САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бэк-энд разработка

Отчет

Лабораторная работа №1

Выполнил:

Борисова Элина

Группа К3341

Проверил:
Добряков Д. И.

Санкт-Петербург

2022 г.

Задача

Нужно написать свой boilerplate на express + TypeORM + typescript.
Должно быть явное разделение на:

- модели
- контроллеры
- роуты

Ход работы

Добавление авторизации и регистрации с JWT:

```
import { Request, Response, NextFunction } from 'express';
import { User } from '../entities/User';
import { AppDataSource } from '../config/dataSource';
import SETTINGS from '../config/settings';

interface AuthenticatedUser {
   id: string;
   username: string;
   email: string;
}

interface AuthenticatedRequest extends Request {
   user?: AuthenticatedUser;
}

export const authMiddleware = async (
   req: AuthenticatedRequest,
   res: Response,
   next: NextFunction
) => {
   const authHeader = req.headers.authorization;

   if (!authHeader) {
       return res.status(401).json({ error: 'Authorization header missing' });
   }

   const [tokenType, token] = authHeader.split(' ');

   if (tokenType !== SETTINGS.JWT_TOKEN_TYPE || !token) {
       return res.status(401).json({
<token>`    error: `Invalid token format. Expected: ${SETTINGS.JWT_TOKEN_TYPE}
       });
   }

   try {
userId:const decoded = jwt.verify(token, SETTINGS.JWT_SECRET_KEY) as JwtPayload & {
       const userRepository = AppDataSource.getRepository(User);

       const user = await userRepository.findOne({
           where: { id: decoded.userId },
           select: ['id', 'username', 'email']
       });
```

```
        if (!user) {
            return res.status(401).json({ error: 'User not found' });
        }

        req.user = {
            id: user.id,
            username: user.username,
            email: user.email
        };

        next();
    } catch (error) {
        if (error instanceof jwt.TokenExpiredError) {
            return res.status(401).json({ error: 'Token expired' });
        }
        if (error instanceof jwt.JsonWebTokenError) {
            return res.status(401).json({ error: 'Invalid token' });
        }
        return res.status(500).json({ error: 'Authentication failed' });
    }
};
```

Далее сделаем разделение на routes и controllers.

Пример контроллера для пользователя, регистарицей и авторизацией с токеном

```
import { Request, Response } from "express";
import { AppDataSource } from "../config/dataSource";
import { User } from "../entities/User";
import jwt from "jsonwebtoken";
import SETTINGS from "../config/settings";
import bcrypt from "bcrypt";


export const registerUser = async (req: Request, res: Response) => {
    const { username, email, password } = req.body;

    try {
        if (!username || !email || !password) {
            return res.status(400).json({ error: "Username, email, and password are required" });
        }

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

        const userRepository = AppDataSource.getRepository(User);
        const user = userRepository.create({
            username,
            email,
            password_hash: hashedPassword
        });

        await userRepository.save(user);

        const token = jwt.sign({ userId: user.id }, SETTINGS.JWT_SECRET_KEY, {
            expiresIn: "1h",
        });
```

```
        res.status(201).json({
            id: user.id,
            username: user.username,
            email: user.email,
            token: token
        });
    } catch (error) {
        console.error("Error creating user:", error);

});     res.status(500).json({ error: "Error creating user", details: error.message
    }
};

export const loginUser = async (req: Request, res: Response) => {
    const { email, password_hash } = req.body;

    try {
        const userRepository = AppDataSource.getRepository(User);
        const user = await userRepository.findOneBy({ email });

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

        const isMatch = await bcrypt.compare(password_hash, user.password_hash);
        if (!isMatch) {
            return res.status(401).json({ error: "Invalid email or password" });
        }

expiresIn: "1h" });= jwt.sign({ userId: user.id }, SETTINGS.JWT_SECRET_KEY, {
        res.json({ token });
    } catch (error) {
        res.status(500).json({ error: "Error logging in" });
    }
};

export const getUsers = async (req: Request, res: Response) => {
    try {
        const userRepository = AppDataSource.getRepository(User);
        const users = await userRepository.find();
        res.json(users);
    } catch (error) {
        res.status(500).json({ error: "Error fetching users" });
    }
};
```

