R3.01 : Développement web

R3.01 : Développement web

Publication date 2022

Table of Contents

List of Figures

Chapter 1. Présentation

1. Objectif

Dans la ressource R1.02:Développementd’interfaces web, vous avez appris à réaliser des pages web statiques. Dans cette ressource nous allons produire des pages Web dynamiques via un serveur HTTP (Hypertext Transfer Protocol). Le langage choisi est PHP (Hypertext Preprocessor), un langage de script impératif orienté objet. Pour la persistance de nos données nous utiliserons le SGBD (Système de Gestion de Base de Données) MariaDB et le moteur de base de données SQLite.

Nous disposons dans cette ressource de 6 créneaux de 1h20 de CM et de 16 créneaux de TD à raison de 2 par semaine. Voici une progression indicative :

Semaines	Objectifs
1..2	Le client serveur
3..4	Le Modèle Vue Contrôleurs
5	Évaluation
6..7	Le framework CodeIgniter
8	Évaluation

2. Évaluations

Les TD (Travaux dirigés) se décomposent en deux grandes parties, l'apprentissage du développement web dynamique en PHP en utilisant le patron de conception MVC (Modèle Vue Contrôleur) et l'extension PDO (PHP Data Objects) pour la persistance, puis l'utilisation du framework CodeIgniter et de l'ORM (Object Relational Mapper) Doctrine. Après chacune de ses parties, vous aurez une évaluation machine, cette ressource contribuera également à la SAE (Situation d'Apprentissage et d'Évaluation).

Chapter 2. Apprentissage de PHP

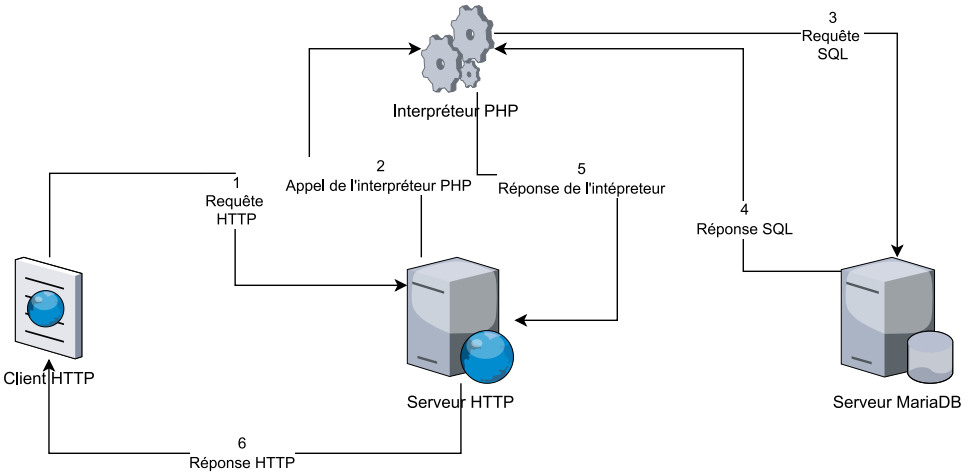
Nous allons apprendre PHP en construisant une application qui permet de commander des produits.

1. Le client serveur

Les protocoles ou environnement client serveur sont un mode de transaction ou le client envoie une requête pour obtenir une réponse du serveur. On dit que le serveur offre des services qui sont sollicités par les clients. Le serveur pouvant lui même être client d'un autre serveur.

Nous aurons un serveur HTTP, un serveur SQL (Structured Query Language) et un interpréteur PHP.

Figure 2.1. Exemple de requête



1. Le client HTTP effectue une requête en spécifiant une URL ;
2. Le serveur HTTP en fonction de l'extension de fichier renvoie directement la page (.htm, .html) ou passe la main à l'interpréteur PHP (.php) ; l'interpréteur PHP interprète les scripts PHP identifiés par la balise `<?php //TODO ?>`, la balise peut apparaître plusieurs fois dans un même fichier mais l'interprétation est unique ;
3. si le script fait appel à un serveur SQL alors une requête SQL est envoyée au serveur SQL ;
4. la réponse du serveur SQL est est renvoyée par le serveur SQL puis utilisée par l'interpréteur PHP ;
5. l'interpréteur fourni le fichier créé à au serveur HTTP ;
6. La réponse est enfin fournie au client HTTP.

   Comme vous l'observez vous aller mettre en oeuvre vos acquis du BUT1, HMTL/CSS, SQL, réseaux. PHP étant un langage orienté objet vous devez également mobiliser vos compétences dans ce domaine.

1.1. Les requêtes

Pour chaque exercice, vous aurez à consulter la documentation officielle PHP : `https://www.php.net/docs.php`.

1.1.1. Exécuter un script sur le serveur

Le serveur HTTP que nous allons utiliser est *srv.infoweb.iut-nantes.univ-nantes.prive*. C'est un serveur apache configuré pour que chaque utilisateur dispose d'un espace de publication, le répertoire `/home/userXXX/public_html` est disponible via l'URL `http://srv-infoweb.iut-nantes.univ-nantes.prive/~userXXX`.

Pour déposer un script sur le serveur vous allez devoir utiliser un protocole de transfert de fichiers à l'IUT, ce sera SFTP (SSH (Secure Shell) File Transfer Protocol). Pour réaliser les transferts, nous allons utiliser un client, ubuntu dispose d'un client graphique utilisable comme suit :

1. ouvrir l'explorateur de fichier (Nautilus) ;
2. utiliser la raccourci clavier **CTRL+L** pour afficher le chemin absolu ;
3. saisir l'url sftp://srv-infoweb.iut-nantes.univ-nantes.prive, vous identifier
4. Vous avez maintenant un point de montage vous permettant d'accéder à vos fichiers distants.

