SIN6U5 : Développement web 2 (Prog. web coté serveur) Utilisation du contexte

Luigi Santocanale
LIF, Aix-Marseille Université

Transparents basés sur le cours de Bertrand Estellon

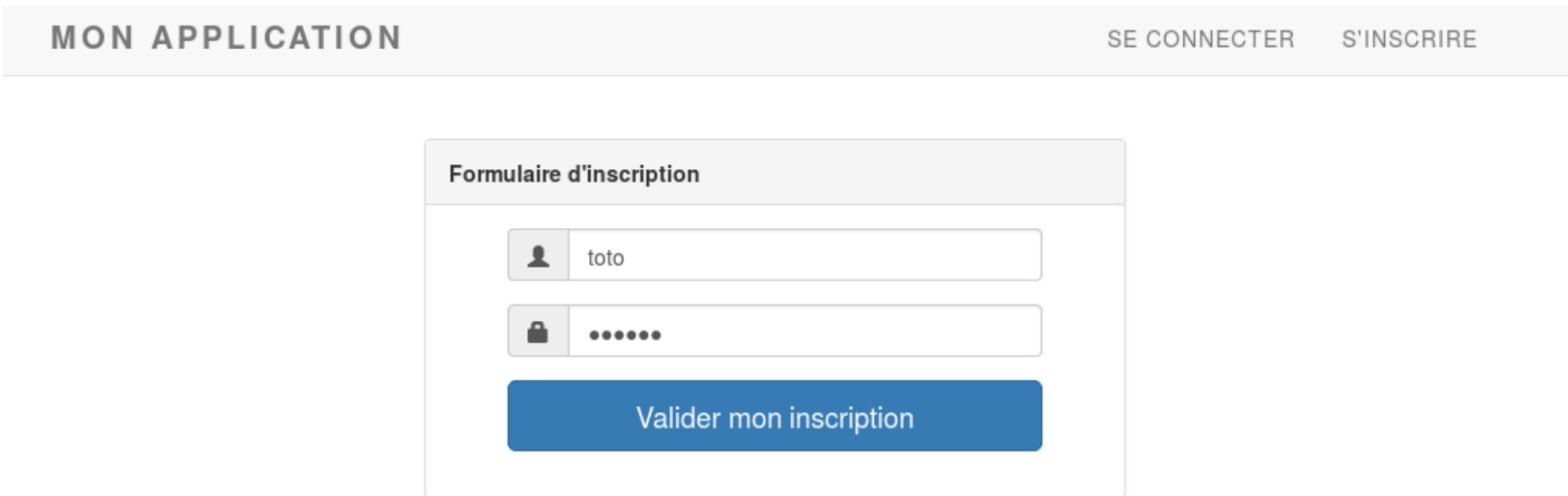
21 février 2018

Plan

Plan

Authentification d'un utilisateur

Une page pour s'inscrire :



Authentification d'un utilisateur

Une page pour s'authentifier :

Les besoins

- ✓ Organisation du projet
- ✓ Gestion des formulaires
- ✓ Utilisation d'une base de données

- ? Stockage des mots de passe
- ? Vérification des mots de passe
- ? Mémorisation de l'authentification

Les besoins

Les URL :

- `users/users_new` : formulaire de création d'un utilisateur
- `users/users_create` : création d'un utilisateur
- `sessions/sessions_new` : formulaire de connexion
- `sessions/sessions_create` : demande de connexion
- `sessions/sessions_destroy` : déconnexion

Deux contrôleurs :