User routes:

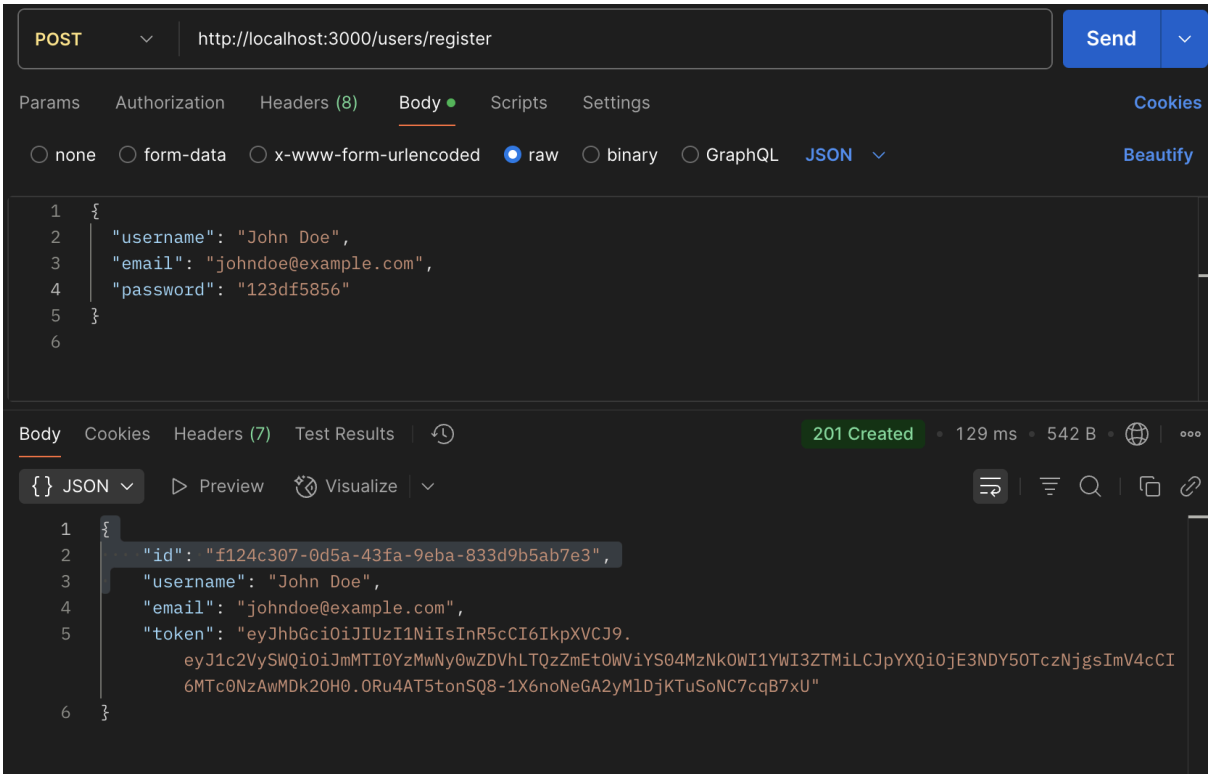
```
import { Router } from "express";
import { getUsers } from "../controllers/user.controller";
import { authMiddleware } from "../middlewares/auth.middleware";
import { loginUser, registerUser } from "../controllers/user.controller";;
const router = Router();
```

