САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бэк-энд разработка

Отчет

Лабораторная работа №1

Выполнила:

Красюк Карина

K3341

Проверил:
Добряков Д. И.

Санкт-Петербург

2025 г.

Задача

Нужно написать свой boilerplate на express + TypeORM + typescript.

Должно быть явное разделение на:

• модели,

• контроллеры,

• роуты.

Ход работы

В ходе работы был создан boilerplate с авторизацией, аутентификацией и регистрацией. Присутствует хэширование паролей, логика разделена на контролеры и сервисы. Документация API отсутствует.

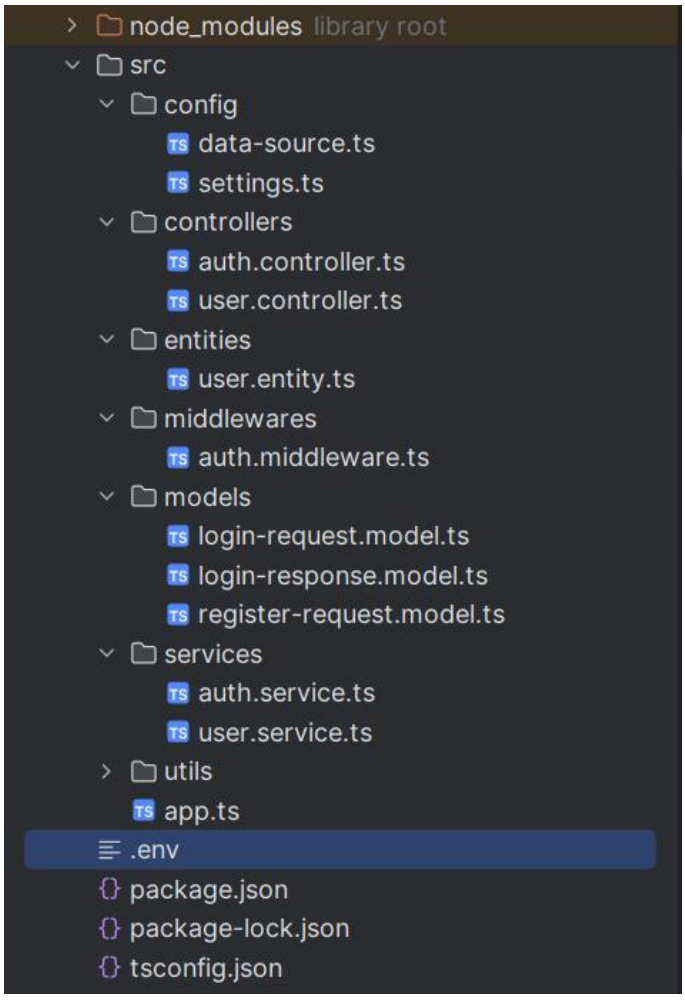


Рисунок 1 – Структура проекта

На рисунке 2,3,4 изображны AuthController, AuthService и AuthMiddleware соответственно. Они требуются для авторизации и регистрации с использованием JWT-токена.

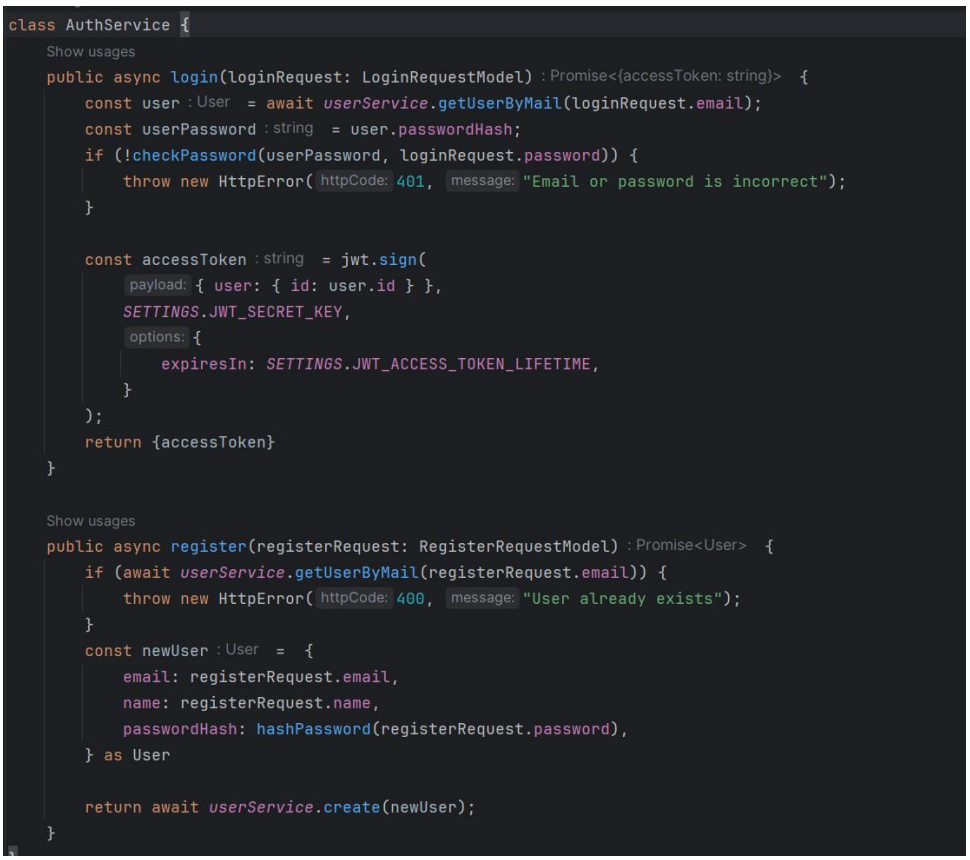
```
import {User} from "../entities/user.entity";
import {LoginRequestModel} from "../models/login-request.model";
import {LoginResponseModel} from "../models/login-response.model";
import authService from "../services/auth.service";
import {RegisterRequestModel} from "../models/register-request.model";
import {Body, Controller, Post} from "routing-controllers";


Show usages
@Controller( baseRoute: '/auth')
export class AuthController {
    no usages
    @Post( route: '/login')
    async login(@Body() loginData: LoginRequestModel): Promise<LoginResponseModel> {
        return await authService.login(loginData);
    }

    no usages
    @Post( route: "/register")
    async register(@Body() registerData: RegisterRequestModel): Promise<User> {
        return await authService.register(registerData);
    }
}

Show usages
export default AuthController;
```

