|  |
| --- |
| 151 – Intégrer des bases de données dans des applications Web |

Documentation du projet Mangabis



Alexis Stella

Version 1 du 13 mars 2021

Module du 1er février 2021 au 12.03.2021

Table des matières

[1 Introduction du module 3](#_Toc66623564)

[2 Projet Mangabis 4](#_Toc66623565)

[2.1 Analyse 4](#_Toc66623566)

[2.1.1 Introduction 4](#_Toc66623567)

[2.1.2 Maquettes 4](#_Toc66623568)

[2.1.3 Use Case Global 7](#_Toc66623569)

[2.1.3.1 Descriptions des acteurs globaux 8](#_Toc66623570)

[2.1.3.2 Descriptions des cas d’utilisations globales 8](#_Toc66623571)

[2.1.4 Diagramme d’activités 9](#_Toc66623572)

[2.1.5 Diagramme de séquence système 10](#_Toc66623573)

[2.1.6 Schéma ER base de données 10](#_Toc66623574)

[2.1.7 Planning 11](#_Toc66623575)

[2.2 Conception 12](#_Toc66623576)

[2.2.1 Diagrammes de classes 12](#_Toc66623577)

[2.2.2 Diagramme de séquence d’interaction 13](#_Toc66623578)

[2.2.3 Schéma relationnel base de données 14](#_Toc66623579)

[2.2.4 Pas de tests 15](#_Toc66623580)

[2.3 Implémentation 16](#_Toc66623581)

[2.3.1 Descente de code 16](#_Toc66623582)

[2.3.2 Tests fonctionnels 24](#_Toc66623583)

[2.3.3 Hébergement 26](#_Toc66623584)

[2.4 Synthèse 29](#_Toc66623585)

[2.4.1 Différence timing 29](#_Toc66623586)

[2.4.2 Conclusion du projet 29](#_Toc66623587)

[3 Conclusion 31](#_Toc66623588)

[3.1 Ce que j’ai aimé 31](#_Toc66623589)

[3.2 Ce que j’ai moins aimé 31](#_Toc66623590)

[4 Sources 32](#_Toc66623591)

# Introduction du module

Dans le module 151, nous allons apprendre à 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. Nous allons aussi apprendre à identifier les informations importantes de protection et de sécurité en tenant compte de la protection des données, et définir les mesures. Nous allons apprendre à 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.

Pour terminer, nous allons apprendre à mettre en œuvre les souhaits de modifications conformément au déroulement prescrit des modifications et définir et exécuter la procédure de test et de remise, la documenter dans un procès-verbal de tests. Si nécessaire, entreprendre les corrections.

# Projet Mangabis

## Analyse

### Introduction

Pour introduire notre projet, nous allons tout d’abord vous expliquez dans les grandes lignes en quoi il consiste. L’application web réalisée dans ce projet est un site d’annonce de mangas d’occasion. Dans le même style que Anibis, nous aurons sur la page d’accueil, la liste des différentes annonces réalisées par les utilisateurs inscrit. Ces annonces auront une image, un titre, un prix, une description, les informations pour contacter le vendeur et un état.

* Image 🡺 L’image du manga à vendre
* Titre 🡺 Le titre de l’annonce (nom du manga, nombre de livre, …)
* Prix 🡺 Le prix de l’annonce
* Description 🡺 La description de l’annonce
* Etat 🡺 L’état du ou des mangas
* Information du vendeur : téléphone, adresse email, nom, prénom…

Nous pourrons aussi sur cette application web, se créer un comte et se connecter pour pouvoir ajouter, modifier ou supprimer nos annonces. Si nous souhaitons nous déconnecter, un bouton sera à disposition. Sur cette applications web, les utilisateurs auront alors un rôle : admin ou default-user.

Pour terminer, l’admin pourra supprimer une annonce d’un utilisateur si celle-ci n’est pas en règle et si nous avons le temps, nous implémenterons que les admins pourront ajouter, supprimer et modifier des utilisateurs.

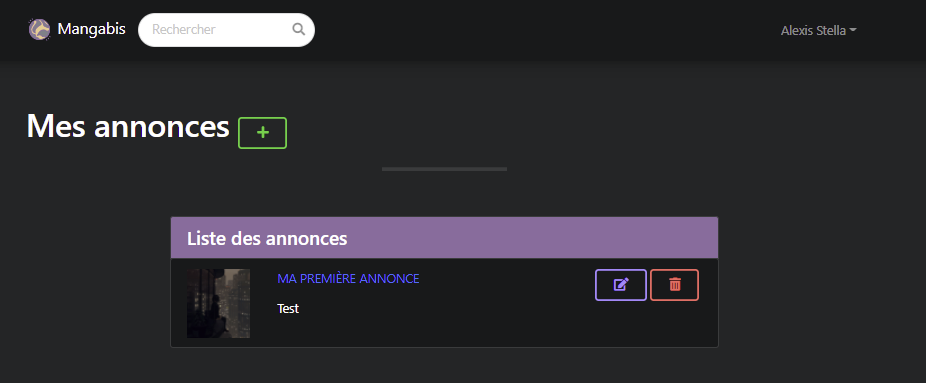
### Maquettes

Voici les différentes maquettes de l’application web que nous allons réaliser lors de ce projet.



Dans l’implémentation de l’application web, il y a eu quelques changements par rapport à la maquette. En effet, la page de gestion des annonces de l’utilisateur a eu quelques changements déjà au niveau de l’url qui passe de « /<utilisateur>/articles » à « /annonces ». Il y a eu aussi des changements visuels, au lieu d’avoir la liste à gauche et à droite la modification/ajout, tout se passe par des modals accessible via des boutons.

Voici le résultat :

