "en" || $_GET['pref_lang'] == "fr"){
44         // Les données entrées en session doivent être filtrées,
45         // même si, dans ce cas, il n'y a pas de danger car on a testé avec ==
46         $_SESSION['preferred_language'] = htmlentities($_GET['pref_lang'],
            ENT_QUOTES, "UTF-8");
47     }else{
48         // Paramètre imprévu, on détruit la donnée de session, au cas où
49         unset($_SESSION['preferred_language']);
50     }
51 }
52 // Si une préférence de langage a été définie, soit dans l'URL, soit en
    session
53 if (isset($_SESSION['preferred_language'])){
54     $PREFERRED_LANG = $_SESSION['preferred_language'];
55 }else{
56     $PREFERRED_LANG = "undef";
57 }
58
59 // Flush des Données de Session, (sauvegarde immédiate sur le disque)
60 session_write_close();
61
62 // Code de la vue :
63 require_once('classes/VueHtmlUtils.php');
64 if (empty($dataError)){ // Code de la vue normale
65     echo CoursPHP\Vue\VueHtmlUtils::enTeteHTML5("Session avec SID Aléatoire", '
        UTF-8', 'myStyle.css');
66     echo "<h1>Session avec <i>SID</i> Aléatoire<br/>Transmis par <code>COOKIE</
        code></h1>";
67     echo "<p>";
68     // Message de bienvenue dans la langue sélectionnée ou en multilingue si
        undef
69     echo getGreeting($PREFERRED_LANG). "<br/>";
70     echo "<a href=\"".$_SERVER['SCRIPT_NAME']."?pref_lang=fr\">Français</a> ou "
71     . "<a href=\"".$_SERVER['SCRIPT_NAME']."?pref_lang=en\">English</a>";
72     echo "</p>";
73     echo CoursPHP\Vue\VueHtmlUtils::finFichierHTML5();
74 }else{ // Appel de la vue d'erreur :
75     require('ex04_vueErreur.php');
76 }
77 ?>

```

7.8 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 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 :



exemples/sessions/ex07_authentification.php

```

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   ?>

```

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

exemples/sessions/classes/VueHtmlUtils.php

```

27  /** Fonction qui retourne le code HTML d'un formulaire de login */
28  public static function getHTML_LoginForm($formAction){
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 sur une
                    connexion sécurisée <i>HTTPS</i> avec <i>SSL</i>.<br/>"
33              . "Votre confidentialité n'est pas garantie !!!</p>";
34      }
35      // Code du formulaire :
36      $htmlCode .= '<form method="POST" action="'. $formAction. '>';
37      $htmlCode .= '<input type="hidden" name="action" value="validateAuth"/>';
38      $htmlCode .= '<p><label for="e-mail">e-mail</label><input type="email" name
                    ="email" size="25"/></p>';
39      $htmlCode .= '<p><label for="motdepasse">Mot de passe</label><input type="
                    password" name="motdepasse" size="25"/></p>';
40      $htmlCode .= '<input class="sansLabel" value="Envoyer" type="submit"/>';
41      $htmlCode .= '</form>';
42      $htmlCode .= "<p>L'adresse <i>e-mail</i> doit être valide et "
43          . "votre mot de passe doit contenir au moins 8 caractères, une minuscule,
                    une majuscule, un chiffre, "
44          . "et un caractère parmi ". htmlentities("#-|.@[]=!&", ENT_QUOTES, "UTF
                    -8"). "., merci de votre compréhension...</p>";
45      return $htmlCode;
46  }

```

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 :



exemples/sessions/ex07_receivePassword.php

```

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 ');
5
6  // Fonction à implémenter : test d'existence du login/mot de passe en BD
7  // Voir le chapitre sur les bases de données...
8  function userPasswordCheckInDatabase($email, $hashedPassword){
9      // TODO : tester si le couple e-mail et mot de passe (après hashage SHA512)
10     // sont bien présents dans la base de données
11     return true;
12 }
13
14 // Test de la forme (regex) du mot de passe et de l'e-mail
15 CoursPHP\Controleur\ValidationRequest::validationLogin($dataError, $email,
16     $password);
17
18 if (empty($dataError)){ // les données d'authentification ont la bonne forme.
19     // On vérifie que le mot de passe (après hashage SHA512)
20     // est bien celui en base de donnée.
21     if (!userPasswordCheckInDatabase($email, hash("sha512" , $password))){
22         // Renvoi d'une erreur de login
23         $dataError["login"] = "Erreur : login ou mot de passe incorrect";
24     }else{
25         CoursPHP\Auth\SessionUtils::createSession($email);
26         // Flush des Données de Session, (sauvegarde immédiate ur le disque)
27         session_write_close ();
28     }
29 }
30
31 require_once('classes/VueHtmlUtils.php ');
32 if (empty($dataError)){ // Code de la vue normale :
33     echo CoursPHP\Vue\VueHtmlUtils::enTeteHTML5("Welcome Page", 'UTF-8',
34         'myStyle.css ');
35     echo "<h1>Persistance de connexion<br/>Exemple de politique de sécurité</
36     h1>";
37     echo "Bienvenue ! Vous êtes convenablement authentifié.<br/>";
38     echo "Pour accéder encore à des données sensibles, "
39     . "<a href=\"./ex08_sessionTestIP_RandomIdCookie.php\">cliquez ici</a>.";
40     echo CoursPHP\Vue\VueHtmlUtils::finFichierHTML5();
41 }else{ // Appel de la vue d'erreur :
42     require('ex07_vueErreur.php ');
43 }
44 ?>

```

exemples/sessions/ex07_validationPasswd.php

```

1  <?php
2  namespace CoursPHP\Controleur;
3  /**
4   * @brief Permet la validation initiale des données reçues via $_REQUEST.
5   * Nettoyage de toutes les chaînes. Initialisation à vide des inputs inexistants
6   */
7  class ValidationRequest {
8      /** @brief Nettoie une chaîne avec filter_var et FILTER_SANITIZE_STRING
9       */
10     private static function sanitizeString($chaine){
11         return isset($chaine) ? filter_var($chaine, FILTER_SANITIZE_STRING) : "";
12     }

```

```

13
14  /** @brief Validation et initialisation des données du login/password
15  * à partir des données reçues dans les tableau superglobal $_REQUEST.
16  */
17  public static function validationLogin(&$dataError, &$email, &$password){
18      if (!isset($dataError)){
19          $dataError = array();
20      }
21      // Test sur la forme des données de login et mot de passe :
22      $wouldBePasswd = $_POST['motdepasse'];
23      if (empty($wouldBePasswd) || !\CoursPHP\Auth\AuthUtils::isStrongPassword(
24          $wouldBePasswd)){
25          $password = "";
26          $dataError["login"] = "Mot de passe incorrect"
27                               . "votre mot de passe doit contenir au moins 8
28                               caractères, "
29                               . "une minuscule, une majuscule, un
30                               chiffre, "
31                               . "et un caractère permis "
32                               . htmlentities("#-./@[]=!&", ENT_QUOTES, "UTF-8"). "</p>";
33      }else{
34          $password = $wouldBePasswd;
35      }
36
37      if (filter_var($_POST["email"], FILTER_VALIDATE_EMAIL) == FALSE){
38          $email = "";
39          $dataError["login"] = "Adresse e-mail invalide.";
40      }else{
41          $email = $_POST["email"];
42      }
43  }
44  }
45  ?>

```

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`

exemples/sessions/classes/AuthUtils.php

```

1 <?php
2 namespace CoursPHP\Auth;
3
4 class AuthUtils{
5
6     /** Fonction qui teste si un mot de passe est suffisamment difficile */
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 expressions régulières...
11        $CharacterDiversityCondition = preg_match("/[a-z]/", $wouldBePasswd)
12                                     && preg_match("/[A-Z]/", $wouldBePasswd)
13                                     && preg_match("/[0-9]/", $wouldBePasswd)
14                                     && preg_match("/[!@#\$%&]/", $wouldBePasswd);
15        return $lengthCondition && $CharacterDiversityCondition;
16    }
17 }

```

18 | ?>

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.



exemples/sessions/ex07_receivePassword.php

```

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');
5
6 // Fonction à implémenter : test d'existence du login/mot de passe en BD
7 // Voir le chapitre sur les bases de données...
8 function userPasswordCheckInDatabase($email, $hashedPassword){
9     // TODO : tester si le couple e-mail et mot de passe (après hashage SHA512)
10    // sont bien présents dans la base de données
11    return true;
12 }
13
14 // Test de la forme (regex) du mot de passe et de l'e-mail
15 CoursPHP\Controleur\ValidationRequest::validationLogin($dataError, $email,
16     $password);
17
18 if (empty($dataError)){ // les données d'authentification ont la bonne forme.
19     // On vérifie que le mot de passe (après hashage SHA512)
20     // est bien celui en base de donnée.
21     if (!userPasswordCheckInDatabase($email, hash("sha512", $password))){
22         // Renvoi d'une erreur de login
23         $dataError["login"] = "Erreur : login ou mot de passe incorrect";
24     }else{
25         CoursPHP\Auth\SessionUtils::createSession($email);
26         // Flush des Données de Session, (sauvegarde immédiate sur le disque)
27         session_write_close();
28     }
29 }
30
31 require_once('classes/VueHtmlUtils.php');
32 if (empty($dataError)){ // Code de la vue normale :
33     echo CoursPHP\Vue\VueHtmlUtils::enTeteHTML5("Welcome Page", 'UTF-8',
34         'myStyle.css');

```

```

34     echo "<h1>Persistance de connexion<br/>Exemple de politique de s curit </
        h1>";
35     echo "Bienvenue ! Vous  tes convenablement authentifi .<br/>";
36     echo "Pour acc der encore   des donn es sensibles, "
37         . "<a href='\"./ex08_sessionTestIP_RandomIdCookie.php\">cliquez ici</a>.";
38     echo CoursPHP\Vue\VueHtmlUtils::finFichierHTML5();
39 }else{ // Appel de la vue d'erreur :
40     require('ex07_vueErreur.php');
41 }
42 ?>

```

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

exemples/sessions/classes/SessionUtils.php

```

1 <?php
2 namespace CoursPHP\Auth;
3
4 class SessionUtils{
5
6     /** @brief fonction de g n ration de l'ID de session al atoire */
7     public static function generateSessionId(){
8
9         // G n ration de 10 octets (pseudo-)al atoires cod s en hexa
10        $cryptoStrong = false; // Variable pour passage par r f rence
11        $octets = openssl_random_pseudo_bytes(10, $cryptoStrong);
12        $mySid = bin2hex($octets);
13        return $mySid;
14    }
15
16    /** Cr ation d'une session de SID al atoire avec l'e-mail (login unique)
17     * @param email e-mail servant de login (identifiant unique de l'utilisateur)
18     * @param role r le de l'utilisateur (admin, visiteur, gestionnaire...)
19     * (voir le chapitre sur le Front Controller)
20     */
21    public static function createSession($email, $role="visitor"){
22        // Le num ro de session al atoire
23        $mySid = self::generateSessionId();
24        session_id($mySid);
25        // Cr ation du cookie avec SID al atoire. Validit  du cookie : 2mn
26        // Un pirate aura besoin de temps pour voler le cookie...
27        setcookie("session-id", $mySid, time()+60*2);
28
29        // L'utilisateur vient de s'identifier.
30        // Dans le cas improbable d'une collision sur le SID,
31        // Mais surtout d'une usurpation d'identit , on d truit la session
32        // avant de red marer une session vide
33        session_start();
34        session_destroy();
35        // On fixe   2mn la dur e de persistance de la session sur le serveur
36        //   partir de chaque connexion. Cela limite le temps pour un hackeur
37        // pour deviner le num ro de session...
38        session_cache_expire(2);
39        session_id($mySid);
40        // D marrage de la session
41        session_start();

```

```

42 // On échappe, même si on sait qu'on a validé l'adresse e-mail....
43 $_SESSION[ 'email' ] = htmlentities($email, ENT_QUOTES, "UTF-8");
44 // On échappe, même si on sait qu'on a validé l'adresse e-mail....
45 $_SESSION[ 'role' ] = htmlentities($role, ENT_QUOTES, "UTF-8");
46 $_SESSION[ 'ipAddress' ] = $_SERVER[ 'REMOTE_ADDR' ];
47 }
48 }
49 ?>

```

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.



exemples/sessions/ex08_sessionTestIP_RandomIdCookie.php

```

1 <?php
2 require_once(dirname(__FILE__). '/classes/AuthUtils.php ');
3 require_once(dirname(__FILE__). '/classes/SessionUtils.php ');
4
5 $dataError = array ();
6 // Test pour voir si l'identifiant de session existe et si la donnée a la
7 // bonne forme
8 // (10 chiffres hexa entre 0 et f)
9 if (!isset($_COOKIE[ 'session-id' ]) ||
10     !preg_match("/^[0-9a-fA-F]{20}$/", $_COOKIE[ 'session-id' ])){
11     $dataError[ 'no-cookie' ] = "Votre cookie a peut-être expiré, "
12     . "Merci de vous connecter à nouveau... ";
13 } else {
14     // On récupère l'ID de session
15     $mySid = $_COOKIE[ 'session-id' ];
16
17     // On récupère les données de session :
18     session_id($mySid);

```



