159 - Intégrer des bases de données dans des applications Web

Rapport personnel

Date de création : 30.01.2023  
Version 1 du 13.02.2023

Luvisotto Marco Baldo

|  |  |  |
| --- | --- | --- |
|  |  | Module du 30.01.2023 au 7.03.2023 |

Table des matières

[1 Introduction 5](#_Toc129529077)

[1.1 Objectifs du module 5](#_Toc129529078)

[2 Théorie 5](#_Toc129529079)

[3 PHP 5](#_Toc129529080)

[3.1 MVC 5](#_Toc129529081)

[3.2 Code & explications 5](#_Toc129529082)

[3.2.1 Session 5](#_Toc129529083)

[3.2.2 POST & GET 6](#_Toc129529084)

[a) POST 6](#_Toc129529085)

[b) GET 6](#_Toc129529086)

[3.2.3 Tableau 7](#_Toc129529087)

[a) Associatif 7](#_Toc129529088)

[nom: Dupont 7](#_Toc129529089)

[prenom: Jean 7](#_Toc129529090)

[age: 35 7](#_Toc129529091)

[b) Numéroté 7](#_Toc129529092)

[3.3 Callback 8](#_Toc129529093)

[La fonction processDate est appelé avec une data en paramètre et la fonction DisplayResult 8](#_Toc129529094)

[3.4 Exercices 8](#_Toc129529095)

[a) Exercice Client Serveur 8](#_Toc129529096)

[4 Projet 11](#_Toc129529097)

[4.1 Analyse 11](#_Toc129529098)

[4.1.1 Introduction & explication projet 12](#_Toc129529099)

[4.1.2 Planification 12](#_Toc129529100)

[4.1.3 Use case 12](#_Toc129529101)

[4.1.4 Maquette 13](#_Toc129529102)

[4.1.5 Schéma ER 14](#_Toc129529103)

[4.1.6 Diagramme activités 14](#_Toc129529104)

[4.2 Conception 15](#_Toc129529105)

[4.2.1 Diagramme séquence system 15](#_Toc129529106)

[4.2.2 Diagramme de classes 16](#_Toc129529107)

[4.2.3 Diagramme de séquence interaction 17](#_Toc129529108)

[4.2.4 Schéma relationnel 18](#_Toc129529109)

[4.3 Implémentation 18](#_Toc129529110)

[4.3.1 SERVEUR 18](#_Toc129529111)

[a) Difficulté rencontrée 19](#_Toc129529112)

[4.3.2 CLIENT 21](#_Toc129529113)

[4.3.3 DB 21](#_Toc129529114)

[a) Difficulté rencontrée 21](#_Toc129529115)

[5 Test 22](#_Toc129529116)

[5.1 Injection 22](#_Toc129529117)

[5.2 Test fonctionnel 22](#_Toc129529118)

[6 Synthèse 23](#_Toc129529119)

[7 Conclusion 23](#_Toc129529120)

# Introduction

Au cours de mon apprentissage, je vais apprendre à mettre en place les points clés pour concevoir, intégrer, sécuriser et tester une application web et sa base de données. Je vais notamment apprendre à analyser les exigences de l'application et de la base de données, identifier les mesures de sécurité nécessaires, intégrer l'application avec la base de données, et définir et exécuter une procédure de test et de remise.

## Objectifs du module

Analyser les exigences d’une application Web et de la base de données, respectivement des éléments de données à lier, définir et documenter la technique de liaison.

2 Identifier les informations importantes de protection et de sécurité en tenant compte de la protection des données, et définir les mesures.

3 Réaliser l’intégration de l’application Web avec la base de données, respectivement aux éléments de données, en prêtant attention aux transactions, à la protection et la sécurité des données.

4 Mettre en œuvre les souhaits de modifications conformément au déroulement prescrit des modifications.

5 Définir et exécuter la procédure de test et de remise, la documenter dans un procès-verbal de tests. Si nécessaire, entreprendre les corrections.

# Théorie

# PHP

## MVC

Le modèle MVC (Modèle-Vue-Contrôleur) est un design pattern pour développer des applications web. Il sépare l'application en trois parties distinctes :

* Modèle : représente les données de l'application et les logiques de gestion de la base de données.
* Vue : gère l'affichage des données pour l'utilisateur final.
* Contrôleur : contrôle les interactions entre le modèle et la vue. Il reçoit les entrées utilisateur, effectue les traitements nécessaires avec le modèle et choisit la vue à afficher.

Ainsi, dans une application PHP utilisant MVC, le contrôleur agit comme un pont entre la vue et le modèle, en gérant les requêtes HTTP, en déterminant les actions à effectuer sur les données et en sélectionnant la vue à afficher.

Les changements apportés aux données sont effectués via le modèle, et la vue s'occupe de les afficher à l'utilisateur. Cela permet une séparation claire entre les différentes parties de l'application et une plus grande flexibilité pour les futures modifications ou mises à jour.

## Code & explications

### Session

Une variable de session est une variable globale en PHP qui permet de conserver les données d'un utilisateur à travers plusieurs pages web dans une session unique.

Elle est stockée sur le serveur et est associée à un identifiant de session qui est envoyé au navigateur client sous forme de cookie. Chaque fois que l'utilisateur accède à une page du site, le serveur vérifie si l'identifiant de session existe et restaure les données de la session associées.

La session est détruite lorsque l'utilisateur ferme le navigateur ou lorsque le délai d'expiration de la session est atteint. Les données peuvent être enregistrées et accédées à l'aide de la superglobale **$\_SESSION** en PHP.