Рисунок 2 – AuthController



```
class AuthService {
    Show usages
    public async login(loginRequest: LoginRequestModel) : Promise<{accessToken: string}>  {
        const user : User  = await userService.getUserByMail(loginRequest.email);
        const userPassword : string  = user.passwordHash;
        if (!checkPassword(userPassword, loginRequest.password)) {
            throw new HttpError( httpCode: 401,  message: "Email or password is incorrect");
        }

        const accessToken : string  = jwt.sign(
            payload: { user: { id: user.id } },
            SETTINGS.JWT_SECRET_KEY,
            options: {
                expiresIn: SETTINGS.JWT_ACCESS_TOKEN_LIFETIME,
            }
        );
        return {accessToken}
    }

    Show usages
    public async register(registerRequest: RegisterRequestModel) : Promise<User>  {
        if (await userService.getUserByMail(registerRequest.email)) {
            throw new HttpError( httpCode: 400,  message: "User already exists");
        }
        const newUser : User  =  {
            email: registerRequest.email,
            name: registerRequest.name,
            passwordHash: hashPassword(registerRequest.password),
        } as User

        return await userService.create(newUser);
    }
```

Рисунок 3 – AuthService

```
const authMiddleware = (
    request: RequestWithUserId,
    response: Response,
    next: NextFunction,
) : Response<any, Record<...>>  => {
    const authHeader : string  = request.headers.authorization;

    try {
        if (!authHeader || !authHeader.startsWith('Bearer ')) {
            return response
                .status( code: 401)
                .send( body: { message: 'Authorization header missing' });
        }

        const token : string  = authHeader.split( separator: ' ')[1];

        const { userId }: JwtPayloadWithUser  = jwt.verify(token, SETTINGS.JWT_SECRET_KEY) as JwtPayloadWithUser;

        request.userId = userId;

        next();
    } catch (error) {
        console.error(error);

        return response
            .status( code: 403)
            .send( body: { message: 'Forbidden: token is invalid or expired' });
    }
};
```

Рисунок 4 – AuthMiddleware

На рисунке 5,6 изображены UserController и UserService, которые требуют предварительной авторизации для их использования

```
@Controller( baseRoute: '/users')
export class UserController {
    no usages
    @Get()
    @UseBefore(authMiddleware)
    public async getUsers() : Promise<User[]>  {
        return await userService.getAll();
    };

    no usages
    @Get( route: "/me")
    @UseBefore(authMiddleware)
    public async me(@Body() request: RequestWithUserId) : Promise<User>  {
        const result : User  = await userService.getById(request.userId);
        if (!result) {
            throw new NotFoundError("User not found");
        }

        return result;
    };

    no usages
    @Get( route: '/:id')
    @UseBefore(authMiddleware)
    public async getById(@Param( name: 'id') id: string) : Promise<User>  {
        const result : User  = await userService.getById(id);
        if (!result) {
            throw new NotFoundError("User not found");
        }

        return result;
    };
```

Рисунок 5 – UserController

```
class UserService {
    private repository : Repository<User>  = dataSource.getRepository(User);

    Show usages
    public async getAll(): Promise<User[]> {
        return await this.repository.find();
    }

    Show usages
    public async getById(id: string): Promise<User> {
        const user : User  = await this.repository.findOneBy( where: {id: id});
        if (!user) return null;
        return user;
    }

    Show usages
    public async getUserByMail(email: string): Promise<User> {
        const user : User  = await this.repository.findOneBy( where: {email: email})
        if (!user) return null;
        return user;
    }

    Show usages
    public async create(user: User): Promise<User> {
        const savedUser : User  = await this.repository.save(user);
        return user;
    }
```

Рисунок 6 – UserService

Вывод

Спроектированный boilerplate станет основой для будущего REST API.