P\_App-Webstore



Cardis Yohan – Cid2a

Vennes

32

G. Sonney

ETML

Table des matières

[1 Spécifications 3](#_Toc167269130)

[1.1 Titre 3](#_Toc167269131)

[1.2 Description 3](#_Toc167269132)

[1.3 Matériel et logiciels à disposition 3](#_Toc167269133)

[1.4 Prérequis 3](#_Toc167269134)

[2 Réalisation 4](#_Toc167269135)

[2.1 Conceptualisation 4](#_Toc167269136)

[2.2 Explication de mon code 5](#_Toc167269137)

[2.2.1 Le client peut accéder à son propre profil en utilisant un lien tel que : 5](#_Toc167269138)

[• https://localhost/users/john 5](#_Toc167269139)

[Seul son profil lui sera rendu visible. 5](#_Toc167269140)

[2.2.2 Il doit être possible d’accéder à votre site de e-commerce de manière sécurisée (https://localhost). Le port utilisé sera le 443. Le certificat sera auto-signé par OpenSSL. 6](#_Toc167269141)

[2.2.3 L’utilisateur devra s’authentifier par la page https://localhost/login Le mot de passe sera haché et salé avant d’être stocké dans la base de données (table t\_users). 7](#_Toc167269142)

[3 Conclusion 8](#_Toc167269143)

[3.1 Bilan des fonctionnalités demandées 8](#_Toc167269144)

[3.2 Bilan de la planification 8](#_Toc167269145)

[3.3 Bilan personnel 8](#_Toc167269146)

[4 Divers 8](#_Toc167269147)

[4.1 Journal de travail 8](#_Toc167269148)

[4.2 Bibliographie 8](#_Toc167269149)

[4.3 Webographie 8](#_Toc167269150)

[5 Annexes 8](#_Toc167269151)

# Spécifications

## Titre

Création d’un site d’e-commerce sécurisé

## Description

Au terme du projet, l’apprenti sera capable de construire une application node.js offrant un accès sécurisé et une gestion des rôles.

## Matériel et logiciels à disposition

• Un ordinateur standard de la section informatique avec Docker Desktop

## Prérequis

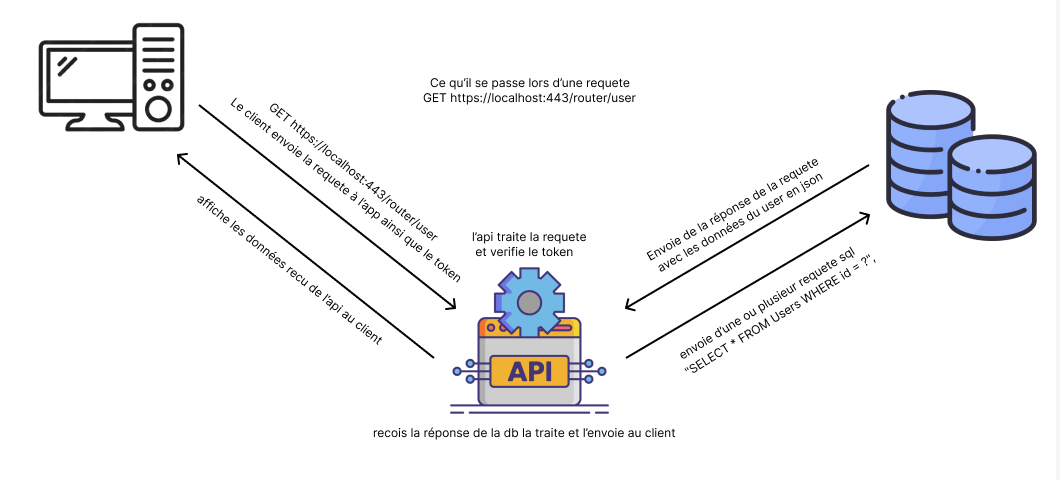
Module 294

Module 187

# 

# Réalisation

## Conceptualisation



## Explication de mon code

Pour simplifier les explications, je vais reprendre tous les points demandés dans le cahier des charges. Pour chacun de ces points, j’expliquerai ce que j’ai fait pour les valider.