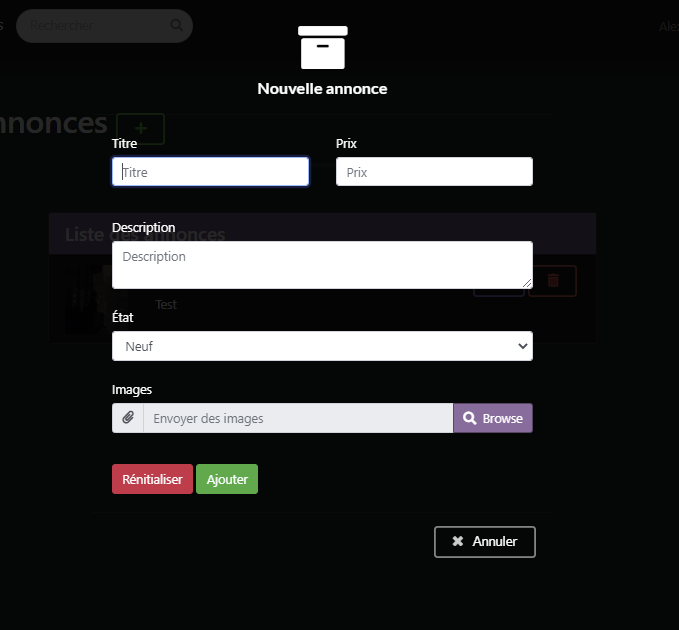


Modification d’une annonce

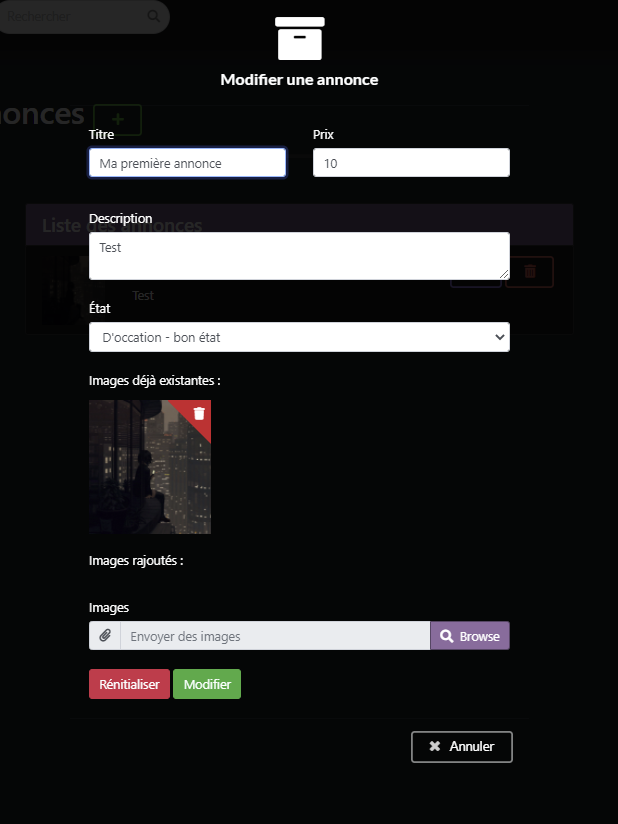
Suppression d’une annonce

Ajout d’une annonce

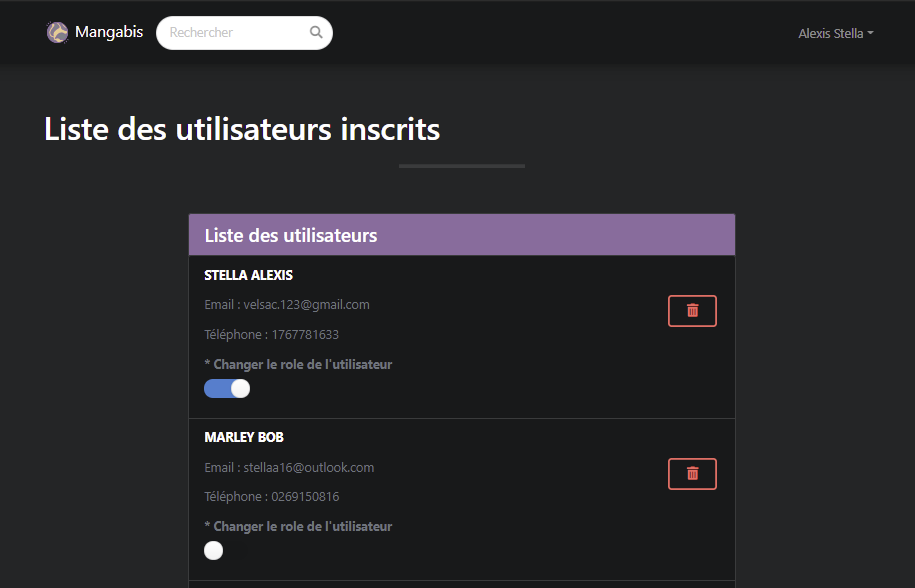
Quand nous cliquons sur l’ajout, nous avons un modal qui apparaît :



De même pour la modification :



La page de modification du profil de l’utilisateur à aussi eu une petite modification. C’était une petite erreur de ma part mais l’utilisateur ne peut pas changer son rôle. Pour la page de gestion de tous les utilisateurs de la base de données, page qui est seulement accessible par un administrateur, des changements visuels ont été fait, voici le résultat :



### Use Case Global

Voici le « Use Case » global de notre application. Cela permet d’avoir une vue globale de notre application :



#### Descriptions des acteurs globaux

* « Visiteur »

Le visiteur est la personne qui n’est pas encore connecté. Il pourra consulter la liste des annonces ajoutés par les utilisateurs, voir l’annonce en détail (dans une nouvelle page), s’inscrire ou se connecter s’il dispose déjà d’un compte.

* « Utilisateur »

L’utilisateur est la personne pouvant se connecter à l’application web. Il dispose d’un rôle normal qui permet de créer une annonce, modifier ou supprimer ces annonces ou encore se déconnecté. Comme le visiteur, il pourra consulter la liste des annonces et les afficher en détails (dans une nouvelle page). Pour terminer, il pourra s’il le souhaite se déconnecter de l’application web.

* « Administrateur »

L’administrateur dispose des mêmes permissions que l’utilisateur. Il peut supprimer des utilisateurs ou des annonces sans que ce soit les siennes. Cela permet d’éviter que des utilisateurs ajoutent des annonces néfastes pour le site ou qui ne respectent pas le règlement

#### Descriptions des cas d’utilisations globales

* « Connexion »

Dans le cas où le visiteur possède déjà un compte, alors il pourra alors se connecter. Les informations seront alors envoyées vers le serveur afin qu’il les contrôle. Si les informations (email, mot de passe) sont correctes, alors il sera connecté et pourra jouir des fonctionnalités disponibles en tant qu’utilisateur connecté. Si la connexion n’est pas correcte, un message d’erreur lui sera envoyé afin qu’il entre à nouveaux ses informations de connexion.

* « Inscription »

Lorsqu’un visiteur se rend sur l’application web, il pourra comme expliqué dans le point précédent, voir les différentes annonces. Il aura le choix entre se connecter s’il possède déjà un compte, sinon il pourra s’inscrire. Lorsqu’il s’inscrit, il devra mettre son nom, prénom, email, numéro de téléphone et son mot de passe. Ces données d’inscription seront alors envoyées du côté du serveur pour qu’il les ajoute dans la base de données. Dans le cas où l’email de l’utilisateur serait déjà utilisé alors, un message d’erreur sera retourné, sinon, il sera redirigé vers la page de connexion et pourra cette fois-ci se connecter pour accéder à l’application web.

* « Déconnexion »

Lorsque l’utilisateur a fini d’utiliser l’application web, il peut alors se déconnecter. Lorsqu’un utilisateur se déconnecte, il est redirigé vers la page d’accueil. Il ne pourra plus ajouter/supprimer/modifier ces articles.

* « Visualisation des articles »

Quand un visiteur arrive sur le site, il pourra sur la page d’accueil, voir toutes les annonces que les différents utilisateurs ont ajouté. Il pourra trier ces annonces en fonction de leur état, leur titre ou encore de leur prix.

* « Modifier article »

Lorsqu’un utilisateur est connecté, il pourra s’il le souhaite, modifier des annonces qu’il aura précédemment crée.

* « Supprimer article »

Lorsqu’un utilisateur est connecté avec un rôle par défaut ou d’administrateur, il pourra supprimer des annonces qu’il aura précédemment crée. Dans le cas de l’administrateur, il pourra tout simplement supprimer n’importe quelle annonce qu’il y a sur le site.

* « Ajout article »

Lorsqu’un utilisateur est connecté, il pourra ajouter une ou plusieurs annonces, de même pour l’administrateur qui peut avec ces propres annonces.

* « Supprimer utilisateur »

Lorsqu’un utilisateur avec un rôle d’administrateur se connectera, il pourra s’il le souhaite supprimer un ou plusieurs utilisateurs.

* « Modifier utilisateur »

Quand un administrateur est connecté, il pourra modifier les utilisateurs inscrit sur le site en changeant seulement leur rôle. Si au contraire la personne connectée est un utilisateur lambda, il ne pourra que changer son nom, prénom, email, mdp ou encore son numéro de téléphone.

### Diagramme d’activités

Voici le diagramme d’activité représentant les différentes actions effectuées lorsqu’un utilisateur ajoute une nouvelle annonce.



Comme nous pouvons le voir, lorsque l’utilisateur fait l’action d’ajouter une annonce, une vérification si l’annonce n’est pas nulle va s’effectuer. Si celle-ci n’est pas nulle alors elle sera envoyée au serveur via une requête POST sinon un message d’erreur sera affiché.

Le serveur va ensuite vérifier si les paramètres attendus sont bien transmis, si oui, le serveur va tester si l’utilisateur est connecté sinon, un message d’erreur sera affiché. Dans le cas où l’utilisateur n’est pas connecté, un message d’erreur sera aussi affiché, sinon la requête SQL cherchant une annonce de même titre sera effectuée à la base de données MariaDB. Si une annonce de même titre existe déjà, une erreur sera retournée sinon une autre requête sera envoyée à la base de données permettant d’ajouter l’annonce.

Pour terminer, si l’annonce a correctement été ajouté, un message de succès sera envoyé sinon un message d’erreur.

### Diagramme de séquence système

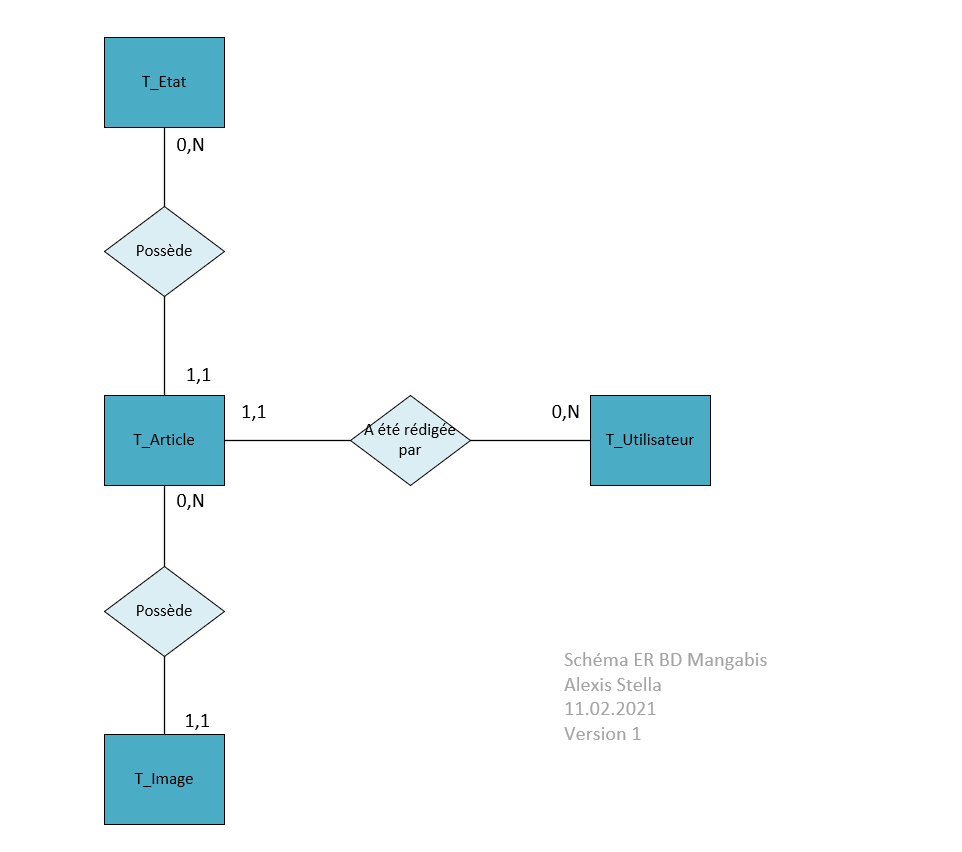
Voici le diagramme de séquence système représentant les actions effectuées entre les différents acteurs lorsqu’un utilisateur ajoute une nouvelle annonce.