Créer un fichier avec l'extension .php contenant un script (`<?php phpinfo(); ?>`) qui lui même utilise la fonction `phpinfo()`[1], déposer ce fichier sur votre espace de publication et déclenchez l'exécution du script via votre navigateur.

Répondez au questions suivantes :

- Quelle est la version de PHP ?
- Dans les protocoles, les requêtes contiennent une entête et un corps, vous pouvez observer à travers votre navigateur l'entête envoyée par votre client
  1. clic droit :inspecter,
  2. network,
  3. rafraîchir,
  4. sélectionner la requête,
  5. header

  Où pouvez vous retrouver les informations du header HTTP dans le résultat de l'exécution du script ?
- Les interpréteurs PHP dispose d'un fichier de configuration (php.ini) qui limite les usages. Ici quelle est la taille maximum d'upload d'un fichier ?
- L'interpréteur PHP peut communiquer avec son environnement et par exemple obtenir des informations depuis le serveur HTTP au travers du tableau `$_SERVEUR`. Une URL HTTP peut contenir des fragments et parmis ses fragments des paramètres peuvent-être passés, les paramètres sont introduis par ?, ils sont sous la forme clef=valeur et sont séparés par &. Dans cet exemple hhttp://www.exemple.com:80/chemin/vers/monfichier.html?clé1=valeur1&clé2=valeur2#QuelquePartDansLeDocument les paramètres sont clé1 et clé2 . Modifier l'URL appelée pour passer en paramètres code_departement avec comme valeur 44, ville avec comme valeur Nantes et département avec comme valeur Loire Atlantique, que contiennent $_SERVER['REQUEST_METHOD'], $_SERVER['QUERY_STRING'] ? Pourquoi l'espace a t-il été transformé en %20 ?

1.1.2. Utiliser les méthodes de requête HTTP

Le protocole HTTP reconnaît des méthodes de requête[2], nous utiliserons dans ce cours :

GET

: pour récupérer des données

POST

: pour envoyer des informations.

Nous utiliserons deux clients : votre navigateur et **curl** un client en ligne de commande. D'autres clients comme postman et Insomnia REST Client existent, nous les utiliserons le semestre prochain.

[1] https://www.php.net/manual/fr/function.phpinfo.php
[2] `https://developer.mozilla.org/fr/docs/Web/HTTP/Methods`

1.1.2.1. la méthode GET

Le trafic en `GET` est principalement généré avec la barre d'URL, les ancres et les formulaires. En PHP, les paramètres sont récupérés avec le tableau `$_GET`[3].

PHP est un langage interprété faiblement typé le type est inféré à l'exécution et une variable peut changer de type[4].

Pour afficher sur la sortie standard PHP dispose de la fonction `echo`[5], de la fonction `print`[6] et de la fonction `var_dump`[7].

En PHP, les tableaux sont associatifs[8].

1.1.2.1.1. Afficher les paramètres

Nous utiliserons comme fragment de paramètres *?pseudo=laurent&password=pass&statut=admin*.

Écrire un script `get1.php` qui utilise `$_GET` et `var_dump` pour produire l'affichage suivant :

```
array(3) { ["pseudo"]=> string(7) "laurent" ["login"]=> string(4) "pass"
["status"]=> string(5) "admin" }
```

Dans un fichier PHP, cohabitent du texte et des scripts, entourer votre script avec les balises `<pre> </pre>`, vous devriez obtenir

```
array(3) {
  ["pseudo"]=>
  string(7) "laurent"
  ["password"]=>
  string(4) "pass"
  ["status"]=>
  string(5) "admin"
}
```

`var_dump` est utilisée pour le débogage, en utilisant `echo` et `$_GET["clef"]` écrire un script `get2.php` qui permet d'avoir la page HTML 5 suivante :

```
pseudo : laurent

password : pass

status : admin
```

Pour rappel une page HTML a la structure suivante :

```
<!doctype html>
<html lang="fr">
<head>
  <meta charset="utf-8">
  <title>Test2</title>
  </head>
<body>
  ...
  <!-- Le reste du contenu -->
  ...
</body>
</html>
```

Nous avons utilisé trois fois `$_GET`, nous pouvons améliorer notre script avec la boucle `foreach`[9]. Une syntaxe alternative est également possible avec `<?php foreach ($tab as $clef=>$valeur): ?> <?php endforeach;?>`.

[3] https://www.php.net/manual/fr/reserved.variables.get.php
[4] https://www.php.net/manual/fr/language.variables.variable.php
[5] https://www.php.net/manual/fr/function.echo.php
[6] https://www.php.net/manual/fr/function.print.php
[7] https://www.php.net/manual/fr/function.var-dump
[8] https://www.php.net/manual/fr/book.array.php
[9] https://www.php.net/manual/fr/control-structures.foreach.php

1.1.2.1.2. Envoyer des paramètres

Nous avons déjà envoyé des paramètres avec la barre d'URL, nous pouvons également utiliser le client en ligne de commande **curl**, vous pouvez obtenir la documentation de **curl** avec `man curl`. Tester et comprendre la commande `curl -v --noproxy "*" -X GET "http://srv-infoweb/~userXXX/chemin_dans_public_html_vers_votre_script/get2.php?pseudo=laurent&login=pass&statut=admin"` *srv-infoweb.iut-nantes.univ-nantes.prive est aliasé, il se nomme*.