### L’ensemble des services web sera conteneurisé.

Pour faire cela, c'était assez simple. Une fois mon application terminée, j'ai récupéré le fichier zip mis à disposition pour le projet.

Ensuite, j'ai supprimé tous les fichiers dans le dossier app pour y copier ceux de mon application.

Après cela, il ne me restait plus qu'à faire les modifications nécessaires pour que le Dockerfile et le docker-compose soient fonctionnels avec mon application.

Tous les services dockerizé :

Une image contenant capture d’écran, texte, logiciel, Logiciel multimédia

Description générée automatiquement

Dockerfile :

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Pour le Dockerfile, j'ai uniquement changé le chemin vers mon application, car ce n’était pas le bon, et modifié le port pour que l’application soit bien disponible sur le port 443, comme demandé dans le cahier des charges.

Docker-compose :

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

Description générée automatiquement

Maintenant, le docker-compose a uniquement été modifié dans le service Node.js, celui de l'application. Voici les différents points qui ont été modifiés :

* Les ports, car dans le cahier des charges, il a été demandé que l’application fonctionne sur le port 443.
* J’ai enlevé les options qui ne me semblaient pas utiles, par exemple l'utilisateur.

Ensuite, je n’ai pas trouvé de solution pour définir l’IP d'un conteneur avant de l’avoir lancé. Donc, lors du premier lancement de l’environnement sur Docker, je suis obligé de faire la commande “docker inspect nom-du-conteneur-de-la-db”, récupérer l’IP et, si nécessaire, la changer dans mon fichier de connexion.

### Le client peut accéder à son propre profil en utilisant un lien tel que :

### • <https://localhost/users/john>

### Seul son profil lui sera rendu visible.





Une image contenant texte, capture d’écran

Description générée automatiquement

Pour commencer, on peut voir que le code commence par récupérer le token dans le stockage local du navigateur.

Ensuite, s'il réussit à le récupérer, il envoie la requête “GET https://localhost/users” avec le token dans l'en-tête. Cette requête est dirigée vers le fichier app.mjs qui la transmet au mainrouter.

La première action de la requête est de récupérer le token, puis de le décoder. Ensuite, grâce à l'ID de l'utilisateur récupéré dans le token, le code envoie une requête SQL à la base de données : "SELECT \* FROM Users WHERE id = ?".

Une fois que l'API reçoit les données de la base de données, elle les conserve dans un tableau au format JSON. Enfin, le frontend (HTML) récupère ce tableau et l'affiche sur la page.

### Il doit être possible d’accéder à votre site de e-commerce de manière sécurisée (https://localhost). Le port utilisé sera le 443. Le certificat sera auto-signé par OpenSSL.

Tout d'abord, j'ai dû créer mes certificats. Pour cela, j'ai exécuté cette commande dans un terminal Git Bash.

La commande :



Cette commande va donc créer deux fichier (server.key et server.cert)

Puis voici comment ils sont utilisés dans mon code :

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

Description générée automatiquement

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

Description générée automatiquement

Pour commencer, le code crée un objet “httpsOptions” qui contient deux propriétés : "key" et "cert". "Key" contient la clé privée du serveur, lue à partir du fichier “server.key”. "Cert" contient le certificat SSL du serveur, lu à partir du fichier ”server.cert“. Ces fichiers sont lus de manière synchrone en utilisant “fs.readFileSync”.

Dans le deuxième code, un serveur HTTPS est créé en utilisant les options définies dans “httpsOptions”. Une fois en fonctionnement, il écoute sur le port défini.

### L’utilisateur devra s’authentifier par la page https://localhost/login Le mot de passe sera haché et salé avant d’être stocké dans la base de données (table t\_users).

Cette fonctionnalité est utilisée uniquement pour le processus de connexion ; par la suite, c'est le token qui prend le relais.