### Schéma ER base de données

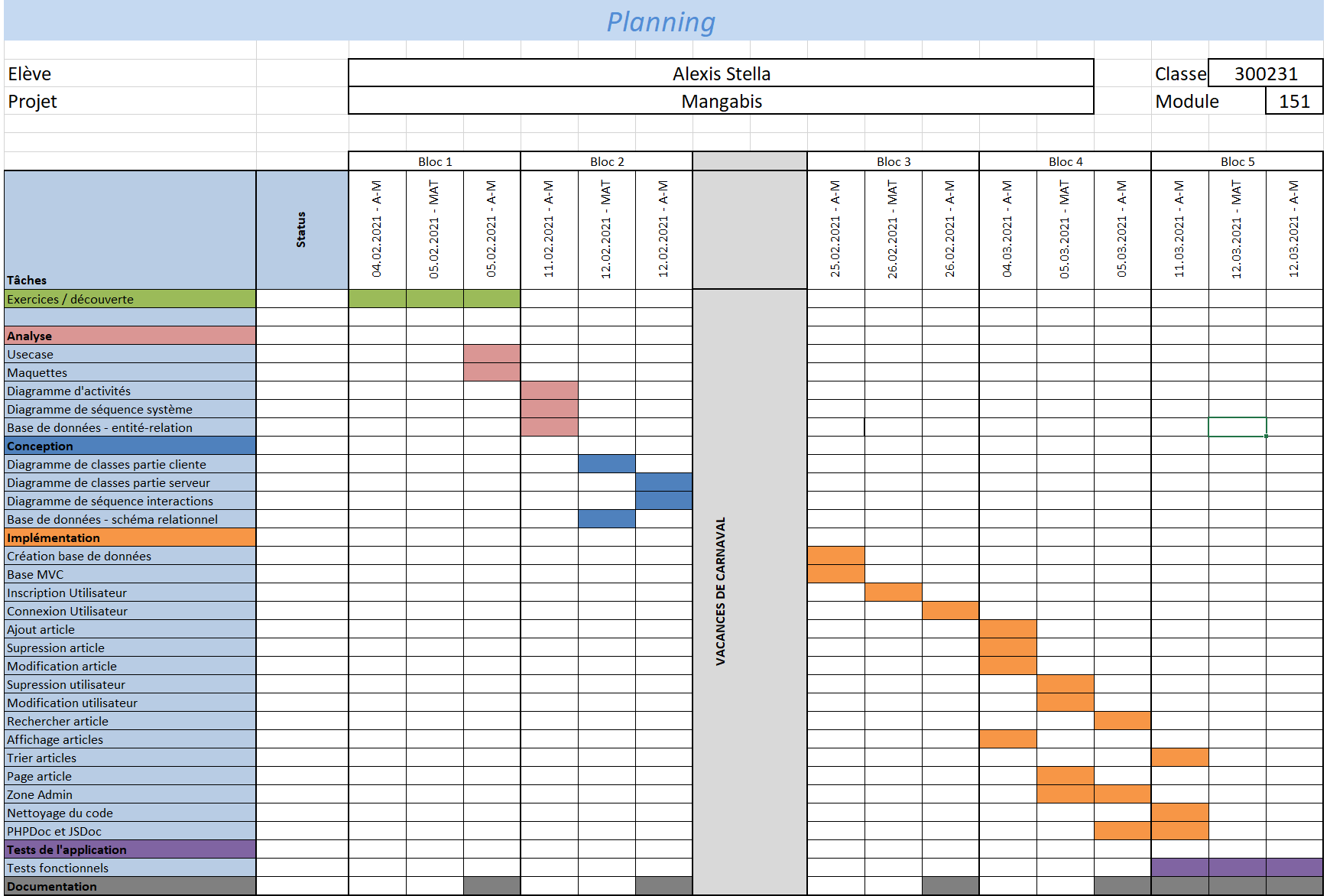
Voici le schéma entité relation de la base de données de notre projet. Comme nous pouvons le voir, une table utilisateur contiendra tous les utilisateurs avec leur mot de passe, nom, prénom, etc. Un utilisateur pourra avoir aucun ou plusieurs articles qui eux possèderont aucune ou plusieurs images. Une annonce aura un état qui représentera l’état de l’objet présenté dans l’annonce.

(La table t\_article dans ce schéma a été remplacée par t\_annonce qui était plus logique)



### Planning

Voici le planning réalisé pour le projet Mangabis. Nous pouvons voir une prévision des choses à faire durant ces 5 semaines. J’ai aussi mis une colonne pendant les vacances car je pense que j’avancerai un peu sur mon projet. Nous pouvons dans le chapitre « Différence timing », voir les différences avec ce que j’avais prévu et ce qui s’est vraiment passé.



## Conception

### Diagrammes de classes

Voici le diagramme de classes de la partie cliente de l’application web. Ce premier diagramme est pour une version dite normale. C’est-à-dire quelle fonctionne avec du jQuery et du javascript lambda.



Comme nous pouvons le voir, un « index.html » servira de vu principale. Nous irons en fonction de l’url, changer le contenue de l’« index.html ». Pour les contrôleurs. Nous avons :

* « indexCtrl.js » contiendra un switch case qui changera de vue en fonction de l’url et aussi les différentes fonctions qui constitue la page d’accueil.
* « profilCtrl.js » contiendra toutes les méthodes pour la page de gestion de profil de l’utilisateur connecté.
* « listeUtilisateursCtrl.js » contiendra toutes les méthodes de la page de gestion de tous les utilisateurs de l’application. Cette page est seulement accessible pas un administrateur.
* « annonceCtrl.js » contiendra toutes les méthodes pour la page d’information d’une annonce.
* « listeAnnoncesCtrl.js » contiendra toutes les méthodes de la page de gestion de toutes les annonces de l’application. Cette page est seulement accessible pas un administrateur.
* « gestionAnnoncesCtrl.js » contiendra toutes les méthodes de la page de gestion des annonces des utilisateurs connectés.
* « registerCtrl.js » contiendra toutes les méthodes de la page d’inscription de l’application web.
* « loginCtrl.js » contiendra toutes les méthodes de la page de connexion de l’application web.
* « navbarCtrl.js » contiendra toutes les méthodes de la barre de navigation de l’application web.

Pour terminer, serviceHttp.js permettra de communiquer avec le serveur qui enverra des informations sous forme de json.

Voici maintenant le diagramme de classes de la partie serveur :



Comme nous pouvons le voir, le serveur fonctionne avec le modèle MVC. Il disposera d’un « main.php » qui sera globalement un gros switch case qui, en fonction de la requête, appellera des méthodes du contrôleur. Le contrôleur appellera les méthodes du Wrk qui lui, appellera les méthodes des sous-workers. Les sous-workers pourront appeler la base de données via le singleton Connexion. Nous avons choisi de créer un singleton pour éviter d’avoir trop de connexion à la base de données ce qui pourrait bloquer celle-ci.

Pour terminer, nous pouvons aussi voir les différents beans de l’application web partie serveur.

### Diagramme de séquence d’interaction

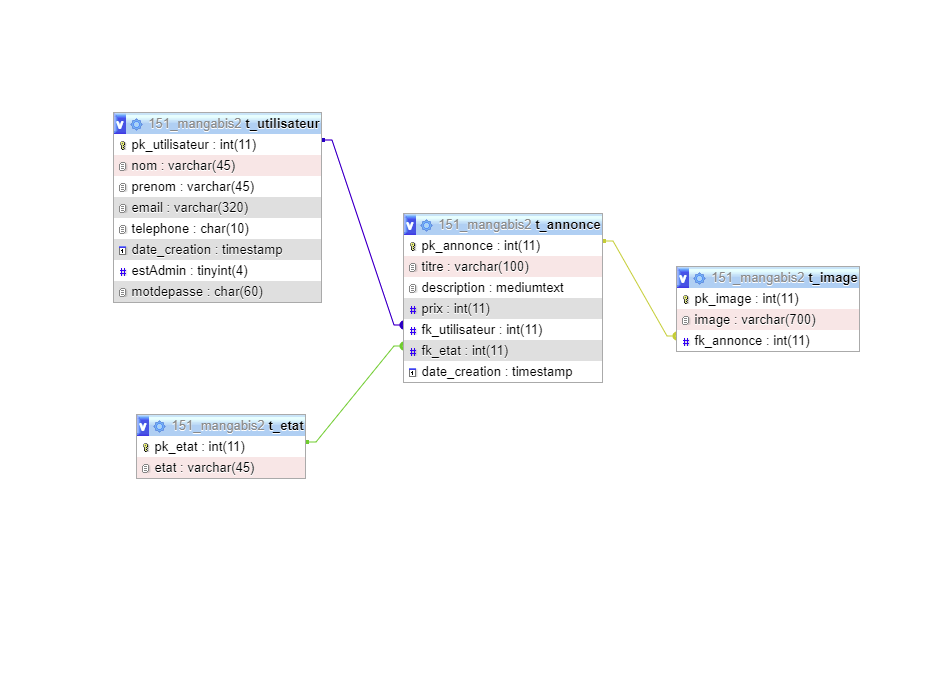
Voici le diagramme de séquence d’interaction de l’application web du projet Mangabis.