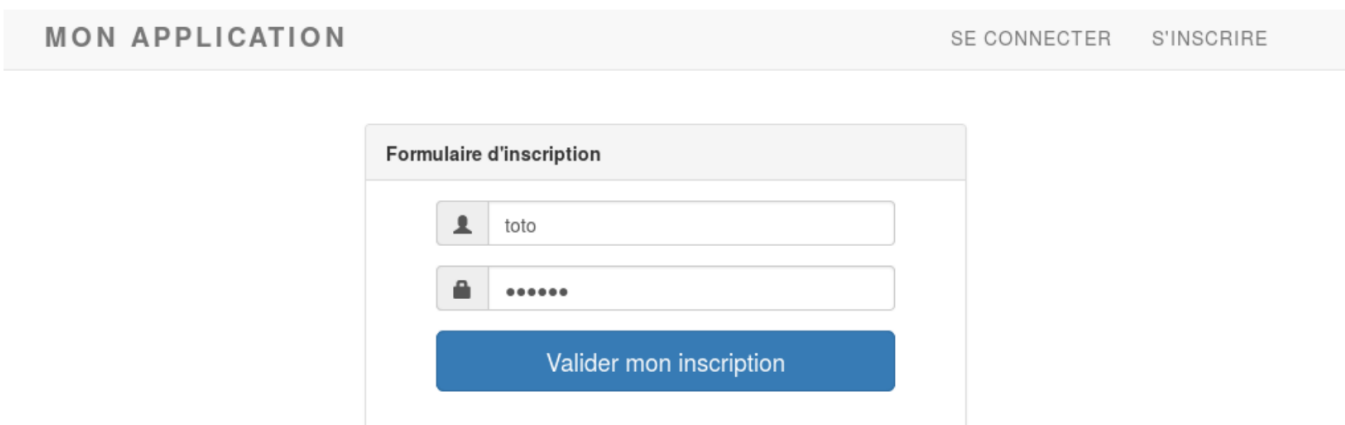
- `Users` : gestion des utilisateurs
- `Sessions` : gestion des sessions

Trois vues :

- `top_bar.php` : barre du haut
- `users_new` : formulaire d'inscription
- `sessions_new` : formulaire de connexion

Formulaire d'inscription

```
<!-- views/users_new.php  -->
<br />
<div class="loginmodal-container">
    <?php if (isset($error)): ?>
        <div class="error"><?= $error ?></div>
    <?php endif; ?>
    <form action="/index.php/users/users_create" method="post">
        <input class="form-control" name="username" type="text" placeholder="Nom utilisateur">
        <input class="form-control" name="password" type="password" placeholder="password">
        <input type="submit" value="Valider mon inscription">
    </form>
</div>
```



Contrôleur `User` : gestion de l'inscription

```php
<?php

// controllers/Users.php

class Users extends Controller {

    public function users_new() {
        $this->loader->load('users_new');
    }

    public function users_create() {
        try {
            $username = filter_input(INPUT_POST, 'username');
            $password = filter_input(INPUT_POST, 'password');
            $user = $this->users->create_user($username, $password);
            $this->sessions->login($user);
            $this->loader->redirect('/users/hello.php');
        } catch (Exception $e) {
            $data = ['error' => $e->getMessage()];
            $this->loader->load('users_new', $data);
        }
    }

}
```

Formulaire de connexion

```
<!-- views/sessions_new.html.php  -->
<br />
<div class="loginmodal-container">
    <?php if (isset($error)): ?>
        <div class="error"><?= $error ?></div>
    <?php endif; ?>

    <form action="/index.php/sessions/sessions_create" method="post">
        <input class="form-control" name="username" type="text"
 ↪   placeholder="Nom utilisateur">
        <input class="form-control" name="password" type="password"
 ↪   placeholder="password">
        <input type="submit" value="Se connecter">
    </form>
</div>
```

Contrôleur `Sessions` : gestion des connexions

```php
<?php
// controllers/Sessions.php

class Sessions extends Controller {

    public function sessions_new() {

    }

    public function sessions_create() {

    }

    public function sessions_destroy() {

    }
}
```

