P\_Secured\_webshop

AUTEUR : Nicola Golaz

Classe : MID2A

Table des matières

[Conceptualisation (schéma) 3](#_Toc192868089)

[Explication du code 4](#_Toc192868090)

[Configuration HTTPS 4](#_Toc192868091)

[Authentification 4](#_Toc192868092)

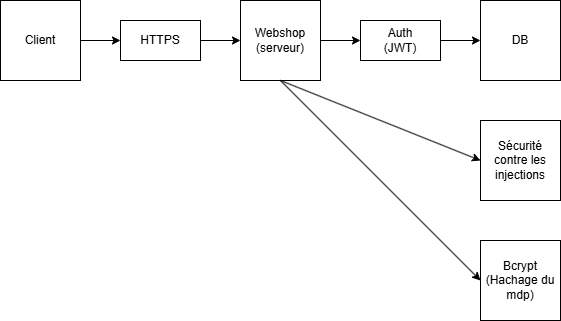
[Administration 8](#_Toc192868093)

[Page de profile 8](#_Toc192868094)

[Protection contre les injections 10](#_Toc192868095)

[Conclusion 11](#_Toc192868096)

# Conceptualisation (schéma)



# Explication du code

## Configuration HTTPS

Pour mettre en place l’accès au site de manière sécurisé sur un serveur https avec un certificat auto-signé par OpenSSL, voici les trois étapes à suivre :

1. Création d’une clé privée :

openssl genrsa -out certificat/key.pem 2048

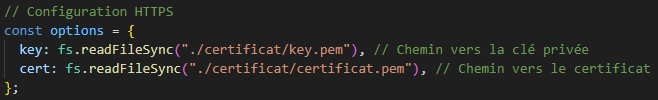
1. Création d’une demande de signature de certificat (CSR) :

openssl req -new -key certificate/key.pem -out certificate/cerificat.csr

1. Generation du certificate auto-signé :

openssl x509 -req -days 365 -in certificate/certificate.csr -signkey certificate/key.pem -out certificat/certificat.pem

Une fois en possession du certificat, il faut l’implémenter dans le code :

La clé privée et le certificat sont récupérée et mis dans la constante « options », et sont ensuite utilisés lors de la création du serveur https.



## Authentification

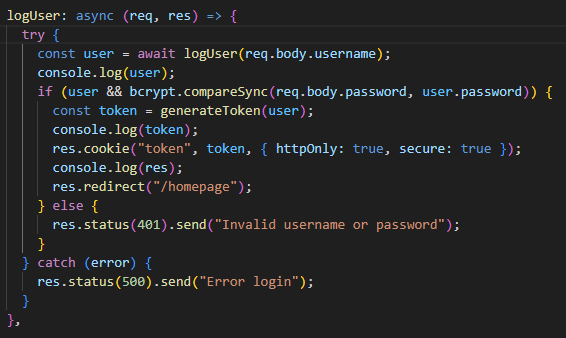
**Bcrypt**

**Bcrypt** est un algorithme de hachage sécurisé conçu pour stocker des mots de passe. Il utilise un **sel** (salt) aléatoire et un facteur de coût ajustable pour ralentir les attaques par force brute. Dans le cadre de ce projet il est utilisé pour hasher les mots de passe lors de la création d’un compte, et aussi pour comparer les hash pendant la connexion.

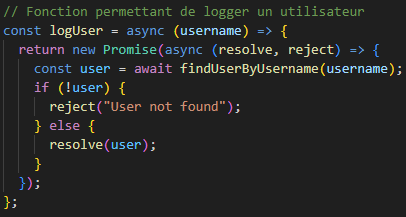
**Login**

L’utilisateur se connecte au « secured\_webshop » en utilisant son nom d’utilisateur et son mot de passe. Voici les trois fonctions nécessaires à la connexion d’un utilisateur :

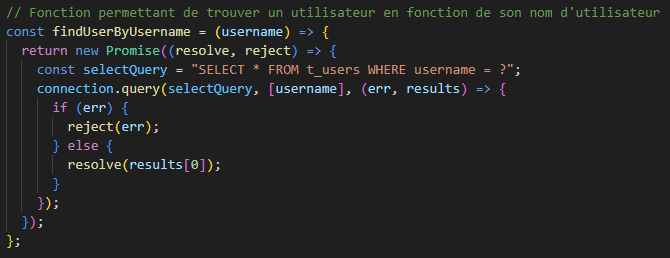
UserController/logUser



mysql/logUser



mysql/findUserByUsername



La première étape est de récupérer l’utilisateur dans la base de données, via son nom d’utilisateur. Ensuite le hash du mot de passe entré par l’utilisateur est comparé avec le hash du mot de passe de la base de données, avec bcrypt.