Voici le code nécessaire pour implémenter cette fonctionnalité :

Une image contenant texte, capture d’écran, Police

Description générée automatiquementUne image contenant texte, capture d’écran, Police, affichage

Description générée automatiquement

La première image illustre le processus de hachage et de salage du mot de passe saisi par l'utilisateur.

Pour le sel :

* “randomBytes(16)” génère 16 octets de données aléatoires.
* “.toString("hex")” convertit ces octets en une chaîne de caractères hexadécimale.

Le sel est utilisé pour renforcer la sécurité du hachage en garantissant que même des mots de passe identiques produiront des hachages différents.

Pour le hachage :

* “crypto.pbkdf2Sync” est une fonction qui utilise l'algorithme PBKDF2 (Password-Based Key Derivation Function 2) pour hacher le mot de passe.
* “salt” est le sel généré précédemment.
* “1000” est le nombre d'itérations de l'algorithme, ce qui renforce la sécurité en augmentant le temps nécessaire pour calculer le hachage.
* “64” est la longueur de la clé dérivée en octets.
* "sha512" spécifie que l'algorithme de hachage SHA-512 doit être utilisé.
* “.toString("hex")” convertit le résultat du hachage en une chaîne de caractères hexadécimale.

Dans le deuxième code, le même processus est effectué, mais ensuite, le hachage de la base de données est comparé pour vérifier si le mot de passe est correct.

### Une page d’administration devra avoir un champ de recherche (Nom du visiteur) et permettre d’afficher tous les utilisateurs ayant tout ou partie de ce nom.

Pour faire cela voici comment j’ai fait,

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

Description générée automatiquement

Dans cette partie du code, c'est la requête que l'API effectue à la base de données pour récupérer les données de l'utilisateur ou des utilisateurs.

Comme on peut le voir, elle commence par vérifier si la requête contient le token, puis ellevérifie ce token. Une fois cela fait, elle vérifie si la personne qui a fait la requête a le rôle d'administrateur, ce qui est nécessaire pour pouvoir utiliser cette fonctionnalité.

Une image contenant texte, capture d’écran, logiciel

Description générée automatiquement

Ici, c'est le côté client. Lorsque l'utilisateur clique sur le bouton, la requête est envoyée à l'API.

La première chose que la requête fait est de vérifier les caractères qui ont été utilisés afin de détecter tout caractère spécial qui pourrait indiquer une attaque par injection. Si la requête passe cette étape, elle est envoyée à l'API.

Ensuite, si des utilisateurs correspondent, le client reçoit les données sous forme de JSON et les affiche. Sinon, certaines erreurs sont gérées et des messages appropriés sont renvoyés.

### Votre page d’administration devra être protégée contre les injections SQL. Sans utiliser sequelize ou tout autre ORM, votre site devra être robuste face au…

Pour valider ce point, je suis parti d'un principe simple : pour mener des attaques par injection, il est nécessaire d'utiliser des caractères spéciaux. C'est pourquoi ils ne sont pas autorisés dans cette barre de recherche. Voici le code qui me permet de réaliser cela.

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

Comme le montre le code ci-dessus, la toute première chose que le code fait est de vérifier si ce que l'utilisateur a entré dans la barre de recherche contient des caractères spéciaux.

C'est ainsi que j'ai conçu mon code pour se prémunir contre les attaques par injection.

# Conclusion

## Bilan personnel

Personnellement, j'ai beaucoup aimé ce projet car il y a un lien direct avec ce qui a été vu durant le module. Malheureusement, je me suis pris beaucoup de retard et heureusement qu'il y a eu l'évaluation à 80% pour me réveiller. Je trouve qu'il y avait des points peu clairs dans ce projet, ce qui m'a porté préjudice.

Par exemple, nous n'avions pas le droit d'utiliser Sequelize, pourtant le seul endroit où j'ai clairement vu que nous n'étions pas censés l'utiliser, c'est dans mon évaluation à 80%, si je ne me trompe pas.