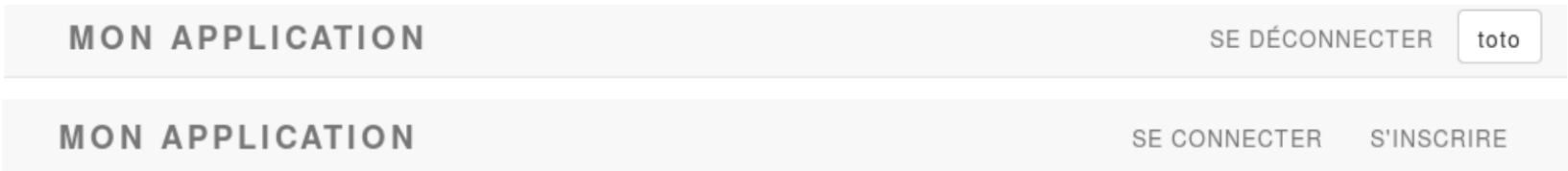
Contrôleur `Sessions` (connexions) II

```
    public function __construct() {
        session_start();
    }

    public function login($user) {
        $_SESSION['logged_user'] = $user;
    }


    public function user_is_logged() {
        return $this->logged_user() !== null;
    }
```

Barre du haut




```
<!-- views/top_bar.html.php  -->
<div class="navbar navbar-static-top">
    <div class="container">
        <div class="navbar-collapse collapse">
            <ul class="nav navbar-nav navbar-right"> <!-- code incomplet pour le style -->
                <?php if ($user_is_logged): ?>
                    <li class="active"><a href="/index.php/sessions/sessions_destroy">Se
  ↪    déconnecter</a></li>
                    <li><button><?= $logged_user->username ?></button></li>
                <?php else: ?>
                    <li><a href="/index.php/sessions/sessions_new">Se connecter</a></li>
                    <li><a href="/index.php/users/users_new">S'inscrire</a></li>
                <?php endif ?>
            </ul>
        </div>
    </div>
</div>
```

Deconnexion

```
public function sessions_destroy() {
    $this->sessions->logout();
    $this->loader->redirect('/index.php');
}
```

Les méthodes des modèles

Le modèle `Users_model` :

- `create_user($username, $password) : User`
- `user_from_id($id) : User`
- `user_from_username($username) : User`

La classe User (`lib/User.php`) :

- `password_is_valid($password) : bool`
- `$username`

Le modèle `Sessions_model` :

- `login($user)`
- `logout()`
- `$user_is_logged` et `$logged_user` dans les vues

Plan

Stockage des mots de passe

Problématique :

- Menace : vol des données de la table contenant les mots de passe
- Un mot de passe est souvent utilisé sur plusieurs sites
- Le vol de mot de passe en clair est une attaque sérieuse

Donc :

ne jamais conserver les mots de passe en clair ! ! !

Vols de mot de passe

Vol des mots de passe stockés sur un site Web.
Piratage de l'autorité qui stocke les mots de passe.

Quelques piratage historique :

- YouPorn, 2012, mots de passe en clair
- Yahoo, 2014, mots de passe chiffrés,
  - vol de données personnel,
  - vol des mots de passe chiffrés, leur et déchiffrage par table Rainbow.

Autres types de vols :

- attaques par « sniffing »
- attaques par « sniffing »sur protocole https.

Voir :
`http://assiste.com/Mots_de_passe_Formes_d_attaques.html`

Stockage des mots de passe

Une bonne fonction de hachage pour notre problème :

- va d'un ensemble possiblement infini vers un ensemble fini ;
- minimise si possible les collisions ;
- doit s'évaluer rapidement ;
- doit être difficilement inversible.

Les attaques possibles pour essayer d'inverser la fonction :

- attaque par brute force (tous les possibles combinaisons)
- attaque par dictionnaire (chercher seulement des combinasoins frequentes, par ex. les mots de la langue naturelle)
- attaque par « Rainbow table » .

Idée pour contrer les attaques par dictionnaire :
ajouter un "sel", c'est-à-dire, une partie aléatoire au mot de passe avant d'appliquer la fonction de hachage afin de rendre difficile l'inversion de la fonction.