Si les hash correspondent, un token JWT est générer, et est ajouté dans les cookies. Ce token permettra d’identifier l’utilisateur pendant sa navigation entre les pages du sites web et de vérifier que seules les requêtes authentifiées peuvent accéder aux ressources protégées.

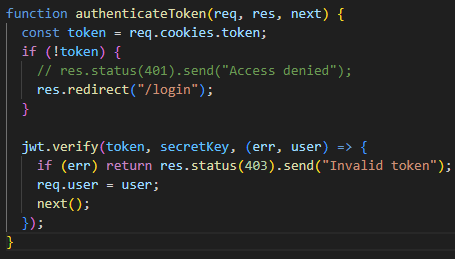
Si la connexion s’est passée correctement, l’utilisateur est redirigé vers la « homepage ».

Si l’utilisateur n’a pas pu être connecté, une erreur est levée (« Invalid username or password » ou « Error login »).

Les routes permettant l’accès à la « homepage » et à la page de profile sont sécurisée en utilisant la fonction authenticateToken, qui vérifie que le token est valide à chaque fois que l’utilisateur passe par ces routes.



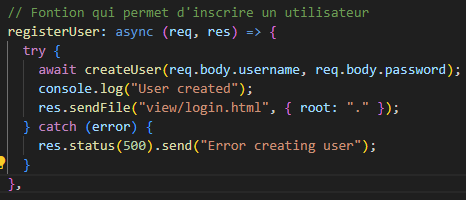
jwt/authenticateToken



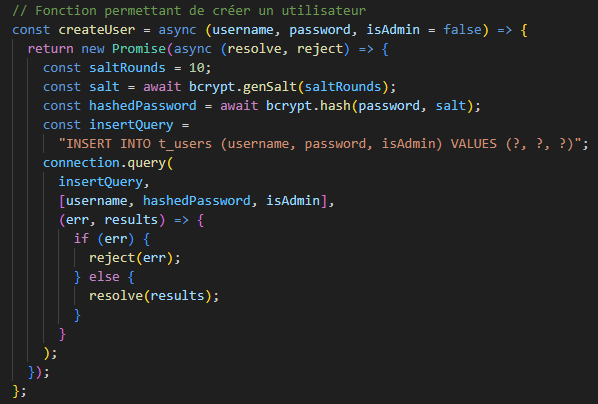
**register**

Un utilisateur se créer un compte dans le « secured\_webshop » en indiquant un nom d’utilisateur (username) et un mot de passe (password). Voici les fonctions nécessaires pour la création d’un compte utilisateur :

UserController/registerUser



mysql/createUser



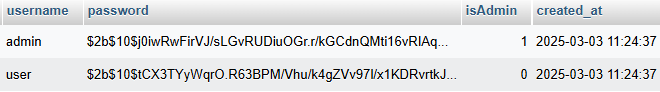
Pour commencer, l’utilisateur indique un nom d’utilisateur et un mot de passe, puis le mot de passe est salé, hashé et ajouté à la base de données avec le « username ». Le mot de passe est salé est hashé grâce à bcrypt, qui le fait automatiquement.

Je précise quand même que c’est le hash du mot de passe qui est stocké dans la base de données, et non le mot de passe en clair.

Des messages d’erreurs sont levés en fonction de la réussite de la création du compte utilisateur.

## Administration

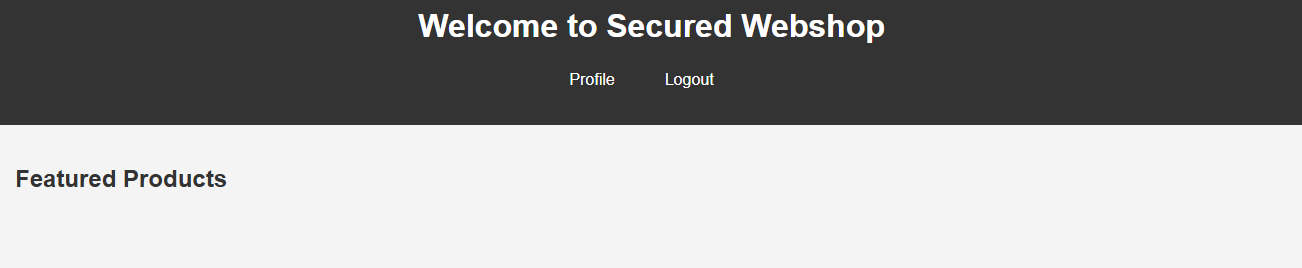
Les utilisateurs du « secured\_webshop » sont divisé en deux groupe, les simples utilisateurs, et les administrateurs. Ces deux types d’utilisateur sont différencier les uns des autres grâce à un simple booléen dans la base de données (1 = true, 0 = false), stocké avec les informations de l’utilisateur :