Pour créer une variable de session en PHP, vous devez d'abord démarrer la session avec la fonction :

|  |
| --- |
| session\_start() |

Ensuite, vous pouvez attribuer des valeurs à des variables de session en utilisant le tableau :

|  |
| --- |
| $\_SESSION[‘Cle’] |

Il est nécessaire d'appeler la fonction session\_start() à chaque début de script PHP qui utilise les variables de session, même si une session a déjà été démarrée.

### POST & GET

Les variables POST et GET ses deux méthodes utilisées pour envoyer des données à un serveur web.

#### POST

La méthode POST est utilisée pour envoyer des données à un serveur web sans que les données soient visibles dans l'URL, ce qui est souvent utilisé pour les formulaires ou les opérations qui doivent être sécurisées.

Exemple : HTML & PHP

|  |
| --- |
| <form action="traitement.php" method="post">  <label>Nom :</label>  <input type="text" name="nom">  <br>  <label>Adresse :</label>  <input type="text" name="adresse">  <br>  <input type="submit" value="Envoyer">  </form> |

|  |
| --- |
| <?php  if ($\_POST) {  $nom = $\_POST['nom'];  $adresse = $\_POST['adresse'];  echo "Bonjour $nom, votre adresse est $adresse.";  }  ?> |

Lorsque l'utilisateur remplit le formulaire et clique sur le bouton "Envoyer", les données sont envoyées au serveur en utilisant la méthode POST, et sont accessibles dans la page de traitement via le tableau associatif $\_POST.

#### GET

La méthode GET, d'autre part, envoie les données dans l'URL, ce qui est utile pour les requêtes de recherche simples. Les données envoyées à travers la méthode GET sont limitées à une certaine longueur, ce qui peut être un problème pour les opérations nécessitant beaucoup de données.

Exemple : URL & PHP

|  |
| --- |
| http://www.sitePerso.com/fichier.php?nom=Olivier |

|  |
| --- |
| if(isset($\_GET['nom'])){  echo 'Bonjour '..$\_GET['nom'];  } |

### Tableau

#### Associatif

Un tableau associatif est un type de tableau qui relie une clé à une valeur.

Cette structure de données permet de stocker plusieurs enregistrements sous forme de paires clé-valeur.

Chaque clé est unique et associée à une valeur spécifique. Les tableaux associatifs sont souvent utilisés pour stocker des informations organisées dans un ordre spécifique.

En PHP, les tableaux associatifs sont créés en utilisant la syntaxe suivante :

|  |
| --- |
| $tableau = array("cle1" => "valeur1", "cle2" => "valeur2", "cle3" => "valeur3"); |

Les clés peuvent être des chaînes de caractères ou des nombres entiers. Les valeurs peuvent être de n'importe quel type de données, comme des chaînes de caractères, des nombres, des tableaux, des objets, etc.

Pour accéder à une valeur spécifique dans un tableau associatif, vous pouvez utiliser la syntaxe suivante :

$valeur = $tableau["cle1"];

Exemple foreach :

|  |
| --- |
| foreach ($personnes as $key => $value) {  echo $key . ": " . $value . "<br>";  } |

Outpout

|  |
| --- |
| nom: Dupontprenom: Jeanage: 35 |

#### Numéroté

Un tableau numéroté en PHP est un type de tableau qui utilise des indices numériques pour accéder à ses éléments.

Voici un exemple de tableau numéroté en PHP :

|  |
| --- |
| <?php  $tableau = array(1, 2, 3, 4, 5); |

## Callback

Un callback en JavaScript est une fonction qui est passée en argument à une autre fonction et qui est appelée ultérieurement par celle-ci. Les callbacks sont souvent utilisés pour définir une logique personnalisée qui sera exécutée à la fin d'une opération asynchrone, telles que les requêtes AJAX ou les timers.

Exemple :

function processData(data, callback) {

// traitement des données

let result = data + 1;

// appel du callback avec le résultat

callback(result);

}

function displayResult(result) {

console.log("Le résultat est:", result);

}

let data = 1;

processData(data, displayResult);

### La fonction processDate est appelé avec une data en paramètre et la fonction DisplayResult

Une fois la fonctionProcessData effectuer elle effectue un callback qui va appeler DisplayResult avec la donnée modifier.

## Exercices

#### Exercice Client Serveur

Cet exercice consistait à créer une application coté client qui compose du html avec les données envoyer par l’application serveur.

Structure site :

Une image contenant texte

Description générée automatiquement

**Client :**

Du coté client on retrouve 3 fichiers :

* **Index.html**

C’est simplement un fichier html qui importe du javascript, il contient un tableau, c’est dans ce tableau que les données vont être affiché

<!doctype html>

<html lang="fr">

<head>

<title>Lecture d'un service</title>

<meta charset="UTF-8"/>

<link rel="stylesheet" type="text/css" href="stylesheets/main.css" />

<script type="text/javascript" src="javascripts/helpers/jquery-1.10.2.min.js"></script>

<script type="text/javascript" src="javascripts/controllers/indexCtrl.js"></script>

</head>

<body>

<div id="conteneur">

<h1>Lire des données au format XML</h1>

<button type="button" id="displayTeams" >Affiche</button>

<table border= "1">

<thead>

<tr>

<td>Id</td>

<td>Club</td>

</tr>

</thead>

<tbody id="tableContent"/>

</table>

</div>

</body>

</html>

* indexCtrl.js

La méthode start (en bas du fichier)

La fonction $.getScript() fait partie de la bibliothèque jQuery et permet de charger un fichier JavaScript de manière asynchrone. Dans ce cas particulier, la fonction charge le fichier "javascripts/services/servicesHttp.js" et appelle une fonction de rappel (callback) une fois le fichier chargé.