### Schéma relationnel base de données

Voici le schéma relationnel de la base de données du projet Mangabis. Comme nous pouvons le voir, cette base de données contiendra 4 tables :

* t\_etat 🡺 Contiendra les états des mangas dans l’annonce : Neuf, d’occasion – bonne état, …
* t\_utilisateur 🡺 Contient les utilisateurs avec leur nom, prénom, téléphone, email, …
* t\_image 🡺 Contient les images que les utilisateurs on mit dans leurs annonces.
* t\_annonce 🡺 Contient les annonces que les utilisateurs ont ajoutés.



### Pas de tests

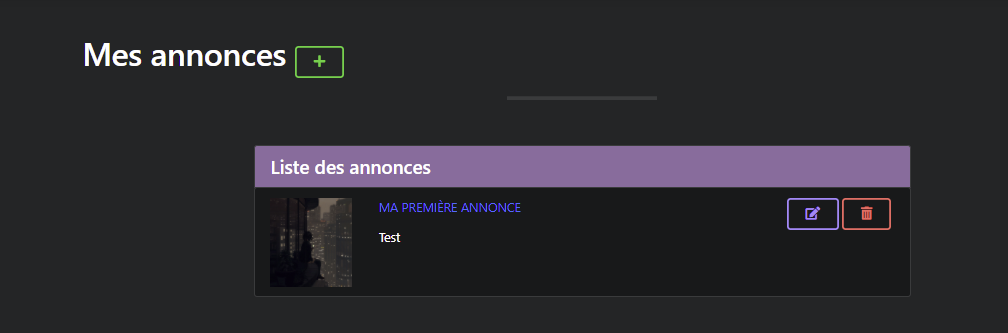
Voici les pas de test de l’application web du projet Mangabis. Ces pas de tests seront effectués afin d’assurer que cette application est bien fonctionnelle.

|  |  |  |
| --- | --- | --- |
| N° | Description | Résultats attendus |
| 1 | Un visiteur peut visualiser toutes les annonces. | Le visiteur a bien accès aux annonces via la page d’accueil. |
| 2 | Un visiteur peut s’inscrire via son nom, prénom, email, numéro de téléphone et son mot de passe. | Le visiteur peut s’inscrire ce qui impacte la base de données. Il peut alors, se connecter. |
| 3 | Un visiteur disposant d’un compte peut se connecter si ces identifiants sont corrects. | Le visiteur peut se connecter via son email et son mot de passe. Il a alors accès à ces annonces et son profil. |
| 4 | Un utilisateur connecté peut ajouter/supprimer/modifier des annonces. | L’utilisateur a bien accès aux fonctionnalités d’ajout, suppression et de modification d’annonces. |
| 5 | Un utilisateur connecté peut modifier son profil (nom, prénom, téléphone, email, mot de passe, etc). | L’utilisateur a bien accès aux modifications de son profil. |
| 6 | Un utilisateur avec un rôle d’admin peut supprimer n’importe quelle annonce de n’importe quel utilisateur. | L’utilisateur avec un rôle d’admin a bien accès à la suppression d’annonces d’utilisateurs. |
| 7 | Un utilisateur avec un rôle d’admin peut supprimer un utilisateur ou modifier son rôle. | L’utilisateur avec un rôle d’admin a bien accès à la suppression d’utilisateurs et de modification de rôle de ceux-ci. |
| 8 | Un utilisateur peut se déconnecter ce qui termine sa session. | L’utilisateur peut bien se déconnecter, sa session est correctement terminée. Il est ensuite redirigé vers la page d’accueil |
| 9 | Un visiteur essaie de se rendre sur un profil utilisateur ou la gestion des annonces d’un utilisateur. | Un message d’erreur est affiché. |
| 10 | Un visiteur essaie de se connecter avec un mauvais email et mot de passe. | Un message d’erreur est bien affiché et le visiteur reste non-connecté. |
| 11 | Un visiteur essaie de s’inscrire mais n’accepte pas le captcha. | Un message d’erreur est affiché et le visiteur n’est pas inscrit. |
| 12 | Un visiteur essaie de s’inscrire mais n’a pas rempli tous les champs. | Un message d’erreur est affiché et le visiteur n’est pas inscrit. |
| 13 | Les injections SQL sont gérées dans chacun des champs que les utilisateurs et les visiteurs doivent entrer. | Les injections SQL sont bien gérées et impossible à effectuer |

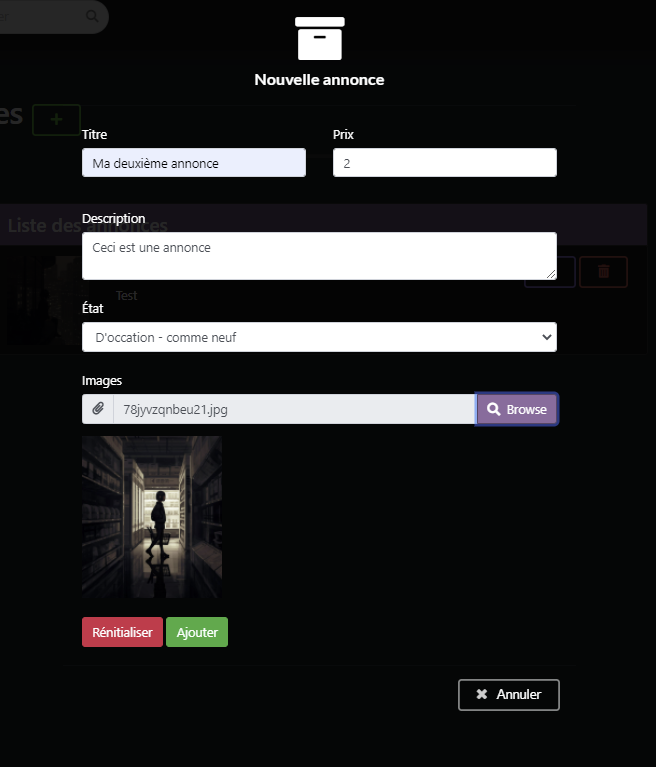
## Implémentation

### Descente de code

Nous allons maintenant voir une descente de code pour l’ajout d’une annonce.



Voici un exemple de remplissage de champs pour l’ajout d’une nouvelle annonce.



1. Client : index.html avec la vue annonces.html ajoutée et son component addAnnonceModal.html.

Quand quelqu’un clique sur le bouton « ajouter » du modal d’ajout d’annonce, si le formulaire est correctement rempli, la méthode actionAddAnnonce sera appelé à l’aide du code HTML suivant :

Si le formulaire est correctement rempli, la méthode actionAddAnnonce est appelée.

<!-- Modal -->

<div class="ui tiny basic modal" id="modalAddAnnonce">

<div class="ui icon header">

<i class="archive icon"></i>

Nouvelle annonce

</div>

<div class="scrolling content">

<form action="javascript:actionAddAnnonce()" enctype="multipart/form-data">

<!-- créer un groupe -->

<div class="form-group row">

<div class="col-sm-6 mb-3 mb-sm-0">

<label for="titre">Titre</label>

<!-- champ texte qui devra contenir le titre de l'annonce -->

<input type="text" class="form-control" id="titre" aria-describedby="emailHelp"

placeholder="Titre" maxlength="100" required/>

</div>

<div class="col-sm-6 mb-3 mb-sm-0">

<!-- créer un groupe -->

<div class="form-group mb-4">

<label for="prix">Prix</label>

<!-- champ nombre qui devra contenir le prix de l'annonce -->

<input type="number" id="prix" class="form-control" placeholder="Prix" required/>

</div>

</div>

</div>

<!-- créer un groupe -->

<div class="form-group mb-4">

<label for="description">Description</label>

<!-- champ texte qui devra contenir la description de l'annonce -->

<textarea id="description" class="form-control" maxlength="500" required

placeholder="Description"></textarea>

</div> <!-- créer un groupe -->

<div class="form-group mb-4">

<label for="etat">État</label>

<!-- champ texte qui devra contenir le mot de passe de l'utilisateur -->

<select class="form-control" id="etat">

<option value="1">Neuf</option>

<option value="2">D'occation - comme neuf</option>

<option value="3">D'occation - bon état</option>

<option value="3">D'occation - assez bon état</option>

</select>

</div>

<!-- créer un groupe -->

<div class="form-group mb-4">

<input type="file"

name="images"

id="images"

hidden

accept="image/x-png,image/gif,image/jpeg"

multiple

class="file"

/>

<label for="browseImages">Images</label>

<div class="input-group mb-3">

<div class="input-group-prepend">

<span class="input-group-text" id="basic-addon1"><i class="fas fa-paperclip"></i></span>