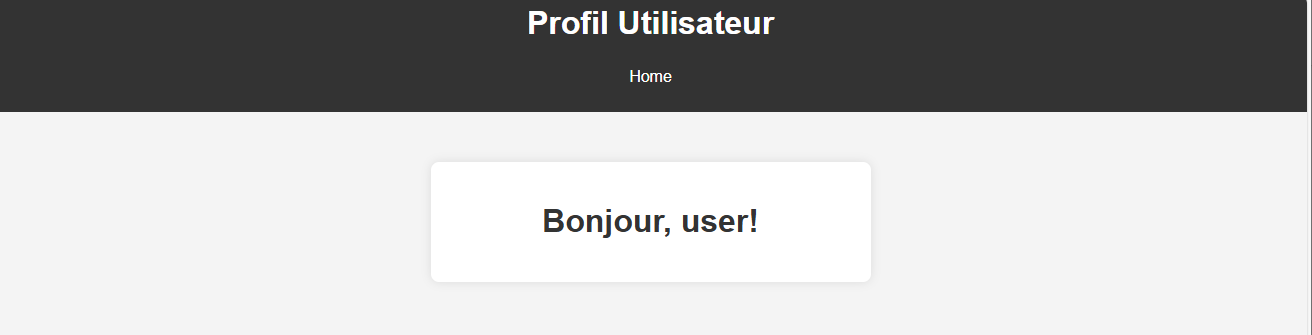
Pour créer un administrateur ou données le rôle à un utilisateur existant, on est obligé de passer directement par la base de données. Un utilisateur créer depuis le site web aura le rôle de simple utilisateur par défaut.

## Page de profile

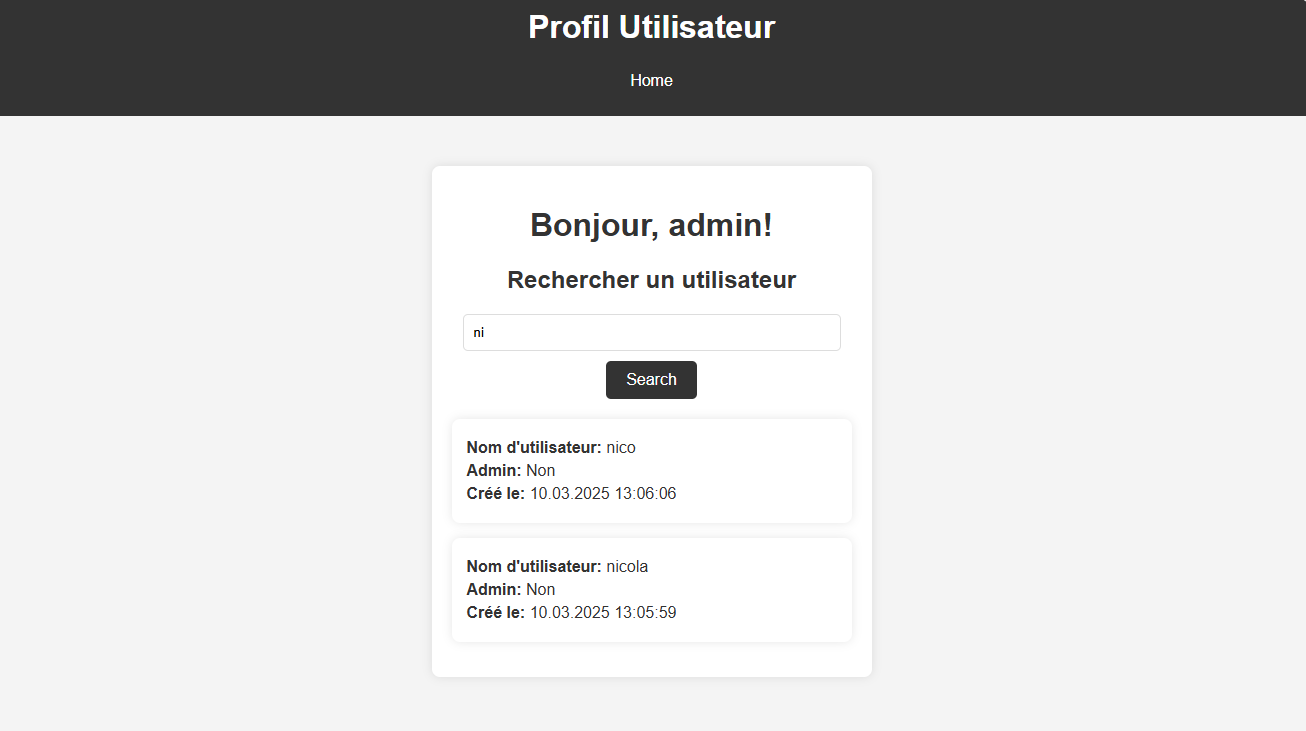
Un utilisateur connecté arrive sur la page home, il peut ensuite accéder à sa page de profile.