```
18 session_start();
19
20 // Test sur les donn es de session et contr le par IP
21 if (empty($_SESSION['email']) || empty($_SESSION['ipAddress']))
22     || $_SESSION['ipAddress'] != $_SERVER['REMOTE_ADDR']) {
23     $dataError["no-session"] = "Votre session a peut- tre expir e, "
24         . "Merci de vous connecter   nouveau...";
25     session_destroy();
26 } else {
27     // Raffinement : on change le SID al atoire, en copiant
28     // la session dans une nouvelle. On r g n re ensuite le cookie
29     // Comme  a, le cookie n'est valable qu'une fois, et l'ID de session aussi
30     // ce qui limite beaucoup la possibilit  d'un  ventuel hacker
31     $backupSession_Email = $_SESSION['email'];
32     // On d truit l'ancienne session
33     session_destroy();
34     // On recr e une session :
35     CoursPHP\Auth\SessionUtils::createSession($backupSession_Email);
36     // Flush des Donn es de Session, (sauvegarde imm diate ur le disque)
37     session_write_close();
38 }
39 }
40
41 // Code de la vue :
42 require_once(dirname(__FILE__).' /classes/VueHtmlUtils.php');
43 if (empty($dataError)) { // Code de la vue normale
44     echo CoursPHP\Vue\VueHtmlUtils::enTeteHTML5("Consultation des Donn es
45         Personnelles", 'UTF-8', 'myStyle.css');
46     echo "<h1>Consultation des Donn es Personnelles</h1>";
47     echo "<p>";
48     echo "Ne le dite   personne : le <i>SID</i> est : <br/>". $mySid;
49     echo "</p><p>";
50     echo "Votre adresse e-mail est : ".$_SESSION['email']. "<br/>";
51     echo "Gr ce   votre adresse e-mail, le serveur peut retrouver vos donn es
52         personnelles<br/>";
53     echo "et vous les afficher. Et d'ailleurs, les voici...<br/>";
54     echo "Pour acc der encore   des donn es sensibles, "
55         . "<a href='\"./ex08_sessionTestIP_RandomIdCookie.php\">cliquez ici</a>.";
56     echo "</p>";
57     echo CoursPHP\Vue\VueHtmlUtils::finFichierHTML5();
58 } else { // Appel de la vue d'erreur :
59     require('ex07_vueErreur.php');
```



FIGURE 7.4 : Re-chargement du script. La connexion persiste, mais le *SID* a changé.



FIGURE 7.5 : 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



FIGURE 8.1 : Création d'une nouvelle base de données



FIGURE 8.2 : Création d'une nouvelle table



FIGURE 8.3 : Exemple de base de données avec deux tables



FIGURE 8.4 : Vue Relationnelle de base de données avec deux tables

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*.



FIGURE 8.5 : Ajout d'un Utilisateur *MySQL*

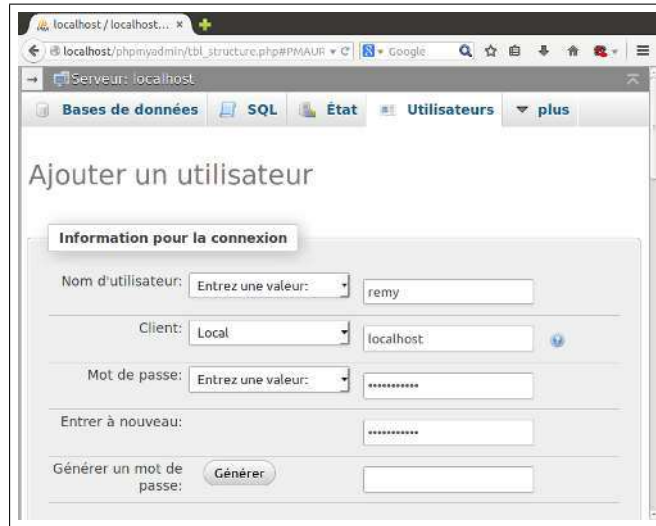


FIGURE 8.6 : Accès de l'Utilisateur *MySQL* Uniquement sur le Serveur Local

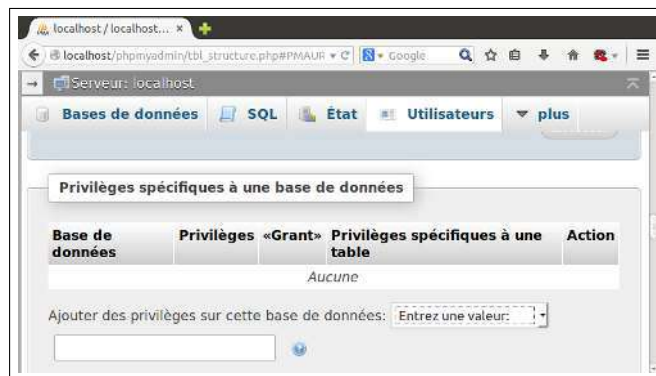


FIGURE 8.7 : Ajout de Privileges Spécifiques d'un Utilisateur sur une Base de Données

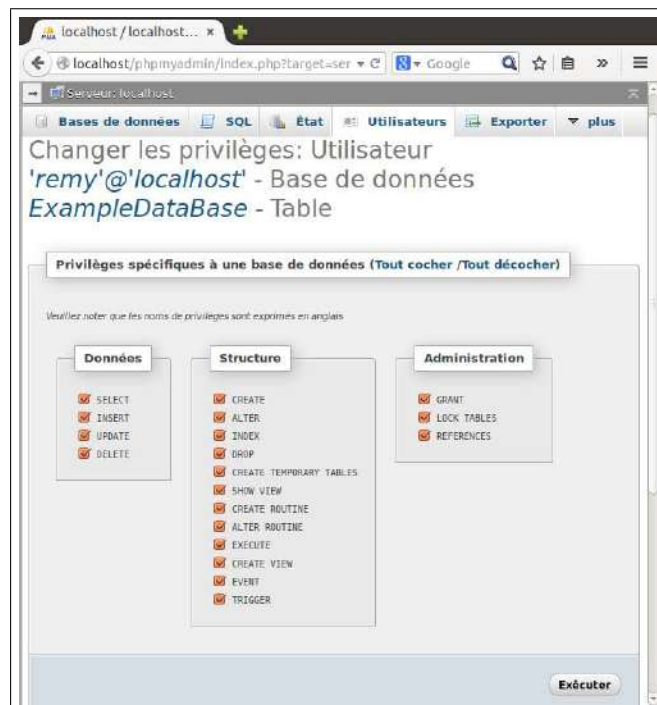


FIGURE 8.8 : Exemple : Donner tous les Droits à l'Utilisateur sur une Base

8.2 Initiation   PDO : connexion, query, destruction

L'extension du langage PHP appel e *PHP Data Objects* fournit une couche d'abstraction pour acc der   des donn es,   l'aide de diff rents *drivers*. Comme exemples de *drivers*, on peut citer *MySQL*, *Oracle*, *SQLite*, *PostgreSQL*, *MS SQL Sever*, etc.

L'int r t de *PDO* est de permettre d'utiliser tous ces *drivers*, qui acc dent   des bases de donn es diff rentes, avec les m mes fonction : les m thodes de *PDO*.

8.2.1  tablir la Connexion   une Base de Donn es

La connexion   la base de donn es se fait avec le nom du driver (ici *mysql*), le nom d'h te, le nom de la base de donn es, le nom d'utilisateur ayant les droits sur la base de donn es, et son mot de passe.

Une  ventuelle exception issue du constructeur de PDO doit absolument  tre g r e avec `try...catch` (ou avec un *handler*), car sinon le message de l'exception s'affiche, r v lant le nom et le mot de passe de l'utilisateur ayant les droits sur la base.



exemples/pdo/ex01_connexionPDO.php

```
1 <?php
2 $mySqlUser = "remy";
3 $mySqlPassword = "my_password";
4 $dataBase = "ExampleDataBase";
5
6 $dataError = array();
7
8 // ON DOIT ABSOLUMENT G RER CETTE EXCEPTION, FAUTE DE QUOI
9 // L'UTILISATEUR DE LA BASE DE DONN ES ET LE MOT DE PASSE
10 // APPARA SENT EN CLAIR !!!!
11 try {
12     // Cr ation de l'instance de PDO (database handler).
13     $dbh = new PDO('mysql:host=localhost;dbname='.$dataBase, $mySqlUser,
14                   $mySqlPassword);
15 } catch (PDOException $e) {
16     $dataError["connexion"] = "Erreur de connexion   la base de donn es."
17                             . "Vous n'avez pas besoin d'en savoir plus...";
18     require("vueErreur.php");
19     die();
20 }
21 // Requete : cha ne de caract res contenant une requ te valide en SQL
22 $requete = 'INSERT INTO Adresse(id, numeroRue, rue, complementAddr, '
23           . 'codePostal, ville, pays) '
24           . 'VALUES ("123456abda", "11", "All e des Pies Jaunes", ';
```

```

25         . 'Bâtiment 2D', "63000", "Clermont-Ferrand", "France") ');
26
27 // Exécution de la requête et mémorisation du résultat :
28 $resultExec = $dbh->exec($requete);
29
30 if ($resultExec === false){
31     $dataError["requete"] = "Problème d'exécution de la requête. "
32         . "(par exemple, une ligne avec cette clé primaire existe déjà, "
33         . "ou encore la requête n'est pas valide sur la base...)";
34     require("vueErreur.php");
35     die();
36 }
37
38 // $resultExec doit donner le nombre de lignes affectées (ici : insérées)
39 if ($resultExec === 0){
40     $dataError["requete"] = "Problème : Aucune ligne n'a été affectée par la
41         requête.";
42     require("vueErreur.php");
43     die();
44 }
45 // Fermeture de la connexion (connexion non persistante)
46 $resultExec = null;
47 $dbh = null;
48
49 // Code de la vue :
50 require_once('classes/VueHtmlUtils.php');
51 echo CoursPHP\Vue\VueHtmlUtils::enTeteHTML5("Ma Première Connexion PDO",
52     'UTF-8', 'myStyle.css');
53 echo "<h1>Initier une Connexion <i>PDO</i></h1>";
54 echo "La requête a bien été exécutée...";
55 echo CoursPHP\Vue\VueHtmlUtils::finFichierHTML5();
56 ?>

```



exemples/pdo/ex01_vueErreur.php

```

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

```



```

8     }
9     echo CoursPHP\Vue\VueHtmlUtils : :finFichierHTML5 ();
10  ?>

```

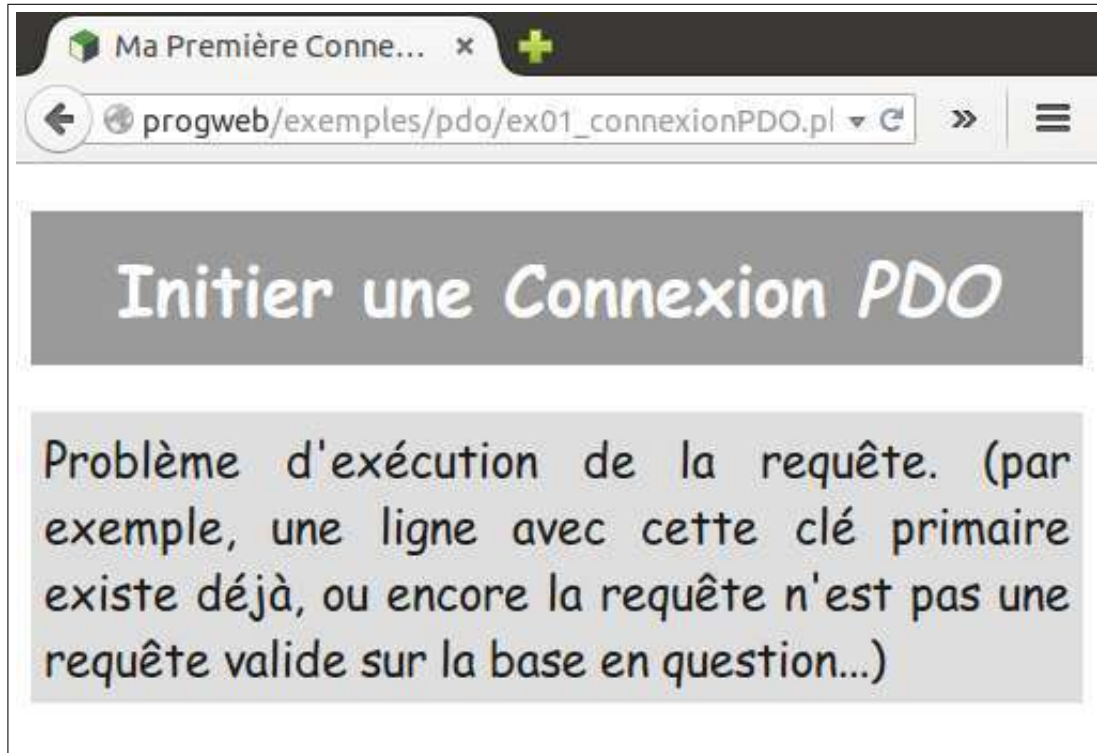


FIGURE 8.9 : Cas o  la cl  primaire existe d j  (ici : rechargement du script)

Voici un exemple o  l'exception g n r e par le constructeur de *PDO* n'est pas g r e. Les donn es d'authentification pour l'acc s   la base de donn es apparaissent en clair chez le client.



exemples/pdo/ex02_withNoTryCatchLeakProblem.php

```

1 <?php
2     $mySqlUser = "remy";
3     $mySqlPassword = "my_password";
4     $dataBase = "ExampleMisSpelledDataBase";
5
6     $dbh = new PDO( 'mysql:host=localhost;dbname='. $dataBase , $mySqlUser ,

```

```

7         $mySqlPassword);
8
9     $requete = 'INSERT INTO Adresse(id, numeroRue, rue, complementAddr, '
10             . '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
20     // Code de la vue :
21     foreach($dbh->query('SELECT * from Adresse') as $row) {
22         print_r($row);
23     }
24     $dbh = null;
25     echo CoursPHP\Vue\VueHtmlUtils::ichierHTML5();
26 ?>

```

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

exemples/pdo/ex03_connectToDatabasePDO.php

```

1 <?php
2     $mySqlUser = "remy";
3     $mySqlPassword = "my_password";
4     $dataBase = "ExampleDataBase";
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     // APPARAÎSENT EN CLAIR !!!!
9     try {
10         // Création de l'instance de PDO (database handler).
11         $dbh = new PDO('mysql:host=localhost;dbname='.$dataBase, $mySqlUser,
12             $mySqlPassword);
13     } catch (PDOException $e) {
14         $dataError['connexion-bd'] = "Erreur de connexion à la base de données."
15             . "Vous n'avez pas besoin d'en savoir plus...";
16         require("vueErreur.php");
17         die();
18     }
19 ?>

```

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.

exemples/pdo/ex03_queryForeachModeAssoc.php

```

1 <?php
2     // Connexion à la base de données :
3     require_once(dirname(__FILE__).'/ex03_connectToDatabasePDO.php');

```



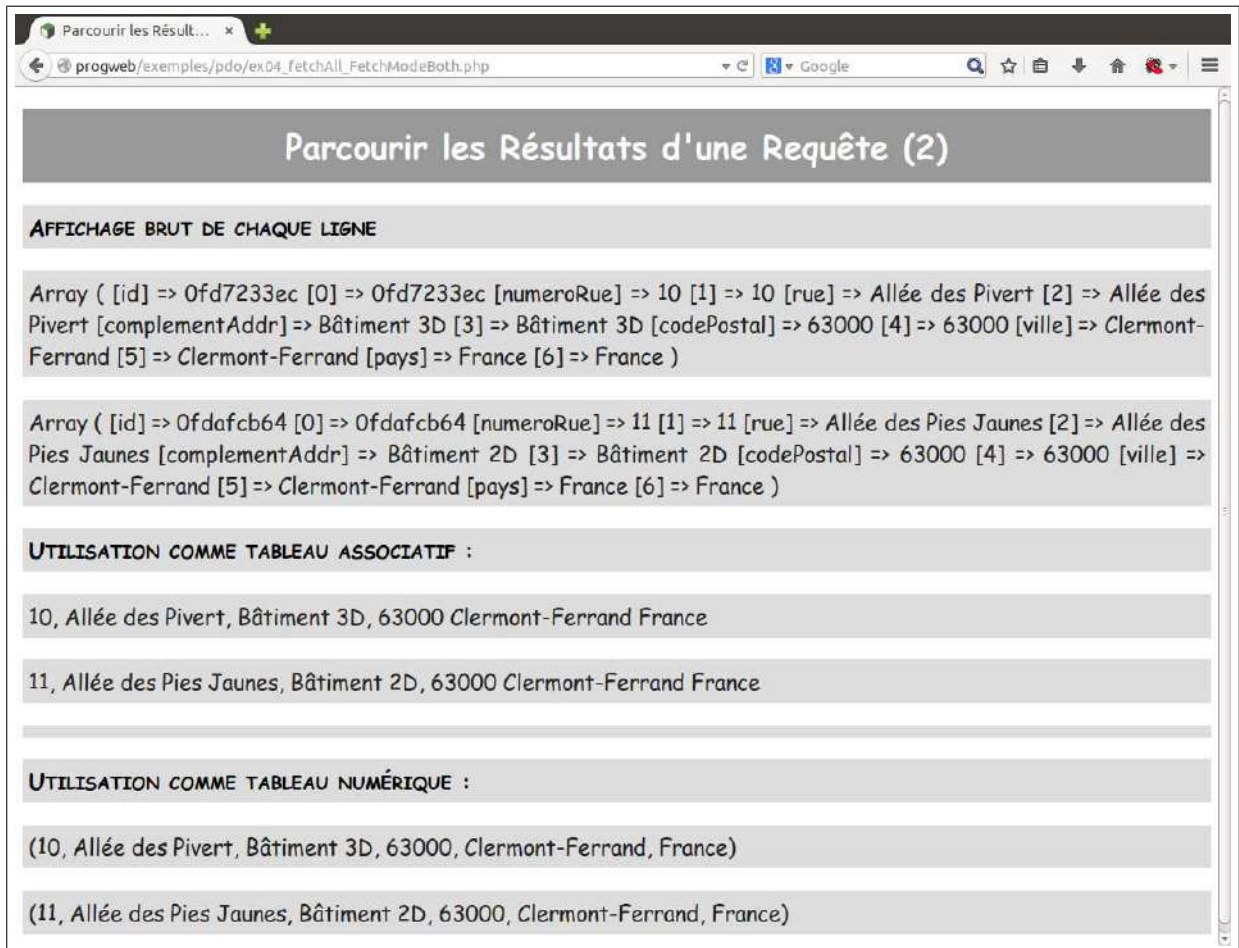
```