Une autre solution est de créer un formulaire[10], créer un formulaire `get2_form.html` qui permet de saisir le pseudo et le mot de passe, le statut sera envoyé avec un `input` de type `hidden`.

Créer un document HTML 5 qui contient 2 ancres qui référencent le même script `get2.php` dont chacune code un statut différent, ici nous passons dans l'URL les informations pour le serveur.

```
je suis admin

je suis visiteur
```

1.1.2.2. la méthode POST

Il est impossible de générer un trafic en `POST` depuis la barre d'URL, en effet les paramètres ne sont plus passés dans l'entête de la requête mais dans son corps. Il nous faudra un formulaire ou un langage comme javascript pour générer du `POST` depuis un navigateur.

Créer le fichier `get_post.php` qui devra afficher les paramètres comme le faisait `get2.php`, ce script devra fonctionner que la requête soit en `GET` ou en Évaluation. Vous aurez à utiliser `isset`[11]qui permet de savoir si une variable est déclarée et est non nulle, vous aurez également à utiliser une conditionnelle[12]. Attention en PHP `$_GET` est toujours déclaré car en vérité il ne teste pas la méthode HTTP mais la présence de paramètres dans le header.

Vous pourrez tester votre code avec **curl** : `curl -v --noproxy "*" -d "pseudo=laurent&login=pass&statut=admin" -H "Content-Type: application/x-www-form-urlencoded" -X POST "http://serveur/~user/chemin/get_post.php"`.

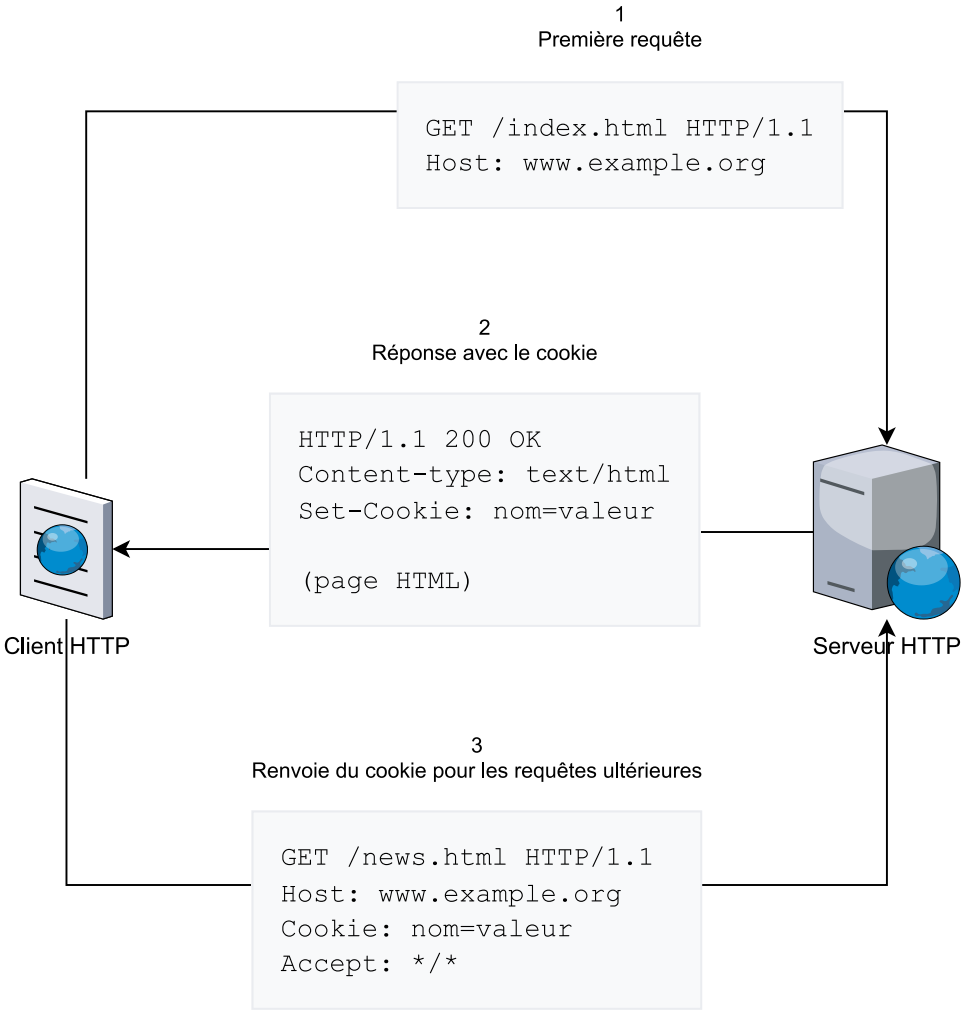
Pour finir créer un formulaire et tester.

1.2. Les cookies et les sessions

Le protocole HTTP est protocole sans état qui n'enregistre pas l'état de session lors d'une communication entre deux requêtes successives. Cependant le protocole HTTP permet de remédier à ce problème en utilisant des cookies. Un cookie est texte court envoyé par un serveur HTTP à un client HTTP, que ce dernier renvoiera les prochaines fois qu'il se connectera aux serveurs partageant le même nom de domaine.