Charger team va appeler la fonction de serviceHTTP.js celle-ci va faire une requête ajax qui va faire un get auprès du serveur, la fonction chargerTeamError ou chargerTeamSuccess va être appeler en fonction du résultat avec en paramètre les data.

      $controller = new Ctrl;

      ajouteCelluleHtml($controller -> getEquipes(), 1);

        function ajouteCelluleHtml($contenu , $id){

          $code = "";

          for ($i=0; $i < sizeof($contenu); $i++) {

            $code = $code."<tr>";

            $code =$code . "<td>";

            $code =$code .$id;

            $code =$code . "<td>";

            $code =$code. $contenu[$i];

            $code = $code."</td>";

            $code =$code . "</tr>";

            $id++;

          }

          echo $code;

\*\*

\* Méthode appelée lors du retour avec succès du résultat

\* @param {type} data

\* @param {type} text

\* @param {type} jqXHR

\*/

function chargerTeamSuccess(data, text, jqXHR) {

// appelé lorsque l'on reçoit les données de la part du PHP

var tblContent = $("#tableContent");

var txt = '';

$(data).find("equipe").each(function() {

alert($(this).find("nom").text());

txt = "<tr><td>" + $(this).find("id").text() + "</td><td>" + $(this).find("nom").text() + "</td></tr>";

$(txt).appendTo(tblContent);

})

}

/\*\*

\* Méthode appelée en cas d'erreur lors de la lecture du webservice

\* @param {type} data

\* @param {type} text

\* @param {type} jqXHR

\*/

function chargerTeamError(request, status, error) {

// appelé s'il y a une erreur lors du retour

alert("erreur : " + error + ", request: " + request + ", status: " + status);

}

/\*\*

\* Méthode "start" appelée après le chargement complet de la page

\*/

$(document).ready(function() {

var butLoad = $("#displayTeams");

$.getScript("javascripts/services/servicesHttp.js", function() {

console.log("servicesHttp.js chargé !");

});

butLoad.click(function(event) {

chargerTeam(chargerTeamSuccess, chargerTeamError);

});

});

var BASE\_URL = "../srv/serveur.php";

/\*\*

\* Fonction permettant de charger les données d'équipe.

\* @param {type} Fonction de callback lors du retour avec succès de l'appel.

\* @param {type} Fonction de callback en cas d'erreur.

\*/

function chargerTeam(successCallback, errorCallback) {

$.ajax({

type: "GET",

dataType: "xml",

url: BASE\_URL,

success: successCallback,

error: errorCallback

});

}

# Projet

## Analyse

### Introduction & explication projet

Lors de ce module nous devons mettre en place une petite application avec le cahier des charges suivant :

* Le projet devra être publié sur serveur emf-informatique.
* Le projet doit comporter du côté client au moins deux zones différentes (visiteur et utilisateur connecté).
* L’application cliente HTML (JS-AJAX-JQuery) et celle serveur PHP doivent être réalisées en Objet.
* Les données seront enregistrées dans une base de données mariaDB.

Pour ce projet j’ai décidé de créer une application permettant de partager des fichiers, on pourra naviguer au travers de rubriques et télécharger les fichiers partagés

L’utilisateur non enregistré pourra visualiser le site au travers de ses rubriques et télécharger les fichier docx ou PDF.

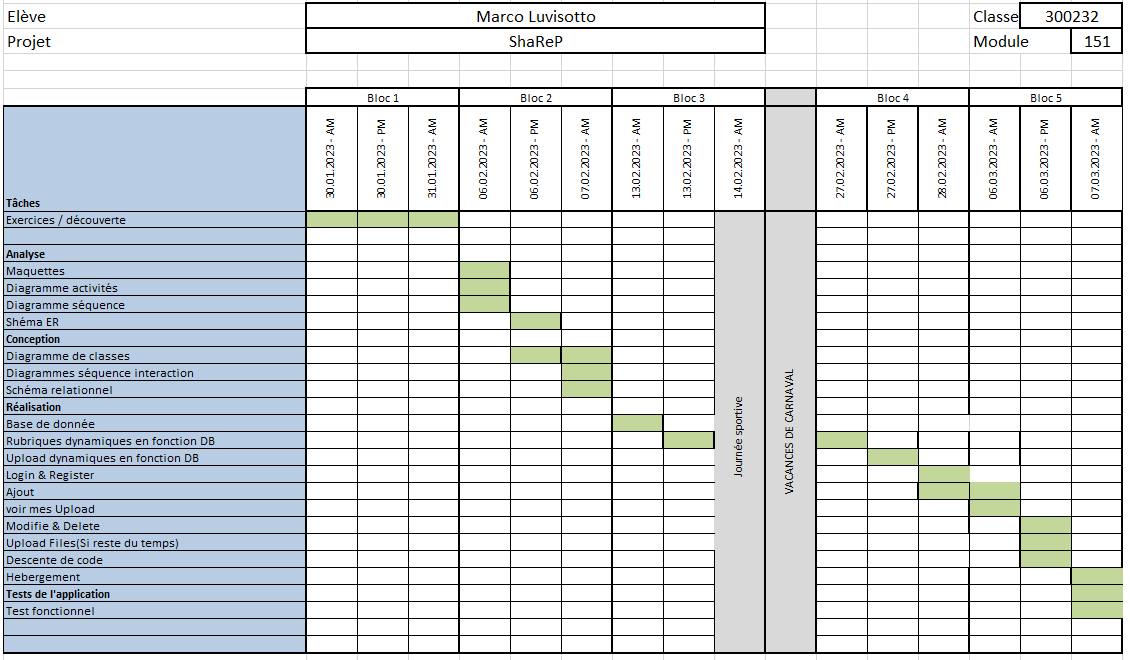
L’utilisateur loguer pourra créer des rubriques (comme un explorateur de fichier) ajouter des fichiers et modifier ses publications.

