Projet P\_Sécurié

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

Description générée automatiquement

Quentin Métroz – CID2A

Vennes

9 semaines

Gaël Sonney

Table des matières

[1 Conceptualisation 2](#_Toc166662864)

[1.1 MCD 2](#_Toc166662865)

[1.2 MLD 3](#_Toc166662866)

[1.3 MPD 3](#_Toc166662867)

[1.4 Schéma d’une requête 3](#_Toc166662868)

[2 Explications du code 4](#_Toc166662869)

[2.1 Fichier pour le serveur 4](#_Toc166662870)

[2.2 Connection à la base de données 4](#_Toc166662871)

[2.3 Routes pour utilisateurs 5](#_Toc166662872)

[2.4 Routes pour le login 6](#_Toc166662873)

[2.5 Explication de OpenSSL 8](#_Toc166662874)

[2.5.1 Commande pour la création des fichiers 8](#_Toc166662875)

[2.5.2 Question et Réponse 8](#_Toc166662876)

[2.5.3 Utilisation des fichiers 9](#_Toc166662877)

[2.6 Explication du Token 9](#_Toc166662878)

[2.7 Explication de la base de données et du Hash 9](#_Toc166662879)

[3 conclusion 9](#_Toc166662880)

[3.1 Conclusion personnelle 9](#_Toc166662881)

[3.2 Piste d’amélioration 9](#_Toc166662882)

[4 annexe 9](#_Toc166662883)

# Conceptualisation

## MCD

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

Description générée automatiquement

## MLD

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

Description générée automatiquement

## MPD

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

Description générée automatiquement

## Schéma d’une requête

Pour une requête https://localhost:443/users/1

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

Description générée automatiquement

# Explications du code

## Fichier pour le serveur

// import le framework express

import express from "express"

// import le router pour les users

import { userRouter } from "./routes/User.mjs";

// import le router pour le login

import { loginRouteur } from "./routes/Login.mjs";

// import la librairie fs pour lire des fichiers

import fs from "node:fs"

// import la librairie https pour monter le serveur

import https from "node:https"

// On fait une instanciation de express

const app = express();

// On défini les options pour monter notre serveur HTTPS

const options = { key: fs.readFileSync("cert/server.key"), cert: fs.readFileSync("cert/server.cert") };

// On crée le serveur https avec options, app et on le fait écouter le port 443

https.createServer(options, app).listen(443)

// On utilise express.json() pour transformer le json en objet JS

app.use(express.json());

// Pour les routes qui commence par https://localhost:443/users on appelle le router useRouter

app.use('/users', userRouter);

// Pour les routes qui commence par https://localhost:443/login on appelle le router loginRouter

app.use('/login', loginRouteur);

## Connection à la base de données

// On défini un variable vide

let connection;

// On essaie d'établir la connection avec la base de données

try {

    // Si vous êtes en local alors voici la bonne connection

    connection = await mysql.createConnection({

        host: "localhost",

        user: "root",

        password: "root",

        port: "3306",

        database: "db\_securityapp"

    });

    // Si vous êtes avec une application dockerisée alors voici la bonne connection

    /\*connection = await mysql.createConnection({

        host: "172.24.0.3",

        user: "root",

        password: "root",

        port: "3306",

        database: "db\_securityapp"

    });\*/

} catch(ex) {

    console.log(ex)

    throw new Error("Cannot connect to the database. Please try again later")

}

## Routes pour utilisateurs

// Première route asynchrone GET sur http://localhost:443/users avec authentification

userRouter.get('/', auth, async (req, res) => {

    // On check si l'utilisateur a passé un username dans l'url

    if (req.query.username) {

        // On limite la recherche pour éviter qu'elle soit trop gourmande en énergie

        if (req.query.name.length < 2) {

          const message = `Le terme de la recherche doit contenir au moins 2 caractères`;

          return res.status(400).json({ message });

        }

        // On mets une limite arbitraire à 3

        let limit = 3;

        // Si l'utilisateur donne une limite alors on réaffecte la variable

        if (req.query.limit) {

          limit = parseInt(req.query.limit, 10);

        }

        // Défini le message de réussite

        const message = "Voici la liste des utilisateurs récupéré";

        // Prend le user avec le username

        const results = await getAllUserLike(connection, req.query.username);

        // Envoie le message et les données sous format json

        res.json(success(message, results));

    }

    const message = "Voici la liste des utilisateurs récupéré";

    // On effectue une commande SQL si elle rate ou un problème durant la requête alors on affiche une erreur

    try {

        // On effectue une requête pour avoir tout les utilisateurs

        const results = await getAllUser(connection);

        // On l'envoie sous format json

        res.json(success(message, results));

    } catch (error) {

        // On défini un message d'erreur

        const message = "Erreur lors de la récupération des utilisateurs" + error;

        // On l'envoie sous format json

        res.status(500).json({ message });

    }

});