Stockage des mots de passe

La fonctions de PHP pour “hacher” un mot de passe :

```php
$hash = password_hash($password, PASSWORD_DEFAULT);
```

La fonctions de PHP pour vérifier un mot de passe :

```php
$password_is_correct = password_verify($password, $hash);
```

Nous pouvons choisir l’algorithme de hachage :

```php
$hash = password_hash($password, PASSWORD_ARGON2I);
```

L’algorithme par défaut est bcrypt (`PASSWORD_BCRYPT`), mais dans le futur cela peut (et doit) changer.

Stockage des mots de passe

Exemple de mot de passe haché par la fonction de PHP :

```
<?=password_hash("zadiahdaajzd", PASSWORD_DEFAULT) ?>
```

Le résultat de l'exécution de la ligne précédente :

```
$2y$10$OCJuVTJ/XiwEMyGLvAc.ze2b0eq2oK7VQMdg01iQBPiBdQZB4qlQe
```

- L'algorithme utilisé est `2y` (bcrypt)
- le paramètres de l'algorithme (le cost) est `10`,
- le sel et le résultat du hachage sont stockés dans la chaîne.

Plan

Le modèle Sessions_model

```php
<?php
// models/Sessions_model.php
class Sessions_model extends Model {
    public function __construct() {

    }
    public function login($user) {

    }
    public function user_is_logged() {

    }
    public function logged_user() {

    }
    public function logout() {

    }
    public function inject_data($data) {

    }
}
```

Le modèle `Sessions_model` II

```
public function __construct() {
    session_start();
}

public function login($user) {
    $_SESSION['logged_user'] = $user;
}


public function user_is_logged() {
    return $this->logged_user() !== null;
}
```

Le modèle `Sessions_model` III

```
    public function logged_user() {
        if (session_id() === "")
            return null;
        if (!isset($_SESSION['logged_user']))
            return null;
        return $_SESSION['logged_user'];
    }

    public function logout() {
        session_destroy();
    }

    public function inject_data($data) {
        $data['logged_user'] = $this->logged_user();
        $data['user_is_logged'] = $this->user_is_logged();
        return $data;
    }
```

Les cookies

Caractéristiques d'un cookie :

- Donnée placée sur l'ordinateur du visiteur
- Les cookies servent à stocker de l'information associée aux visiteurs
- Le visiteur peut interdire ou supprimer les cookies
- Le visiteur peut modifier les informations contenues dans les cookies
- Chaque site peut écrire un nombre limité de cookies sur chaque client

Les cookies

La demande d'enregistrement d'un cookie chez le visiteur se fait dans le header de la réponse faite par le serveur au client :

Requête :
GET /toto.html HTTP/1.0
Host : example.com
Referer : http ://example2.com/
User-Agent : Mozilla/5.0 (X11 ; Linux x86_64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/45.0.2454.85 Safari/537.36

Réponse :
HTTP/1.1 200 OK
Date : Wed, 02 Sep 2015 12 :53 :36 GMT
Server : Apache
Set-Cookie : key=value
Content-Type : text/html
Content-Length : 59
Last-modified : Wed, 02 Sep 2015 10 :33 :36 GMT

Les cookies

Pour modifier le header et demander l'inscription d'un cookie :

```
setcookie('key', 'value');
```

...ou avec des informations supplémentaires :

```
setcookie('key',
  'value',
  time()+3600,
  '/path/',
  'www.example.com');
```

Les cookies

Les cookies associées à une URL sont envoyés à chaque requête du client :

Requête :
GET /toto.html HTTP/1.0
Host : example.com
Referer : http ://example2.com/
User-Agent : Mozilla/5.0 (X11 ; Linux
x86_64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/45.0.2454.85
Safari/537.36
Cookie :
key1=value1&key2=value2

Réponse :
HTTP/1.1 200 OK
Date : Wed, 02 Sep 2015
12 :53 :36 GMT
Server : Apache
Content-Type : text/html
Content-Length : 59
Last-modified : Wed, 02 Sep 2015
10 :33 :36 GMT ..