### Planification

Pour planifier notre travail nous utilisons deux plannings,

* Un fichier Excel : permet à l’apprenti de voir s’il est dans les temps.
* Un planning sur GitHub : vérifier l’avancement du projet

Le planning ci-dessous contient tous les points de l’analyse, conception et implémentation.

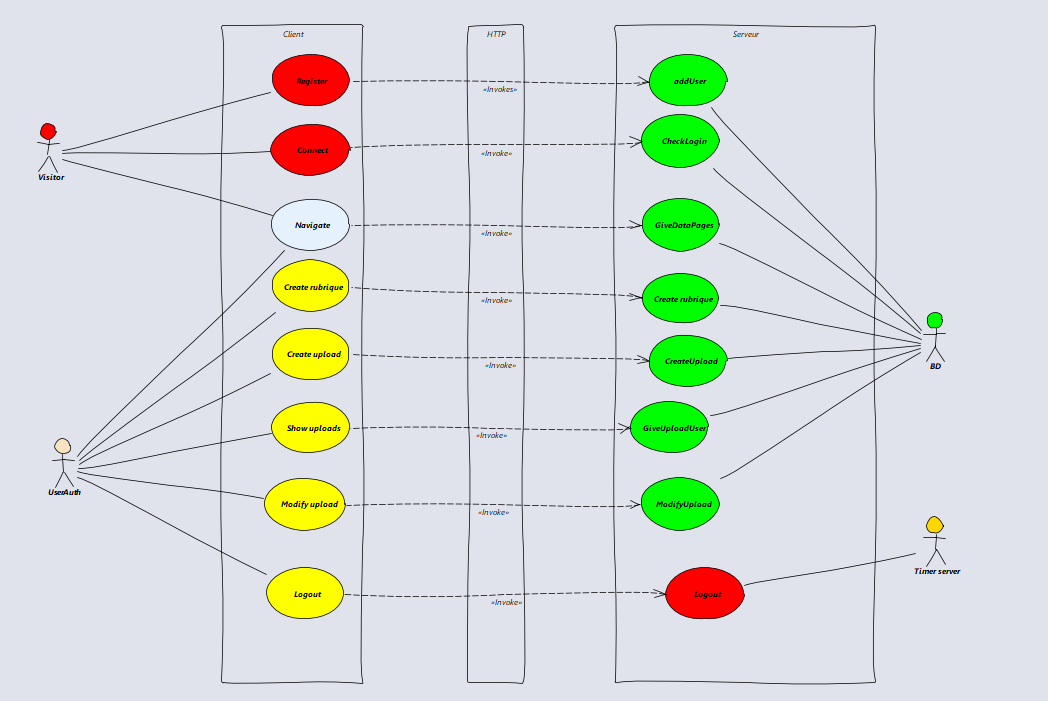


### Use case

Ce use case montre toutes les fonctionnalités disponibles de mon application.

L’user peut :

* Se connecter / s’inscrire
* Naviguer
* Si connecté
  + Créer une rubrique
  + Créer un upload
  + Vois ses publications
  + Modifier ses publications
  + Se déconnecter



### Maquette

Mon site est formé de la manière suivante.

**Non connecté :**

* Home page avec des rubriques, on peut naviguer à travers et télécharger des Uploads
* Une page de login
* Un page registrer

**Connecté**

* Un espace perso ou l’on peut voir et modifier ses publications
* Un espace d’ajout de rubrique et Upload

**Schéma**

(Schéma vectorisé donc lisible)



### Schéma ER

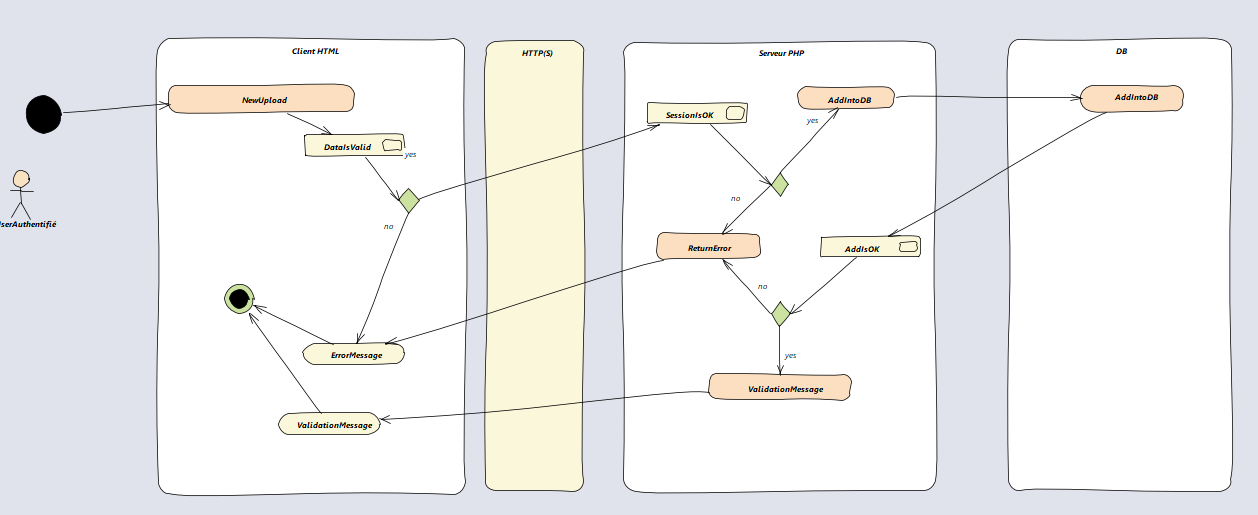
Ma base de données est très simple elle est structurée de la manière suivante.