</div>

<input type="text"

class="form-control"

disabled

placeholder="Envoyer des images"

aria-label="Envoyer des images"

aria-describedby="basic-addon1"

/>

<div class="input-group-append">

<button class="browse btn btn-primary"

id="browseImages"

type="button">

<i class="fas fa-search"></i>&nbsp;

Browse

</button>

</div>

</div>

<div class="row" id="imageListAddAnnonce"></div>

</div>

<button class="btn btn-danger" type="reset">Rénitialiser</button>

<button type="submit" class="btn btn-success">Ajouter</button>

</form>

</div>

<div class="actions">

<div class="ui red basic cancel inverted button">

<i class="remove icon"></i>

Annuler

</div>

</div>

</div>

1. Client : gestionAnnonceCtrl.js

Quand la méthode « actionAddAnnonce » est appelée, elle va tout d’abord récupérer les informations des champs et les stocker dans différentes variables. Elle va ensuite créer un objet annonce avec ces informations Elle va après ça récupérer les images qui ont été choisies via le browser et le mettre dans un formdata. Elle ajoute ensuite l’objet annonce et l’action qui est « POST\_ANNONCE » dans ce formdata. Elle va ensuite caché le modal et appeler la fonction addAnnonce() de httpServ.

function actionAddAnnonce() {

let titre = document.getElementById("titre").value;

let description = document.getElementById('description').value;

let prix = document.getElementById('prix').value;

let etat = document.getElementById('etat').value;

let images = document.getElementById("images");

let annonce = new Annonce();

annonce.setTitre(titre);

annonce.setDescription(description);

annonce.setFkEtat(etat);

annonce.setPrix(prix);

annonce.setFkUtilisateur(window.userConnected.pk\_utilisateur);

let formData = new FormData();

Array.from(images.files).forEach((file) => {

formData.append('images[]', file);

});

formData.append('annonce', JSON.stringify(annonce));

formData.append('action', "POST\_ANNONCE");

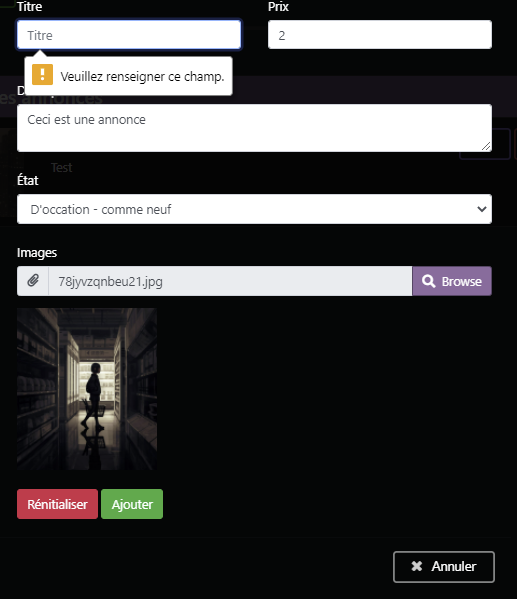
$('#modalAddAnnonce').modal('hide');

addAnnonce(formData,

[…]

PS : Dans le code ci-dessus nous ne testons pas si les champs sont vides car c’est déjà fait avec le code HTML et du coté de serveur.

Sous forme HTML :



Erreur si un champ est vide

1. Client : httpServ.js

La fonction addAnnonce effectue une requête POST vers le serveur en spécifiant les informations suivantes : action POST\_ANNONCE, l’annonce et les images. Tout ça se trouve dans le formdata précédemment créé.

function addAnnonce(formData, callback) {

$.ajax({

type: "POST",

dataType: "json",

url: SERVER\_MAIN,

data: formData,

success: callback,

error: callback,

contentType: false,

processData: false

});

}

1. Serveur : main.php

Le serveur reçoit la requête POST qu’il contrôle ensuite que cette dernière contienne bien les paramètres action et annonce.

Si tous ces paramètres sont corrects, alors il appelle la méthode « nouvelleAnnonce » avec comme paramètres : l’objet annonce qui est sous forme de JSON et les images. Sachant qu’une annonce n’a pas obligatoirement une/des images, le paramètre « images » est testé via la méthode isset. Si $\_FILES[‘images’] est vide, la valeur null est passée en paramètre sinon $\_FILES[‘images’] est passé.

Si au contraire les paramètres action et annonce ne sont pas là, un json avec un message d’erreur sera renvoyé au client et bien sûr avec le code d’erreur 500.

case "POST":

if (isset($\_POST['action'])) {

switch ($\_POST['action']) {

case 'POST\_ANNONCE':

if (isset($\_POST['annonce'])) {

$ctrl->nouvelleAnnonce($\_POST['annonce'], isset($\_FILES['images']) ? $\_FILES['images'] : null);

} else {

http\_response\_code(500);

echo json\_encode(array("error" => true, "message" => "Paramètre 'annonce' manquant !"));

}

break;

} else {

http\_response\_code(500);

echo json\_encode(array("error" => true, "message" => "Paramètre 'action' manquant !"));

}

break;

1. Serveur : Ctrl.php

Cette méthode permet d'ajouter une nouvelle annonce à la base de données. Elle prend en paramètre l’objet qui représente le json contenant les informations de l'annonce à ajouter et les images à ajouter à l’annonce. Cette méthode permet d'ajouter une nouvelle annonce à la base de données. Elle prend en paramètre l’objet qui représente le json contenant les informations de l'annonce à ajouter et les images à ajouter à l’annonce. Cette méthode va alors tester si l’utilisateur est connecté. S’il est connecté, la méthode « addAnnonce » du worker principale sera appelé avec les différents paramètres reçus précédemment.

Pour terminer elle un message d'erreur/succès et un boolean « error » qui, quand il est à true, veut dire que le retour est une erreur ou false si ce n'est pas une erreur. La méthode va aussi retourner le code 401 quand l'utilisateur n'est pas connecté, 500 quand un problème est survenu et 200 quand tout est bon.

public function nouvelleAnnonce($data, $images)

{

if ($this->wrk->isConnected()) {

if ($this->wrk->addAnnonce($data, $images)) {

echo json\_encode(array("error" => false, "message" => "L'annonce a correctement été ajouté !"));

} else {

http\_response\_code(500);

echo json\_encode(array("error" => true, "message" => "Erreur lors de l'ajout de l'annonce !"));

}

} else {

http\_response\_code(401);

echo json\_encode(array("error" => true, "message" => "Vous n'avez pas l'autorisation d'ajouter une nouvelle annonce !"));

}

}

1. Serveur : Wrk.php

Le worker principale va tout simplement faire appel au sous-worker « WrkAnnonce » qui effectuera la méthode « addAnnonce ».

public function addAnnonce($data, $images)

{

return $this->wrkAnnonce->addAnnonce($data, $images);

}

1. Serveur: WrkAnnonce.php

Cette fonction permet d'ajouter une nouvelle annonce dans la base de données. Elle reçoit en paramètre $data qui représente les informations de l'annonce à ajouter et $images qui représente la/les images à ajouter pour l'annonce. Une annonce peut avoir aucune image.

Cette fonction va ensuite tester si $data est bien définie et créer une variable $array qui décode $data qui est un JSON comme expliqué précédemment. Elle va après ça, tester si les valeurs rentrées dans le JSON sont définies.

Ensuite, elle va créer la requête SQL en respectant les conditions pour éviter les injections SQL. Elle va ensuite ajouter cette requête à une transaction.

1. Serveur :WrkDatabase

Voici le code pour l’ajout d’une transaction :

public function addQueryToTransaction($query, $params)

{

$res = false;

if ($this->pdo->inTransaction()) {

$maQuery = $this->pdo->prepare($query);

$res = $maQuery->execute($params);

}

return $res;

}

Si l’ajout à la transaction s’est bien effectué (true), la variable pk\_annonce reçoit la pk de la dernière annonce ajouté et va ensuite, si des images sont contenues dans la variable $images, les ajouter via la méthodes addImages() puis si toutes les images ont été ajouté correctement, un commit est effectué. Si cependant aucunes images n’étaient présentes dans la variable $images, un commit est directement fait. Si par malheur une erreur survient, un rollback est effectué.

public function addAnnonce($data, $images)

{

$result = false;

if (isset($data)) {

$array = json\_decode($data);

if (isset($array) && !empty($array)) {

if ($this->database->startTransaction()) {

try {

$titre = htmlspecialchars($array->titre);

$description = htmlspecialchars($array->description);

$prix = htmlspecialchars($array->prix);

$fkUtilisateur = htmlspecialchars($array->fkUtilisateur);

$fkEtat = htmlspecialchars($array->fkEtat);

if (isset($titre, $description, $prix, $fkEtat, $fkUtilisateur)) {

$sql = "INSERT INTO t\_annonce (titre, description, prix, fk\_utilisateur ,fk\_etat) VALUES (?,?,?,?,?)";