4
5 // Stockage des données résultat de la requête dans une variable
6 // De type PDOStatement
7 $statement = $dbh->query( 'SELECT * from Adresse ');
8 // On veut récupérer les lignes comme des tableaux Associatifs
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 objets 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).



exemples/pdo/ex04_fetchAll_FetchModeBoth.php

```

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 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
11     pas ...) ";
12     die();
13 }
14 $statement->setFetchMode(PDO::FETCH_BOTH);
15

```

```
16 // Pour pouvoir parcourir trois fois les résultats , on copie ceux-ci
17 // dans un grand tableau.
18 // Ça peut utiliser beaucoup de mémoire s'il y a beaucoup de lignes...
19 $stabResultats = $statement->fetchAll();
20
21 require_once( 'classes/VueHtmlUtils.php' );
22 echo CoursPHP\Vue\VueHtmlUtils::enTeteHTML5("Parcourir les Résultats d'une
    Requête", 'UTF-8', 'myStyle.css');
23 echo "<h1>Parcourir les Résultats d'une Requête (2)</h1>";
24 echo "<p>";
25 echo "<strong>Affichage brut de chaque ligne</strong>";
26 echo "</p>";
27
28 foreach($stabResultats 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($stabResultats 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 $row['pays'];
45     echo "</p>";
46 }
47 echo "</p>";
48
49 echo "<p>";
50 echo "<strong>Utilisation comme tableau numérique :</strong>";
51 echo "</p>";
52 foreach($stabResultats 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 objets 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 exhaustive!!!) 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

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 remplacer les ? dans la requête. **Ces valeurs, qui peuvent correspondre à des inputs utilisateur, sont automatiquement filtrées, 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. Nous en voyons une. Voyons déjà un exemple d'insertion d'une adresse dans une table.

L'adresse est saisie dans un formulaire :

exemples/pdo/ex05_formAdresse.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_requetesPrepareses.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="numeroRue" 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` :

exemples/pdo/ex06_retrieveInputPosts.php

```

1 <?php
2 $numeroRue="";
3 if (isset($_POST['numeroRue'])){
4     $numeroRue = filter_var($_POST['numeroRue'], FILTER_SANITIZE_STRING);
5 }
6
7 $rue="";
8 if (isset($_POST['rue'])){
9     $rue = filter_var($_POST['rue'], FILTER_SANITIZE_STRING);
10 }
11
12 $complementAddr="";
13 if (isset($_POST['complementAddr'])){
14     $complementAddr = filter_var($_POST['complementAddr'],
15         FILTER_SANITIZE_STRING);
16 }
17 $codePostal="";
18 if (isset($_POST['codePostal'])){
19     $codePostal = filter_var($_POST['codePostal'], FILTER_SANITIZE_STRING);
20 }
21 $ville="";
22 if (isset($_POST['ville'])){
23     $ville = filter_var($_POST['ville'], FILTER_SANITIZE_STRING);
24 }
25 $pays="France";
26 if (isset($_POST['pays']) && $_POST['pays'] != ""){
27     $pays = filter_var($_POST['pays'], FILTER_SANITIZE_STRING);

```

28 ?>

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).

exemples/pdo/ex07_requetesPreparees.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 sont des ?)
7 $statement = $dbh->prepare('INSERT INTO Adresse(id, numeroRue, rue, '
8                          . 'complementAddr, codePostal, ville, pays) '
9                          . 'VALUES (?, ?, ?, ?, ?, ?, ?)');
10
11 // Test en supposant le mode de gestion d'erreurs PDO: ERRMODE_SILENT
12 // (sinon, en mode PDO: ERRMODE_EXCEPTION il faudrait utiliser try...catch)
13 if ($statement === false){
14     $dataError['preparation-query'] = "Problème de préparation de la requête."
15     . "(par exemple, la syntaxe de la requête est invalide "
16     . "pour le driver utilisé...)";
17     require("vueErreur.php");
18     die();
19 }
20
21 // Liaison de variables avec les " ? " de la requête préparée :
22 // Le premier paramètre de bindParam est ici le numéro du " ? "
23 // en commençant par 1.
24 // Lors de l'exécution de la requête, chaque " ? " sera remplacé par
25 // le contenu de la variable correspondante.
26 $statement->bindParam(1, $id);
27 $statement->bindParam(2, $numeroRue);
28 $statement->bindParam(3, $rue);
29 $statement->bindParam(4, $complementAddr);
30 $statement->bindParam(5, $codePostal);
31 $statement->bindParam(6, $ville);
32 $statement->bindParam(7, $pays);
33
34 // Récupération des données du formulaires et affectation des variables
35 // $numeroRue, $rue, $complementAddr, $codePostal, $ville, $pays
36 // à partir des données utilisateur (tableau $_POST)
37 require(dirname(__FILE__).' /ex06_retrieveInputPosts.php ');
38
39 // Génération d'un $id difficile à deviner.
40 $id = hash("sha512", $numeroRue.$rue.$complementAddr.$codePostal.$ville.$pays)
41     ;
42 $id = substr($id, 0, 10); // respect de la forme des ID (10 chiffres hexa)

```



```

43 // Exécution de la requête. (Tous les "?" de la requête ont été liés à des
    // variables)
44 if ($statement->execute() === false){
45     $dataError["execute-query"] = "Problème d'exécution de la requête. "
46     . "(par exemple, une ligne avec cette clé primaire $id existe déjà...)";
47     require("vueErreur.php");
48 }else{ // Code de la vue :
49     require_once('classes/VueHtmlUtils.php');
50     echo CoursPHP\Vue\VueHtmlUtils::enTeteHTML5("Requêtes Préparées", 'UTF-8',
51     'myStyle.css');
52     echo "<h1>Requêtes Préparées (1)</h1>";
53     echo "<p>La requête a été exécutée.</p>";
54     echo CoursPHP\Vue\VueHtmlUtils::finFichierHTML5();
55 }
56
57 // Fermeture de la connexion (connexion non persistante)
58 $statement = null;
59 $dbh = null;
60 ?>

```

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

exemples/pdo/ex08_requetesPreparesSelect.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 sont des ?
7 $statement = $dbh->prepare('SELECT * FROM Adresse WHERE codePostal = ?');
8
9 // Test en supposant le mode de gestion des erreurs PDO:ERRMODE_SILENT
10 // (sinon, en mode PDO:ERRMODE_EXCEPTION il faudrait utiliser try...catch)
11 if ($statement === false){
12     $dataError['preparation-query'] = "Problème de préparation de la requête. "
13     . "(par exemple, la syntaxe de la requête est invalide "
14     . "pour le driver utilisé...)";
15     require("vueErreur.php");
16     die();
17 }
18
19 // Liaison de la variable $_GET['codePostal'] avec le "?" de la requête :
20 // Le premier paramètre de bindParam est ici le numéro du "?", à savoir 1
21 $statement->bindParam(1, $_GET['codePostal']);
22
23 // Test d'erreur en supposant le mode de gestion des erreurs PDO:
24 // ERRMODE_SILENT
25 // (sinon, avec le mode PDO:ERRMODE_EXCEPTION il faudrait utiliser avec try
26 // ...catch)
27 if ($statement->execute() !== false){ // Code de la vue :
28     require_once('classes/VueHtmlUtils.php');
29     echo CoursPHP\Vue\VueHtmlUtils::enTeteHTML5("Requêtes Préparées",
30     'UTF-8', 'myStyle.css');
31     echo "<h1>Requêtes Préparées (2)</h1>";
32     // Affichage des résultats de la requête
33     foreach ($statement as $row){

```

```

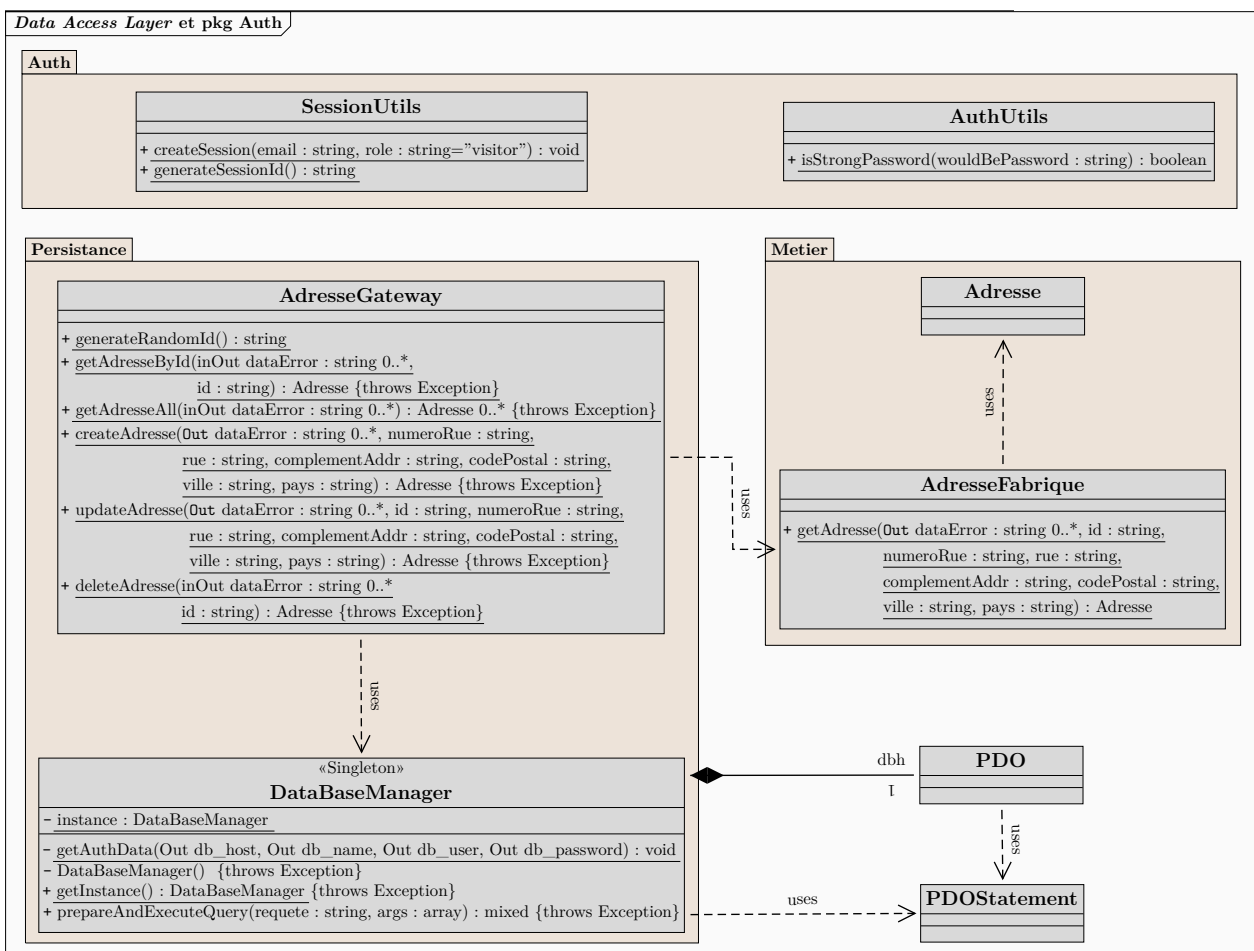
32     echo "<p>";
33     echo $row[ 'numeroRue ']. ", ". $row[ 'rue ']. ", ";
34     if (!empty($row[ 'complementAddr ']))
35         echo $row[ 'complementAddr ']. ", ";
36     echo $row[ 'codePostal ']. " ";
37     echo $row[ 'ville ']. " ";
38     echo $row[ 'pays '];
39     echo "</p>";
40 }
41 echo CoursPHP\Vue\VueHtmlUtils : :finFichierHTML5();
42 }else{ // Erreur lors de l'exécution de la requête
43     $dataError[ "execute-query" ] = "Problème d'exécution de la requête. "
44         . "(par exemple, une ligne avec cette clé primaire $id existe déjà...)";
45     require( "vueErreur.php" );
46     die();
47 }
48
49 // Fermeture de la connexion (connexion non persistante)
50 $statement = null;
51 $dbh = null;
52 ?>

```

Chapitre 9

Couche d'Accès aux données

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

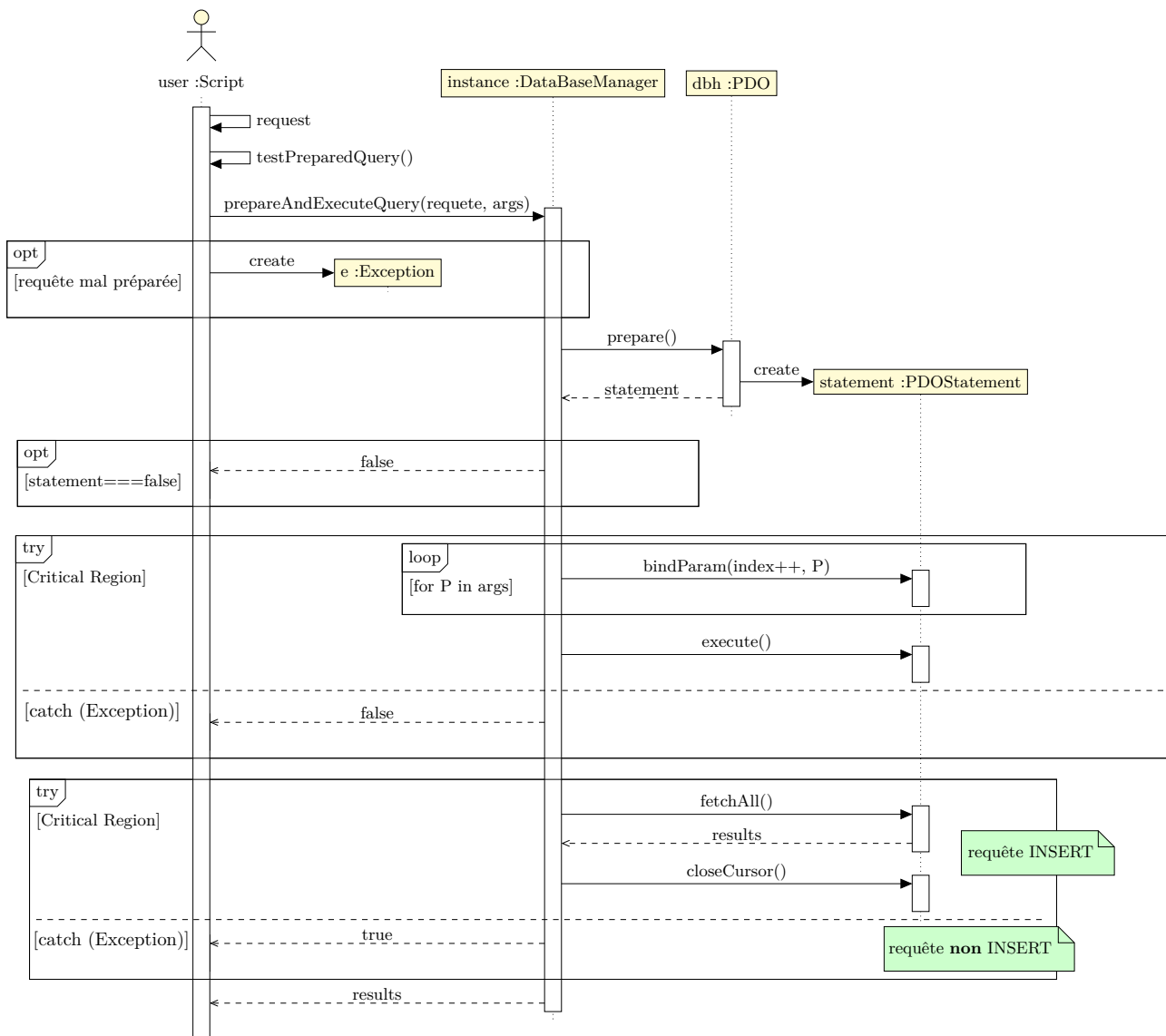
Nous présentons ici une classe `DataBaseManager` gestionnaire de connexion à une base de données. La connexion est persistante, c'est à dire que l'on ne va pas réinitialiser la connexion

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 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()`

`exemples/pdo/ex12_classeDataBaseManager.php`