* Des user
* Des Upload qui sont lié à des user et heading
* Des heading qui sont lié à d’autres heading



### Diagramme activités

L’activité de ce diagramme correspond à l’ajout d’un nouveau post

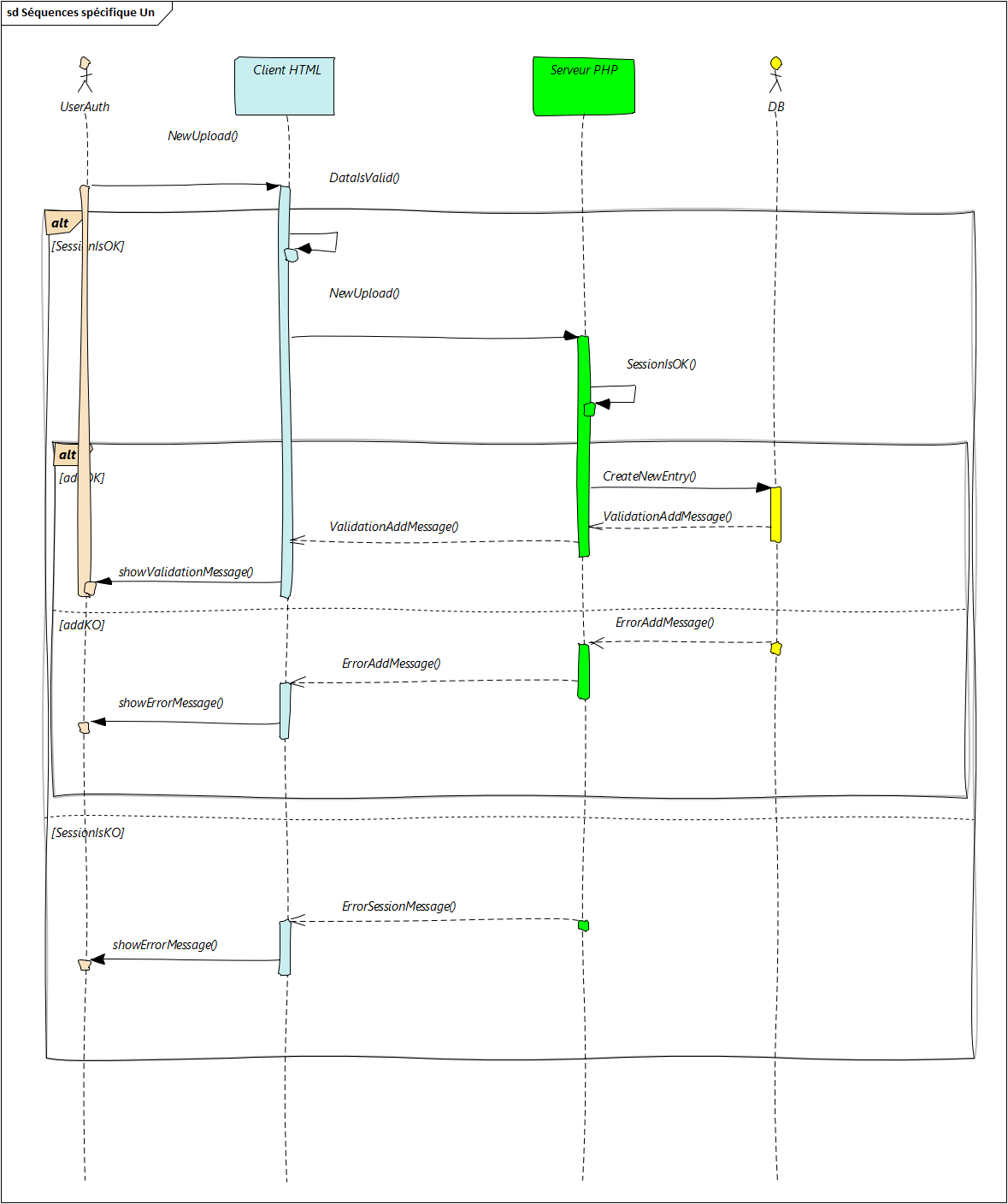


## Conception

### Diagramme séquence system

Ce diagramme de séquence represente l’ajout d’un nouveau fichier.

* Si les données sont valides le client demande au serveur d’ajouter des données
* Si la session est ok il ajoute dans la base de données le fichier si c’est ok sinon il envoie un message d’erreur si l’ajout est ok il renvoie un message, sinon il renvoie un message d’erreur.

