P-183\_WebStore



(Une image originale représentant le projet)

Table des matières

[1 Spécifications 3](#_Toc308526316)

[1.1 Titre 3](#_Toc308526317)

[1.2 Description 3](#_Toc308526318)

[1.3 Matériel et logiciels à disposition 3](#_Toc308526319)

[1.4 Prérequis 3](#_Toc308526320)

[1.5 Cahier des charges 3](#_Toc308526321)

[1.5.1 Objectifs et portée du projet (objectifs SMART) 3](#_Toc308526322)

[1.5.2 Caractéristiques des utilisateurs et impacts 3](#_Toc308526323)

[1.5.3 Fonctionnalités requises (du point de vue de l’utilisateur) 3](#_Toc308526324)

[1.5.4 Contraintes 3](#_Toc308526325)

[1.5.5 Travail à réaliser par l'apprenti 4](#_Toc308526326)

[1.5.6 Si le temps le permet … 4](#_Toc308526327)

[1.5.7 Méthodes de validation des solutions 4](#_Toc308526328)

[1.6 Les points suivants seront évalués 4](#_Toc308526329)

[1.7 Validation et conditions de réussite 4](#_Toc308526330)

[2 Planification Initiale 4](#_Toc308526331)

[3 Analyse 4](#_Toc308526332)

[3.1 Opportunités 4](#_Toc308526333)

[3.2 Document d’analyse et conception 4](#_Toc308526334)

[3.3 Conception des tests 5](#_Toc308526335)

[3.4 Planification détaillée 5](#_Toc308526336)

[4 Réalisation 5](#_Toc308526337)

[4.1 Dossier de Réalisation 5](#_Toc308526338)

[4.2 Modifications 5](#_Toc308526339)

[5 Tests 5](#_Toc308526340)

[5.1 Dossier des tests 5](#_Toc308526341)

[6 Conclusion 5](#_Toc308526342)

[6.1 Bilan des fonctionnalités demandées 5](#_Toc308526343)

[6.2 Bilan de la planification 5](#_Toc308526344)

[6.3 Bilan personnel 5](#_Toc308526345)

[7 Divers 6](#_Toc308526346)

[7.1 Journal de travail 6](#_Toc308526347)

[7.2 Bibliographie 6](#_Toc308526348)

[7.3 Webographie 6](#_Toc308526349)

[8 Annexes 6](#_Toc308526350)

# Spécifications

## Titre

Webstore

## Description

Création d’un site sécurisé

## Matériel et logiciels à disposition

Un ordinateur standard de la section informatique avec Docker Desktop.

## Les points suivants seront évalués

* Le rapport
* Le journal de travail
* Le code et les commentaires
* Les documentations de mise en œuvre et d’utilisation

## Validation et conditions de réussite

* Compréhension du travail
* Possibilité de transmettre le travail à une personne extérieure pour le terminer, le corriger ou le compléter
* Etat de fonctionnement du produit livré

# Réalisation

## Dockerisation des services

Lors de la mise en place de notre projet, plusieurs fichiers nous ont été fourni. Premièrement un fichier docker-compose.yml qui servira à créer trois containers :

* Mysql (Pour la base de données)
* PhpMyAdmin (Pour gérer la base de données avec une interface)
* NodeJs (Pour notre application, il faut tenir compte que le docker-compose appèle le dockerfile de nodejs)

Le container de nodejs est créer grâce à deux étape, tout d’abord le fichier nodejs.dockerfile permet de créer l’image du container. Puis le dockercompose appelle ce fichier pour que celui-ci se lance avec les deux autres.

Pour mettre en place ces différent container nous devons nous rendre dans le dossier où se situe le fichier docker-compose.yml puis dans un terminal de commande exécuter la commande suivante.

*Docker compose up -d*

Ces trois containers vont interagir entre eux, spécialement le container de notre application et le container de base de données, car nous allons établir une connexion pour que notre application puisse interagir avec celle-ci.

Pour ce faire, nous devons installer les package *mysql2* avec *npm*, puis dans notre application établir la connexion.

Une image contenant texte, capture d’écran, logiciel, Police

Description générée automatiquement

Tout d’abord nous importons *mysql2* puis nous créons la connexion grâce à *mysql.createConnection()*, en insérant les bonne information de notre container ainsi que notre base de données (Attention la base de données doit être créer sinon la connexion de pourra pas se faire).

L’host, correspond à l’Ip du container de base de données. Nous pouvons l’obtenir en exécutant la commande *docker inspect <nom-du-container-db>.*

Le port est celui par défaut du container mysql.