Les cookies

La lecture des cookies se fait via la variable `$_COOKIE` :

```
<?php echo $_COOKIE['key1'] ;
echo $_COOKIE['key2'] ;
```

Les cookies ne sont pas immédiatement accessibles après `setcookie` :

```
setcookie('key','value');
echo $_COOKIE['key']; /* pas défini ou valeur
 ↪  présente dans la requête du client. */
```

Les cookies sont des couples de chaînes de caractères et fonctionnent comme les données liées à `GET` et `POST`. Il est donc possible de stocker des tableaux en mettant des clés de la forme `key[]`.

Les cookies

Pour supprimer le contenu d'un cookie :

```
setcookie('key');
```

La suppression du cookie est effective au rechargement de la page.

Les sessions

Peut-on conserver le fait que l'utilisateur est identifié avec un cookie ?

```
<?= $_COOKIE['is_authenticated'] ?>
```

Non ! Le client peut modifier la valeur des cookies.

Peut-on conserver le mot de passe et l'identifiant de l'utilisateur dans un cookie et vérifier à chaque fois que le couple est correct ?
Non ! Un mot de passe ne doit jamais être stocké en clair.

Peut-on fournir au client un identifiant que seul lui et le serveur connaissent afin d'identifier de façon "certaine" le client ?
Oui ! Cet identifiant sera stocké dans les cookies du client.