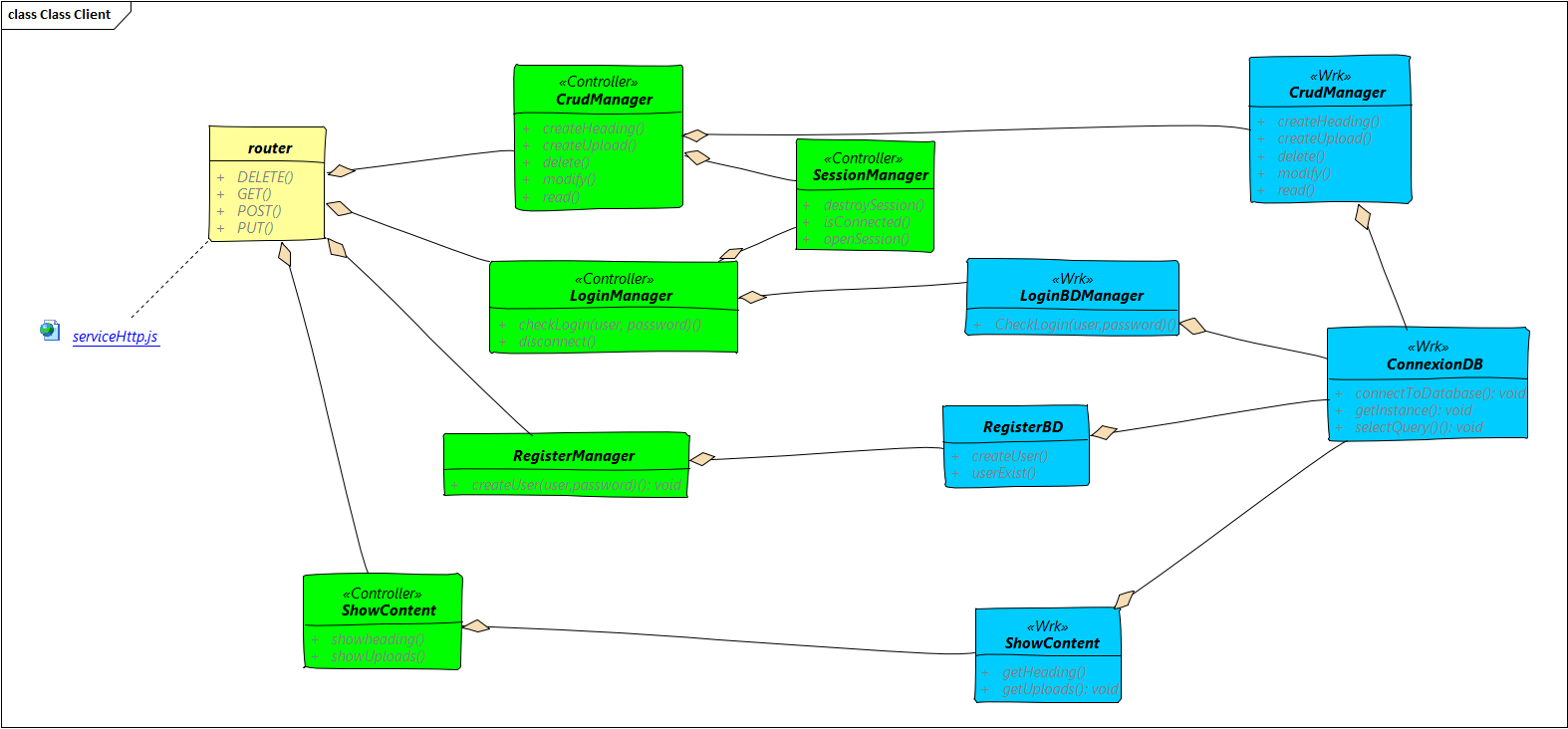


### Diagramme de classes

**Serveur :**

Ce diagramme correspond à l’architecture MVC de mon serveur.

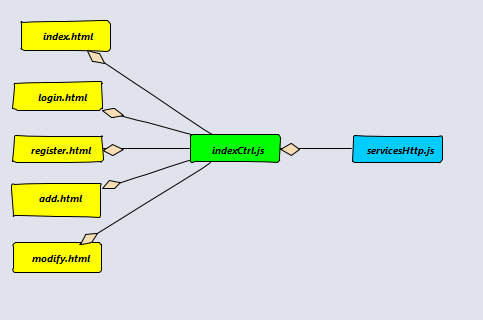
* Le router redirige les requêtes recu du client vers les bons contrôleurs
* Session manager est le fichier qui permet de gérer la session des utilisateurs
* Les autres contrôleurs sont assez explicites
* Le worker Crud permet d’éditer de supprimer et ajouter de nouvelles entrées
* Le wrk ConnexionDB effectue les requêtes



**Client :**

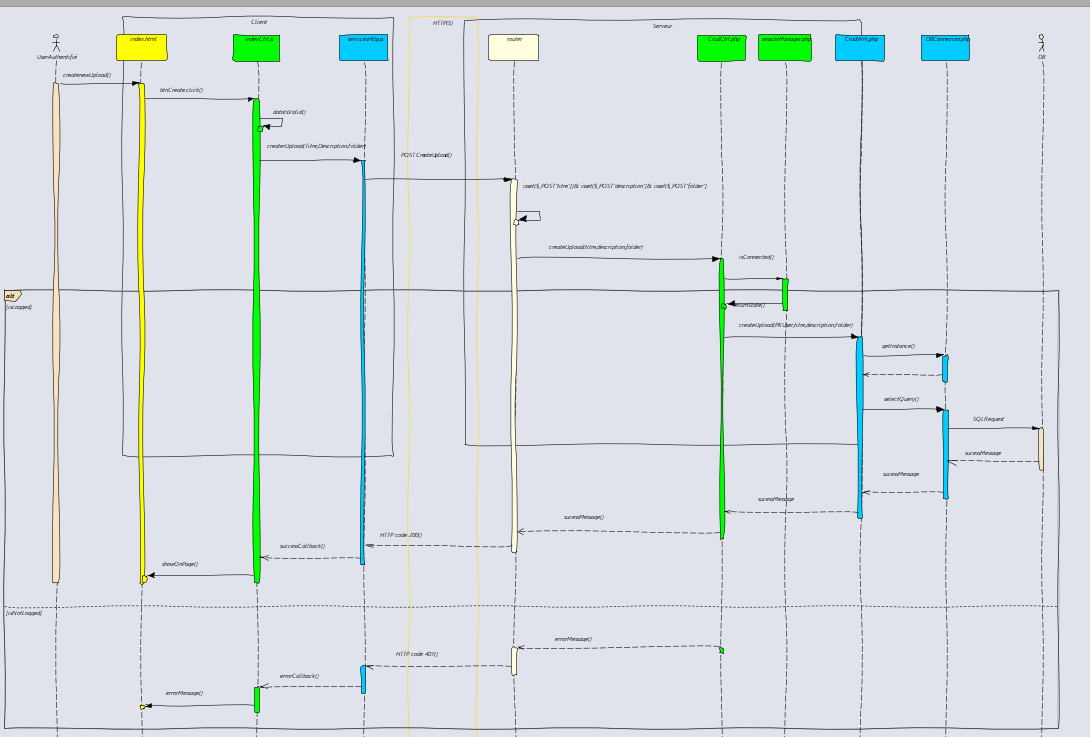
Ce modèle MVC correspond à mon client.

