**Étape 1 : Configuration initiale**

**1. Installer les dépendances nécessaires :**

npm install express sequelize mysql2 dotenv bcrypt jsonwebtoken express-validator helmet csurf cookie-parser cors

npm install --save-dev nodemon

- Express : Framework web.

- Sequelize : ORM pour MySQL.

- bcrypt : Hashage des mots de passe.

- jsonwebtoken : Gestion des tokens.

- helmet : Renforce la sécurité des headers HTTP.

- csurf : Protection CSRF.

- express-validator : Validation des données.

- cookie-parser : Gestion des cookies.

- dotenv : Gestion des variables d'environnement.

**dotenv** est une bibliothèque Node.js utilisée pour charger des variables d'environnement à partir d'un fichier .env dans process.env. Cela permet de gérer de manière sécurisée les configurations sensibles, telles que les clés API, les mots de passe, ou les paramètres de connexion aux bases de données, sans les exposer dans le code source.

* **Sécurité** : Éviter de coder en dur des informations sensibles dans le code source.
* **Flexibilité** : Faciliter la configuration des variables pour différents environnements (développement, test, production).
* **Simplicité** : Les variables sont accessibles globalement via process.env.

**2. Configurer nodemon dans package.json :**

Nodemon est un outil utilisé dans le développement Node.js pour surveiller les modifications apportées aux fichiers d'une application et redémarrer automatiquement le serveur lorsque ces fichiers changent. Cela améliore l'efficacité des développeurs en éliminant la nécessité de redémarrer manuellement le serveur à chaque modification du code.

"scripts": {

"start": "node app.js",

"dev": "nodemon app.js"

}

**Lancement en développement :**

**npm run dev**

**3. Structure du projet :**

secure-app/

├── controllers/

├── dto/

├── entities/

├── middlewares/

├── repositories/

├── routes/

├── services/

├── views/

├── config/

├── server.js

├── .env

**4. Configurer .env :**

PORT=3000

DB\_HOST=localhost

DB\_USER=root

DB\_PASSWORD=

DB\_NAME=secure\_app

JWT\_SECRET=your\_jwt\_secret

JWT\_EXPIRES\_IN=1h

**Étape 2 : Modèle de données avec Sequelize**

**1. Configurer Sequelize dans config/database.js :**

const { Sequelize } = require('sequelize');

const dotenv = require('dotenv');

dotenv.config();

const sequelize = new Sequelize(process.env.DB\_NAME, process.env.DB\_USER, process.env.DB\_PASSWORD, {

host: process.env.DB\_HOST,

dialect: 'mysql',

});

module.exports = sequelize;

**2. Créer l'entité User dans entities/User.js :**

const { DataTypes } = require('sequelize');

const sequelize = require('../config/database');

const User = sequelize.define('User', {

id: {

type: DataTypes.INTEGER,

primaryKey: true,

autoIncrement: true,

},

username: {

type: DataTypes.STRING,

allowNull: false,

unique: true,

},

email: {

type: DataTypes.STRING,

allowNull: false,

unique: true,

},

password: {

type: DataTypes.STRING,

allowNull: false,

},

role: {

type: DataTypes.ENUM('user', 'admin'),

defaultValue: 'user',

},

});

module.exports = User;

**3. Synchroniser les modèles dans server.js :**

const sequelize = require('./config/database');

const Product = require('./entities/Product');

const Category = require('./entities/Category');

const User = require('./entities/User');

sequelize.sync().then(() => {

console.log('Database synchronized');

});

**Étape 3 : Gestion des Routes et des Contrôleurs**

**1. Créer les routes routes/auth.js :**

const express = require('express');

const { register, login } = require('../controllers/authController');

const router = express.Router();

router.get('/register', (req, res) => { res.render('register'); // Affiche la vue register.ejs});

router.post('/register', register);

router.post('/login', login);

module.exports = router;

2. Créer le contrôleur controllers/authController.js :

const bcrypt = require('bcrypt');

const jwt = require('jsonwebtoken');

const { validationResult } = require('express-validator');

const User = require('../entities/User');

const register = async (req, res) => {

try {

const errors = validationResult(req);

if (!errors.isEmpty()) return res.status(400).json({ errors: errors.array() });

const { username, email, password } = req.body;

const hashedPassword = await bcrypt.hash(password, 10);

const newUser = await User.create({ username, email, password: hashedPassword });

res.status(201).json({ message: 'User registered', user: newUser });