[10] `https://developer.mozilla.org/fr/docs/Web/HTML/Element/Form`
[11] `https://www.php.net/manual/fr/function.isset.php`
[12] `https://www.php.net/manual/fr/control-structures.if.php`

Figure 2.2. Les cookies HTTP

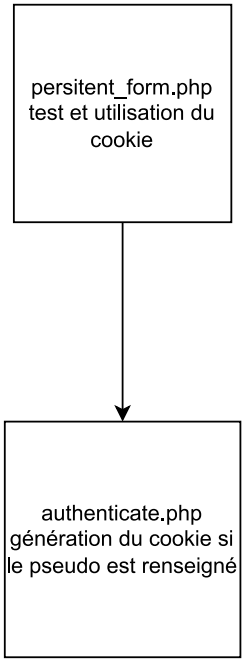


Les cookies sont utilisés pour la personnalisation, la gestion des sessions, le pistage, ...

1.2.1. La personnalisation

Depuis HTML5 les données peuvent-être validées par le client et la mémoire conservée dans le navigateur, les mêmes fonctionnalités peuvent être offertes par PHP. Nous allons créer un formulaire `perisitent_form.php` qui demandera le pseudonyme et le mot de passe, si ce formulaire est consulté à nouveau alors le pseudonyme doit rester pré-rempli, ce formulaire aura pour cible `authenticate.php` qui créera le cookie.

Figure 2.3. Cookie pour la personnalisation



Le création d'un cookie est réalisée avec `setcookie()`[13]. Attention, selon la configuration du serveur, la création du cookie comme toute manipulation de l'entête HTTP doit avoir lieu avant toute écriture sur la sortie standard. La lecture est réalisée avec le table `$_COOKIE`[14].

Quelques conseils :

- utiliser `htmlentities()`[15] pour le contenu de votre cookie;
- dans votre navigateur, vous pouvez observer et supprimer les cookies stockés.

1.2.2. La gestion des sessions

Nous allons augmenter améliorer notre système.

1.2.2.1. Effectuer une direction

Dans authenticate.php si l'authentification échoue alors une redirection est effectuée vers `persitent_form.php` sinon ok doit-être affiché. La redirection en `HTTP` est en trois étapes : [16]

1. Le client demande une ressource au serveur ;
2. le serveur répond au client avec un code 30X que la ressource est disponible à une autre adresse et lui fourni l'adresse,
3. le client consulte la nouvelle adresse.

Les manipulations de l'entête en PHP se font avec avec `header()`[17], l'en-tête à modifier sera la "`Location:`'.

La liste des utilisateurs est définie par le tableau :

```
$users = array(
   "jojo" => array("password" => "pass1", "status" => "administrator"),
   "raoul" => array("password" => "pass2", "status" => "visitor"),
```

[13] `https://www.php.net/manual/fr/function.setcookie.php`
[14] `https://www.php.net/manual/fr/reserved.variables.cookies.php`
[15] `https://www.php.net/manual/fr/function.htmlentities.php`
[16] `https://developer.mozilla.org/fr/docs/Web/HTTP/Redirections`
[17] `https://www.php.net/manual/fr/function.header.php`

```
   "roméo" => array("password" => "pass3", "status" => "customer"),
    );
```

La fonction `array_key_exists()` peut vous aider mais n'est pas indispensable[18].

1.2.2.2. Utiliser une variable de session

Maintenant que nous avons pu nous authentifier, nous allons créer une variable de session[19], dans cette variable de session vous stockerez le pseudo et le status puis vous effectuerez une redirection vers `site.php`.

Si la variable de session n'est pas positionnée alors `site.php` devra afficher accès refusé sinon le pseudo sera affiché et en fonction du status différents affichage :

visitor

: consulter

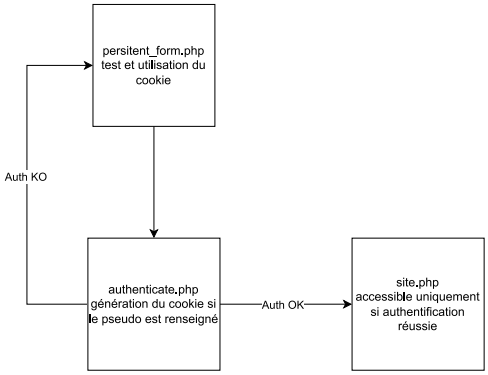
customer

: consulter, acheter

administrator

: consulter, acheter, administrer

Figure 2.4. Session



1.3. Pour aller plus loin : Offrir des contenus différentiés

Il est possible pour le client HTTP en spécifiant dans son entête un type MIME (Multipurpose Internet Mail Extensions) de choisir pour une même URL un contenu différent. Écrire un fichier `mime.php` qui si le type est

text/plain

: renvoie

```
pseudonyme: jojo
```

text/html

: renvoie une page web en HTML5

application/json

: renvoie

```
{
  "pseudonyme": "jojo"
}
```

[18] `https://www.php.net/manual/fr/function.array-key-exists.php`
[19] `https://www.php.net/manual/fr/session.examples.basic.php`

L'en-tête `Content-Type` sert à indiquer le type MIME de la ressource sollicitée et ne peut spécifié pour `GET`, vous devrez donc utiliser `POST`. Les tests se feront avec **curl**.

En PHP, header permettra de spécifier le type MIME de la réponse et `$_SERVER["CONTENT_TYPE"]` permet de connaître le type MIME demandé.