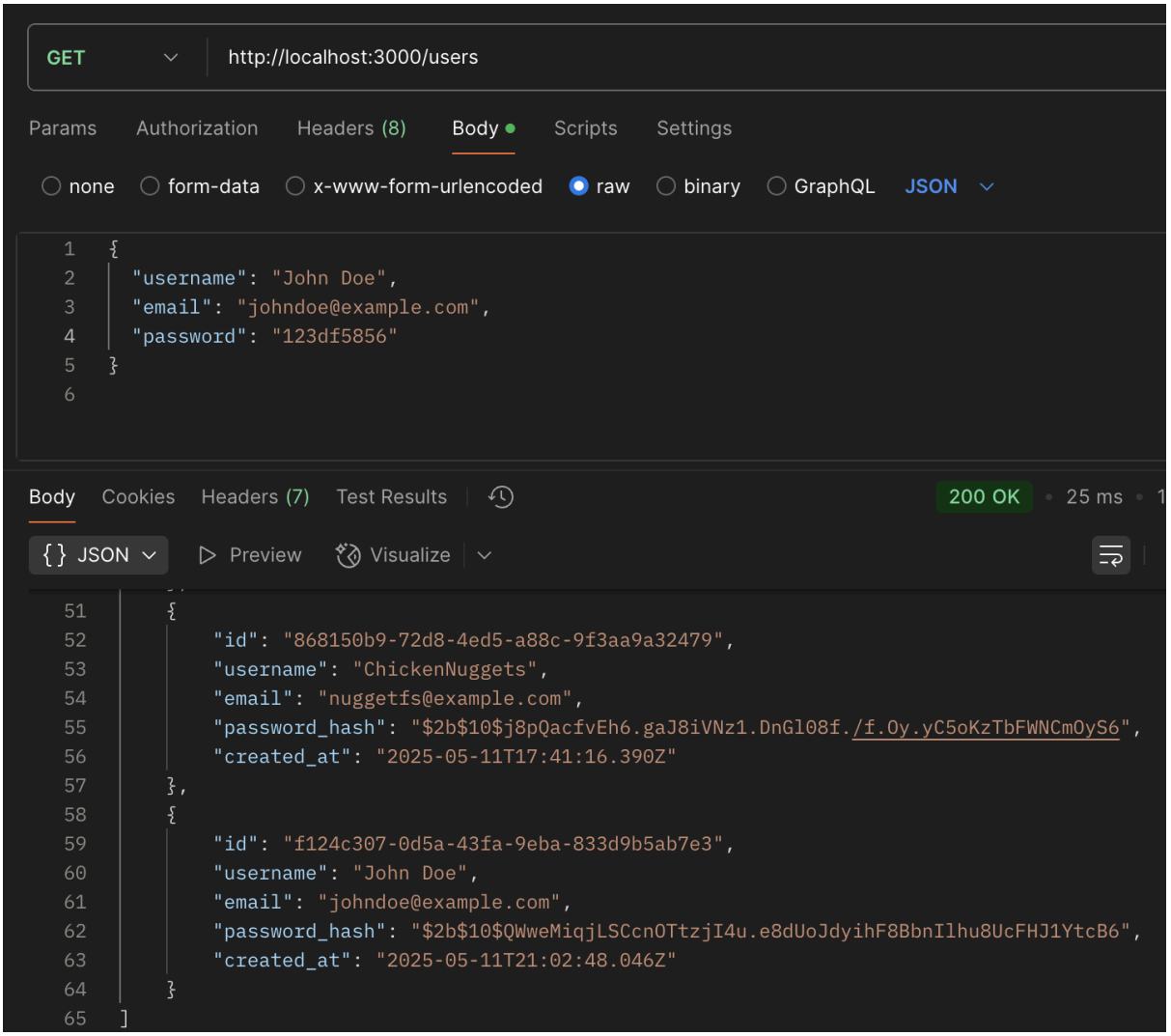
```
router.get("/", getUsers);

router.post("/login", loginUser);
router.post("/register", registerUser);

export default router;
```

Контроллеры остальных сущностей приведены в репозетории.
Проверим регистрацию пользователя:





Вывод

В данной лабораторной работе был реализован backend boilerplate на базе Express, TypeORM и TypeScript с чётким разделением на модели, контроллеры и роуты.