## HTTPS

Pour pouvoir accéder à notre site de manière sécurisé avec le protocole HTTPS, nous avons besoin de plusieurs choses.

Tout d’abord nous avons besoin d’une clé et d’un certificat :

Une image contenant Police, texte, capture d’écran, Graphique

Description générée automatiquement

Pour récupérer ses deux fichiers, j’ai exécuté la commande suivant avec le CLI de OpenSSL.



Celui-ci va ensuite nous demander des informations pour pouvoirs générer ces deux fichiers. Puis lorsque nous avons générer ses deux fichiers nous pouvons les intégrer a notre application.

Une image contenant texte, Police, capture d’écran, nombre

Description générée automatiquement

Puis dans notre code pour mettre en place le certificat SSL, nous devons importer les deux modules suivants.

Une image contenant Police, texte, capture d’écran, ligne

Description générée automatiquement

Ils vont nous permettre de lire les deux fichier *cert* et *key* et créer la connexion avec le HTTPS.

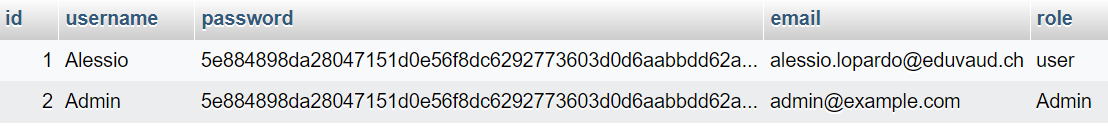
Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Pour se faire nous créons une constante *option* qui contient le contenu de nous deux fichiers. Puis nous créons le « server » à l’aide de ses options en spécifiant que nous le lançons sur notre *app* et en écoutant le port 8080.

## Authentification par mot de passe

Pour effectuer la vérification de notre mot de passe, il nous faut tout d’abord ajouter un utilisateur avec un mot de passe hashé dans notre base de données.



Ensuite nous devons installer les package « crypto » pour notre application. On se rend donc à l’emplacement de notre « package.json » puis nous exécutons la commande suivante pour effectuer l’installation.

npm install crypto

Puis lorsque nous avons installé les packages, nous pouvons créer un nouveau fichier dans notre projet et nous appellerons « crypto.js ». Dans ce fichier nous allons importer crypto, et nous allons définir deux fonctions.

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Ces deux fonctions vont nous permettre de comparer si deux hash sont identique, donc par la suite, si oui cela voudra dire que l’utilisateur sera identifié.

« hashPassword » permet de hasher un mot de passe entré en paramètre avec du sha256 avec comme output du hexadécimal.

Puis « comapreHashPassword » permet de comparer le hash de deux mots de passe entré en paramètre. Car le premier paramètre sera hashé grâce à la première fonction puis le deuxième paramètre est le mot de passe hashé que nous devons comparer.

Maintenant, nous devons appeler ces fonctions lorsque l’utilisateur va s’identifier sur notre site. Nous nous rendons donc dans notre fichier login pour modifier notre route de login.

Nous allons récupérer les informations du user avec le username entré par l’utilisateur (attention à faire une requête préparée pour éviter les injections SQL), puis nous avec le mot de passe récupéré (qui lui est hashé), nous allons comparer les deux mots de passes en hashant avant le mot de passe entré par l’utilisateur pour faire la comparaison des hash. Puis si la comparaison est correcte alors nous générons le Json Web Token.

Une image contenant texte, capture d’écran, logiciel

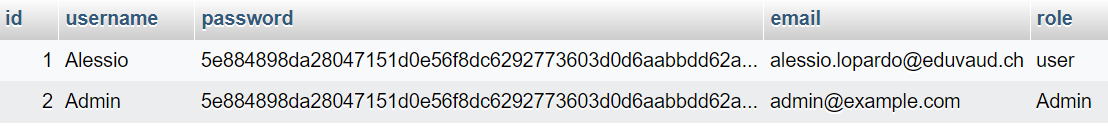
Description générée automatiquement

TODO :

* Faire présentation
* Faire mini interface

## Administration

Pour la page Admin, nous devons rajouter dans notre base de données un champs « Rôle » pour chaque utilisateur, par défaut nous allons lui attribuer la valeur « user ». Ensuite nous pouvons créer un nouvel utilisateur est lui attribuer le rôle « Admin »



Ensuite nous devons récupérer ce champs « rôle » dans notre token, nous allons donc rajouter cette donnée lorsqu’on signe celui-ci.

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Ensuite, lorsque nous avons rajouter le rôle dans le token, nous allons définir le « req.user » avec la valeur du token lorsque nous le vérifions lorsqu’un utilisateur se login.

