TP: Implémentation d'un Système d'Authentification avec Node.js, Prisma et JWT

Votre Nom

November 7, 2024

Introduction

Ce TP a pour but d'implémenter un système d'authentification avec les fonctionnalités suivantes :

- Inscription d'un utilisateur (register)
- Connexion d'un utilisateur (login)
- Déconnexion d'un utilisateur (logout)

Nous allons utiliser **Node.js** pour le backend, **Prisma** pour la gestion de la base de données, et **JWT (JSON Web Token)** pour la gestion des sessions.

Prérequis

Avant de commencer, vous devez avoir les outils suivants installés :

- Node.js
- Prisma
- Express.js
- JWT (jsonwebtoken)
- Une base de données (SQLite ou PostgreSQL)

Configuration du projet

1. Initialisation du projet Node.js

Dans un terminal, créez un nouveau dossier et initialisez un projet Node.js :

```
mkdir auth-system
cd auth-system
npm init -y
```

2. Installation des dépendances

Installez les dépendances nécessaires :

```
npm install express prisma jsonwebtoken bcryptjs dotenv
```

3. Configuration de Prisma

Exécutez la commande suivante pour initialiser Prisma :

```
npx prisma init
```

Cela va créer un fichier `prisma/schema.prisma`. Modifiez ce fichier pour définir votre modèle d'utilisateur :

```
generator client {
  provider = "prisma-client-js"
}

datasource db {
  provider = "sqlite"
  url      = "file:./dev.db"
}

model User {
  id        Int    @id @default(autoincrement())
  email     String @unique
  password  String
  createdAt DateTime @default(now())
}
```

Ensuite, effectuez la migration pour créer la base de données :

```
npx prisma migrate dev --name init
```

4. Création des fichiers principaux

Dans le dossier `src`, créez les fichiers suivants :

- `index.js` : Le point d'entrée de votre application.
- `authController.js` : Le contrôleur contenant les fonctions d'authentification.
- `authRouter.js` : Le routeur pour gérer les routes d'authentification.
- `authMiddleware.js` : Le middleware pour vérifier les tokens JWT.

Implémentation

1. Contrôleur d'authentification

Voici le code pour `authController.js` :

```
import bcrypt from 'bcryptjs';
import jwt from 'jsonwebtoken';
import { PrismaClient } from '@prisma/client';
import dotenv from 'dotenv';

dotenv.config();

const prisma = new PrismaClient();
const JWT_SECRET = process.env.JWT_SECRET || 'secret_key';

export async function register(req, res) {
    const { email, password } = req.body;
    const existingUser = await prisma.user.findUnique({ where: { email } });
    if (existingUser) {
        return res.status(400).json({ message: 'Email déjà utilisé' });
    }
```

```
    const hashedPassword = await bcrypt.hash(password, 10);
    const newUser = await prisma.user.create({
        data: {
            email,
            password: hashedPassword,
        },
    });

    res.status(201).json({ message: 'Utilisateur créé avec succès', userId: newUse
}

export async function login(req, res) {
    const { email, password } = req.body;
    const user = await prisma.user.findUnique({ where: { email } });
    if (!user) {
        return res.status(400).json({ message: 'Utilisateur introuvable' });
    }

    const validPassword = await bcrypt.compare(password, user.password);
    if (!validPassword) {
        return res.status(400).json({ message: 'Mot de passe incorrect' });
    }

    const token = jwt.sign({ userId: user.id }, JWT_SECRET, { expiresIn: '1h' });

    res.status(200).json({ message: 'Connexion réussie', token });
}

export async function logout(req, res) {
    // Ici, avec JWT, vous n'avez pas besoin de supprimer le token côté serveur.
    // Le client peut simplement supprimer le token de son stockage local ou cooki
    res.status(200).json({ message: 'Déconnexion réussie' });
}
```

2. Routeur d'authentification

Voici le code pour `authRouter.js` :

```
import express from 'express';
import { register, login, logout } from './authController.js';

const router = express.Router();

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

export default router;
```

3. Middleware pour vérifier le token

Voici le code pour `authMiddleware.js` :

```
import jwt from 'jsonwebtoken';
import dotenv from 'dotenv';

dotenv.config();

const JWT_SECRET = process.env.JWT_SECRET || 'secret_key';

export function authenticateToken(req, res, next) {
    const token = req.headers['authorization']?.split(' ')[1];
    if (!token) return res.status(401).json({ message: 'Accès refusé' });

    jwt.verify(token, JWT_SECRET, (err, user) => {
        if (err) return res.status(403).json({ message: 'Token invalide' });
        req.user = user;
        next();
    });
}
```

4. Serveur Express

Voici le code pour `index.js` :

```
import express from 'express';
```

```
import dotenv from 'dotenv';
import authRouter from './authRouter.js';

dotenv.config();

const app = express();
const port = 3000;

app.use(express.json());

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

app.listen(port, () => {
    console.log(`Serveur démarré sur le port ${port}`);
});
```

Tests

Pour tester votre application, vous pouvez utiliser un client HTTP comme Postman ou Insomnia.

- Pour l'inscription : `POST /auth/register`
- Pour la connexion : `POST /auth/login`
- Pour la déconnexion : `POST /auth/logout`

Conclusion

Dans ce TP, nous avons mis en place un système d'authentification utilisant Node.js, Prisma, et JWT. Ce système permet l'inscription, la connexion, et la déconnexion des utilisateurs. Vous pouvez étendre ce projet en ajoutant des fonctionnalités comme la gestion des rôles ou l'authentification par OAuth.