$params = array($titre, $description, $prix, $fkUtilisateur, $fkEtat);

if ($this->database->addQueryToTransaction($sql, $params)) {

$pk\_annonce = $this->database->getLastId('t\_annonce');

if (isset($images)) {

$imagesAdd = false;

for ($i = 0; $i < count($\_FILES['images']['name']); $i++) {

$imagesAdd = $this->refWrk->addImage($\_FILES['images']['name'][$i], $\_FILES['images']['tmp\_name'][$i], $\_FILES['images']['type'][$i], $pk\_annonce);

}

if ($imagesAdd) {

$result = $this->database->commitTransaction();

}

} else {

$result = $this->database->commitTransaction();

}

}

}

} catch (Exception $e) {

$this->database->rollbackTransaction();

}

}

}

}

return $result;

}

1. Serveur :WrkImage

La fonction addImage va tout simplement ajouter une image à la base de données avec la pk de l’annonce reçue en paramètre et en même temps l’ajouter dans un dossier du même nom que la pk de l’annonce.

public function addImage($image, $tmp\_name, $type, $pk\_annonce)

{

$result = false;

if (isset($image, $tmp\_name, $type, $pk\_annonce) && $pk\_annonce > 0) {

$ready = true;

if (!$this->database->isInTransaction()) {

$ready = $this->database->startTransaction();

}

if ($ready) {

try {

$newImageName = $this->renameImage($image);

$advertisementFolder = self::UPLOAD\_PATH . $pk\_annonce . "/";

$query = "INSERT INTO t\_image (image, fk\_annonce) VALUES (?,?)";

$params = array(htmlspecialchars($advertisementFolder . $newImageName), htmlspecialchars($pk\_annonce));

if ($this->database->addQueryToTransaction($query, $params)) {

if (!file\_exists($advertisementFolder)) {

@mkdir($advertisementFolder, 0777);

}

$result = move\_uploaded\_file($tmp\_name, $advertisementFolder . $newImageName);

}

} catch (Exception $e) {

$this->database->rollbackTransaction();

}

}

}

return $result;

}

1. Serveur : WrkDatabase.php

Voici le code pour le commit.

public function commitTransaction()

{

$res = false;

if ($this->pdo->inTransaction()) {

$res = $this->pdo->commit();

}

return $res;

}

Voici le code pour un rollback :

public function rollbackTransaction()

{

$res = false;

if ($this->pdo->inTransaction()) {

$res = $this->pdo->rollBack();

}

return $res;

}

1. Client: gestionAnnoncesCtrl

Le serveur va alors retourner si tout est bon un succès ou une erreur. Si c’est un succès, les annonces sont rechargées dans la navbar et dans le tableau. Pourquoi dans la navbar ? Car elle a une barre de recherche et qu’elle contient toutes les annonces.

function (data) {

if (!data.error) {

succes(data.message);

loadAnnonces(function () {

loadConnecteNavbarSection();

loadTableAnnonces();

});

} else {

erreur(data);

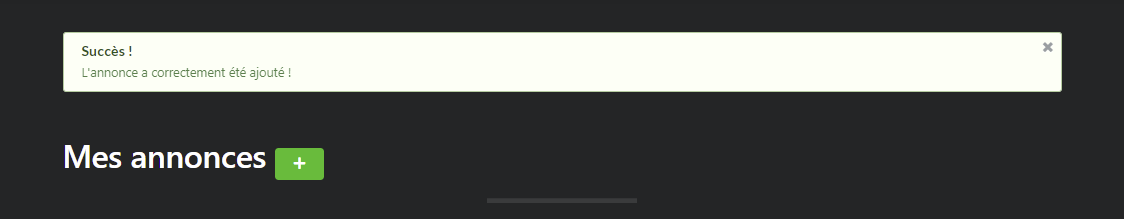
}

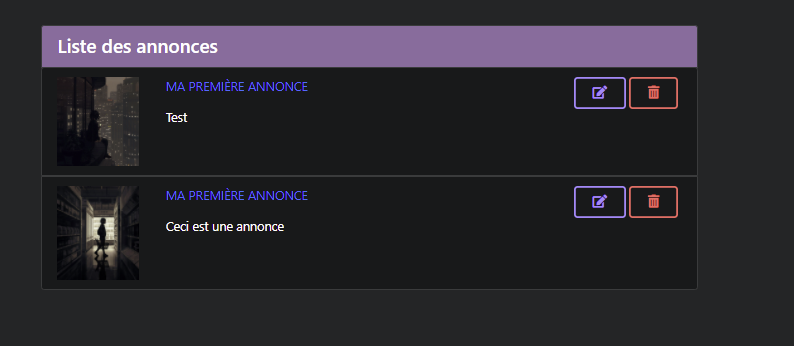
}

);

}

Voici le résultat sur le site web :





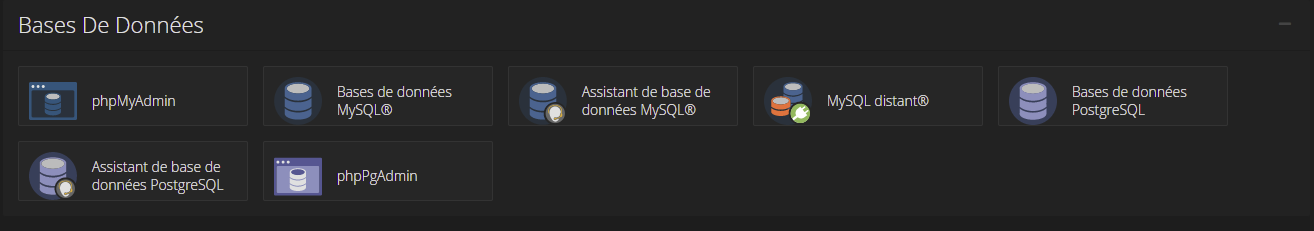
### Tests fonctionnels

Voici la liste des résultats de tests fonctionnels effectués le 12.03.2021 avec Lionel Ding en tant qu’attaquant/utilisateur.

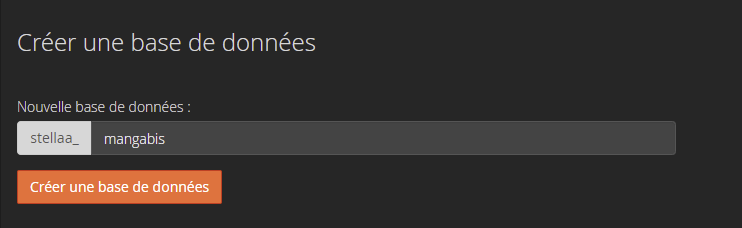
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| N° | Description | Résultat attendu | Résultat obtenu | OK ? |
| 1 | Un visiteur peut visualiser toutes les annonces. | Le visiteur a bien accès aux annonces via la page d’accueil. | Le visiteur a bien accès aux annonces via la page d’accueil. | OK |
| 2 | Un visiteur peut s’inscrire via son nom, prénom, email, numéro de téléphone et son mot de passe. | Le visiteur peut s’inscrire ce qui impacte la base de données. Il peut alors, se connecter. | Le visiteur peut s’inscrire ce qui impacte la base de données. Il peut alors, se connecter. | OK |
| 3 | Un visiteur disposant d’un compte peut se connecter si ces identifiants sont corrects. | Le visiteur peut se connecter via son email et son mot de passe. Il a alors accès à ces annonces et son profil. | Le visiteur peut se connecter via son email et son mot de passe. Il a alors accès à ces annonces et son profil. | OK |
| 4 | Un utilisateur connecté peut ajouter/supprimer/modifier des annonces. | L’utilisateur a bien accès aux fonctionnalités d’ajout, suppression et de modification d’annonces. | L’utilisateur a bien accès aux fonctionnalités d’ajout, suppression et de modification d’annonces. | OK |
| 5 | Un utilisateur connecté peut modifier son profil (nom, prénom, téléphone, email, mot de passe, etc). | L’utilisateur a bien accès aux modifications de son profil. | L’utilisateur a bien accès aux modifications de son profil et ne peut pas modifier son rôle. | OK |
| 6 | Un utilisateur avec un rôle d’admin peut supprimer n’importe quelle annonce de n’importe quel utilisateur. | L’utilisateur avec un rôle d’admin a bien accès à la suppression d’annonces de tous les utilisateurs. | L’utilisateur avec un rôle d’admin a bien accès à la suppression d’annonces de tous les utilisateurs. | OK |
| 7 | Un utilisateur avec un rôle d’admin peut supprimer un utilisateur ou modifier son rôle. | L’utilisateur avec un rôle d’admin a bien accès à la suppression d’utilisateurs et de modification de rôle de ceux-ci. | L’utilisateur avec un rôle d’admin a bien accès à la suppression d’utilisateurs et de modification de rôle de ceux-ci. | OK |
| 8 | Un utilisateur peut se déconnecter ce qui termine sa session. | L’utilisateur peut bien se déconnecter, sa session est correctement terminée. Il est ensuite redirigé vers la page d’accueil | L’utilisateur peut bien se déconnecter, sa session est correctement terminée. Il est ensuite redirigé vers la page d’accueil  (il est aussi déconnecté de la base de données) | OK |
| 9 | Un visiteur essaie de se rendre sur un profil utilisateur ou la gestion des annonces d’un utilisateur. | Un message d’erreur est affiché | Un message d’erreur est affiché. | OK |
| 10 | Un visiteur essaie de se connecter avec un mauvais email et mot de passe. | Un message d’erreur est bien affiché et le visiteur reste non-connecté. | Un message d’erreur est bien affiché et le visiteur reste non-connecté. | OK |
| 11 | Un visiteur essaie de s’inscrire mais n’accepte pas le captcha. | Un message d’erreur est affiché et le visiteur n’est pas inscrit. | Un message d’erreur est affiché et le visiteur n’est pas inscrit. | OK |
| 12 | Un visiteur essaie de s’inscrire mais n’a pas rempli tous les champs. | Un message d’erreur est affiché et le visiteur n’est pas inscrit. | Un message d’erreur est affiché et le visiteur n’est pas inscrit. | OK |
| 13 | Les injections SQL sont gérées dans chacun des champs que les utilisateurs et les visiteurs doivent entrer. | Les injections SQL sont bien gérées et impossible à effectuer | Les injections SQL sont bien gérées et impossible à effectuer | OK |

### Hébergement

Pour l’hébergement sur le cpanel, c’est très facile. J’ai préalablement créé le fichier SQL de la base de données avec les tables, … qui a été fourni. Je me suis ensuite redu sur le cpanel et je me suis rendu dans l’onglet « Base de données » et j’ai cliqué sur « Bases de données MySQL ».



