examen partielle

francois TOYI

February 2025

1 Travail à réaliser

1.1 Implémentation du modèle UML avec prisma

```
model User {
  id        Int      @id @default(autoincrement())
  username  String   @unique
  password  String
  email     String   @unique
  tasks     Task[]
}

model Task {
  id          Int      @id @default(autoincrement())
  title       String
  description String?
  completed   Boolean  @default(false)
  userId      Int
  user        User     @relation(fields: [userId], references: [id])
}
```

1.2 Importance des migrations en prisma et écrivons la commande permettant d'appliquer la migrations initial

Les migrations dans **Prisma** sont essentielles car elles permettent de synchroniser le schéma de la base de données avec les modifications apportées au modèle de données. Elles assurent l'intégrité des données et facilitent l'évolution du modèle au fil du développement.

Commande pour appliquer la migration initiale :

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

1.3 Implémentons une routes (/register) et hachons le mode passe avant insertion

Route /register

```
this.router.get('/register', this.authController.register.bind(this.authController));
```

Logique de (/register):

```
 // import express from "express";

import UserService from "../services/UserService.js";

import bcrypt from 'bcryptjs';
import jwt from 'jsonwebtoken';
import dotenv from "dotenv";

dotenv.config()

export default class AuthController{

    userService;

    constructor(){
        // initialize user service here
        this.userService = new UserService();
    }


   async register(req,res) {
        const data = req.body;

        try {
            const hashedPassword = await bcrypt.hash(data.password, 10);
            data.password = hashedPassword;
            const user = await this.userService.create(data);
            res.status(status.HTTP_200_OK).json(user);
        } catch (error) {
            res.status(400).json({ message: error.message });
        }
    }
}
```

1.4 Implémentons une route /login qui renvoie un token JWT en cas de succès

Exemple de route /login

```
this.router.get('/login', this.authController.login.bind(this.authController));
```

Logique pour le login

```
    // import express from "express";

import UserService from "../services/UserService.js";

import bcrypt from 'bcryptjs';
import jwt from 'jsonwebtoken';
import dotenv from "dotenv";

dotenv.config()

export default class AuthController{

    userService;

    constructor(){
        // initialize user service here
        this.userService = new UserService();
    }

    async login(req,res) {
        const {email,password} = req.body;
        // res.send("login");
        try {
            const user = await this.userService.findByEmail(email);
            // console.log(user);

            if(!user){
                return res.status(404).json({ message: "Invalid email or password" });
            }

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

            // console.log(isValidPassword);


            if (!isValidPassword) {
                return res.status(401).json({ message: "Invalid email or password" });
            }


            const payload = {
                id: user.id,
```

```
                name : user.name,
                email : user.email,
            }


            const token = jwt.sign(payload, process.env.JWT_SECRET, { expiresIn: '1h' });
            res.json({ token });
        } catch (error) {
            console.error(error);
            res.status(500).json({ message: "An error occurred while trying to login" });
        }
    }
}
```

1.5 Developpons un middleware d'authentification permettant de protéger nos routes privées .

```
dotenv.config();

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

const authMiddlware = (req,res,next) => {
    // console.log(req.originalUrl);

    if (req.originalUrl != '/auth/login') {
        // console.log("uuuuuuuuuuuuuuuuuuuuuu");
        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();
        });
    }
    // console.log(token);
    next()
}
```

En suite on ajoute notre middleware à notre projet comme suit :

```
app.use(authMiddlware)
```

1.6 Implémentons les routes pour les tâches

1.6.1 POST /tasks/:user_id : Création d'une nouvelle tâche pour l'utilisateur spécifié.

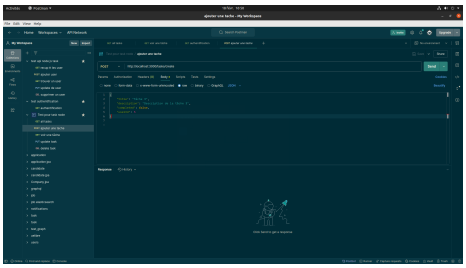

Figure 1: ajouter une tache

1.6.2 GET /tasks/:user_id : Récupération de toutes les tâches de l'utilisateur connecté.

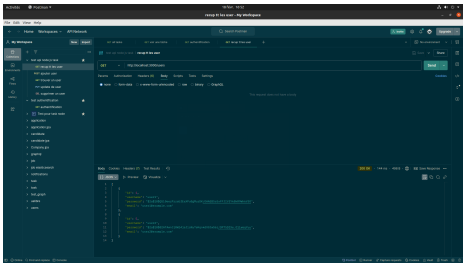

Figure 2: Toutes les tâches