P\_APP-183



Rotzetter Ethan – MID2A

Lausanne, Venne

24 P

Gaël Sonney

Table des matières

[1. conceptualisation (schéma) 2](#_Toc192510301)

[2. explications sur le code 3](#_Toc192510302)

[1.1 Profil du client 3](#_Toc192510303)

[1.2 HTTPS 4](#_Toc192510304)

[1.3 Authentification par mot de passe 5](#_Toc192510305)

[1.4 Vérification du token JWT 8](#_Toc192510306)

[1.5 Administration 9](#_Toc192510307)

[1.6 Protection contre les injections SQL 12](#_Toc192510308)

[1.7 Utilisation de bcrypt 12](#_Toc192510309)

[1.8 Versioning 13](#_Toc192510310)

[3. conclusion sur le travail fourni et sur l’attitude face au projet 13](#_Toc192510311)

[4. Webographie 13](#_Toc192510312)

# conceptualisation (schéma)

Une image contenant capture d’écran, conception

Description générée automatiquement

Figure Schéma

# explications sur le code

## Profil du client



Figure 2 Le contrôleur de profile

Le code au-dessus permet d’afficher la page de profile si l’utilisateur est connecté. Il permet aussi de déconnecter en effaçant les cookies avec une redirection vers la page d’accueil.



Figure Le code du Router profile

Le code ci-dessus permet de vérifier si un utilisateur qui veut aller sur la page profile est connecté, c’est vérifié à l’aide de la méthode « auth », qui reprend le token présent dans le cookie. Si l’utilisateur n’est pas connecté, ce-dernier sera redirigé vers la page de connexion. Si l’utilisateur a les droits administrateurs, alors un lien indisponible pour les autres apparaîtra. Si une erreur est attrapée, le message d’erreur apparaitra dans la console et l’utilisateur sera redirigé sur la page d’accueil.

## HTTPS

Voici les étapes de réalisation pour avoir HTTPS :

Je commence par faire cette commande :

openssl genpkey -algorithm RSA -out privkey.key

Une image contenant texte, capture d’écran, Police

Le contenu généré par l’IA peut être incorrect.

Puis celle là:

openssl req -new -key privkey.key -out request.csr

Une image contenant texte, Appareils électroniques, capture d’écran, logiciel

Le contenu généré par l’IA peut être incorrect.

Et pour finir :

openssl x509 -req -in request.csr -signkey privkey.key -out certificate.crt -days 365

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

Le contenu généré par l’IA peut être incorrect.

**Résumé des commandes:**

générer une clé privée :

- openssl genpkey -algorithm RSA -out privkey.key

créer une demande de signature de certificat (CSR) :

- openssl req -new -key privkey.key -out request.csr

signer cette demande pour produire un certificat auto-signé :

- openssl x509 -req -in request.csr -signkey privkey.key -out certificate.crt -days 365

## Authentification par mot de passe





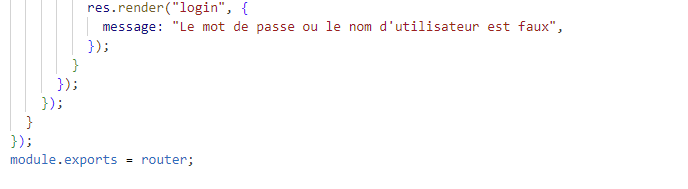
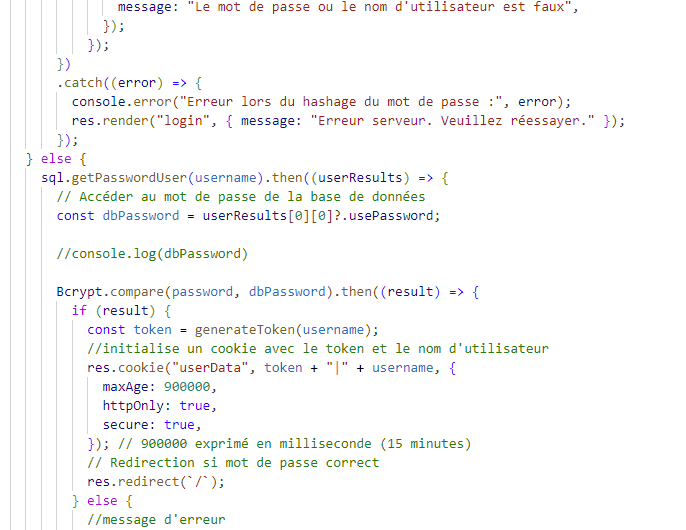