Il y a autant de vue car elles sont tellement différentes qu’il m’a fallu différents fichiers.



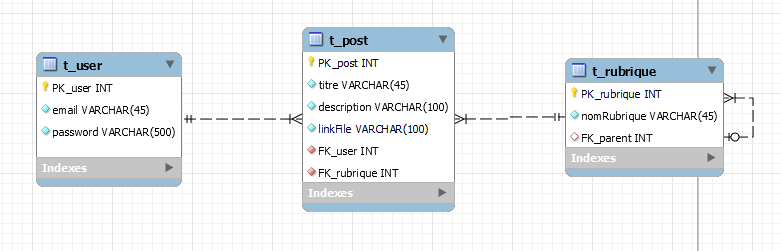
### Diagramme de séquence interaction

Ce diagramme d’interaction montre l’ajout d’un upload



### Schéma relationnel

Voici mon schéma relationnel, il est exactement basé sur mon schéma ER



## Implémentation

La documentation se trouve sous forme de phpdoc et jsdoc

### SERVEUR

Coté serveur mon code est séparé en quatre partie

* Router : reçois toutes les requêtes du client
* Beans : permet de construire les rubriques & Posts
* Controller : permet de gérer les sessions et de rediriger les requêtes vers les bonnes fonctions des worker
* Worker : permet de faire des ajout, lecture, modification et suppression des rubriques et postes

Voici tous mes fichiers coté serveur

Une image contenant texte

Description générée automatiquement

Par rapport à ma conception on retrouve juste un fichier de config pour la base de données « configConnedxion.php » et dans service « HttpErrorHandler.php » qui permet de retourner des messages d’erreur ainsi qu’un code d’erreur.

**Mot de passe inscription et connexion**

Pour hasher un mot de passe j’ai utilisé la fonction qui suit

$password = password\_hash($password, PASSWORD\_DEFAULT);

Pour vérifier le mot de passe

password\_verify($password, $user['passwordUser']

#### Difficulté rencontrée

**RETOUR**

Pour revenir en arrière lorsque l’on parcourt nos rubriques on doit pouvoir revenir en arrière (rubrique précédente), pour ce faire je sauvegarde dans la session la dernière rubrique visitée et j’appelle mes rubriques avec cette valeur lors d’un retour en arrière.

public function getLasRubrique()

{

array\_pop($\_SESSION['lastRubrique']);

$lastValue = end($\_SESSION['lastRubrique']);

return $lastValue;

}

public function setLastRubrique($rubrique)

{

if (!isset($\_SESSION['lastRubrique'])) {

$\_SESSION['lastRubrique'] = array();

$arr = $\_SESSION['lastRubrique'];

$arr[] = "home";

$\_SESSION['lastRubrique'] = $arr;

}

$arr = $\_SESSION['lastRubrique'];

$arr[] = $rubrique;

$\_SESSION['lastRubrique'] = $arr;

}

**TIMEOUT**

Pour set le timeout j’ai dû utiliser la fonction time de PHP car la session par défaut de 30 min ne fonctionnait pas

// vérification de la variable de session

if (isset($\_SESSION['LAST\_ACTIVITY']) && (time() - $\_SESSION['LAST\_ACTIVITY'] > 1800)) {

// last request was more than 30 minutes ago

session\_unset(); // unset $\_SESSION variable for the run-time

session\_destroy(); // destroy session data in storage

return false;

}

$\_SESSION['LAST\_ACTIVITY'] = time(); // update last activity time stamp

if (isset($\_SESSION['user'])) {

return true;

} else {

return false;

}

**AJOUT FICHIER BINAIRE**

Pour ajouter un fichier binaire il suffit de le décodé (car encodé en base 64) puis de l’ajouter dans la base de données.

public function ajoutPost($titre, $description, $idRubrique, $file)

{

$base64 = base64\_decode($file);

$session = new SessionManager();

if (!$session->isLoggedIn()) {

$error = new HttpErrorHandler();

return $error->setHttpError(403, "Vous netes pas connecté");

}

try {

return $this->crudDB->ajoutNewPost(strip\_tags($titre), strip\_tags($description), strip\_tags($idRubrique), $base64);

} catch (Exception $e) {

print "Erreur !: " . $e->getMessage() . "<br/>";

die();