} catch (err) {

res.status(500).json({ error: 'Internal Server Error' });

}

};

const login = async (req, res) => {

try {

const { email, password } = req.body;

const user = await User.findOne({ where: { email } });

if (!user) return res.status(401).json({ error: 'Invalid credentials' });

const isMatch = await bcrypt.compare(password, user.password);

if (!isMatch) return res.status(401).json({ error: 'Invalid credentials' });

const token = jwt.sign({ id: user.id, role: user.role }, process.env.JWT\_SECRET, {

expiresIn: process.env.JWT\_EXPIRES\_IN,

});

res.cookie('jwt', token, { httpOnly: true });

res.status(200).json({ message: 'Login successful', token });

} catch (err) {

res.status(500).json({ error: 'Internal Server Error' });

}

};

module.exports = { register, login };

3. Connecter les routes dans app.js :

const express = require('express');

const path = require('path');

const helmet = require('helmet');

const csrf = require('csurf');

const cookieParser = require('cookie-parser');

const authRoutes = require('./routes/auth');

const app = express();

// Configurer EJS

app.set('view engine', 'ejs');

app.set('views', path.join(\_\_dirname, 'views'));

// Middleware

app.use(express.urlencoded({ extended: true }));

app.use(express.json());

app.use(helmet());

app.use(cookieParser());

app.use(csrf({ cookie: true })); // Protection CSRF

// Middleware global pour injecter le token CSRF dans les vues

app.use((req, res, next) => {

    res.locals.csrfToken = req.csrfToken(); // Ajoute csrfToken à res.locals

    next();

});

app.use('/auth', authRoutes);

// Lancer le serveur

const PORT = process.env.PORT || 3000;

app.listen(PORT, () => {

    console.log(`Server running on port ${PORT}`);

});

**Étape 4 : Sécurisation**

1. Protection contre les injections SQL :

- Sequelize paramètre automatiquement les requêtes pour éviter les injections SQL.

2. Protection contre XSS :

- Helmet prévient les attaques via les headers.

- Validez les entrées utilisateur avec express-validator.

3. Protection CSRF :

- Utilisez csurf pour générer des tokens CSRF.

4. Hachage des mots de passe :

- Utilisez bcrypt pour hasher les mots de passe.

5. JWT sécurisé :

- Utilisez un secret fort pour signer les tokens.

- Configurez les cookies en mode `httpOnly` pour éviter leur manipulation.

**Étape 5 : Gestion des Rôles**

1. Créer un middleware pour les rôles dans middlewares/authMiddleware.js :

const jwt = require('jsonwebtoken');

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

const token = req.cookies.jwt;

if (!token) return res.status(403).json({ error: 'Unauthorized' });

try {

const decoded = jwt.verify(token, process.env.JWT\_SECRET);

req.user = decoded;

next();

} catch (err) {

res.status(403).json({ error: 'Unauthorized' });

}

};

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

if (req.user.role !== 'admin') return res.status(403).json({ error: 'Forbidden' });

next();

};

module.exports = { isAuthenticated, isAdmin };

**Étape 6 : Frontend Views**

1. Créer la vue dans views/register.ejs :

<!-- Formulaire d'inscription -->

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Register</title>

</head>

<body>

    <h1>Register</h1>

    <form action="/auth/register" method="POST">

        <!-- Si vous utilisez CSRF, ajoutez le token -->

        <% if (csrfToken) { %>

            <input type="hidden" name="\_csrf" value="<%= csrfToken %>">

        <% } %>

        <label for="username">Username:</label>

        <input type="text" id="username" name="username" required>

        <br>

        <label for="email">Email:</label>

        <input type="email" id="email" name="email" required>

        <br>

        <label for="password">Password:</label>

        <input type="password" id="password" name="password" required>

        <br>

        <button type="submit">Register</button>

    </form>

</body>

</html>

Créer la vues dans views/login.ejs :

<!-- Formulaire de connexion -->

<form action="/auth/login" method="POST">

<input type="email" name="email" placeholder="Email" required />

<input type="password" name="password" placeholder="Password" required />

<button type="submit">Login</button>

</form>