Figure 4 Images du backend de login

Au début de ce présentation du code, il a deux routes, la première pour afficher la page de connexion et la seconde pour chercher et connecter un utilisateur. Lorsqu’un utilisateur entre ses informations de connexion et les valide, ces informations sont récupérées puis un test est fait pour savoir si le nom d’utilisateur saisi est une lettre dont la position est un nombre pair ou impair. Si le nombre retourné par la fonction est paire alors on sait que le mot de passe de l’utilisateur à été haché selon la première méthode (la méthode faite maison). Autrement c’est l’autre méthode qui a été utilisé (avec Bcrypt).

Avec la première méthode, le mot de passe saisi par l’utilisateur est haché puis on retrouve dans la base de données le mot de passe haché. Ensuite le sel est extrait du mot de passe de la base de données, et le code teste si les deux mots de passes correspondent. Si oui, alors l’utilisateur sera connecté et un cookie contenant le token et le nom d’utilisateur sera initialisé. Si non, l’utilisateur une erreur indiquant que soit le mot de passe soit le nom d’utilisateur est faux.

Dans la seconde méthode, une librairie externe est utilisée, il s’agit bien évidemment de Bcrypt. Une méthode de Bcrypt consiste à comparer deux mot deux passe et renvoie une valeur booléenne. Si la valeur renvoyée est juste alors un nouveau cookie sera initialiser avec un token et le nom d’utilisateur. Et l’utilisateur sera renvoyé (connecté) vers la page d’accueil. Si c’est faux, l’utilisateur aura une erreur indiquant que soit le mot de passe soit le nom d’utilisateur est faux.

## Vérification du token JWT



Figure Middleware "auth"

Ce middleware vérifie la validité d’un token extrait d’un cookie. Ce code utilise la librairie JsonWebToken. Si le token a pu être décodé, l’utilisateur pourra aller sur la page protégée. Dans le cas contraire, une redirection sera faite pour qu’il puisse se connecter.

## Administration



Figure Contrôleur de Admin

L’image ci-dessus montre dévoile le code pour du contrôleur <<Admin>>, ce code récupère les informations dans les cookies et s’il n’y en a pas l’utilisateur est redirigé vers la page de connexion. Si des informations sont disponibles, alors une requête SQL sera faite pour chercher une booléenne qui définit si un utilisateur est administrateur ou pas. S’il est admin, la page sera dévoilée, sinon une redirection est faite vers la page d’accueil. En cas d’erreur, l’utilisateur sera renvoyé vers une page d’erreur avec un lien pour la page d’accueil.



Figure 7 Route de Admin

Ce code permet d’afficher la page quand l’utilisateur va sur la page « /admin » et permet de faire une recherche sur tous les utilisateurs de la base de données et retourne un résultat par rapport à ce qui a été saisi dans la barre de recherche.



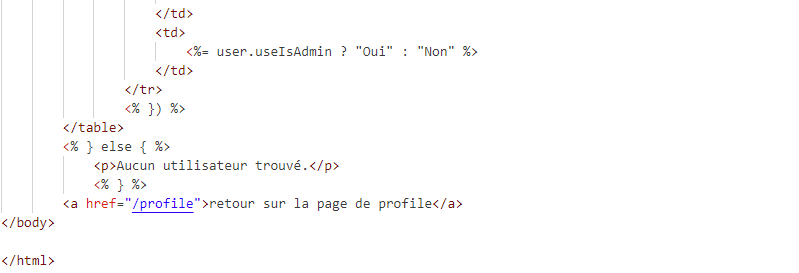


Figure 8 Page ejs de admin

Le code html ci-dessus permet d’afficher un tableau contenant le résultat d’une recherche ou un tableau contenant tous les utilisateurs. Une barre de recherche est présente et deux boutons, un pour réafficher le tableau de base et un pour valider la saisi dans la barre de recherche. Si aucun utilisateur n’a été trouvé une phrase s’affichera indiquant qu’il n’y a pas de résultat. Un lien est présent pour retourner sur la page de profile.