```

1 <?php
2 namespace CoursPHP\Persistence;
3 /**
4  * @brief Classe permettant de gérer la connexion à une base de données
    
```

```
5  * L'ex cution de requ tes SQL avec pr paration "offerte service compris".
6  * La classe est g r e avec le pattern SINGLETON, qui permet
7  * d'avoir un exemplaire unique du gestionnaire de connexion,
8  * pour une connexion persistante.
9  * La classe encapsule compl tement PDO, y compris les exceptions.
10 */
11 class DataBaseManager{
12     /* Gestionnaire de connexion   la base de donn es avec PDO
13     * Variable de classe. */
14     static private $dbh = null;
15
16     /** R f rence de l'unique instance de la classe.
17     * Variable de classe, initialement null */
18     static private $instance=null;
19
20     /** @brief Donn es n cessaires   la connexion par le constructeur de PDO.
21     * Les valeurs pourraient  tre initialis es   partir d'un
22     * fichier de configuration s par  (par exemple classe Config)
23     */
24     private static function getAuthData(&$db_host, &$db_name,
25                                         &$db_user, &$db_password){
26         $db_host="mysql:host=localhost;";
27         $db_name="dbname=ExampleDataBase";
28         $db_user="remy";
29         $db_password="my_password";
30     }
31
32     /**
33     * @brief Constructeur priv . Cr e une instance de PDO (connexion BD SQL)
34     * Personne ne peut cr er des instances   tire larigot
35     * car dans le singleton il ne doit y avoir qu'une seule instance
36     * R cup re les exception PDO.
37     * @throws exception personnalis e en cas d'exception PDO
38     */
39     private function __construct(){
40         try {
41             self::getAuthData($db_host, $db_name, $db_user, $db_password);
42             // Cr ation de l'instance de PDO (database handler).
43             self::$dbh = new \PDO($db_host.$db_name, $db_user, $db_password);
44             // Rendre les erreurs PDO d tectables et g rables par exceptions :
45             self::$dbh->setAttribute(\PDO::ATTR_ERRMODE, \PDO::ERRMODE_EXCEPTION);
46         }catch (\PDOException $e){
47             throw new \Exception("Erreur de connexion   la base de donn es. "
48                                 ."Vous n'avez pas besoin d'en savoir plus...");
49         }
50     }
51
52     /**
53     * @brief M thode statique publique d'acc s   l'unique instance.
54     * Si l'instance n'existe pas, on la cr e.
55     * On retourne ensuite l'unique instance.
56     */
57     public static function getInstance()
58     {
59         if (null === self::$instance) {
60             self::$instance = new self;
```

```

61     }
62     return self::$instance;
63 }
64
65 /**
66  * @brief Prépare et exécute une requête.
67  * @param string $requete : requête avec des ?
68  * @param string[] $args : arguments à lier (binder) aux ? dans la requête
69  * @return false si la requête échoue,
70  *         true si succès mais pas de résultats (requête différente de SELECT),
71  *         ou résultats du SELECT dans un tableau à double entrée PHP standard
72  */
73 public function prepareAndExecuteQuery($requete, $args){
74     // récupération du nombre d'arguments :
75     $numargs = count($args);
76
77     // Une requête préparée ne doit pas contenir de guillemets !!!
78     if (empty($requete) || !is_string($requete) ||
79         preg_match('/(\\"|\')+/', $requete) !== 0)
80     {
81         throw new \Exception("Erreur concernant la sécurité. "
82             . "Requête incomplètement préparée.");
83     }
84
85     // On ne laisse pas remonter d'exceptions PDO
86     try{
87         // Préparation de la requête
88         $statement = self::$dbh->prepare($requete);
89         if ($statement !== false){
90             // On parcourt les arguments en commençant au deuxième
91             // on commence après le paramètre $requete
92             for ($i=1 ; $i <= $numargs; $i++){
93                 // Lien entre l'argument et le "?" numéro i
94                 // (rappel : les "?" sont numérotés à partir de 1)
95                 $statement->bindParam($i, $args[$i-1]);
96             }
97
98             // Exécution de la requête préparée :
99             $statement->execute();
100         }
101     }catch (\Exception $e){
102         return false;
103     }
104
105     if ($statement === false){
106         return false;
107     }
108
109     try{
110         // Transfert des résultats de la requête dans un array
111         $results = $statement->fetchAll(\PDO::FETCH_ASSOC);
112         // destruction des données du PDOstatement
113         $statement->closeCursor();
114     } catch (\PDOException $e){
115         // La requête a été exécutée mais pas de résultats
116         // La requête n'est pas de type SELECT...

```

```

117     $results = true;
118 }
119
120 // Libération via la grabage collector
121 $statement = null;
122
123 return $results; // retour des données de requête
124 }
125
126 /**
127  * @brief on interdit le clonage (pour le pattern singleton).
128  */
129 private function __clone() {}
130 }
131 ?>

```

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*.

exemples/pdo/ex13_testSingletonPDO.php

```

1 <?php
2     require_once(dirname(__FILE__).' /classes/DataBaseManager.php ');
3
4     try{
5         $statement = CoursPHP\Persistence\DataBaseManager::getInstance()->
6             prepareAndExecuteQuery(
7                 'SELECT * FROM Adresse WHERE codePostal = ?',
8                 array(isset($_GET['codePostal']) ? $_GET['codePostal'] : ""));
9     }catch (Exception $e){
10         $dataError[] = $e->getMessage();
11         require("vueErreur.php");
12         die();
13     }
14
15     if ($statement === false){
16         // Erreur lors de l'exécution de la requête
17         $dataError[] = "Problème lors de la préparation de la requête. "
18             . "(par exemple, la donnée codePostal passée par GET est invalide...)";
19         require("vueErreur.php");
20         die();
21     }
22
23     // Code de la vue :
24     require_once('classes/VueHtmlUtils.php');
25     echo CoursPHP\Vue\VueHtmlUtils::enTeteHTML5("Classe Singleton de Connection",
26         'UTF-8', 'myStyle.css');
27
28     // Affichage des résultats de la requête
29     foreach ($statement as $row){
30         echo "<p>";
31         echo $row['numeroRue']. " ", ". $row['rue']. " ", ";
32         if (!empty($row['complementAddr']))
33             echo $row['complementAddr']. " ", ";
34         echo $row['codePostal']. " ";
35         echo $row['ville']. " ";
36         echo $row['pays'];
37         echo "</p>";

```

```

36     }
37     CoursPHP\Vue\VueHtmlUtils ::finFichierHTML5 ();
38     ?>

```

9.3 Classes *Gateway* : Persistance des Objets Métiers

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. 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 5.4) pour retourner les résultats (ou les erreurs) à des fin, par exemple, d'affichage.

Remarques.

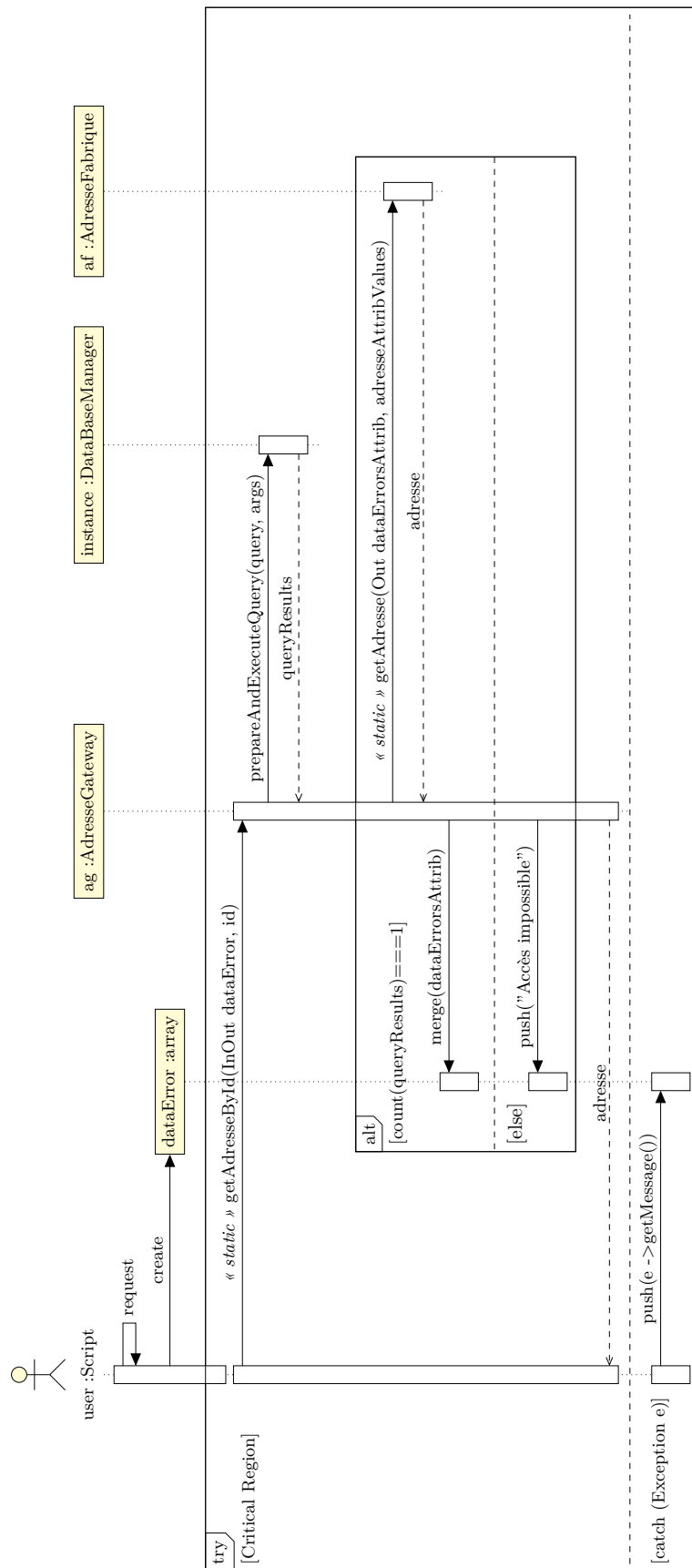
1. Dans l'implémentation ci-dessous, pour la création d'une nouvelle instance, nous proposons une recette de cuisine pour la génération de l'*id* unique de l'adresse. Signalons que cette technique ne passe pas à l'échelle, notamment pour une utilisation dans le cadre de *Web Services*. Dans ce cas, des techniques robustes, appelées *Universal Unique Identifier (UUID)* peuvent être utilisées, et des implémentations en *PHP* existent sous forme de bibliothèques.
2. 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 jointures entre les tables correspondantes (voir la partie 13.5 pour un exemple d'application comportant une jointure).

exemples/pdo/classes/AdresseGateway.php

```

1 <?php
2 namespace CoursPHP\Persistance;
3
4 class AdresseGateway{
5
6     /**
7     * @brief Génère un ID unique de 10 chiffres hexa aléatoires (soit 5 octets).
8     * Dans la pratique, préférer une implémentation d'UUID (ID unique universel).
9     */
10    public static function generateRandomId()
11    {
12        // Génération de 5 octets (pseudo-)aléatoires codés en hexa
13        $cryptoStrong = false; // Variable pour passage par référence
14        $octets = openssl_random_pseudo_bytes(5, $cryptoStrong);
15        return bin2hex($octets);
16    }
17
18    /**
19    * Permet de récupérer une adresse à partir de son ID.
20    * @param dataError : données d'erreurs (couple type => message) par référence

```

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

```

21  * @param id : clé primaire de l'adresse à récupérer
22  * @return instance d'Adresse en cas de succès, undefined sinon.
23  */
24  public static function getAdresseById(&$dataError, $id){
25      if (isset($id)){
26          // Exécution de la requête via la classe de connexion (singleton)
27          // Les exceptions éventuelles, personnalisées, sont gérées plus haut
28          $queryResults = DataBaseManager::getInstance()->prepareAndExecuteQuery(
29              'SELECT * FROM Adresse WHERE id = ?', array($id));
30          // Si l'exécution de la requête a fonctionné
31          if (isset($queryResults) && is_array($queryResults)){
32              // si une et une seule adresse a été trouvée
33              if (count($queryResults) == 1){
34                  $row = $queryResults[0];
35                  $adresse = \CoursPHP\Metier\AdresseFabrique::getAdresse(
36                      $dataErrorAttributes,
37                      $row['id'], $row['numeroRue'], $row['rue'],
38                      $row['complementAddr'], $row['codePostal'],
39                      $row['ville'], $row['pays']);
40                  $dataError = array_merge($dataError, $dataErrorAttributes);
41              }else{
42                  $dataError['persistence'] = "Adresse d'ID \"$.id.\" introuvable.";
43              }
44          }else{
45              $dataError['persistence'] = "Impossible d'accéder aux données.";
46          }
47      }else{
48          $dataError['persistence'] = "Adresse d'ID \"$.id.\" introuvable.";
49      }
50      return $adresse;
51  }
52
53  /**
54   * Permet de récupérer une collection d'adresses présentes dans la table.
55   * @param dataError : données d'erreurs (couple type => message) par référence
56   * @return collection d'Adresses en cas de succès, collection vide sinon.
57   */
58  public static function getAdresseAll(&$dataError){
59
60      // Exécution de la requête via la classe de connexion (singleton)
61      // Les exceptions éventuelles, personnalisées, sont gérées plus haut
62      $queryResults = DataBaseManager::getInstance()->prepareAndExecuteQuery(
63          'SELECT * FROM Adresse', array());
64
65      // Construction de la collection des résultats (fabrique)
66      $collectionAdresse = array();
67      // Si l'exécution de la requête a fonctionné
68      if ($queryResults !== false){
69          // Parcours des lignes du résultat de la requête :
70          foreach ($queryResults as $row){
71              // Ajout d'une adresse dans la collection :
72              $adresse = \CoursPHP\Metier\AdresseFabrique::getAdresse(
73                  $dataErrorAttributes,
74                  $row['id'], $row['numeroRue'], $row['rue'],
75                  $row['complementAddr'], $row['codePostal'],
76                  $row['ville'], $row['pays']);

```