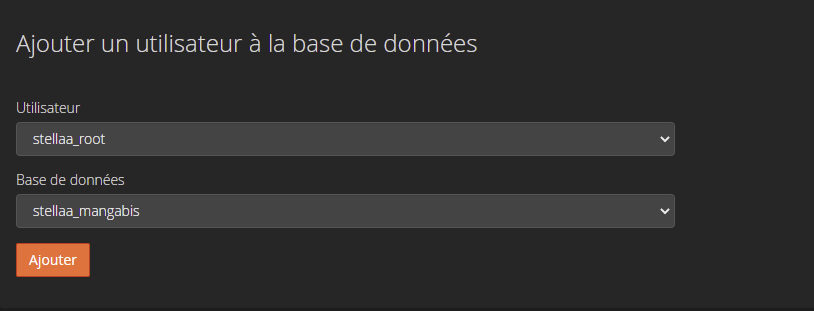
J’ai ensuite créé une nouvelle base de données avec comme nom : « mangabis ».



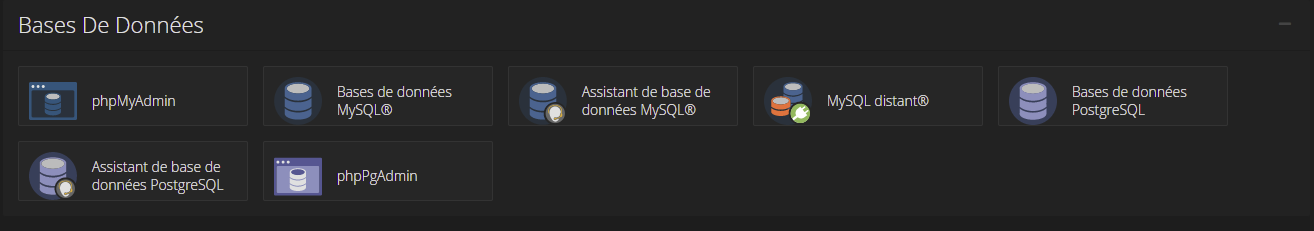
Après ça, j’ai créé un nouvel utilisateur que j’ai nommé root et dont j’ai généré un mot de passe.



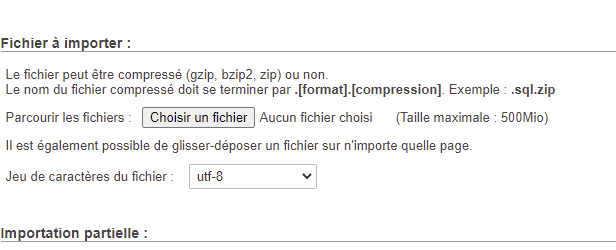
Je l’ai ensuite associé à la base de données mangabis :



Je me suis ensuite rendu dans phpMyAdmin :

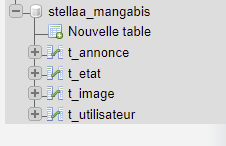


Dans phpMyAdmin, sous stellaa\_mangabis, nous pouvons nous rendre dans « Importer », puis cliquer sur « Choisir un fichier » et sélectionner le fichier « stellaa\_mangabis » :





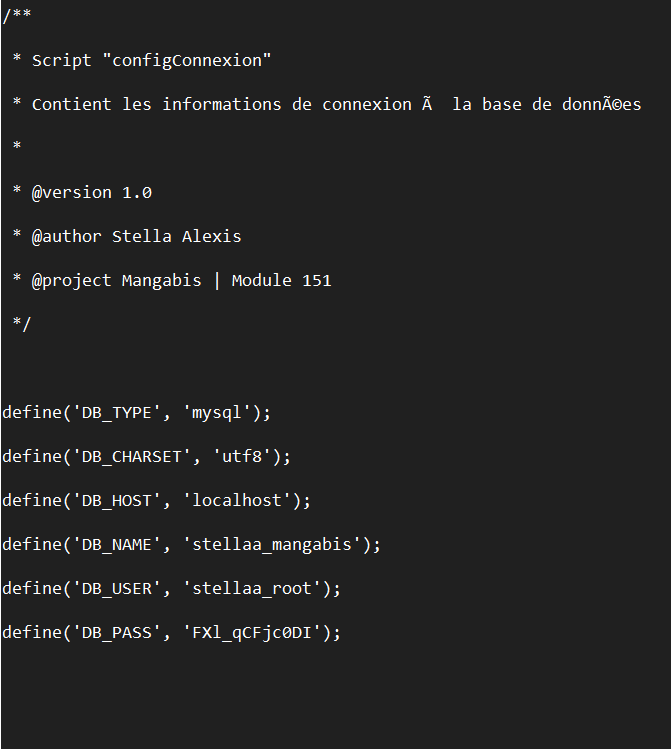
Il nous reste à cliquer sur « Exécuter » et le tour est joué !



Une fois la database déplacée sur CPanel, j’ai déplacé le code du site web en passant par un accès FTP avec l’application WinSCP. Il me restait juste à changer sur le client dans l’indexCtrl, le lien de l’API par :

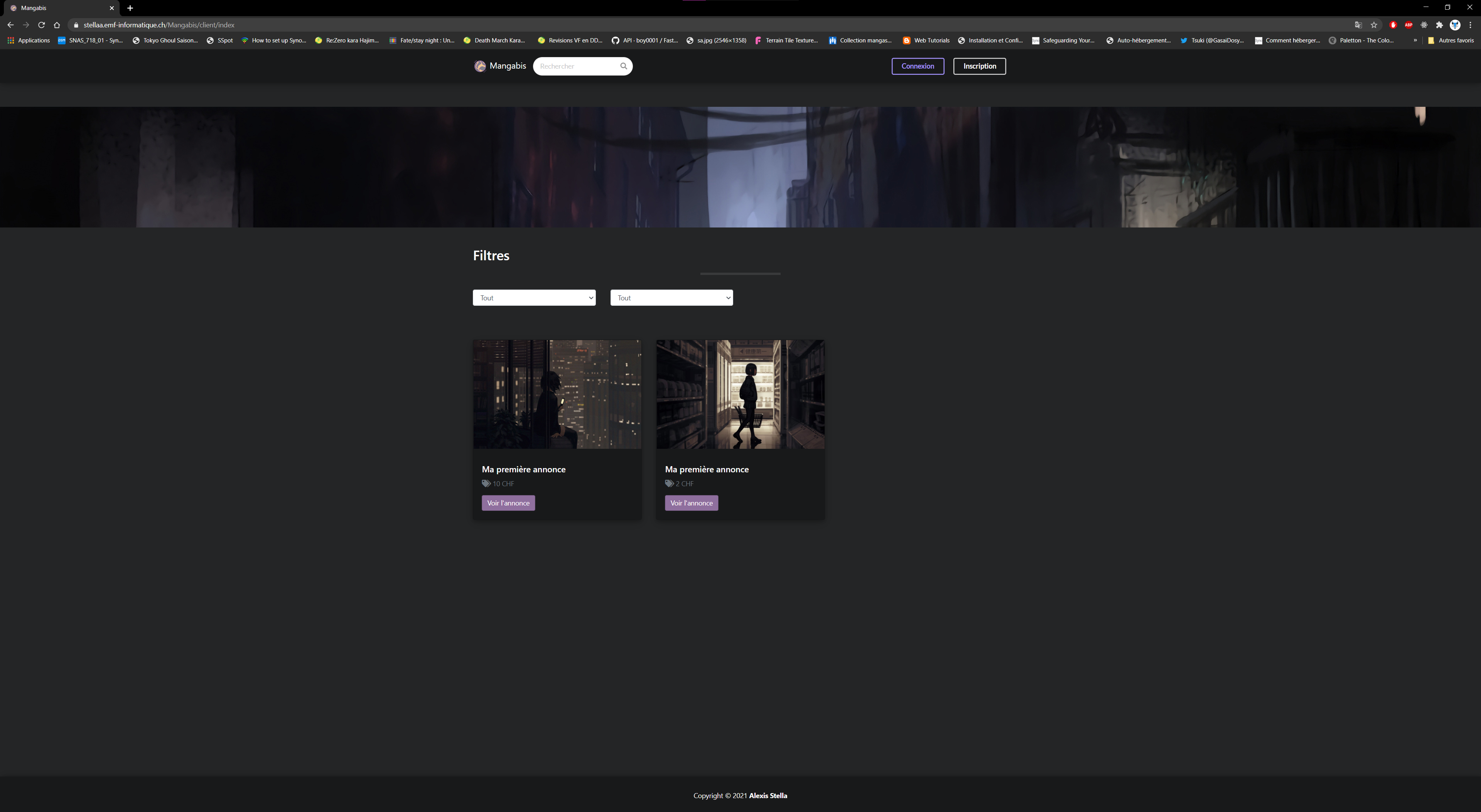


Et dans la configuration de la base de données, mettre les informations de l’utilisateur précédemment créé et le nom de la base de données.

