P-183\_WebStore



Table des matières

[1 Spécifications 3](#_Toc167531738)

[1.1 Titre 3](#_Toc167531739)

[1.2 Description 3](#_Toc167531740)

[1.3 Matériel et logiciels à disposition 3](#_Toc167531741)

[1.4 Les points suivants seront évalués 3](#_Toc167531742)

[1.5 Validation et conditions de réussite 3](#_Toc167531743)

[2 Réalisation 3](#_Toc167531744)

[2.1 Dockerisation des services 3](#_Toc167531745)

[2.2 HTTPS 6](#_Toc167531746)

[2.3 Authentification par mot de passe 7](#_Toc167531747)

[2.4 Administration 8](#_Toc167531748)

[2.5 Protection contre les injections SQL 10](#_Toc167531749)

[2.6 Profile du client 10](#_Toc167531750)

[3 Conclusion 11](#_Toc167531751)

[3.1 Bilan des fonctionnalités demandées 11](#_Toc167531752)

[3.2 Bilan de la planification 11](#_Toc167531753)

[3.3 Bilan personnel 11](#_Toc167531754)

[4 Divers 11](#_Toc167531755)

# 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)

De plus, nous allons modifier ce fichier docker-compose.yml et rajouter un fichier nommé « vuejs.dockerfile » pour ajouter un container qui contiendra la partie FrontEnd de notre application.

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 containers qui sont MySQL et PhpMyAdmin.

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. (Attention à l’ajout de package, il faut reconstruire les images et les containers)

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>* (Attention cette adresse ip peut changer au fil du temps que nous recréons nos containers, si cela ce produit, il faut mettre à jour l’adresse IP dans le BackEnd)*.*

Le port est celui par défaut du container mysql.

Pour la partie FrontEnd en vue, puisque nous l’avons ajouté nous même le fichier docker-compose.yml ne contient pas de container pour vue. Nous pouvons donc créer un nouveau service dans celui-ci et créer un fichier nommé « vuejs.dockerfile ».

Le container VueJS et similaire à celui de NodeJS, il nous faut uniquement modifier quelques informations dans notre dockerfile.

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Ici, nous disons d’exécuté la commande pour installer le CLI de vue, puis nous créons le répertoire de notre application pour le container et nous copions le fichier « package.json » qui contient toutes les informations sur les dépendance, dans celui-ci.

Puis nous lui disons d’exécuté la commande pour installer les packages dans ce répertoire, nous copions le projet dans ce même répertoire, nous lui disons d’exposer le port 5173 qui est celui par défaut de Vue et pour finir nous lui ajouter les différentes commandes pour qu’il puisse démarrer notre FrontEnd qui sont « npm run dev », avec comme référence le fichier « main.js » qui est le point de départ de notre application FrontEnd.

Ensuite, nous devons rajouter le container dans notre fichier dockercompose pour qu’il puisse exécuter notre fichier dockerfile en plus de lancer tous les autres containers.

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Ici, cette partie est très similaire à celle de NodeJS également, nous modifier uniquement les informations pour qu’il prenne le bon dockerfile, ainsi que les différents volumes qui doivent être les même que ceux créer grâce à notre dockerfile, puis nous spécifiant les ports sur lesquelles notre machine hôtes et le container vont communiquer et la commande pour démarrer les services.

Maintenant lorsque nous executons la commande pour lancer notre docker-compose, un nouveau container contenant vuejs s’affiche, cependant lorsque nous allons sur notre navigateur, et que nous recherchons <http://localhost:5173> qui ne s’affiche et la page ne fonctionne pas. Ceci est dû que le « localhost » fonctionne mais puisqu’il est exécuté dans le container il ne fonctionne que dans celui-ci, nous n’arrivons donc pas à l’atteindre depuis notre machine hôte.

Il faut que nous rajoutions deux petits paramètres dans le fichier « viteconfig.js » de notre projet FrontEnd.