```
77     $collectionAdresse [] = $adresse;
78     $dataError = array_merge($dataError, $dataErrorAttributes);
79     }
80 }else{
81     $dataError['persistence'] = "Probl me d'acc s aux donn es.";
82 }
83
84 return $collectionAdresse;
85 }
86
87 /**
88  * @brief Met   jour une adresse (Update)
89  * @return l'instance d'Adresse (erreurs + donn es de l'adresse saisie)
90  */
91 public static function updateAdresse(&$dataError, $id, $numeroRue,
92                                     $rue, $complementAddr,
93                                     $codePostal, $ville, $pays){
94
95     // Tentative de construction d'une instance (et filtrage)
96     $adresse = \CoursPHP\Metier\AdresseFabrique::getAdresse($dataErrorAttributes
97
98         ,
99         $id, $numeroRue, $rue, $complementAddr,
100         $codePostal, $ville, $pays);
101     // Si la forme des attributs sont corrects (expressions r guli res - setters
102     )
103     if (empty($dataErrorAttributes)){
104         // Ex cution de la requ te de mis   jour :
105         $queryResults = DataBaseManager::getInstance()->prepareAndExecuteQuery(
106             'UPDATE Adresse SET numeroRue=?, rue=?, '
107             . 'complementAddr=?, codePostal=?, ville=?, pays=? '
108             . 'WHERE id=?',
109             array($adresse->getNumeroRue(),
110                 $adresse->getRue(),
111                 $adresse->getComplementAddr(),
112                 $adresse->getCodePostal(),
113                 $adresse->getVille(),
114                 $adresse->getPays(),
115                 $adresse->getId()
116             )
117         );
118         if ($queryResults == false){
119             $dataError["persistence"] = "Probl me d'acc s aux donn es.";
120         }
121     }else{
122         $dataError = array_merge($dataError, $dataErrorAttributes);
123     }
124
125     return $adresse;
126 }
127
128 /**
129  * @brief Ins re une nouvelle adresse (Create)
130  * @return l'instance d'Adresse (erreurs + donn es de la adresse saisie)
131  */
132 public static function createAdresse(&$dataError, $numeroRue,
133                                     $rue, $complementAddr,
```

```

131         $codePostal, $ville, $pays){
132
133     // Tentative de construction d'une instance (et filtrage)
134     $adresse = \CoursPHP\Metier\AdresseFabrique::getAdresse($dataErrorAttributes
135         ,
136         "0000000000", $numeroRue, $rue, $complementAddr,
137         $codePostal, $ville, $pays);
138
139     // Si la forme des attributs sont corrects (expressions régulières -
140         setters)
141     if (empty($dataErrorAttributes)){
142         // Exécution de la requête d'insertion :
143         $queryResults = false;
144         $count = 0;
145         // Boucle en cas de collision de l'ID
146         // Autre possibilité : utiliser le nombre de lignes (COUNT)
147         // ou le maximum par ordre alpha ou hexa des ID de la table pour calculer
148         // un ID qui n'est pas présent dans la table. Concaténer avec de l'alé
149         // atoire
150         while ($queryResults == false && $count <= 3){
151             $adresse->setId(self::generateRandomId()); // Identifiant aléatoire
152             $count++;
153             $queryResults = DataBaseManager::getInstance()->prepareAndExecuteQuery(
154                 'INSERT INTO Adresse(id, numeroRue, rue, '
155                 . 'complementAddr, codePostal, ville, pays) '
156                 . 'VALUES (?, ?, ?, ?, ?, ?, ?)',
157                 array($adresse->getId(),
158                     $adresse->getNumeroRue(),
159                     $adresse->getRue(),
160                     $adresse->getComplementAddr(),
161                     $adresse->getCodePostal(),
162                     $adresse->getVille(),
163                     $adresse->getPays()
164                 )
165             );
166         }
167         // 4 collisions d'affilée, c'est très louche...
168         if ($queryResults == false){
169             $dataError["persistance"] = "Problème d'exécution de la requête.";
170         }
171     }else{
172         $dataError = array_merge($dataError, $dataErrorAttributes);
173     }
174     return $adresse;
175 }
176 /**
177  * @brief Supprime une adresse à partir de son ID.
178  * Retourne le modèle de données (erreurs + données de l'Adresse supprimée)
179  */
180 public static function deleteAdresse(&$dataError, $id){
181     // Test si l'adresse existe et récupérations données à supprimer
182     $dataErrorIdSearch = array();
183     $adresse = self::getAdresseById($dataErrorIdSearch, $id);

```

```

184
185 if (empty($dataErrorIdSearch)){
186     // Exécution de la requête via la classe de connexion (singleton)
187     // Les exceptions éventuelles, personnalisées, sont gérées par le
188     // Contrôleur
189     $queryResults = DataBaseManager::getInstance()->prepareAndExecuteQuery(
190         'DELETE FROM Adresse WHERE id=?',
191         array($id)
192     );
193     if ($queryResults == false){
194         $dataError["persistence"] = "Problème d'exécution de la requête.";
195     }
196 }else{
197     $dataError = array_merge($dataError, $dataErrorIdSearch);
198 }
199 return $adresse;
200 }
201 }
202 ?>

```

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



exemples/pdo/ex15_testAdresseGateway.php

```

1 <?php
2     require_once(dirname(__FILE__). '/classes/DataBaseManager.php ');
3     require_once(dirname(__FILE__). '/classes/AdresseGateway.php ');

```

```

4  require_once(dirname(__FILE__).'/classes/Adresse.php');
5  require_once(dirname(__FILE__).'/classes/AdresseView.php');
6  require_once(dirname(__FILE__).'/classes/AdresseFabrique.php');
7
8  $dataError = array();
9
10 // Recherche d'adresse avec ID inexistant :
11 try{
12     $adresseByIdFail = CoursPHP\Persistence\AdresseGateway::getAdresseById(
13         $dataError, "dummyId");
14
15 }catch (Exception $e){
16     $dataError [] = $e->getMessage();
17 }
18
19 // Création d'une adresse
20 try{
21     $adresseCree = CoursPHP\Persistence\AdresseGateway::createAdresse(
22         $dataError, "1", "Boulevard des Entiers Longs",
23         "", "63000", "Clermont-Ferrand", "France");
24 }catch (Exception $e){
25     $dataError [] = $e->getMessage();
26 }
27
28 // Mise à jour d'une adresse
29 try{
30     $adresseMaj = CoursPHP\Persistence\AdresseGateway::updateAdresse($dataError,
31         $adresseCree->getId(),
32         "1", "Boulevard des Entiers Longs", "Complément Utile",
33         "63000", "Clermont-Ferrand", "France");
34 }catch (Exception $e){
35     $dataError [] = $e->getMessage();
36 }
37
38 // Mise à jour d'une adresse avec erreur de forme d'un attribut (Code Postal)
39 try{
40     $adresseMaj = CoursPHP\Persistence\AdresseGateway::updateAdresse($dataError,
41         $adresseCree->getId(),
42         "1", "Boulevard des Entiers Longs", "Complément Utile",
43         "63000@", "Clermont-Ferrand", "France");
44 }catch (Exception $e){
45     $dataError [] = $e->getMessage();
46 }
47
48 // Recherche d'adresse par ID :
49 try{
50     $adresseById = CoursPHP\Persistence\AdresseGateway::getAdresseById(
51         $dataError, $adresseCree->getId());
52 }catch (Exception $e){
53     $dataError ["exception"] = $e->getMessage();
54 }
55
56 // Récupération de toutes les adresses
57 try{
58     $adresseAll= CoursPHP\Persistence\AdresseGateway::getAdresseAll($dataError);
59 }catch (Exception $e){

```

```
60     $dataError [] = $e->getMessage ();
61 }
62
63 // Suppression d'une adresse
64 try{
65     $adresseDeleted = CoursPHP\Persistence\AdresseGateway::deleteAdresse(
66         $dataError, $adresseCree->getId());
67 }catch (Exception $e){
68     $dataError [] = $e->getMessage ();
69 }
70
71 // Code de la vue :
72 require_once( 'classes/VueHtmlUtils.php ');
73 echo CoursPHP\Vue\VueHtmlUtils::enTeteHTML5("Classe Singleton pour Connexion",
74     'UTF-8', 'myStyle.css ');
75 echo "<h1>Test de la <i>Gateway</i> d'Adresse</h1>";
76
77 // Liste des erreurs :
78 echo "<h2>Erreurs détectées :</h2>";
79 if (empty($dataError)){
80     echo "<p>Aucune erreur. "</p>";
81 }
82 foreach ($dataError as $errorMsg){
83     echo "<p>". $errorMsg. "</p>";
84 }
85
86 echo "<h2>Résultats obtenus :</h2>";
87 // Affichage des résultats de la requête
88
89 echo "<p>";
90 echo "Adresse par ID : "
91     .CoursPHP\Vue\AdresseView::getHtmlCompact($adresseById). "<br/>";
92 echo "Adresse supprimée : "
93     .CoursPHP\Vue\AdresseView::getHtmlCompact($adresseDeleted). "</p>";
94
95 echo "<p>";
96 echo "<strong>Toutes les Adresses : </strong><br>";
97 foreach ($adresseAll as $adresse){
98     echo CoursPHP\Vue\AdresseView::getHtmlCompact($adresse). "<br/>";
99 }
100 echo "</p>";
101
102 CoursPHP\Vue\VueHtmlUtils::finFichierHTML5();
103 ?>
```

Quatrième partie
Conception d'Architectures Avancées



Table of Contents

10 Analyse Fonctionnelle	163
10.1 <i>Storyboards</i>	163
10.2 Diagrammes de Cas d'Utilisations	164
11 Organisation des Répertoires et Configuration	165
11.1 Organisation des Répertoires	165
11.2 <i>Autoload</i>	166
11.3 La classe <i>Config</i> : éviter les <i>URL</i> en dût	168
12 Architectures <i>MVC</i>	172
12.1 Le Contrôleur	172
12.2 Le Modèle	176
12.3 Les Vues	180
13 Utilisateurs et <i>Front Controller</i>	182
13.1 <i>Storyboards</i>	182
13.2 Diagramme de Cas d'Utilisation	183
13.3 Le <i>Front-Controller</i>	183
13.4 Gestion de l'Authentification	190
13.4.1 Modèle et <i>Gateway</i> de la table <i>User</i>	190
13.4.2 Gestion des sessions et des <i>cookies</i>	192
13.5 Gestion de plusieurs classes métier	193
13.5.1 Exemple de classes métiers avec agrégation	193
13.5.2 Structuration des contrôleurs	193

Chapitre 10

Analyse Fonctionnelle

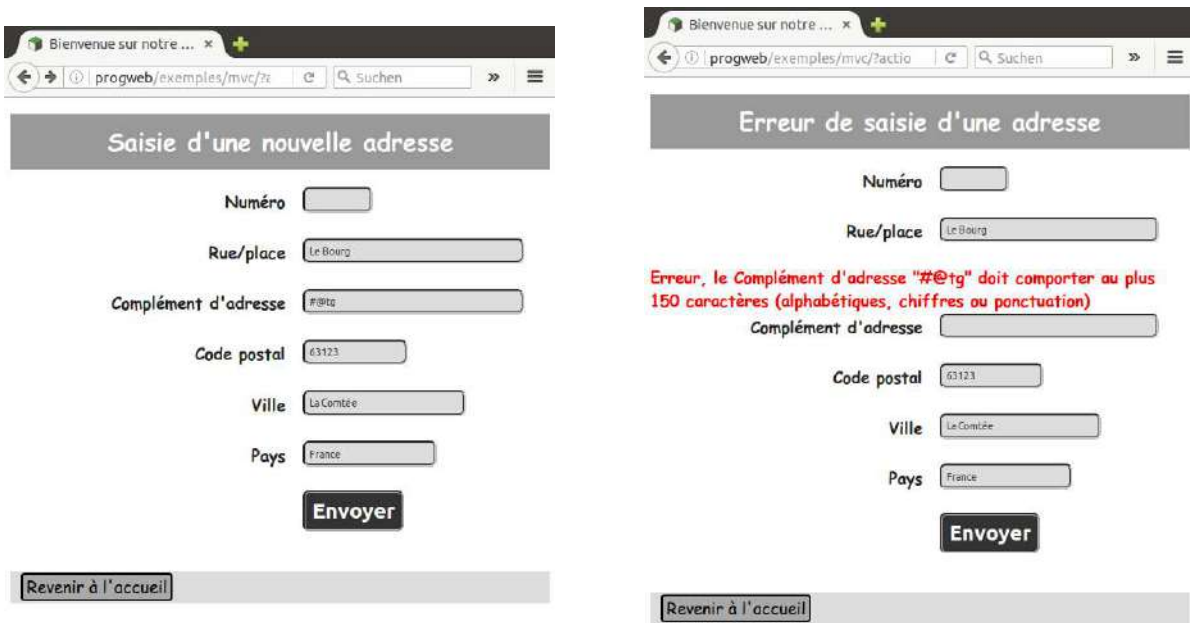
10.1 Storyboards

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



(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.

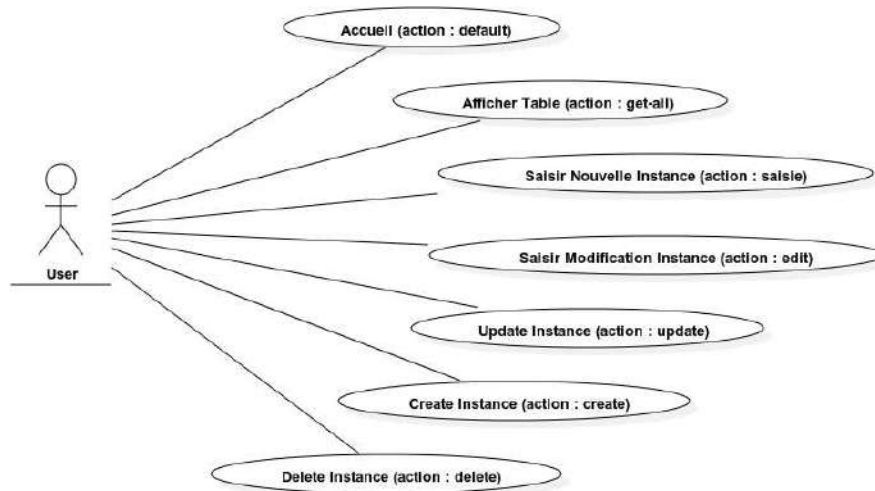


FIGURE 10.3 : *Use Case Diagram* : Les actions possibles pour l'utilisateur

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/` ont été étudiées dans le chapitre 5. Les classes du répertoire `Vue/` ont été étudiées dans ce même chapitre 5. et dans le chapitre 2. La classe `DataBaseManager` du répertoire `Persistence/` 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 `ValidationRequest`, qui a été abordée dans les parties 5.6 (pour la validation d'adresses) et 7.8 (pour la validation du *login/password*), les classes du répertoire `Controleur`, ainsi que les classes du répertoire `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
|   |-- ValidationRequest.php
|-- css
|   |-- defaultStyle.css
|-- Metier
|   |-- AdresseFabrique.php
|   |-- Adresse.php
|   |-- ExpressionsRegexUtils.php
|-- Modele
|   |-- ModelAdresse.php
|   |-- ModelCollectionAdresse.php
|   |-- Model.php
|-- Persistence
|   |-- 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` d clare et impl mente une m thode *callback* qui sera appel e lors de l'utilisation dans le programme d'une classe (ou d'un *trait*) inconnu(e). La m thode *callback* en question cherche alors dans les r pertoires un fichier source *PHP* dont le nom correspond   celui de la classe en question, et charge ce fichier source pour d finir la classe «   la vol e ».

Nous pr sentons une classe `Autoload` conforme   la norme *PSR-4* (pour *PHP Standard Recommendations*, voir la partie 2.1.2). Suivant cette norme, les *namespaces* du modules comportent tous un pr fixe, appel  *Vendor Name* du module. Dans nos exemples, ce pr fixe est le *namespace* : `\CoursPHP`. Le but de ce pr fixe est de garantir que des *frameworks* diff rents ne produiront pas de collisions dans les noms de *namespaces*, de classe, etc. qui poseraient des probl mes d'interop rabilit .

En outre, le chemin relatif complet (d finissant aussi le sous-r pertoire du r pertoire racine de notre module) vers le fichier source de la classe doit  tre nomm  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 d terminer automatiquement l'emplacement du fichier source de la classe   partir de son nom complet.

La transformation du nom complet de la classe en chemin vers le fichier source est illustr e ci-dessous :

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

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

exemples/mvcLatex/exPoly/ex01_autoload.php