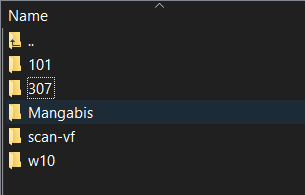


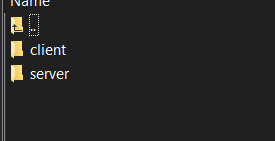
Nous pouvons maintenant accéder à notre site :

ATTENTION : Quand nous allons sur ce site, nous devons être absolument en https, sinon si non essayons de nous connecter, les sessions ne seront pas enregistrées



PS : Pour l’emplacement du site, il doit être placé à la racine dans un dossier nommé « Mangabis ». Ensuite nous devrons y retrouver le client et le serveur.

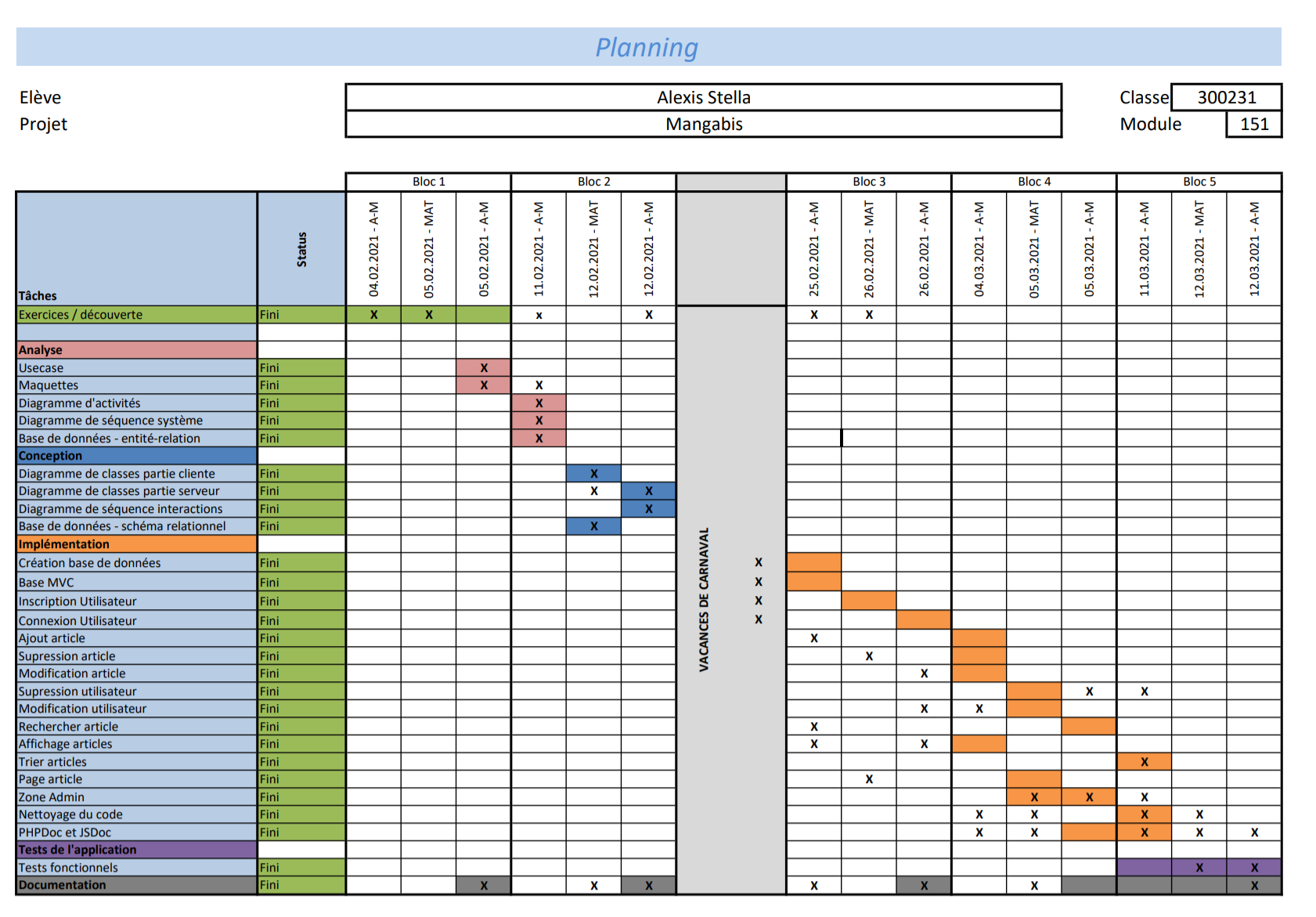




## Synthèse

### Différence timing

Voici le planning final du projet Mangabis du module 151. Comme nous pouvons le voir par rapport au planning de l’analyse c’est qu’il y a eu des choses pas prévues :



En effet, comme nous pouvons le voir, j’avais prévu un bloc pour les exercices. Ce qui s’est réellement passé c’est que nous avons fait des exercices jusqu’au bloc 3. Pour l’analyse et la conception j’ai été assez bon au niveau de la prévision. Pour ce qui est de l’implémentation, nous pouvons voir que j’ai pas mal avancé durant les vacances au niveau du login et du register. J’ai alors été un peu décalé pour le reste car j’avais pris un peu d’avance. Nous pouvons aussi voir que pour la modification d’un article, j’ai pris du retard. Ce retard s’explique par la complexité qui n’était pas prévu. J’ai alors décidé de faire les autres fonctionnalités plus m’attaquer à la modification. Pour terminer, nous pouvons voir que j’avais prévu le bloc 5 pour de la documentation, ce qui n’a pas été respecté. Sinon globalement, nous pouvons voir toutes les fonctionnalités sont implémentées.

### Conclusion du projet

Pour en conclure avec ce projet, je pourrai dire que je me suis bien sortie. En effet, j’ai réussi à réaliser tout ce que je souhaitai au niveau de l’application. J’ai même réussi à la rendre un minimum « user friendly ». Cependant, j’ai eu plus de mal pour la PHP Doc et la JS Doc. En effet, je n’avais pas prévu que ça prenne autant de temps, je compte alors, pour la prochaine fois, faire cette doc en même temps que je code même si c’est un peu barbant.

Au niveau de la documentation du projet, j’ai eu beaucoup de mal pour faire la descente de code car je trouve très compliquer d’expliquer avec des mots faciles, le fonctionnement de l’application avec toutes les étapes quelle effectue pour une action. Sinon tout le reste a été et je pense avoir réussi le module si ma documentation est bonne.

# Conclusion

En conclusion, dans le module 151, nous avons appris à 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. Nous avons aussi appris à identifier les informations importantes de protection et de sécurité en tenant compte de la protection des données, et définir les mesures. Nous avons appris à 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.

Pour terminer, nous avons appris à mettre en œuvre les souhaits de modifications conformément au déroulement prescrit des modifications et définir et exécuter la procédure de test et de remise, la documenter dans un procès-verbal de tests. Si nécessaire, entreprendre les corrections.

## Ce que j’ai aimé

Dans ce module j’ai beaucoup apprécié le fait que ce soit un projet personnel. Je trouve qu’un projet personne est beaucoup mieux car nous pouvons clairement se rendre compte de la réalité des choses. Par exemple, nous avons tendance à viser un peu trop haut et à penser que certaines choses qui nous semblait facile à première vue, est finalement plus compliqué que prévu. Dans mon cas c’était la modification d’une annonce. J’ai du trouvé un compromis pour la modification des images mais j’ai quand même réussi. J’ai aussi beaucoup aimé les introductions du jour avec la petite vidéo/PowerPoint. En bref je recommande ce module et surtout ce professeur 😊 qui est très cool et proche de ces élèves ce que je trouve très important quand on souhaite faire aimer un module.

## Ce que j’ai moins aimé

J’ai moins aimé le fait que nous faisions des exercices pendant les trois premières semaines. Je trouve, personnellement, que ça aurait été mieux de tous les faire le bloc un et ensuite partir sur le projet.

# Sources

Powerpoints du professeur

StackOverFlow