Une image contenant texte, capture d’écran, logiciel

Description générée automatiquement

Ensuite lorsque nous avons données ces informations au req.user, nous avons accès au rôle de l’utilisateur quand celui-ci est connecter (donc grâce au token).

Nous pouvons donc définir notre fonction qui va permettre de vérifier le rôle de l’utilisateur dans notre fichier User.

Une image contenant texte, capture d’écran, affichage, Police

Description générée automatiquement

Cette fonction attend en paramètre un rôle (ici nous le définirons plus tard lors de l’appelle de la fonction). Puis elle va récupérer le rôle de l’utilisateur grâce au « req.user.userRole » et le donné comme valeur à une constante, puis nous allons comparer cette constante avec le rôle que nous avons passé en paramètre. Si c’est juste alors on continue, sinon la fonction nous renvoie un statue « 403 ».

Puis pour utiliser notre fonction et donc vérifier les rôles des utilisateurs, nous appelons notre fonction sur les routes exactement comme le « auth ».



Cela veut dire que lorsque nous définis serons une route qui ne peut être accéder uniquement par des Administrateurs, nous appellerons notre fonction de cette manière. De plus si nous voulions par la suite ajouter d’autre rôle, nous pourrions également le faire de cette manière

## Protection contre les injections SQL

Lorsque nous voulons effectuer des requêtes vers notre base de données de manière sécurisé et se protéger des injections SQL. Nous parlons de requêtes paramétrées.

Exemple :



Nous définissons notre « query », puis lors de la condition de recherche « WHERE » nous entrons notre condition de recherche, ici nous voulons sélectionner un username en particulier, puis nous rajouter uniquement un « ? » et non directement la valeur entrée par l’utilisateur.

De cette manière, lorsque nous appelons notre query dans la deuxième ligne nous ajouter le champs « [username] » qui est défini par la valeur entrée par l’utilisateur de cette manière notre fonction « connection.query » va automatiquement géré l’entré et permet d’éviter les injections SQL.

## Profile du client

Après avoir défini notre « req.user » cela va être beaucoup plus simple de récupérer les données de l’utilisateur, car grâce au req.user nous avons récupérer les données principales de l’utilisateur qui sont stocker dans le token (dans ce cas-là ce qui nous intéresse est l’id).

Nous pouvons donc définir une nouvelle route User.

Une image contenant texte, capture d’écran, affichage, logiciel

Description générée automatiquement

Celle-ci va être très simple. Tout d’abord, il ne faut pas oublier d’ajouter la partie authentification grâce au « auth ». Puis nous allons définir notre requête préparée pour récupérer l’utilisateur connecter de la base de données. Puisque nous définissons notre « req.user » et que celui-ci possède l’id de l’utilisateur, nous pouvons directement effectuer notre recherche selon l’id. Ensuite nous effectuons une rapide gestion des erreurs, si notre requête revoie une erreur ou si notre requêtes nous renvoie aucun utilisateur. Pour finir nous envoyons le résultat avec un « res.json ».

# Conclusion

## Bilan des fonctionnalités demandées

Il s’agit de reprendre point par point les fonctionnalités décrites dans les spécifications de départ et de définir si elles sont atteintes ou pas, et pourquoi.

Si ce n’est pas le cas, estimer en « % » ou en « temps supplémentaire » le travail qu’il reste à accomplir pour terminer le tout.

## Bilan de la planification

Distinguer et expliquer les tâches qui ont généré des retards ou de l'avance dans la gestion du projet. Indiquer les différence entre les planifications initiales et détaillées avec le journal de travail.

## Bilan personnel

Si c’était à refaire:

Qu’est-ce qu’il faudrait garder ? Les plus et les moins ?

Qu’est-ce qu’il faudrait gérer, réaliser ou traiter différemment ?

Qu’est que ce projet m’a appris ?

Suite à donner, améliorations souhaitables, …

Remerciements, signature, etc.

# Divers

## Journal de travail

Date, activité (description qui permet de reproduire le cheminement du projet), durée, liens et références sur des documents externes. Lorsqu’une activité de recherches a été entreprise, il convient d’énumérer ce qui a été trouvé, avec les références.

## Bibliographie

Références des livres, revues et publications utilisés durant le projet.

## Webographie

Références des sites Internet consultés durant le projet.

# Annexes

Listing du code source (partiel ou, plus rarement complet)

Guide(s) d’utilisation et/ou guide de l’administrateur

Etat ou « dump » de la configuration des équipements (routeur, switch, robot, etc.).

Extraits de catalogue, documentation de fabricant, etc.