2. L'approche MVC

Le patron de conception ou motif d'architecture logiciel MVC (Modèle Vue Contrôleur) permet de séparé la logique métiers de l'affichage, il est composé de trois parties :

Le modèle

qui gère les données et la logique métiers

Le contrôleur

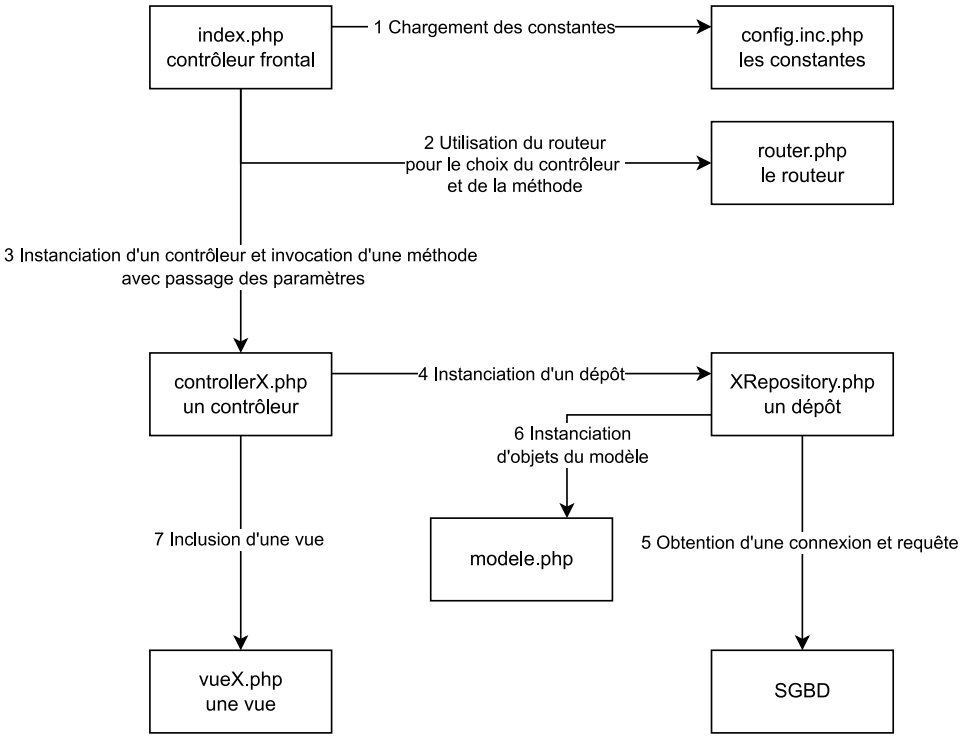
qui gère les demandes utilisateur en sollicitant le modèle et en modifiant la vue

La vue

qui est l'interface l'utilisateur

Nous allons réaliser une implémentation du MVC adaptée au web.

Figure 2.5. Notre MVC



2.1. Le contrôleur frontal et l'inclusion des constantes

Le contrôleur frontal sera le seul fichier PHP qui pourra être appelé directement, les autres seront inclus depuis celui-ci. Créer l'arborescence suivante :

```
-td2
  -app
    -config
      -config.inc.php
    -index.php
```

```
  -system
```

app contiendra notre application et system des utilitaires.

Nous allons commencer par forcer l'appel du contrôleur frontal, apache permet de redéfinir localement sa configuration avec un fichier .htaccess, ce fichier redéfini la configuration sur le répertoire qui le contient et ceux qui sont inclus. Vous le mien que vous devrez placer dans td2/app et adapter :

```
RewriteEngine On

FallbackResource /~berdjugin-jf/php/td2/app/index.php
```

RewriteEngine On active le module de réécriture d'apache utile pour la directive FallbackResource[20], cette directive indique que si un fichier n'est pas trouvé alors index.php est servi. Tester avec des URL inclues dans td2.

Nous allons depuis index.php inclure config.inc.php qui contiendra la définition de nos constantes. Pour inclure un autre script vous devez utiliser une des structures de contrôle suivante : include, include_one, require, require_one[21]. Il existe également des possibilités d'auto-chargement, nous les utiliserons bientôt dans cette partie[22].

Dans config.inc.php définir avec define[23] la constante HOME pour qu'elle contienne le chemin absolu avec le répertoire td2. Vous devrez utiliser la constante magique __DIR__[24], la concaténation qui est le . en PHP et la constante pré-définies DIRECTORY_SEPARATOR. Faites afficher cette constante dans index.php. Vous devez obtenir l'affichage d'un chemin absolu similaire à celui-ci

```
/chemin_vers_td2/td2/app/config/../
```

Les inclusions sont calculées à partir du premier fichier chargé, index.php pour nous. Pour faciliter les inclusions nous allons définir un tableau constant pour stocker l'URL du site et les informations sur la base de données :

```
const CFG = array(
        "db" => array(
                "host" => HOME."data".DIRECTORY_SEPARATOR,
                "port" => null,
                "database" => "madb.db",
                "login" => "",
                "password" => "",
                "options" => array(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION),
                "exec" => "PRAGMA foreign_keys = ON;"
           ),
          "siteURL" => "http://localhost/~jub/php/td2/app/" //votre url
        );
```

Nous utiliserons SQLite un moteur SQL léger qui peut-être exécuté en local.

Une fois les constantes de l'application définies, notre contrôleur frontal va faire appel à un routeur.

2.2. Le routage

Nous avons vu que dans l'architecture un client effectue une requête dans l'attente d'une réponse. Le routeur a pour objectif d'analyser la requête pour déclencher l'exécution d'une méthode d'un contrôleur avec d'éventuels paramètres.