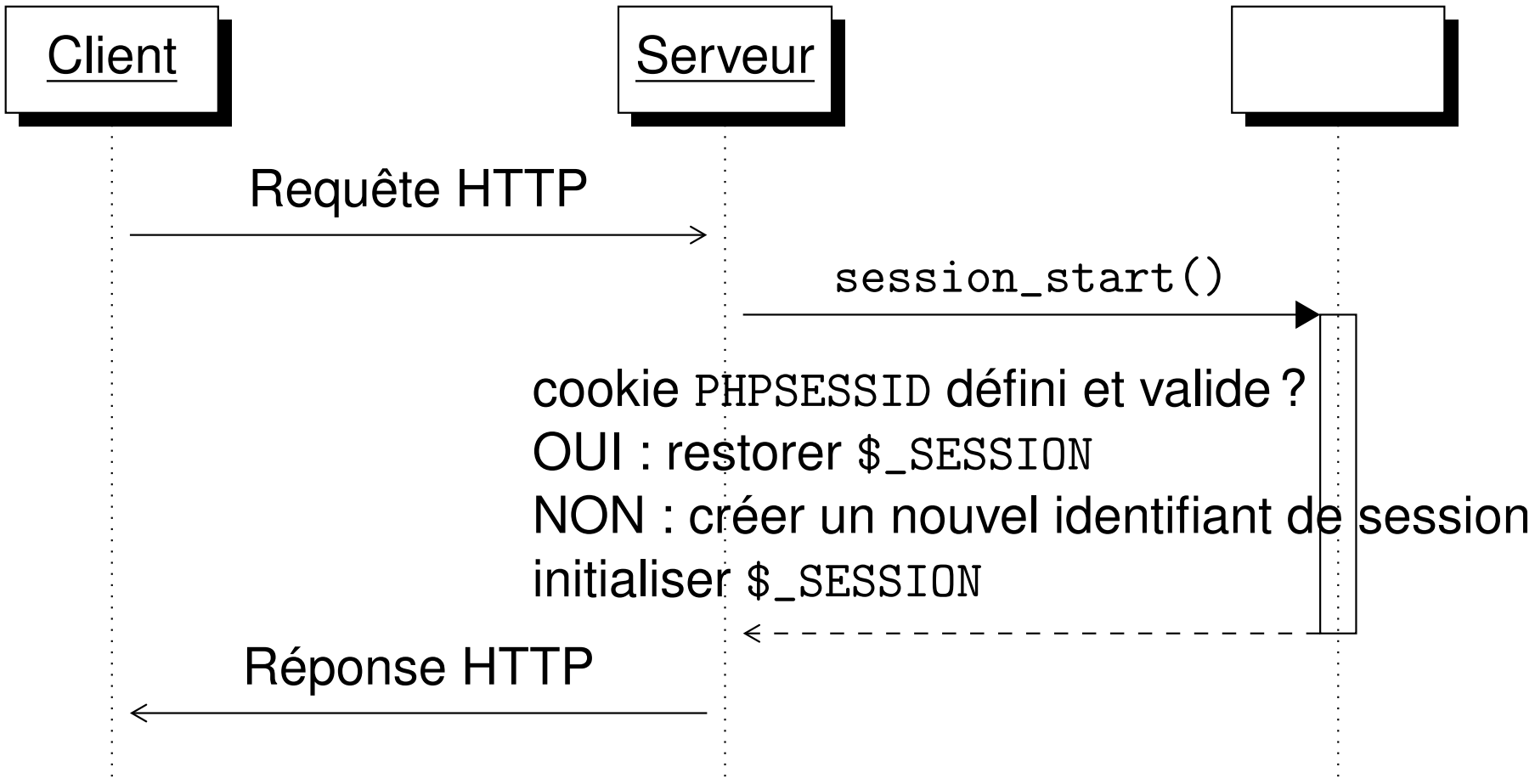
Les sessions PHP

Objectif : Conserver des informations d'un même client entre les pages sans qu'elles puissent être modifiées par le client.

Les étapes du mécanisme des sessions :

1. au début de chaque page, l'appel de `session_start()` crée une session ou restaure la session trouvée sur le serveur via un identifiant.
2. Au moment de la création de la session, chaque utilisateur se voit attribuer un identifiant. Il est transmit entre les pages via un cookie.
3. Le tableau `$_SESSION permet` de stocker des données de la session.
4. `session_destroy()` permet de supprimer la session.

Création d'une session PHP



Persistance des données REST

REST : Representational state transfer

- Par sa nature, le protocole HTTP est « stateless »
- Les cookies ont été introduits en 1994 avec Netscape 0.9
- Intégrés dans Internet Explorer 2 en 1995
- . . . afin de pouvoir créer une appli de e-commerce (gestion du panier . . .)

- La nature « stateless » n'est pas autant mauvaise :
  - application est plus simple à entretenir ;
  - indépendance entre client et serveur ;
  - absence de gestion d'état du client sur le serveur permet la répartition des requêtes sur plusieurs serveurs ;
  - etc.
- Avec d'autres (RESTful) frameworks la persistance des données est mise en ouvre seulement par le cookies.
- Les valeurs des cookies sont transmis cryptés.

Remarques pratiques

- La fonction `setcookie` modifie l'header de la réponse HTTP.
- Cette fonction doit être appelée avant de toute sortie.
- De même, pour la fonction `session_start`.

Ce code ne marchera pas :

```
Bonjour
<?php
if(isset($_SESSION['prenom'])){
    echo $_SESSION['prenom'];
}else{
    echo 'visiteur';
    session_start();
}
?>
```

PS : très mavais code, tout de même.

Remarque pratiques II

De même pour celui la :

```php
<?php
include 'file.php';
session_start();
?>
```

```php
<?php
// fichier.php

// Aucun affichage sur le tampon de sortie
// Même seulement des commentaires

?>
```

Un exemple : CSRF

CSRF : cross site request forging

- Alice obtient le formulaire pour créditer son compte depuis Bob,
- Elle transmet ce formulaire à Bob (en prétendant que c'est autre chose) pre-complété avec une somme important d'argent
- Bob poste ce formulaire avec se identifiant de session.
- Résultat : Alice est riche, Bob fait faillite.

Prévention :

- Ajout d'un token unique à tout formulaire,
- dont la validité est à vérifier au moment de la validation des données.
- L'unicité implémentée via les sessions.