```

1 <?php
2 namespace CoursPHP\Config;
3 /**
4  * @brief classe Autoload : permet de charger automatiquement les classes.
5  * La m thode autoloadCallback() permet de charger le code source
6  * d'une classe dont le nom est pass  en param tre.
7  * Pour cela, la m thode load() d clare autoloadCallback()
8  * par un appel   spl_autoload_register()
9  */
10 class Autoload{
11
12     /** Pr fixe principal des namespaces du projet */
13     public static $vendorNamespace;
14
15     /**
16     * @brief Enregistrement du callback d'autoload.
17     * @param vendorNamespace Pr fixe principal des namespaces du projet
18     */
19     public static function load_PSR_4($vendorNamespace){
20         self::$vendorNamespace = $vendorNamespace;
21         // Enregistrement d'un callback charg  d'inclure les classes.
22         // Il peut y en avoir plusieurs, appel s successivement
23         // en cas d' chec des premier...
24         spl_autoload_register('CoursPHP\Config\Autoload::autoloadCallback_PSR_4');
```

```

25 }
26
27 /**
28  * @brief Callback d'Autoload suivant la norme PSR-4,
29  * Cette méthode est appelée automatiquement en cas d'instanciation
30  * d'une classe inconnue. La méthode charge alors la classe en question.
31  *
32  * @param class : nom de la classe à charger.
33  *
34  * @note L'arborescence des répertoires et les noms de fichiers PHP
35  * contenant les classes doivent coïncider avec les sous-namespace
36  * de la classe pour trouver directement le répertoire contenant
37  * le fichier source de la classe.
38  */
39 public static function autoloadCallback_PSR_4($class) {
40
41     // La classe a-t-elle le bon préfixe de namespace ?
42     $longueurVendorNamespace = strlen(self::$vendorNamespace);
43     if (strncmp(self::$vendorNamespace, $class, $longueurVendorNamespace) !== 0)
44     {
45         // Echec de l'autoloader. Espérons qu'il y en a un deuxième...
46         return;
47     }
48
49     // On enlève le préfixe "Vendor Namespace".
50     $relativeClass = substr($class, $longueurVendorNamespace);
51
52     // Chemin vers le fichier source de la classe :
53     $filePath = $rootDirectory.str_replace('\\', '/', $relativeClass).'.php';
54
55     // si le fichier existe
56     if (file_exists($filePath)) {
57         // Chargement de la classe :
58         require($filePath);
59     }
60 } // fin de la classe Autoload
61 ?>

```

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) :



exemples/mvcLatex/autoload/testAutoload.php

```

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 $adresse1 = new CoursPHP\Metier\Adresse("0af46d3bd9", '10', 'allée du net',
11     'Quartier de l'avenir', '63000', 'Clermont-Ferrand', 'France');
12
13 echo \CoursPHP\Vue\VueHtmlUtils::enTeteHTML5('Ma première classe PHP',
14     'UTF-8', 'http://progweb/exemples/mvc/css/defaultStyle.css');
15 echo "<h1>Test d'Autoload</h1>";
16 echo "<p>";
17 echo "<strong>Adresse :</strong><br/>".
18     \CoursPHP\Vue\AdresseView::getHtmlCompact($adresse1). "<br/>";
19 echo "</p>";
20 echo \CoursPHP\Vue\VueHtmlUtils::finFichierHTML5();
21 ?>

```

11.3 La classe Config : éviter les URL en dût

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 `index.php`, 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 `index.php`).

Voici un fichier qui illustre le calcul des différentes parties de l'*URL* complète du *script* :

exemples/mvcLatex/autoload/testURI.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 coupe l'URL du script au niveau de l'extension ".php"
7 $scriptWithoutExtention = explode(".php", $_SERVER['PHP_SELF'])[0];
8 // 2) puis on s'arrête au dernier slash (pour enlever la basename du script)
9 $longueurRootURI = strpos($scriptWithoutExtention, '/');
10 // 3) On prend le début de l'URL en coupant à la bonne longueur

```



```

11 $rootURI = substr($_SERVER['REQUEST_URI'], 0, $longueurRootURI);
12
13 // chargement de l'autoload pour autochargement des classes
14 require_once($rootDirectory . '/Config/Autoload.php');
15 \CoursPHP\Config\Autoload::load_PSR_4('CoursPHP\');
16
17 echo \CoursPHP\Vue\VueHtmlUtils::enTeteHTML5("Calcul de l'URI",
18     'UTF-8', 'http://progweb/exemples/mvc/css/defaultStyle.css');
19 echo "<h1>Calcul de l'<i>URI</i> de la racine du <i>MVC</i></h1>";
20 echo "<ul>";
21 echo "<li><strong>$_SERVER['REQUEST_URI'] </strong> <code>".$_SERVER['
22     REQUEST_URI']. "</code></li>";
23 echo "<li><strong>$_SERVER['PHP_SELF'] </strong> <code>".$_SERVER['PHP_SELF'
24     ]. "</code></li>";
25 echo "<li><strong>Script sans extension </strong> <code>".
26     $scriptWithoutExtention. "</code></li>";
27 echo "<li><strong><i>URI</i> de la racine du <i>MVC</i> </strong> <code>".
28     $rootURI. "</code></li>";
29
30 echo "<li><strong>$_SERVER['SERVER_NAME'] </strong> <code>".$_SERVER['
31     SERVER_NAME']. "</code></li>";
32 echo "<li><strong><i>URL</i> de la racine du <i>MVC</i> </strong> <code>http
33     ://
34     ".$_SERVER['SERVER_NAME'].
35     $rootURI. "</code></li>";
36 echo "<li><strong>Nom du fichier source <i>PHP</i> </strong> <code>".
37     . basename($_SERVER['PHP_SELF']
38     ). "</code></li>";
39 echo "<li><strong><i>URL</i> absolue complète de la requête </strong> <code>
40     http ://
41     ".$_SERVER['SERVER_NAME']. $rootURI. "/" . basename(
42     $_SERVER['PHP_SELF']). "?" .
43     $_SERVER['QUERY_STRING']. "</code></li>";
44 echo "</ul>";
45 echo \CoursPHP\Vue\VueHtmlUtils::finFichierHTML5();
46 ?>
    
```

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`.

exemples/mvcLatex/exPoly/ex02_config.php

```
1 <?php
2 namespace CoursPHP\Config;
3 /**
4  * @brief Classe de configuration
5  * Donne accès aux paramètres spécifiques concernant l'application
6  * telles que les chemins vers les vues, les vues d'erreur,
7  * les hash pour les ID de sessions, etc.
8  */
9 class Config
10 {
11     /** @brief Données nécessaires à la connexion à la base de données.
12     * Les valeurs pourraient être initialisées à partir d'un
13     * fichier de configuration séparé (require('configuration.php'))
14     * pour faciliter la maintenance par le webmaster.
15     */
16     public static function getAuthData(&$db_host, &$db_name,
17                                       &$db_user, &$db_password){
18         $db_host="mysql:host=localhost;";
19         $db_name="dbname=ExampleDataBase";
20         $db_user="remy";
21         $db_password="my_password";
22     }
23
24     /**
25     * @brief retourne le tableau des (chemins vers les) vues
26     */
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     /**
42     * @brief retourne le tableau des (chemins vers les) vues d'erreur
43     */
44     public static function getVuesErreur(){
45         // Racine du site
46         global $rootDirectory;
47         // Répertoire contenant les vues d'erreur
48         $vueDirectory = $rootDirectory."Vue/vues/";
49         return array(
50             "default" => $vueDirectory."vueErreurDefault.php",
51             "saisieAdresseCreate" => $vueDirectory."vueErreurSaisieCreate.php",
```

```
52     "saisieAdresseUpdate" => $vueDirectory."vueErreurSaisieUpdate.php"
53     );
54 }
55
56 /**
57  * @brief retourne l'URI (sans le nom d'hôte du site et sans la query string)
58  * du répertoire la racine de notre architecture MVC.
59  * Exemple : pour l'URL http://example.org/path/to/my/mvc/?action=goToSleep,
60  * l'URI est : /path/to/my/mvc/
61  */
62 public static function getRootURI(){
63     global $rootURI; // variable globale initialisée dans l'index.php
64     return $rootURI;
65 }
66
67 /**
68  * @brief retourne le tableau des (URLS vers les) feuilles de style CSS
69  */
70 public static function getStyleSheetsURL(){
71     // Répertoire contenant les styles css
72     // Le nettoyage par filter_var évite tout risque d'injection XSS
73     $cssDirectoryURL = filter_var(
74         "http://".$_SERVER['SERVER_NAME'].self::getRootURI()."/css/",
75         FILTER_SANITIZE_URL);
76     return array("default" => $cssDirectoryURL."defaultStyle.css");
77 }
78
79 /**
80  * @brief Génère 10 chiffres hexa aléatoires (soit 5 octets) :
81  */
82 public static function generateRandomId()
83 {
84     // Génération de 5 octets (pseudo-)aléatoires codés en hexa
85     $cryptoStrong = false; // Variable pour passage par référence
86     $octets = openssl_random_pseudo_bytes(5, $cryptoStrong);
87     return bin2hex($octets);
88 }
89 }
90 ?>
```

Chapitre 12

Architectures *MVC*

12.1 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 `index.php`) 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.

exemples/mvcLatex/exPoly/ex03_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 // 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.

exemples/mvcLatex/exPoly/ex04_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 constructeur 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['id']) ? $_REQUEST['id'] : "";
59     $id = filter_var($rawId, FILTER_SANITIZE_STRING);
60     // Construction du modèle et implémentation de la persistance :
61     $modele = \CoursPHP\Model\ModelAdresse::getModelAdresse($id);
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     $modele = \CoursPHP\Model\ModelCollectionAdresse::getModelAdresseAll();
75     // test d'erreur :
76     if ($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\Model\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['id']) ? $_REQUEST['id'] : "";
97     $id = filter_var($_REQUEST['id'], FILTER_SANITIZE_STRING);
98     // Construction du modèle et implémentation de la persistance :
99     $modele = \CoursPHP\Model\ModelAdresse::getModelAdresse($id);
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     // valider ou nettoyer les inputs (par exemple : filter_var)
112     ValidationRequest::validationAdresse($id, $numeroRue, $rue, $complementAddr,
113                                         $codePostal, $ville, $pays);
114     // Construction du modèle et implémentation de la persistance :
115     $modele = \CoursPHP\Model\ModelAdresse::getModelAdresseUpdate(
116         $id, $numeroRue, $rue, $complementAddr, $codePostal,
117         $ville, $pays);
118     // test d'erreur :
119     if ($modele->getError() === false){ // Appel de la vue
120         require (\CoursPHP\Config\Config::getVues()["afficheAdresse"]);
121     }else{ // Appel de la vue d'erreur
122         if (!empty($modele->getError()['persistance'])){
123             // Erreur d'accès à la base de donnée
124             require (\CoursPHP\Config\Config::getVuesErreur()["default"]);
125         }else{ // Appel de la vue d'erreur
126             // Erreur de saisie (attribut incorrect)
127             require (\CoursPHP\Config\Config::getVuesErreur()["saisieAdresseUpdate"
128                 ]);
129         }
130     }
131 }
132
133 /** @brief Implemente l'action "create" : crée une instance dans la BD
134 */
135 private function actionCreate(){
136     // valider ou nettoyer les inputs (par exemple : filter_var)
137     ValidationRequest::validationAdresse($id, $numeroRue, $rue, $complementAddr,
138                                         $codePostal, $ville, $pays);
139     // Construction du modèle et implémentation de la persistance :
140     $modele = \CoursPHP\Model\ModelAdresse::getModelAdresseCreate(
141         $numeroRue, $rue, $complementAddr, $codePostal, $ville, $pays);
142     // test d'erreur :
143     if ($modele->getError() === false){ // Appel de la vue
144         require (\CoursPHP\Config\Config::getVues()["afficheAdresse"]);
145     }else{ // Appel de la vue d'erreur
146         if (!empty($modele->getError()['persistance'])){
147             // Erreur d'accès à la base de donnée
148             require (\CoursPHP\Config\Config::getVuesErreur()["default"]);
149         }else{
150             // Erreur de saisie (attribut incorrect)
151             require (\CoursPHP\Config\Config::getVuesErreur()["saisieAdresseCreate"
152                 ]);
153         }
154     }
155 }
156
157 /** @brief Implemente l'action "delete" : supprime une instance via son ID
158 */
159 private function actionDelete(){
160     // ID de l'instance à supprimer
161     $id = filter_var($_REQUEST['id'], FILTER_SANITIZE_STRING);

```



```

160 // Construction du modèle et implémentation de la persistance :
161 $modele = \CoursPHP\Modele\ModelAdresse::deleteAdresse($id);
162 // test d'erreur :
163 if ($modele->getError() === false){ // Appel de la vue
164     require (\CoursPHP\Config\Config::getVues()["afficheAdresse"]);
165 }else{ // Appel de la vue d'erreur
166     require (\CoursPHP\Config\Config::getVuesErreur()["default"]);
167 }
168 }
169 }
170 ?>

```

Comme expliqué au chapitre 5, si les données issues du tableau `$_REQUEST` sont utilisées, elles doivent être systématiquement filtrées (validées ou nettoyées, voir partie 5.6).

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 12.2 ci-dessous. Le contrôleur appelle ensuite la vue, qui sera éventuellement une vue d'erreur en cas de tableau d'erreurs non vide.

12.2 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 renvoie 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*.

exemples/mvcLatex/exPoly/ex05_model.php

```

1 <?php
2 namespace CoursPHP\Modele;
3 /**
4  * @brief classe de base pour toutes les classes contenant des modèles.
5  * Cette classe vise seulement à factoriser le code concernant les données
6  * d'erreurs (tableau associatif dont les valeurs sont des messages d'erreur).
7  */
8 class Model{
9     /**
10     * @brief Dictionnaire d'erreurs (couples type => message)
11     */
12     protected $dataError;
13
14     /**
15     * @brief return false en l'absence d'erreurs, la collection d'erreurs sinon
16     * @return un tableau associatif dont les valeurs sont des messages d'erreur.
17     */
18     public function getError(){
19         if (empty($this->dataError)){

```

```

20     return false;
21 }
22     return $this->dataError;
23 }
24
25 /*
26  * @brief Constructeur
27  * @param un tableau associatif dont les valeurs sont des messages d'erreur.
28  * (par exemple un tableau vide, au début d'un traitement)
29  */
30 public function __construct($dataError){
31     $this->dataError = $dataError;
32 }
33 }
34 ?>

```

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 persistance.

exemples/mvcLatex/exPoly/ex06_modelAdresse.php