Le format d'URL retenu est */contrôleur/méthode/param1/param2*. Si une méthode n'est pas fournie alors la méthode index sera choisie. Par exemple */Home* déclenche la méthode index du contrôleur Home, */User/add/jojo/44* déclenche la méthode add du contrôleur User avec les paramètres *jojo* et *44*.

Nous allons créer un classe sans attribut ce qui peut-être contestable mais qui laisse la possibilité d'évolution comme par exemple de choisir les routes depuis un fichier de configuration, une route étant une association entre une URL et l'exécution d'une méthode d'un contrôleur.

[20] https://httpd.apache.org/docs/trunk/fr/mod/mod_dir.html#fallbackresource
[21] https://www.php.net/manual/fr/function.include.php
[22] https://www.php.net/manual/fr/language.oop5.autoload.php
[23] https://www.php.net/manual/fr/function.define.php
[24] https://www.php.net/manual/fr/language.constants.magic.php

Créer dans `system` une classe `Router` dans `Routeur.php`[25].

```
-td2
  -system
    -Router.php
```

```
<?php

/**
 * Class Router
 */
class Router
{
    /**
     * permet d'éxécuter la méthode d'un controleur choisi
     * si la méthode est omise alors index est choisi
     * Les URL saisies sont de la forme
     *      site/controleur
     *      site/controleur/method
     *      site/controleur/method/param1/...
     */
    function route()
    {
        $scriptName = $_SERVER["SCRIPT_NAME"];
        $requestURI = $_SERVER["REQUEST_URI"];

        //Le script name contient index.php on le supprime
        $prefix = substr($scriptName, 0, strlen($scriptName) - 9);
        //Le suffixe est le requestURI privé du préfix
        $suffix = substr($requestURI,strlen($prefix));

        $params = explode("/", $suffix);

        if (count($params) == 0) {
            echo "no controller found";
            die();
        }

        $controller = $params[0];
        array_shift($params);

        if (count($params)==0)
            $controllerMethod="index";
        else
            $controllerMethod=$params[0];

        array_shift($params);
        try {
            $controllerinstance = new $controller();
            $controllerinstance->$controllerMethod($params);
        }
        catch (Error $e){
            echo "route not found $controller $controllerMethod ";
            var_dump($params);

        };
    }
}
```

Pour que nos tests soient efficients, nous allons également créer un contrôleur qui sera choisi par le routage :

```
-td2
  -app
    -controller
      -Home.php
```

```
class Home
{
```

[25] `https://www.php.net/manual/fr/language.oop5.php`

```
//A terme, il n'y aura pas de echo dans un contrôleur
//Les echos seront dans les vues

    function index(){
        echo "index"

    }
    function method(){
        echo "method";
    }
    function methodeWithParameter(array $params){
        var_dump($params);
    }
}
```

Si $controller est le nom de votre contrôleur; $controllerMethod celui de la méthode et $params le tableau de paramètres alors le code suivant permet d'instancier un contrôleur et d'appeler la méthode.

```
$controllerinstance = new $controller();
$controllerinstance->$controllerMethod($params);
```

Le routeur sera instancié dans index.php et la méthode route sera invoquée :

```
$router = new Router();
$router->route();
```

Nous allons ici utiliser le chargement automatique des classes, lors de l'instanciation d'un objet, PHP chargera automatiquement la classe, placer le code suivant dans votre index.php après l'inclusion de config.inc.php et avant l'instanciation du routeur [26]:

```
spl_autoload_register(function($class) {
        $prefix = '..'. DIRECTORY_SEPARATOR . "system".DIRECTORY_SEPARATOR;

        if (file_exists($prefix.DIRECTORY_SEPARATOR.$class . '.php')) {
            require_once($prefix.DIRECTORY_SEPARATOR.$class . '.php');
        }
        $prefix = '..'. DIRECTORY_SEPARATOR . "app".DIRECTORY_SEPARATOR;
        if (file_exists($prefix."controller".DIRECTORY_SEPARATOR.$class . '.php')) {
            require_once($prefix."controller".DIRECTORY_SEPARATOR.$class . '.php');
        }
        if (file_exists($prefix."model".DIRECTORY_SEPARATOR."entity"
            .DIRECTORY_SEPARATOR.$class . '.php')) {
            require_once($prefix."model".DIRECTORY_SEPARATOR."entity"
              .DIRECTORY_SEPARATOR.$class . '.php');
        }
        if (file_exists($prefix."model".DIRECTORY_SEPARATOR."repository"
            .DIRECTORY_SEPARATOR.$class . '.php')) {
            require_once($prefix."model".DIRECTORY_SEPARATOR."repository"
              .DIRECTORY_SEPARATOR.$class . '.php');
        }
    });
```

A ce stade, vous pouvez tester pour vérifier si les méthodes de vos contrôleurs sont bien appelées.