## Routes pour le login

// Cette fonction gére l'authentification pour une route

const auth = (req, res, next) => {

    // On récupére le jeton d'authentification avec le Bearer

    const authorizationHeader = req.headers.authorization;

    // On check si le jeton est vide

    if (!authorizationHeader) {

        // On défini un message d'erreur

        const message = `Vous n'avez pas fourni de jeton d'authentification. AJoutez-en dans l'en-tête de la requête.`;

        // On retourne un statu 401 et le message en format JSON

        return res.status(401).json({ message });

    } else {

        // On prend le token en enlevant le bearer au début

        const token = authorizationHeader.split(" ")[1];

        // On vérifie le token

        const decodedToken = jwt.verify(

            token,

            privateKey,

            (error, decodedToken) => {

                // Si il y a une erreur dans le décodage du token alors on donne une erreur

                if (error) {

                    // On défini un message d'erreur

                    const message = `L'utilisateur n'est pas autorisé à accéder à cette ressource.`;

                    // On retourne un statu 401 et le message, erreur en format json

                    return res.status(401).json({ message, data: error });

                }

                // On met l'id dans une variable

                const userId = decodedToken.userId;

                // On check si l'id donner dans le body n'est pas vide et si il correspond pas au celui du token

                if (req.body.userId && req.body.userId !== userId) {

                    // On défini un messgage d'erreur

                    const message = `L'identifiant de l'utilisateur est invalide`;

                    // On renvoie un statu 401 et le message sous format json

                    return res.status(401).json({ message });

                } else {

                    // On met le boolean de admin dans une variable

                    const isAdmin = decodedToken.admin

                    // Si il est pas admin alors on regarde la methode

                    if(!isAdmin) {

                        // C'est la méthode GET et qu'il a pasé un id pour autoriser unqieument que cett route à l'utilisateur

                        if(req.method == "GET" && req.params.id) {

                            // On check si l'id de l'utilisateur est égal à l'id passé dans le body

                            if(req.params.id != userId) {

                                // On défini le message d'erreur

                                const message = `L'utilisateur n'est pas autorisé à accéder à cette ressource.`;

                                // On retourne le statu 401 et le message et l'erro sous format json

                                return res.status(401).json({ message, data: error });

                            }

                        } else {

                            // On déini le message d'erreur

                            const message = `L'utilisateur n'est pas autorisé à accéder à cette ressource.`;

                            // On retourne le statu 401 et le message et l'erro sous format json

                            return res.status(401).json({ message, data: error });

                        }

                    }

                    // Si tout se passe bien alors l' utilisateur est autorisé à éxécuter

                    next();

                }

            }

        );

    }

};

## Explication de OpenSSL

### Commande pour la création des fichiers

openssl req -nodes -new -x509 -keyout server.key -out server.cert

### Question et Réponse

* Country Name (2 letter code) [AU]: CH
* State or Province Name (full name) [Some-State]:Vaud
* Locality Name (eg, city) []:Lausanne
* Organization Name (eg, company) [Internet Widgits Pty Ltd]:ETML
* Organizational Unit Name (eg, section) []:ETML
* Common Name (e.g. server FQDN or YOUR name) []:Quentin
* Email Address []:[quentin.metroz@eduvaud.ch](mailto:quentin.metroz@eduvaud.ch)

### Utilisation des fichiers

Nous utilisons les fichiers en options pour la création du serveur HTTPS. Qui permet donc de faire du https au lieu de http.

## Explication du Token

Le but du token est d’identifier l’utilisateur. Nous créerons ce token avec la librairie [jsonwebtoken](https://www.npmjs.com/package/jsonwebtoken). Il sert donc surtout à l’authentification et à reconnaître l’utilisateur. Dans notre projet, le token permet de savoir si l’utilisateur à accès à certaine route par exemple s’il n’est pas admin il n’a pas accès à ajouter ou modifier un utilisateur.

## Explication de la base de données et du Hash

On communique avec la base de données grâce à [mysql2](https://www.npmjs.com/package/mysql2). Dans la base de données, seulement le mot de passe est hasher, le reste est en clair. Le hash est fait avec la méthode ‘sha-512’. Les mots de passes sont hasher au moment d’ajouter ou de modifier un utilisateur. La librairie utilisé s’appelle [crypto](https://www.npmjs.com/package/crypto-js).

# conclusion

## Conclusion personnelle

J'ai trouvé ce projet intéressant car il m’apprit à passer de HTTP à HTTPS.

Il y a aussi le fait de faire plus nous-même comme utiliser crypto plutôt bcrypt ou utiliser mysql2 plutôt que sequelize.

Cela permet de coder une API REST sans trop de librairie et de faire la sécurité nous-même.

## Piste d’amélioration

Je pense qu’il manque un peu de contenu sur le projet.

# annexe

Pour plus d’information sur le code : [Repo GitHub](https://github.com/quemet/P_183SecurityApp)