CRSF II

```php
<?php
session_start();
if (empty($_SESSION['token'])) {
    $_SESSION['token'] = bin2hex(random_bytes(32));
}
$token = $_SESSION['token'];
//$token = '';
?>
```

```php
<!DOCTYPE html>
<html>
    <head>
    </head>
    <body>
        <form method="POST" action="/answerForm.php">
            <input type="text" name="money" placeholder="Combien
 ↪  d'argent ?" />
            <input type="hidden" name="token" value="<?=$token?>" />
            <input type="submit" value="Envoyer" />
        </form>
    </body>
</html>
```

CRSF III

```php
<?php
session_start();
$token=filter_input(INPUT_POST,'token');
if($token ===null or !hash_equals($_SESSION['token'], $token)) {
    // Log this as a warning and keep an eye on these attempts
    $error = 'Attaque CSRF ?' ;
           } else {
    $money = filter_input(INPUT_POST,'money');
}
?>
```

```php
<!DOCTYPE html>
<html>
    <head>
    </head>
    <body>
        <?php if(isset($error)): ?>
            Erreur : <?=$error?>
        <?php else: ?>
            Argent : <?=$money?>
        <?php endif; ?>
    </body>
</html>
```

Le modèle *Sessions_model*

```php
<?php
// models/Sessions_model.php
class Sessions_model extends Model {
    public function __construct() {

    }
    public function login($user) {

    }
    public function user_is_logged() {

    }
    public function logged_user() {

    }
    public function logout() {

    }
    public function inject_data($data) {

    }
}
```

Le modèle `Sessions_model` II

```
public function __construct() {
    session_start();
}

public function login($user) {
    $_SESSION['logged_user'] = $user;
}


public function user_is_logged() {
    return $this->logged_user() !== null;
}
```

Le modèle Sessions_model III

```
public function logged_user() {
    if (session_id() === "")
        return null;
    if (!isset($_SESSION['logged_user']))
        return null;
    return $_SESSION['logged_user'];
}

public function logout() {
    session_destroy();
}

public function inject_data($data) {
    $data['logged_user'] = $this->logged_user();
    $data['user_is_logged'] = $this->user_is_logged();
    return $data;
}
```

Plan

Les méthodes des modèles

Le modèle `Users_model` :

- `create_user($username, $password) : User`
- `user_from_id($id) : User`
- `user_from_username($username) : User`

La classe User :

- `password_is_valid($password) : bool`
- `$username`

Le modèle `Sessions_model` :

- `login($user)`
- `logout()`
- `$user_is_logged` et `$logged_user` dans les vues

Injection automatique de données dans les vues

Jusqu'à maintenant, seul les contrôleurs sont responsable de compléter les vues, en passant un tableau au loader :

```
$this->loader->load('mavue');
```

Comment un modèle peut transmettre de données aux vues :

- donnée utilisateur ;
- menu spécifique au rôle d'un utilisateur ;
- etc.

Injection automatique : `Model`

Ajout d'une méthode `inject_data` aux modèles :

```php
<?php

// core/Model.php

class Model {

    private static $static_db;
    protected $db;

    public function __construct() {

    }

    public static function init() {

    }

    public function inject_data($data) {

    }

}

Model::init();
```

Injection automatique : `Model`

```php
public function __construct() {
    $this->db = self::$static_db;
}

public static function init() {
    global $config;
    self::$static_db = new PDO('sqlite:' . $config['database']);
}


public function inject_data($data) {
    return $data;
}
```

Injection automatique : `Loader`

```php
<?php
// core/Loader.php
class Loader {
    private $models = [];
    public function view($view, $data = []) {

    }
    public function load($view = null, $data = []) {

    }
    public function site_url($url) {

    }
    public function define_helper() {

    }

    }
    public function add_model($model_name, $model) {

    }
    private function inject_model_data($data) {

    }
    public function redirect($url) {
```