2.3. Les contrôleurs

Nous venons de voir que nos contrôleurs sont invoqués par le routeur après analyse de l'url. Les contrôleurs vont solliciter les modèles pour obtenir, créer, modifier des informations avant d'appeler des vues. Nous allons afficher ces étapes avant de les implémenter réellement :

```
class Home
{
    //Création d'un attribut pour accèder au modèle
    public function __construct()
```

[26] Avec des espaces de nommage et une convention pour les répertoires, l'autoload peut-être simplifiée.

```
    {
        //instanciation de l'attribut
    }

    function index(){
        //Solicitation du modèle
        //Solicitation de la vue

    }

}
```

2.4. Le modèle

Nous allons utiliser le patron de conception dépôt ou repository, le repository est sollicité par le contrôleur, il sollicite la source de donnée et renvoie des entités. Commençons par créer une interface :

```
-td2
  -app
    -model
      -repository
        -ProductRepositoryInterface.php
```

```
Interface ProductRepositoryInterface
{
    public function findAll(): array;
}
```

Nous allons également créer notre entité, elle ne contient que des attributs privés avec des getters, des setters et un constructeur sans paramètre[27]:

```
-td2
  -app
    -model
      -entity
        -ProductEntity.php
```

```
class ProductEntity
{
    private int $id;
    private string $name;
    private float $price;
    private int $quantity;


    /**
     * @return int
     */
    public function getId(): int
    {
        return $this->id;
    }

    /**
     * @return string
     */
    public function getName(): string
    {
        return $this->name;
    }

    /**
     * @return float
     */
    public function getPrice(): float
    {
        return $this->price;
```

[27] En PHP toute classe possède un constructeur par défaut qui est sans paramètre.

```
    }

    /**
     * @return int
     */
    public function getQuantity(): int
    {
        return $this->quantity;
    }

    /**
     * @param int $id
     */
    public function setId(int $id): void
    {
        $this->id = $id;
    }

    /**
     * @param string $name
     */
    public function setName(string $name): void
    {
        $this->name = $name;
    }

    /**
     * @param float $price
     */
    public function setPrice(float $price): void
    {
        $this->price = $price;
    }

    /**
     * @param int $quantity
     */
    public function setQuantity(int $quantity): void
    {
        $this->quantity = $quantity;
    }

}
```

A vous de créer une classe `MemoryProductRepository` dans `repository` qui implémente `ProductRepositoryInterface` et qui dans `findAll()` renvoie un tableau de `ProductEntity`.

Modifier votre contrôleur `Home` pour que l'attribut soit de type ProductRepositoryInterface et lui affecter dans le constructeur une instance de `MemoryProductRepository` enfin dans la méthode `index` appeler la méthode findAll() du `repository` et vérifier le résultat avec un `var_dump()`.

2.5. La vue

A ce stade, notre contrôleur dispose des données du modèle et souhaite les afficher, c'est la mission de la vue. Nous pourrions comme pour le Router créer une classe mais dans un soucis de simplification nous allons directement inclure la page web. Avec une inclusion, la page aura accès à l'ensemble des variables disponibles dans le contrôleur donc aux données qui viennent du modèle ce qui est souhaité mais la page aura également accès au attributs du contrôleur, ce qui n'est pas souhaitable. Créer une vue et y faire afficher vos produits.

```
-td2
  -app
    -view
      -home.php
```

```
<!doctype html>
<html lang="fr">
<head>
    <meta charset="utf-8">
```

```
    <title>Mon magasin</title>
</head>
<body>
<table>
    <thead>
        <tr>
            <th> id </th>
            <th> name </th>
            <th> price </th>
            <th> quantity </th>
        </tr>
    </thead>
    <tbody>
    <?php ?>
       <tr>
        <td> <?=  ?> </td>
        <td> <?=  ?> </td>
        <td> <?=  ?> </td>
        <td> <?=  ?> </td>
       </tr>
    <?php ?>
    </tbody>
</table>
</body>
</html>
```

3. La persistance

Nous avons écrit beaucoup de code, pour au final produire un site statique, en effet, il renvoie toujours les mêmes informations. Les exécutions des scripts étant indépendantes, nous ne pouvons pas par exemple ajouter ou supprimer des produits. Nous allons utiliser un moteur base de données de pour assurer la persistance de nos données entre deux exécutions puis nous rajouterons des fonctionnalités à notre application.

3.1. Création et peuplement de la base

Nous allons utiliser SQlite pour créer la base madb.db dans data :

```
-td2
  -data
    -madb.db
    -create_and_populate_madb.sql
```

Commençons par créer un script de création et de peuplement de la base de données (create_and_populate_madb.sql) :

```
CREATE TABLE IF NOT EXISTS product (
        id INTEGER NOT NULL PRIMARY KEY,
        name TEXT NOT NULL UNIQUE,
        price FLOAT NOT NULL,
        quantity INTEGER NOT NULL DEFAULT 0
);

INSERT OR REPLACE INTO product (id,name,price, quantity) VALUES (1,'p1',10.0,1);
INSERT OR REPLACE INTO product (id,name,price, quantity) VALUES (2,'p2',20.0,1);
```

Pour exécuter ce script, nous allons utiliser la commande sqlite3[28].

Pour exécuter notre script de création depuis data utiliser : **sqlite3 madb.db '.read create_and_populate_madb.sql'**.

Vous connecter sur la base (**sqlite3 madb.db**), vérifier les tables (**.table**), vérifier le schéma (**.schema**), afficher le contenu (**select**), ajouter un nouvel enregistrement (**insert**), modifier le nouvel enregistrement (**update**), le supprimer (**delete**).