Une image contenant Police, texte, capture d’écran

Description générée automatiquement

Nous spécifions le « host » qui nous permet d’accepter toutes les adresses, de ce fait nous pourrons accéder au localhost du container, et nous rajoutons le port 5173 pour nous assurer qu’il écoute bien sur ce port.

Maintenant tout fonctionne.

ChatGPT à aider à résoudre les problèmes liés à la dockersisation du container VueJS

## 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 à 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.

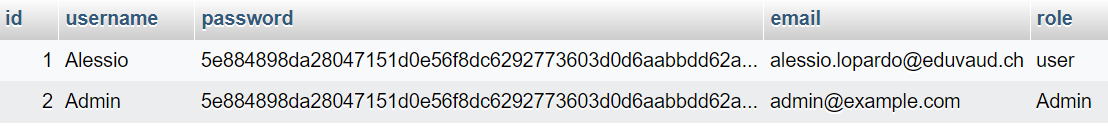
Une image contenant texte, capture d’écran, Police

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.

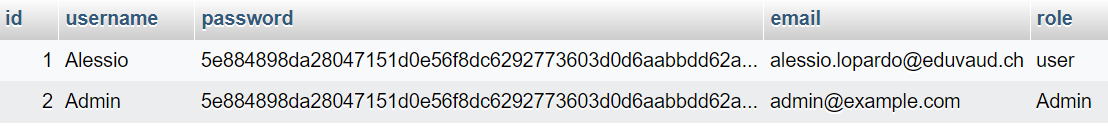
Nous allons récupérer les informations du user avec le username entré par l’utilisateur (attention à faire une requête préparée pour éviter les injections SQL), puis nous avec le mot de passe récupéré (qui lui est hashé), nous allons comparer les deux mots de passes en hashant avant le mot de passe entré 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

## 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 requête nous renvoie aucun utilisateur. Pour finir nous envoyons le résultat avec un « res.json ».

# Conclusion

## Bilan des fonctionnalités demandées

Toutes les fonctionnalités demandées sont présentes, de plus un FrontEnd en VueJS a été rajouter avec la Dockeristation de ce service. Malheureusement la communication entre le frontend et le backend fonctionne uniquement sur mon pc portable, j’ai essayé de cloner mon repo github sur un autre ordinateur mais la communication de se fait pas. Le problème vient probablement du certificat SSL qui n’est pas « valide » de ce fait la communication http et https ne se fait pas. J’ai tout de même pu faire la démo de mon projet.

## Bilan de la planification

Aucun outil de gestion de projet tel que trello, github project, etc...

Un journal de travaille avec les différentes tâches et problème rencontré à été documenter au fil du projet.

## Bilan personnel

Le fait d’associer un environnement de développement sur docker et la conception d’un site sécurisé à été très pédagogique pour l’apprentissage de ces deux matières. J’ai aimé voir la progression de mon site avec les problèmes rencontré et les possibilités infinies d’ajout de fonctionnalité. La partie FrontEnd est moins chargé que le backend mais pourquoi pas continuer cette partie-là dans le futur. Mon backend à beaucoup de pistes d’amélioration tel que l’utilisation de Controller mais ceci me laisse de nouvelle piste à explorer pour m’améliorer. ChatGPT m’a bien aidé vers la fin du projet, avec la charge de travail de tous les projets de cette fin d’année je ne voulais pas perdre de temps sur certain problème ou ajout de fonctionnalité. Je suis très retissent à l’utilisation de cet outil car je n’aime pas quand le travail est fait à ma place ou que le travaille ajouter fonctionne mais que je ne le comprenne pas. C’est pour cela que je me suis assuré de comprendre que toutes les informations que chatGPT me fournissait pour avoir une connaissance de ce que j’implémentais dans mon code et pour avoir les capacités pour débugger ces parties là si je rencontre des problèmes.

# Divers

Un fichier gitignore a été ajouter à la racine du repos Git pour ne pas push les dossier node\_modules.