Injection automatique : `Loader`

```
    public function add_model($model_name, $model) {
        $this->models[$model_name] = $model;
    }

    private function inject_model_data($data) {
        foreach ($this->models as $model) {
            $data = $model->inject_data($data);
        }
        return $data;
    }
```

Injection automatique : `Controller`

```php
<?php

// core/Controller.php

class Controller {

    protected $loader;

    public function __construct() {

    }

    private function init_models() {

    }

    private function init_model($model) {

    }

    protected function compute_error($form_data, $filters) {

    }

}
```

Injection automatique : `Controller`

```php
public function __construct() {
    $this->loader = new Loader();
    $this->init_models();
}

private function init_models() {
    global $config;
    foreach ($config['models'] as $model) {
        $this->init_model($model);
    }
}
```

Attention à la dernière ligne :

```php
private function init_model($model) {
    $class = $model . '_model';
    require "models/$class.php";
    $variable = strtolower($model);
    $this->$variable = new $class();
    $this->loader
            ->add_model($variable, $this->$variable);
}
```

Redéfinition de la méthode *inject_data* dans *Sessions_model*

```php
<?php
// models/Sessions_model.php
class Sessions_model extends Model {
    public function __construct() {

    }
    public function login($user) {

    }
    public function user_is_logged() {

    }
    public function logged_user() {

    }
    public function logout() {

    }
    public function inject_data($data) {

    }
}
```

Redéfinition de la méthode *inject_data* dans *Sessions_model*

```
public function inject_data($data) {
    $data['logged_user'] = $this->logged_user();
    $data['user_is_logged'] = $this->user_is_logged();
    return $data;
}
```

Le modèle `Users`

```php
<?php
// models/Users_model.php
class Users_model extends Model {
    const str_error_username_format = '...';
    const str_error_password_format = '...';
    private function check_username($username) {

    }
    private function check_password($password) {

    }
    private function check_format_with_regex($variable, $regex,
    ↪  $error_message) {

    }
    public function user_from_id($id) {

    }
    public function user_from_username($username) {

    }
    private function user_from_query($query, $array) {

    }
    public function create_user($username, $password) {
```

Le modèle `Users`

```
    private function check_username($username) {
        $this->check_format_with_regex($username, '/^[0-9a-zA-Z]{2,10}$/',
↪     self::str_error_username_format);
    }

    private function check_password($password) {
        $this->check_format_with_regex($password, '/^[^\s]{5,10}$/',
↪     self::str_error_password_format);
    }


    private function check_format_with_regex($variable, $regex, $error_message) {
        $result = filter_var($variable, FILTER_VALIDATE_REGEXP, array(
            'options' => array('regexp' => $regex)));
        if ($result === false || $result === null) {
            throw new Exception($error_message);
        }
    }
```

Le modèle `Users`

```php
public function user_from_id($id) {
    return $this->user_from_query(
                    'SELECT * FROM users WHERE id = ?', [$id]);
}

public function user_from_username($username) {
    $this->check_username($username);
    return $this->user_from_query(
                    'SELECT * FROM users WHERE username = ?', [$username]);
}


private function user_from_query($query, $array) {
    try {
        $statement = $this->db->prepare($query);
        $statement->execute($array);
        $users = $statement->fetchAll();
        if (count($users) == 0)
            return null;
        return User::from_array($users[0]);
    } catch (PDOException $e) {
        throw new Exception('Impossible d\'effectuer la demande.');
    }
}
```

Le modèle `Users`

```
    public function create_user($username, $password) {
        try {
            $this->check_username($username);
            $this->check_password($password);
            $statement = $this->db->prepare(
                    'INSERT INTO users(username, password) VALUES (?, ?)');
            $hash = password_hash($password, PASSWORD_DEFAULT);
            $statement->execute([$username, $hash]);
            $id = $this->db->lastInsertId();
            return new User($id, $username, $hash);
        } catch (PDOException $e) {
            throw new Exception('Impossible d\'inscrire l\'utilisateur.');
        }
    }
```