[28] https://www.sqlite.org/cli.html

3.2. Création d'un repository avec une source de données

Nous allons commencer par créer un singleton qui nous permettra d'obtenir une instance de PDO et via cette instance d'obtenir une connexion :

```
-td2
  -system
    -SPDO.php
```

```
class SPDO
{
    private $connexion;
    private static $PDOInstance;

    private function __construct(string $dsn,
                                 ?string $username = null,
                                 ?string $password = null,
                                 ?array $options = null,
                                 ?string $exec = null)
    {
        $this->connexion = new \PDO($dsn,$username , $password, $options);
        $this->connexion->exec($exec);
    }

    public static function getInstance(string $dsn,
                                       ?string $username = null,
                                       ?string $password = null,
                                       ?array $options = null,
                                       ?string $exec = null):SPDO
    {
        if(is_null(self::$PDOInstance))
        {
            self::$PDOInstance = new SPDO($dsn,$username , $password, $options, $exec);
        }
        return self::$PDOInstance;
    }

    public function getConnexion(): PDO{
        return $this->connexion;
    }
}
```

Cette classe possède un constructeur privé et un `new` est donc exclus, l'instanciation reposera sur l'utilisation de la méthode statique `getInstance()` qui renvoierra une nouvelle instance de PDO si elle n'a pas déjà été créée ou le cas échéant l'instance déjà créée. L'appel d'une méthode statique est réalisée via `::`, n'oubliez pas ce que vous avez défini dans `config.inc.php`. Dans `index.php` obtenir une connexion et la faire afficher, le dsn sera au format `sqlite:chemin/madb.db`[29].

Le front contrôleur n'est pas le bon endroit pour obtenir la connexion, en effet toutes les pages n'en n'auront pas besoin. Créer une classe DbProductRepository qui implémente ProductRepositoryInterface et qui a comme attribut une connexion (déplacer le code précédemment créé). Modifier le contrôleur Home pour qu'il utilise ce nouveau dépôt.

Enfin dans la méthode `findAll()` vous allez réaliser une requête préparée PDO : `prepare()`[30], `execute()`[31], fetchAll()[32] avec le mode `PDO::FETCH_CLASS` et la classe `ProductEntity`.

Nous avons un site dynamique mais avec une seule fonctionnalité, nous allons y remédier.

4. Faisons grandir l'application

Nous allons ajouter des fonctionnalités : la création, la modification et la suppression d'un produit.

[29] `https://www.php.net/manual/fr/ref.pdo-sqlite.connection.php`
[30] `https://www.php.net/manual/fr/pdo.prepare.php`
[31] `https://www.php.net/manual/fr/pdostatement.execute.php`
[32] `https://www.php.net/manual/fr/pdostatement.fetchall.php`

Commencer par créer un contrôleur Produit.

4.1. Afficher un produit

Nous allons afficher un produit si il existe et une erreur sinon.

Il nous faut une route :

Product/display/id

: en GET qui pour afficher la vue du produit, elle conduit au contrôleur Product pour exécuter la méthode display avec comme paramètre un tableau qui contient l'id.

Il nous faut une nouvelle méthode du dépôt :

findById(int $id):?ProductEntity

: qui renvoie le produit si il existe ou l'entité sinon.

Il nous faut également la vue

displayProduct

: qui affiche le produit

Nous allons commencer par le modèle, en vous inspirant de findAll, créer une requête préparée pour afficher un produit. L'id est une entier reçu en paramètre, nous allons l'utiliser dans une requête préparée :

1. Utiliser prepare[33] avec :id comme paramètre nommé
2. Réaliser l'association entre le paramètre effectif de la méthode ($id) et le paramètre nommé (:id) avec bindParam[34].
3. Exécuter la requête.
4. Récupérer le résultat avec fetch en le PDO::FETCH_ASSOC.
5. Créer un objet de type Product si possible
6. Renvoyer le résultat.

Pour tester le modèle, rajouter la méthode display dans le contrôleur et tester avec plusieurs URL.

Quelques conseils :

- Tout ce qui vient du serveur est string[35], intval[36] et floatval[37] permettent de transtyper.

4.2. Créer d'un produit

Pour créer un produit, il faut deux routes :

Product/add

: en GET sans paramètre pour afficher le formulaire de création

Product/addProduct

: en POST avec dans le corps id, name, price, quantity pour modifier le modèle.

[33] https://www.php.net/manual/en/pdo.prepare
[34] https://www.php.net/manual/fr/pdostatement.bindparam
[35] https://www.php.net/manual/fr/language.types.string.php
[36] https://www.php.net/manual/fr/function.intval.php
[37] https://www.php.net/manual/fr/function.floatval

Pour le dépôt, nous ajouterons une méthode :

add(ProductEntityProduct $product):ProductEntity

qui reçoit un produit en paramètre, essaye de le créer puis le récupère dans la base

Pour les vues, nous aurons le formulaire de saisie :

addProduct

formulaire permettant en POST d'envoyer à Product/addProduct

displayProduct

pour afficher le produit créé ou une erreur si la création est impossible

4.3. Modifier un produit

Pour modifier vous devez modifier la vue home.php pour rajouter une colonne, cette colonne contiendra une ancre vers la route Product/update/id.

La méthode update sollicitera le modèle pour à partir de l'id récupérer le produit puis affichera la vue updateProduct qui contiendra un formulaire prérempli et l'id masqué. Ce formulaire conduira à la méthode updateProduct qui réalisera la mise à jour avant d'afficher le produit ajouté avec Product/display/id. Il faudra également ajouter une méthode update(int $id, ProductEntity $product):ProductEntity au dépôt, $id sera l'identifiant du produit à modifier et $product les nouvelles valeurs.

4.4. Supprimer un produit

Pour supprimer un produit vous devez modifier la vue home.php pour rajouter une colonne, cette colonne contiendra une ancre vers la route Product/delete/id. Dans le dépôt rajouter la méthode delete(int $id): bool. Une fois le modèle modifié, vous pouvez faire une redirection vers /Home.

5. Limitons les accès

Chapter 3. Mise en oeuvre d'un framework