Voici ce que verra un simple utilisateur connecter dans la page de profile :



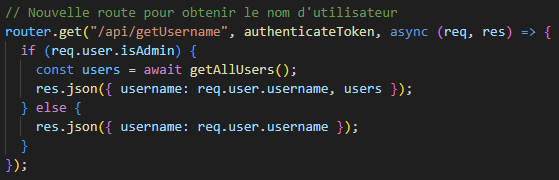
La page de profile d’un administrateur lui permet de chercher les informations des autres utilisateurs dans la base de données, via une barre de recherche.



La gestion des rôles et l’affichage des utilisateurs lors d’une recherche est implémentée de la manière suivante :

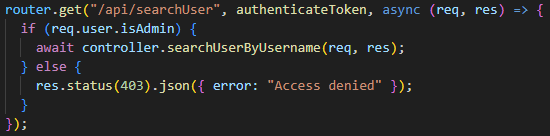
Le code récupère d'abord le nom d'utilisateur depuis le serveur via une requête GET à la route /api/getUsername. La réponse JSON met à jour le texte d'un élément HTML avec l'ID username pour afficher le nom d'utilisateur. Si la réponse contient une propriété users, cela signifie que l’utilisateur est administrateur, l'élément HTML avec l'ID admin-section devient donc visible. En cas d'erreur, celle-ci est enregistrée dans la console.

Route GET /api/getUsername

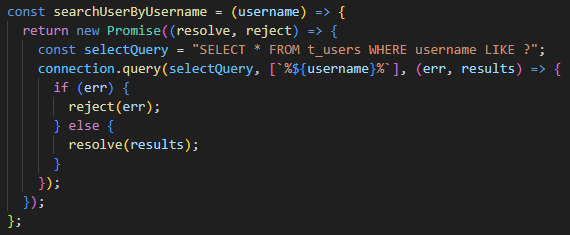


Ensuite, un écouteur d'événements est ajouté au bouton avec l'ID searchButton. Lorsqu'il est cliqué, la valeur du champ de saisie avec l'ID searchUsername est récupérée. Si le champ est vide, une alerte demande de saisir un nom d'utilisateur. Sinon, une requête GET est envoyée à la route /api/searchUser avec le nom d'utilisateur en paramètre. Si la réponse échoue, une erreur est levée. Si elle réussit, les données des utilisateurs sont traitées, les résultats précédents sont vidés, et des éléments HTML sont créés pour afficher les informations des utilisateurs, y compris le nom d'utilisateur, le statut d'administrateur et la date de création. En cas d'erreur, une alerte affiche le message d'erreur.

Route GET /api/searchUser



mysql/searchUserByUsername



## Protection contre les injections

Les deux formulaires sont protégés contre les injections SQL grâce à l'utilisation de requêtes préparées avec des paramètres. Les requêtes préparées permettent de séparer le code SQL des données entrées par l'utilisateur, ce qui empêche les attaquants d'injecter du code SQL pouvant nuire à l’application.

Exemple de requête paramétrée :



# Conclusion

La réalisation de ce projet m’a permis d’apprendre différente façon de mettre en place la sécurité dans une application web, comme sécuriser l’accès avec HTTPS, hacher les mots de passe avec Bcrypt, et authentifier les utilisateurs grâce au token JWT.

Pour conclure, ce projet m’a beaucoup appris, dans la manière de sécuriser une application web et de penser la sécurité dès le début du développement. Je vais pouvoir utiliser ces bonnes pratiques dans mes projets futurs.