```

1 <?php
2 namespace CoursPHP\Modele;
3 /**
4  * @brief Classe Modèle pour stocker une Adresse
5  * Construit un modèle de données pour les vues affichant une unique adresse.
6  * Les données peuvent venir d'un formulaire ou d'un accès à la BD.
7  */
8 class ModelAdresse extends Model
9 {
10     /** Instance d'adresse, données métier du modèle */
11     private $adresse;
12
13     /** Titre principal de la vue */
14     private $title;
15
16     /** Donne accès à la donnée d'adresse */
17     public function getData()
18     {
19         return $this->adresse;
20     }
21
22     /** Donne accès au titre à afficher */
23     public function getTitle()
24     {
25         return $this->title;
26     }
27
28     /** @brief Retourne un modèle avec une adresse par défaut
29     * (par exemple pour créer un formulaire vide)
30     */

```

```
31 public static function getModelDefaultAdresse(){
32     $model = new self(array());
33     // Appel de la couche d'accès aux données :
34     $model->adresse = \CoursPHP\Metier\Adresse::getModelDefaultAdresse();
35     $model->title = "Saisie d'une adresse";
36     return $model;
37 }
38
39 /** @brief Retourne un modèle avec une adresse à partir de son ID
40  * par accès à la base de données.
41  */
42 public static function getModelAdresse($id){
43     $model = new self(array());
44     // Appel de la couche d'accès aux données :
45     $model->adresse = \CoursPHP\Persistence\AdresseGateway::getModelAdresse($id);
46     $model->dataError = $model->dataError;
47     $model->title = "Affichage d'une adresse";
48     return $model;
49 }
50
51 /** @brief Modifie une adresse dans la base de données.
52  */
53 public static function getModelAdresseUpdate($id, $numeroRue, $rue,
54     $complementAddr, $codePostal, $ville, $pays){
55     $model = new self(array());
56
57     // Appel de la couche d'accès aux données :
58     $model->adresse = \CoursPHP\Persistence\AdresseGateway::updateAdresse(
59         $model->dataError, $id, $numeroRue, $rue,
60         $complementAddr, $codePostal, $ville, $pays);
61     $model->title = "L'adresse a été mise à jour";
62     return $model;
63 }
64
65 /** @brief Insère une adresse en créant un nouvel ID dans la BD
66  */
67 public static function getModelAdresseCreate($numeroRue, $rue,
68     $complementAddr, $codePostal, $ville, $pays){
69     $model = new self(array());
70
71     // Appel de la couche d'accès aux données :
72     $model->adresse = \CoursPHP\Persistence\AdresseGateway::createAdresse(
73         $model->dataError, $numeroRue, $rue,
74         $complementAddr, $codePostal, $ville, $pays);
75     $model->title = "L'adresse a été insérée";
76     return $model;
77 }
78
79 /** @brief Supprime une adresse dans la BD et retourne l'adresse.
80  */
81 public static function deleteAdresse($id){
82     $model = new self(array());
83     // Appel de la couche d'accès aux données :
84     $model->adresse = \CoursPHP\Persistence\AdresseGateway::deleteAdresse(
85         $model->dataError, $id);
86     $model->title = "Adresse supprimée";
```

```

87     return $model;
88     }
89 }
90 ?>

```

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 une collection. 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 persistance.

exemples/mvcLatex/exPoly/ex07_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 {
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     */
18     private function __construct(){
19         $this->collectionAdresse = array();
20         $this->dataError = array();
21     }
22
23     /** brief Retourne un modèle avec la collection de toutes les adresses
24     * par accès à la base de données.
25     */
26     public static function getModelAdresseAll(){
27         $model = new self(array());
28         // Appel de la couche d'accès aux données :
29         $model->collectionAdresse = \CoursPHP\Persistence\AdresseGateway::
30             getAddressAll($model->dataError);
31         return $model;
32     }
33 }
34 ?>

```



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.

12.3 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 :

exemples/mvcLatex/exPoly/ex08_vueAfficheAdresse.php

```

1 <?=\CoursPHP\Vue\VueHtmlUtils : :enTeteHTML5( 'Bienvenue sur notre site ',
2   'UTF-8', \CoursPHP\Config\Config : :getStyleSheetsURL() [ 'default' ]) ?>
3
4 <h1>= $modele-&gt;getTitle() ?&gt;&lt;/h1&gt;
5
6 &lt;?=\CoursPHP\Vue\AdresseView : :getHtmlDevelopped( $modele-&gt;getData() ) ?&gt;
7 &lt;p&gt;
8   &lt;a href="&lt;?=\CoursPHP\Config\Config : :getRootURI() ?&gt;"&gt;Revenir à l'accueil &lt;/a&gt;
9 &lt;/p&gt;
10 &lt;?=\CoursPHP\Vue\VueHtmlUtils : :finFichierHtml5() ;?&gt;
</pre

```

Voyons maintenant la vue qui affiche toutes les adresses :

exemples/mvcLatex/exPoly/ex09_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&id=\". $adresse->getId(). \"'>supprimer </a>
12         </td>";
13      echo "<td><a href='\"?action=edit&id=\". $adresse->getId(). \"'>modifier </a></td>";
14      echo "<td>\". \CoursPHP\Vue\AdresseView : :getHtmlCompact( $adresse). \"</td>";
15      echo "<tr>";
16   }
17   echo "</tbody></table>";
18 <?=\CoursPHP\Vue\VueHtmlUtils : :finFichierHtml5() ;?>

```

Voyons enfin, par exemple, la vue d'erreur concernant la saisie incorrecte d'une adresse.

exemples/mvcLatex/exPoly/ex10_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*



FIGURE 13.1 : *Storyboards* : Vues accessible avec le rôle de visiteur

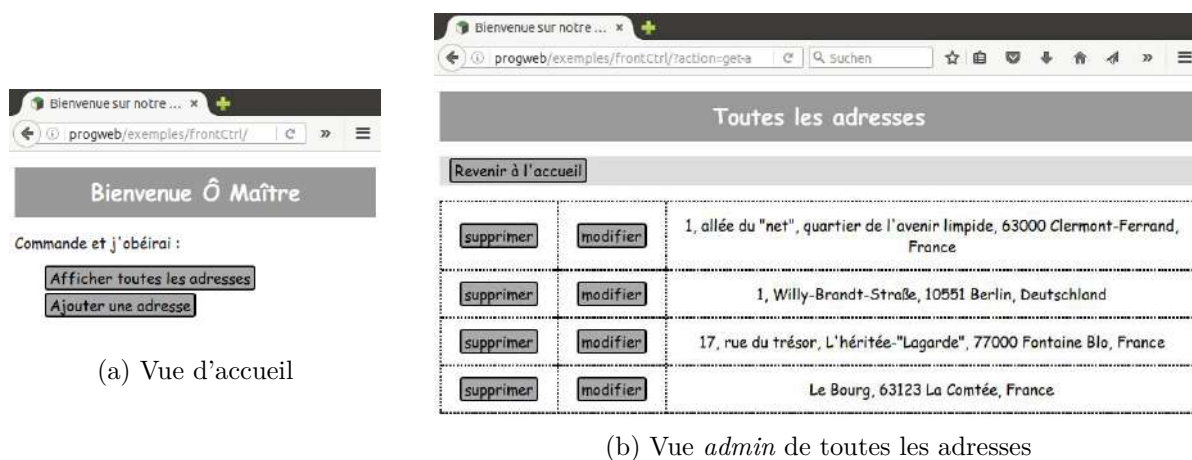


FIGURE 13.2 : *Storyboards* : Vue accessibles avec le rôle d'administrateur.

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 un type d'utilisateurs générique *User*, avec deux spécialisations : *Visitor* et *Admin*.

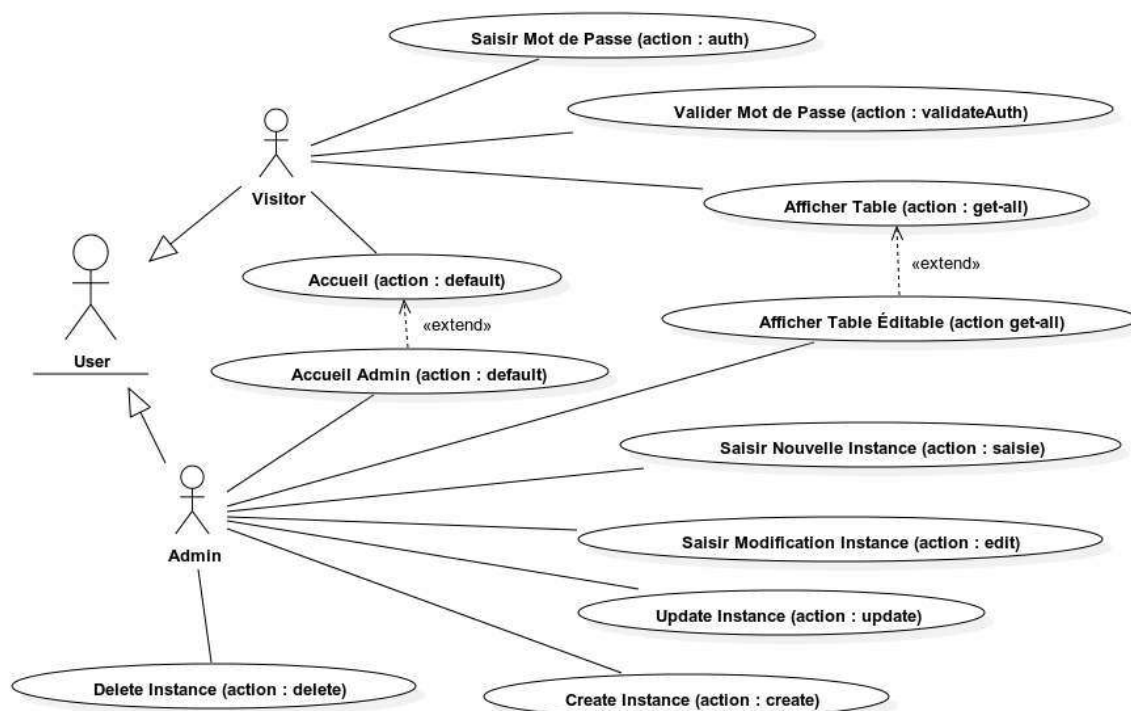


FIGURE 13.3 : *Use Case Diagram* : Les actions possibles pour les deux types d'utilisateurs

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 cr er une instance du contr leur adapt  pour l'action avec le r le de l'utilisateur.

exemples/frontCtrl/ex01_index.php

```
1 <?php
2 // R pertoire 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 enl ve 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'arr te au dernier slash (pour enlever la basename du script)
11 $longueurRootURI = strrpos($scriptWithoutExtention, '/');
12 // 4) On prend le d but de l'URL en coupant   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 // Cr ation de l'instance du contr leur (voir Controleur.php)
20 $ctrl = new \CoursPHP\Controleur\ControleurFront();
21 ?>
```

exemples/frontCtrl/ex02_controleurFront.php

```
1 <?php
2 namespace CoursPHP\Controleur;
3 /**
4  * @brief Le ControleurFront identifie l'action et le r le de l'utilisateur
5  * Dans le cas o  l'utilisateur a des droits insuffisants pour l'action,
6  * le ControleurFront affiche une vue d'authentification ou un vue d'erreur.
7  * Sinon, ControleurFront instancie le contr leur adapt  pour les r le et action
8  * Il g re aussi les exceptions et appelle le cas  ch ant une vue d'erreur.
9  */
10 class ControleurFront {
11     /**
12      * @brief C'est dans le constructeur que le contr leur fait son travail.
13      */
14     function __construct() {
15         // R cup ration d'une  ventuelle exception, d'o  qu'elle vienne.
16         try{
17             // R cup ration de l'action
18             $action = isset($_REQUEST['action']) ? $_REQUEST['action'] : "";
19
20             // L'utilisateur est-il identifi  ? Si oui, quel est son r le ?
21             $modele = \CoursPHP\Auth\Authentication::restoreSession();
22             $role = ($modele->getError() == false) ? $modele->getRole() : "";
23
24             // On distingue des cas d'utilisation, suivant l'action et le r le
25             switch($action){
26
27                 // 1) Actions accessibles uniquement aux visiteurs (r le par d faut)
28                 case "auth": // Vue de saisie du login/password
```

```

29     case "validateAuth": // Validation du login/password
30         $publicCtrl = new ControleurVisitor($action);
31         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()["authentication"]);
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 Model(array('exception' => $e->getMessage()));
60     require (\CoursPHP\Config\Config::getVuesErreur()["default"]);
61 }
62 }
63 }
64 ?>

```

Voici le contrôleur spécialisé pour les actions accessibles à un visiteur non authentifié.

exemples/frontCtrl/ex03_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 constructeur 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     $modele = new \CoursPHP\Modele\Model(array());
39     require (\CoursPHP\Config\Config::getVues()["authentication"]);
40 }
41
42 /**
43  * @brief Implemente l'action "validateAuth"
44  * Validation du login/password et création de session.
45  */
46 private function actionValidateAuth(){
47     ValidationRequest::validationLogin($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()["authentication"]);
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['id']) ? $_REQUEST['id'] : "";
63     $id = filter_var($rawId, FILTER_SANITIZE_STRING);
64     $modele = \CoursPHP\Modele\ModelAdresse::getModelAdresse($id);
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.

exemples/frontCtrl/ex04_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 constructeur 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['id']) ? $_REQUEST['id'] : "";
49     $id = filter_var($rawId, FILTER_SANITIZE_STRING);
50     $modele = \CoursPHP\Model\ModelAdresse::getModelAdresse($id);
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     $modele = \CoursPHP\Model\ModelCollectionAdresse::getModelAdresseAll();
62     if ($modele->getError() === false){
63         require (\CoursPHP\Config\Config::getVues()["afficheCollectionAdresseAdmin
64             "]);
65     }else{
66         require (\CoursPHP\Config\Config::getVuesErreur()["default"]);
67     }
68 }
69
70 /** @brief Implemente l'action "saisie" : Affiche un formulaire vierge
71 */
72 private function actionKeyIn(){
73     $modele = \CoursPHP\Model\ModelAdresse::getModelDefaultAdresse();
74     require (\CoursPHP\Config\Config::getVues()["saisieAdresseCreate"]);
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['id']) ? $_REQUEST['id'] : "";
82     $id = filter_var($_REQUEST['id'], FILTER_SANITIZE_STRING);
83     $modele = \CoursPHP\Model\ModelAdresse::getModelAdresse($id);
84     if ($modele->getError() === false){
85         require (\CoursPHP\Config\Config::getVues()["saisieAdresseUpdate"]);
86     }else{
87         require (\CoursPHP\Config\Config::getVuesErreur()["default"]);
88     }
89 }
90
91 /** @brief Implemente l'action "update" : Met à jour une instance dans la BD
92 */
93 private function actionUpdate(){
94     // valider ou nettoyer les inputs (par exemple : filter_var)
95     ValidationRequest::validationAdresse($id, $numeroRue, $rue, $complementAddr,
```

```

95         $codePostal, $ville, $pays);
96     $modele = \CoursPHP\Modele\ModelAdresse::getModelAdresseUpdate($id,
97         $numeroRue, $rue, $complementAddr, $codePostal, $ville, $pays);
98     if ($modele->getError() === false){
99         require (\CoursPHP\Config\Config::getVues()["afficheAdresse"]);
100    }else{
101        if (!empty($modele->getError()['persistence'])){
102            // Erreur d'accès à la base de donnée
103            require (\CoursPHP\Config\Config::getVuesErreur()["default"]);
104        }else{
105            // Erreur de saisie
106            require (\CoursPHP\Config\Config
107                :getVuesErreur()["saisieAdresseUpdate"]);
108        }
109    }
110 }
111
112 /** @brief Implemente l'action "create" : Crée une instance dans la BD
113  */
114 private function actionCreate(){
115     // valider ou nettoyer les inputs (par exemple : filter_var)
116     ValidationRequest::validationAdresse($id, $numeroRue, $rue, $complementAddr,
117         $codePostal, $ville, $pays);
118     $modele = \CoursPHP\Modele\ModelAdresse::getModelAdresseCreate($numeroRue,
119         $rue, $complementAddr, $codePostal, $ville, $pays);
120     if ($modele->getError() === false){
121         require (\CoursPHP\Config\Config::getVues()["afficheAdresse"]);
122     }else{
123         if (!empty($modele->getError()['persistence'])){
124             // Erreur d'accès à la base de donnée
125             require (\CoursPHP\Config\Config::getVuesErreur()["default"]);
126         }else{
127             // Erreur de saisie
128             require (\CoursPHP\Config\Config
129                 :getVuesErreur()["saisieAdresseCreate"]);
130         }
131     }
132 }
133
134 /** @brief Implemente l'action "delete" : Supprime une instance via son ID
135  */
136 private function actionDelete(){
137     // ID de l'instance à supprimer
138     $id = filter_var($_REQUEST['id'], FILTER_SANITIZE_STRING);
139     $modele = \CoursPHP\Modele\ModelAdresse::deleteAdresse($id);
140     if ($modele->getError() === false){
141         require (\CoursPHP\Config\Config::getVues()["afficheAdresse"]);
142     }else{
143         require (\CoursPHP\Config\Config::getVuesErreur()["default"]);
144     }
145 }
146 }
147 ?>

```

13.4 Gestion de l'Authentification

La vue d'authentification, ainsi que les utilitaires de g n ration d'*HTML* correspondants sont exactement les m mes que dans la partie 7.8 et les autres  l ments (sessions, *cookies*) en sont fortement inspir s. La plus grande diff rence par rapport   la partie 7.8 est l'impl mentation effective de la v rification d'existence du couple *login/password* dans la base de donn es, utilisant une *Gateway* d'utilisateur sur le mod le de la *DAL* (partie 9.3).