}

}

**Problème lié injection html**

Htmlspécialchars ne fonctionnant pas avec xml j’ai utilisé la fonction

strip\_tags

### CLIENT

Mon coté client est séparé en trois partie

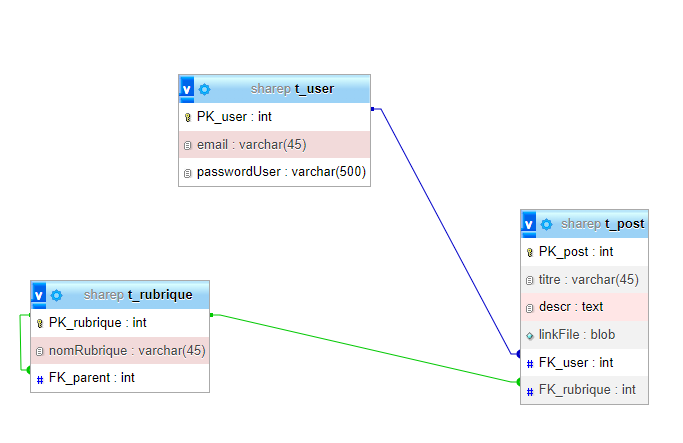
* Vue : fichier html de mon app
* Javascript : permet de controller les actions de l’utilisateur
* Stylsheets : le style de mon app

Chaque page html contient un contrôleur pour chaque vue

### DB

Ma base de données à subit un changement, le champ description est dorénavant égal à descr, car la description est une variable native à SQL. Le champ file à été modifier en long Blob pour stocker des fichiers binaires.

Voici un aperçu de ma base de données dans phpMyAdmin.



#### Difficulté rencontrée

Lors de l’ajout de fichier j’ai effectué des requêtes de plusieurs mégas, par défaut localhost à une longueur de requête défini à un méga maximum, les commandes ci-dessous permettent d’afficher la valeur maximale d’une requête ainsi que comment la modifier

show variables like 'max\_allowed\_packet'

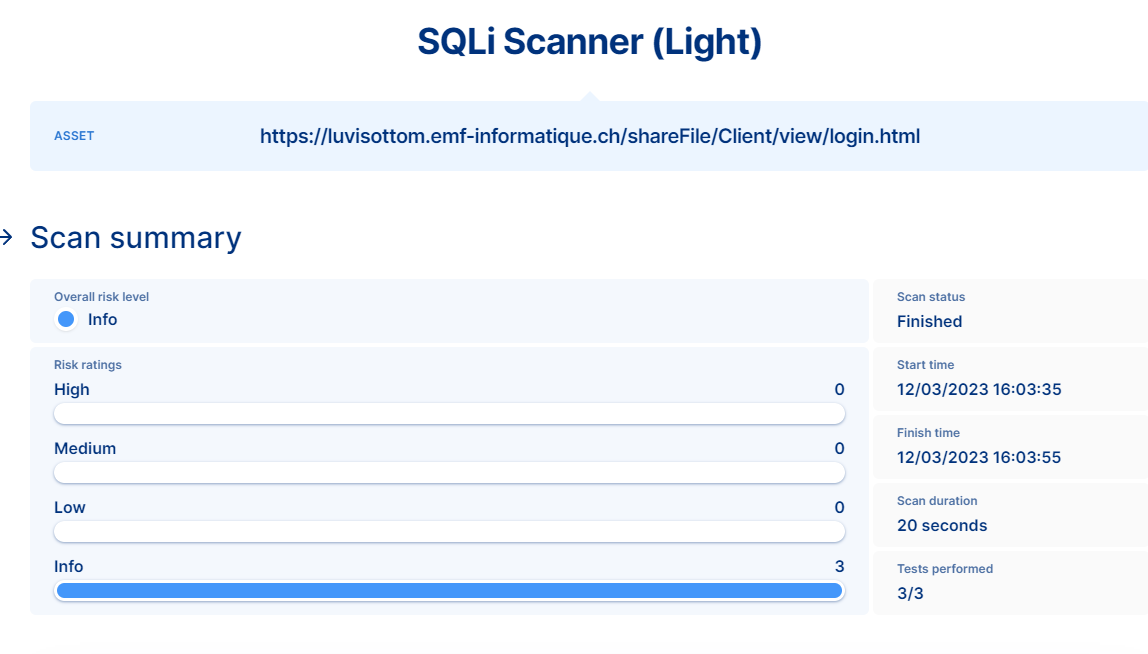
set global max\_allowed\_packet=33554432

# Test

## Injection

|  |  |  |
| --- | --- | --- |
| Test | Résultat attendu | Résultat obtenu |
| Injection html dans en ajoutant un titre à <h1>test</h1> upload | Titre <h1>test</h1> | Titre <h1>test</h1> |
| Injection js <script>alert(‘test’)</script> | Resultat attendu rien ne se passe | Resultat attendu rien ne se passe |
| Injection formulaire connexion sql ‘bob’ or 1=1 -- | Mot de passe incorrect | Mot de passe incorrect |

J’ai trouvé un site qui vérifiait si un site était vulnérable à l’injection SQL.



## Test fonctionnel

Une image contenant table

Description générée automatiquement

# Synthèse

Le projet réalisé est selon moi fonctionnel, même si beaucoup d’axe d’amélioration sont à apporter.

* Gestion erreur longueur / format contenu formulaire
* Récupération mot de passe
* La vue html n’est pas très dynamique d’un serveur à l’autre, beaucoup d’url à changer.
* Les rubriques vides ne doivent pas apparaitre (recursive request)
* Ergonomie
* Les fichiers doivent être chargé uniquement lors du téléchargement
* …

# Conclusion

J’ai beaucoup aimé ce module, j’ai choisi un projet ambitieux, ne sachant pas que les interruptions et la conception prendrait autant de temps.

Pour ce projet uniquement la conception du modèle mvc2 m’a été utiles.

J’ai beaucoup appris lors de ce module et je suis fière de ce que j’ai produit