САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бэк-энд разработка

Отчет

Лабораторная работа №1

Выполнил:

Бархатова Наталья

Группа К3339

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

Я поменяла контроллеры на routing-controllers. Пример на модели Posts

```
import {
  JsonController,
  Get,
  Post as HttpPost,
  Put,
  Delete,
  Param,
  Body,
  HttpCode,
  OnUndefined,
  UseBefore
} from "routing-controllers";
import { AppDataSource } from "../AppDataSource";
import { AuthMiddleware } from '../middlewares/AuthMiddleware';
import { Post as PostEntity } from "../models/Post";

const postRepo = AppDataSource.getRepository(PostEntity);

@JsonController("/posts")
@UseBefore(AuthMiddleware)
export class PostController {
  @HttpPost()
  @HttpCode(201)
  async createPost(@Body() postData: Partial<PostEntity>) {
    const post = postRepo.create(postData);
    const savedPost = await postRepo.save(post);
    return savedPost;
  }

  @Get()
  async getAllPosts() {
    return await postRepo.find();
  }

  @Get("/:id")
```

```
  @OnUndefined(404)
  async getPostById(@Param("id") id: number) {
    const post = await postRepo.findOne({ where: { id: id.toString() },
relations: ["user"] });
    if (!post) {
      return { message: "Post not found" };
    }
    return post;
  }

  @Put("/:id")
  @OnUndefined(404)
  async updatePost(@Param("id") id: number, @Body() body:
Partial<PostEntity>) {
    const post = await postRepo.findOne({ where: { id: id.toString() } });
    if (!post) {
      return { message: "Post not found" };
    }
    postRepo.merge(post, body);
    const updatedPost = await postRepo.save(post);
    return updatedPost;
  }

  @Delete("/:id")
  @OnUndefined(404)
  async deletePost(@Param("id") id: number) {
    const result = await postRepo.delete(id);
    if (result.affected === 0) {
      return { message: "Post not found" };
    }
    return { message: "Post deleted successfully" };
  }
}
```

Остальные контроллеры изменены аналогично. Так как для такого типа контроллеров роуты создаются автоматически, я удалила папку routes.

Затем была реализована аутентификация по JWT. Для этого был реализован класс AuthService, который нужен для шифрования паролей и создания JWT-токена.

```
import jwt from 'jsonwebtoken';
import bcrypt from 'bcryptjs';
import { User } from '../models/User';
import dotenv from 'dotenv';

dotenv.config();

export class AuthService {
  private jwtSecret = process.env.JWT_SECRET as string;

  async hashPassword(password: string): Promise<string> {
    const salt = await bcrypt.genSalt(10);
```

```
    return await bcrypt.hash(password, salt);
  }

  async comparePassword(password: string, hashedPassword: string):
Promise<boolean> {
    return await bcrypt.compare(password, hashedPassword);
  }

  generateToken(user: User): string {
    const payload = { id: user.id, email: user.email };
    return jwt.sign(payload, this.jwtSecret, { expiresIn: '1h' });
  }

  async validateToken(token: string): Promise<jwt.JwtPayload> {
    return new Promise<jwt.JwtPayload>((resolve, reject) => {
      jwt.verify(token, this.jwtSecret, (err, decoded) => {
        if (err) {
          reject(new Error('Invalid token'));
        } else {
          resolve(decoded as jwt.JwtPayload);
        }
      });
    });
  }
}
```

Класс AuthMiddleware отвечает за проверку наличия header-а авторизации и валидирует токен.

```
import { ExpressMiddlewareInterface, Middleware } from 'routing-
controllers';
import { BadRequestError } from 'routing-controllers';
import { AuthService } from '../services/AuthService';

export class AuthMiddleware implements ExpressMiddlewareInterface {
  private authService = new AuthService();

  async use(req: any, res: any, next: Function): Promise<any> {
    const token = req.headers['authorization']?.split(' ')[1];
    if (!token) {
      throw new BadRequestError('No token provided');
    }

    try {
      const decoded = this.authService.validateToken(token);
      req.user = decoded;
      next();
    } catch (error) {
      throw new BadRequestError('Invalid token');
    }
  }
}
```