13.4.1 Mod le et *Gateway* de la table *User*

Nous consid rons dans notre mod lisation que toutes les classes concernant l'authentification des utilisateurs (y compris les mod les et *gateway*) font partie du *package Auth*. Nous verrons plus loin dans ce chapitre comment articuler cela avec les donn es m tier concernant l'utilisateur (donn es personnelles comme le nom, l'adresse, le num ro de t l phone, etc.).

exemples/frontCtrl/ex07_modelUser.php

```
1 <?php
2 namespace CoursPHP\Auth;
3 /**
4  * @brief Classe Mod le pour les donn es de l'utilisateur
5  * e-mail (qui sert ici de login), r le (visitor, admin, etc.)
6  * Les donn es peuvent venir d'une session ou d'un acc s   la BD.
7  */
8 class ModelUser extends \CoursPHP\Modele\Model
9 {
10     /** adresse e-mail de l'utilisateur */
11     private $email;
12     /** role de l'utilisateur */
13     private $role;
14
15     public function __construct($dataError){
16         parent::__construct($dataError);
17     }
18
19     /** Permet d'obtenir l'adresse e-mail (login) */
20     public function getEmail(){
21         return $this->email;
22     }
23
24     /** Permet d'obtenir le r le (et donc les droits) */
25     public function getRole(){
26         return $this->role;
27     }
28
29     /** @brief Remplie les donn es de l'utilisateur   partir
30     * du login/password par acc s   la BD (UserGateway)
31     */
32     public static function getModelUser($email, $hashedPassword){
33         $model = new self(array());
34         // Appel de la couche d'acc s aux donn es :
35         $model->role = UserGateway::getRoleByPassword($model->dataError,
36                                                     $email, $hashedPassword);
37         if ($model->role !== false){
38             $model->email = $email;
```

```

39     }else{
40         $model->dataError[ 'login ' ] = "Login ou mot de passe incorrect.";
41     }
42     return $model;
43 }
44
45 /** @brief Remplie des données de l'utilisateur à partir de la session
46 */
47 public static function getModelUserFromSession($email, $role){
48     $model = new self(array());
49     $model->role = $role;
50     $model->email = $email;
51     return $model;
52 }
53 }
54 ?>

```

exemples/frontCtrl/ex08_userGateway.php

```

1 <?php
2 namespace CoursPHP\Auth;
3
4 class UserGateway{
5     /**
6      * Vérifie que le couple login/password existe dans la table User
7      * @return le rôle de l'utilisateur si login/password valide, une erreur sinon
8      */
9     public static function getRoleByPassword(&$dataError, $email,
10                                             $hashedPassword){
11         // Exécution de la requête via la classe de connexion (singleton)
12         // Les exceptions éventuelles, personnalisées, sont gérées par le Contrôleur
13         $queryResults = \CoursPHP\Persistence\DataBaseManager
14             ::getInstance()->prepareAndExecuteQuery(
15                 'SELECT * FROM User WHERE email = ?',
16                 array($email)
17             );
18         // Si la requête a fonctionné
19         if ($queryResults !== false){
20             // Si un utilisateur avec cet email existe
21             if (count($queryResults) == 1){
22                 $row = $queryResults[0];
23             }
24             // Si l'email n'existe pas en BD ou le mot de passe ne correspond pas
25             if (count($queryResults) != 1 || $row[ 'password ' ] != $hashedPassword){
26                 $dataError[ 'login ' ] = "Adresse e-mail ou mot de passe incorrect";
27                 return "";
28             }
29             return $row[ 'role ' ];
30         }else{
31             $dataError[ 'login ' ] = "Impossible d'accéder à la table des utilisateurs";
32             return "";
33         }
34     }
35 }
36 ?>

```


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 7.8.

exemples/frontCtrl/ex06_authentication.php

```
1 <?php
2 namespace CoursPHP\Auth;
3 /**
4  * Permet d'initier une session après saisie du login/password.
5  * Permet aussi de restaurer la session d'un utilisateur déjà authentifié.
6  */
7 class Authentication {
8     /**
9      * @brief Test du login/password dans la table User et création d'une session
10     * @return Un modèle avec les données de l'utilisateur pour gestion des rôles
11     * Le modèle contient un tableau d'erreur non vide si l'identification échoue
12     */
13     public static function checkAndInitiateSession($login, $password, $dataError){
14         // On vérifie que le mot de passe (après hashage SHA512)
15         // est bien celui en base de donnée.
16         if (!empty($dataError)){
17             return new \CoursPHP\Modele\Model($dataError);
18         }
19         $userModel = ModelUser::getModelUser($login, hash("sha512", $password));
20         if ($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 expiré, "
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 démarrage 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 régénère 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 = $_SESSION['role'];
68 // On détruit l'ancienne session
69 session_destroy();
70 // On recrée une session :
71 SessionUtils::createSession($backupSessionEmail, $backupSessionRole);
72 // Flush des Données de Session, (sauvegarde immédiate sur le disque)
73 session_write_close();
74 }
75 return $userModel;
76 }
77 }
78 ?>

```

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 peut 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 Controller*, qui, en fonction de l'action et du rôle de



FIGURE 13.4 : Vue général affichant une collection de personnes.

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.
- `ControleurAdminPersonne.php` pour la gestion des actions concernant les personnes dans le rôle d'administrateur.

Voici le code du *Front Controller* :

```
exemples/metierFrontArchi/ex01_ControleurFront.php
```

```

1 <?php
2 namespace CoursPHP\Controleur;
3 /**
4  * @brief Le ControleurFront 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'authentification ou une vue d'erreur.
7  * Sinon, ControleurFront instancie le contrôleur adapté pour le rôle et l'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 constructeur que le contrôleur fait son travail.
13      */
14     function __construct() {
15         try{
16             // Récupération de l'action
17             $action = isset($_REQUEST['action']) ? $_REQUEST['action'] : "";
18
19             // L'utilisateur est-il identifié ? Si oui, quel est son rôle ?
20             $modele = \CoursPHP\Auth\Authentication::restoreSession();
21             $role = ($modele->getError() == false) ? $modele->getRole() : "";
22
23             // On distingue des cas d'utilisation, suivant l'action
24             switch($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": // Création d'une nouvelle Adresse dans la BD
37                 case "adresse-delete": // Suppression 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": // Création d'une nouvelle Personne dans la BD
49                 case "personne-delete": // Suppression d'une Personne à partir de son
50                     ID
51                     if ($role == "admin"){
52                         $adminCtrl = new ControleurAdminPersonne($action);
53                     }else{
54                         require (\CoursPHP\Config\Config::getVues()["authentification"]);
55                     }
56                     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 }
83 }catch (\Exception $e){ // Page d'erreur par défaut
84     $modele = new Model(array('exception' => $e->getMessage()));
85     require (\CoursPHP\Config\Config::getVuesErreur()["default"]);
86 }
87 }
88 }
89 ?>

```

Voici à titre d'exemple, le code du *ControllerAuth* gérant les actions associées à l'authentification :

exemples/metierFrontArchi/ex02_ControlleurAuth.php

```

1 <?php
2 namespace CoursPHP\Controleur;
3 /**
4  * @brief ControleurAuth identifie l'action et appelle la méthode
5  * pour construire le modèle pour l'action liée à l'authntification.
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 constructeur 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     $modele = new \CoursPHP\Modele\Model(array());
32     require (\CoursPHP\Config\Config::getVues()["authentication"]);
33 }
34
35 /**@brief Implemente l'action "validateAuth" : validation du login/password
36 *
37 */
38 private function actionValidateAuth(){
39     ValidationRequest::validationLogin($dataError, $email, $password);
40     $modele = \CoursPHP\Auth\Authentication::checkAndInitiateSession($email,
41         $password, $dataError);
42     if ($modele->getError() == false){
43         require (\CoursPHP\Config\Config::getVues()["defaultAdmin"]);
44     }else{
45         require (\CoursPHP\Config\Config::getVues()["authentication"]);
46     }
47 }
48 ?>

```

Voici, toujours à titre d'exemple, le code du *ControllerAdminPersonne* gérant les actions associées aux personnes :

exemples/metierFrontArchi/ex06_ControleurAdminPersonne.php

```

1 <?php
2 namespace CoursPHP\Controleur;
3 /**
4  * @brief ControleurAdminPersonne identifie l'action et appelle la méthode
5  * pour construire le modèle correspondant à l'action avec le rôle "admin".
6  * 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     /**
12     * @brief C'est dans le constructeur 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 "personne-get": // Affichage d'une Personne à partir de son ID
18                 $this->actionGet();
19                 break;

```

```
20     case "personne-get-all": // Affichage de toutes les Personne's
21         $this->actionGetAll();
22         break;
23     case "personne-saisie": // Saisie d'une nouvelle Adresse
24         $this->actionSaisie();
25         break;
26     case "personne-edit": // Saisie des modifications d'une Adresse
27         $this->actionEdit();
28         break;
29     case "personne-update": // Met à jour une Adresse dans la BD
30         $this->actionUpdate();
31         break;
32     case "personne-create": // Cration d'une nouvelle Adresse dans la BD
33         $this->actionCreate();
34         break;
35     case "personne-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['id']) ? $_REQUEST['id'] : "";
49     $id = filter_var($rawId, FILTER_SANITIZE_STRING);
50     $modele = \CoursPHP\Modele\ModelPersonne::getModelPersonne($id);
51     if ($modele->getError() === false){
52         require (\CoursPHP\Config\Config::getVues()["affichePersonneAdmin"]);
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     $modele = \CoursPHP\Modele\ModelCollectionPersonne::getModelPersonneAll();
62     if ($modele->getError() === false){
63         require (\CoursPHP\Config\Config::getVues()["
64             afficheCollectionPersonneAdmin"]);
65     }else{
66         require (\CoursPHP\Config\Config::getVuesErreur()["default"]);
67     }
68 }
69
70 /** @brief Implemente l'action "saisie" : affiche un formulaire vierge
71  */
72 private function actionSaisie(){
73     $modele = \CoursPHP\Modele\ModelPersonne::getModelDefaultPersonne();
74     require (\CoursPHP\Config\Config::getVues()["saisiePersonneCreate"]);
75 }
```

```

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['id']) ? $_REQUEST['id'] : "";
82     $id = filter_var($_REQUEST['id'], FILTER_SANITIZE_STRING);
83     $modele = \CoursPHP\Modele\ModelPersonne::getModelPersonne($id);
84     if ($modele->getError() === false){
85         require (\CoursPHP\Config\Config::getVues()["saisiePersonneUpdate"]);
86     }else{
87         require (\CoursPHP\Config\Config::getVuesErreur()["default"]);
88     }
89 }
90
91 /** @brief Implemente l'action "update" : met à jour une instance dans la BD
92 */
93 private function actionUpdate(){
94     // valider ou nettoyer les inputs (par exemple : filter_var)
95     ValidationRequest::validationPersonne($id, $nom);
96     $modele = \CoursPHP\Modele\ModelPersonne::getModelPersonneUpdate($id, $nom);
97     if ($modele->getError() === false){
98         require (\CoursPHP\Config\Config::getVues()["affichePersonne"]);
99     }else{
100         if (!empty($modele->getError()['persistance'])){
101             // Erreur d'accès à la base de donnée
102             require (\CoursPHP\Config\Config::getVuesErreur()["default"]);
103         }else{
104             // Erreur de saisie
105             require (\CoursPHP\Config\Config
106                     :getVuesErreur()["saisiePersonneUpdate"]);
107         }
108     }
109 }
110
111 /** @brief Implemente l'action "create" : crée une instance dans la BD
112 */
113 private function actionCreate(){
114     // valider ou nettoyer les inputs (par exemple : filter_var)
115     ValidationRequest::validationPersonne($id, $nom);
116     $modele = \CoursPHP\Modele\ModelPersonne::getModelPersonneCreate($nom);
117     if ($modele->getError() === false){
118         require (\CoursPHP\Config\Config::getVues()["affichePersonne"]);
119     }else{
120         if (!empty($modele->getError()['persistance'])){
121             // Erreur d'accès à la base de donnée
122             require (\CoursPHP\Config\Config::getVuesErreur()["default"]);
123         }else{
124             // Erreur de saisie
125             require (\CoursPHP\Config\Config
126                     :getVuesErreur()["saisiePersonneCreate"]);
127         }
128     }
129 }
130

```



```

131  /** @brief Implemente l'action "delete" : supprime une instance via son ID
132  */
133  private function actionDelete(){
134      // ID de l'instance à supprimer
135      $id = filter_var($_REQUEST['id'], FILTER_SANITIZE_STRING);
136      $modele = \CoursPHP\Model\ModelPersonne::deletePersonne($id);
137      if ($modele->getError() === false){
138          require (\CoursPHP\Config\Config::getVues()["affichePersonne"]);
139      }else{
140          require (\CoursPHP\Config\Config::getVuesErreur()["default"]);
141      }
142  }
143 }
144 ?>

```

Voici enfin la vue affichant une collection de personnes avec les adresses agrégées (voir figure 13.4) :

exemples/metierFrontArchi/Vue/vues/vueCollectionPersonneAdmin.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">Ajouter une personne </a>
7
8 <?php
9     foreach ($modele->getData() as $personne){
10         echo "<div>"
11             . \CoursPHP\Vue\PersonneView::getHtmlDevelopped($personne, true);
12         echo "<a href='\"?action=personne-edit&id=\""
13             . $personne->getId()."\">modifier le nom</a>";
14         echo "<a href='\"?action=adresse-saisie&idPersonne=\""
15             . $personne->getId()."\"></a>Ajouter une adresse </a> ";
16         echo "<a href='\"?action=personne-delete&id=\""
17             . $personne->getId()."\"></a>Supprimer la personne </a> ";
18         echo "</div>";
19     }
20 ?>
21 <?=\CoursPHP\Vue\VueHtmlUtils :finFichierHtml5(); ?>

```