## Protection contre les injections SQL

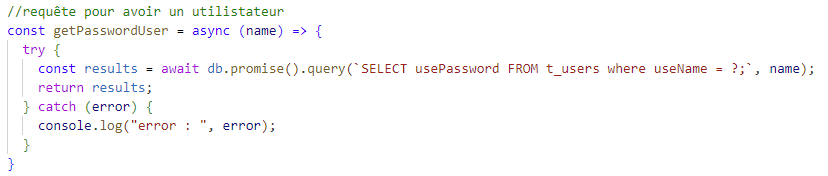


Figure Exemple de protection avec une requête pour avoir un mot de passe

Dans cet exemple, la protection contre les injections est appliquée. Dans la requête SQL, il y a un point d’interrogation qui prendra la valeur de la variable name.

## Utilisation de bcrypt

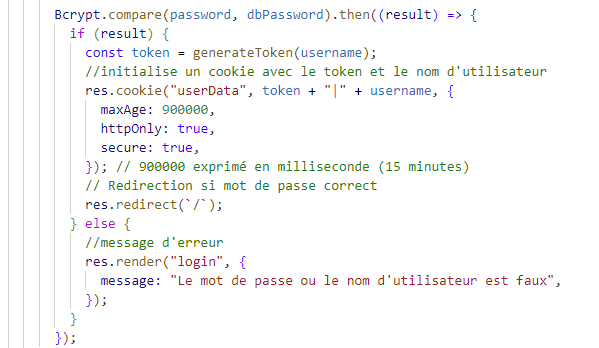


Figure Bcrypt pour le login

Le code ci-dessus permet de comparer le mot de passe saisi et celui trouvé dans la bd. Puis en fonction du résultat (True ou False) un token sera alors générer et un nouveau cookie sera défini et on renvoie l’utilisateur sur la page d’accueil.

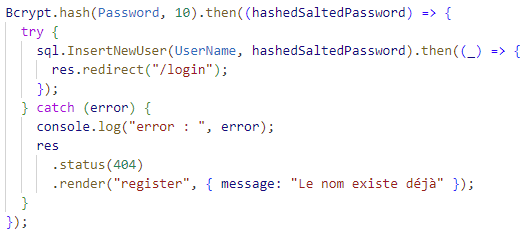


Figure Bcrypt pour le registre

Ici, une fois les informations saisies, le mot de passe est salé et haché puis un nouvel utilisateur est créé dans la base de données. Et enfin, ce-dernier est redirigé vers le login.

## Versioning

Le versioning est fait dans mon repos [GitHub](https://github.com/EthanRotzette/secured_webshop) avec les commits.

# conclusion sur le travail fourni et sur l’attitude face au projet

Je pense que le travail fournit correspond à ce qui a été demandé. Et aussi je suis fier de mon projet et des connaissances que ça m’a apporté.

Pour l’attitude face au projet, Je pense que j’ai perdu une quantité de temps absurde au début du projet car j’étais perdu comme je n’avais pas eu ou très peu de cours dessus. Mais, j’ai apprécié faire ce projet malgré un peu de stresse face à la quantité de travail.

# Webographie

Voici une liste approximative des sites utilisés pour le projet :

(Je mentionne des sites dans le code)

* [Créer un serveur HTTPS | NodeJS](https://node-js.fr/server/ssl.html)
* [Node JS fs.readFileSync() Method - GeeksforGeeks](https://www.geeksforgeeks.org/node-js-fs-readfilesync-method/)
* [ChatGPT](https://chatgpt.com/)