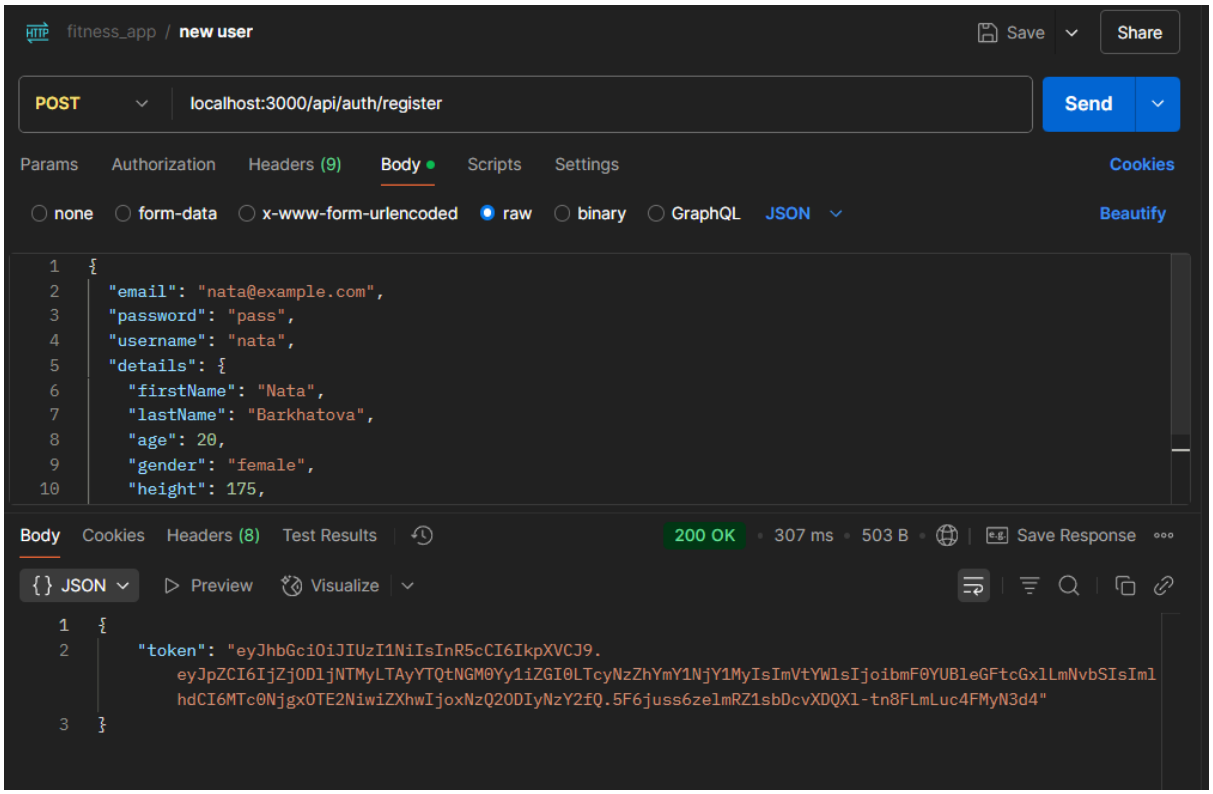
Класс AuthController содержит эндпоинты для логина и регистрации пользователя.

```
import { JsonController, Post, Body, BadRequestError } from 'routing-
controllers';
import { User } from '../models/User';
import { UserDetails } from '../models/UserDetails';
import { AppDataSource } from '../AppDataSource';
import { AuthService } from '../services/AuthService';

@JsonController('/auth')
export class AuthController {
  private userRepository = AppDataSource.getRepository(User);
  private userDetailsRepository = AppDataSource.getRepository(UserDetails);
  private authService = new AuthService();

  @Post('/register')
  async register(@Body() userData: { email: string; password: string;
username: string; details: UserDetails }) {
    const { email, password, username, details } = userData;

    const existingUser = await this.userRepository.findOne({ where: [{
email }] });
    if (existingUser) {
      throw new BadRequestError('User with this email already exists');
    }

    const hashedPassword = await this.authService.hashPassword(password);

    const user = this.userRepository.create({
      email,
      password: hashedPassword,
      username,
      details: this.userDetailsRepository.create(details)
    });

    await this.userRepository.save(user);

    const token = this.authService.generateToken(user);
    return { token };
  }

  @Post('/login')
  async login(@Body() loginData: { email: string; password: string }) {
    const { email, password } = loginData;

    const user = await this.userRepository.findOne({ where: { email },
relations: ['details', 'posts', 'progress', 'trainingPlans'] });
    if (!user) {
      throw new BadRequestError('Invalid credentials');
    }
```

```
    const isPasswordValid = await
this.authService.comparePassword(password, user.password);
    if (!isPasswordValid) {
      throw new BadRequestError('Invalid credentials');
    }

    const token = this.authService.generateToken(user);
    return { token };
  }
}
```

На контроллеры, которые подразумевают использование только после аутентификации, я навесила аннотацию `@UseBefore(AuthMiddleware)`

Демонстрация работы JWT. Создаем нового юзера, в качестве ответа получаем токен.



Пароль хранится в бд в зашифрованном виде

3	6c89c532-02a4-4c4c-bdb4-7276abf56653	nata@example.com	$2b$10$eoUjYt6d.Zq1xbJNgCUbYuka7K8mlu0X4mXd2Ha3GPfE.sWJJqs...	nata

Теперь при указании этого токена в header, запрос будет проходить дальше.

Вывод

В ходе работы была реализована JWT-аутентификация с использованием routing-controllers. Создан сервис для генерации и проверки токенов, настроен middleware для защиты маршрутов. Реализованы регистрация и логин с возвратом JWT. Это обеспечило безопасный доступ к API только для авторизованных пользователей.