САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бэк-энд разработка

Отчет

Лабораторная работа №1

Выполнил:

Гнеушев Владислав

К3339

Проверил:
Добряков Д. И.

Санкт-Петербург

2025 г.

Задача

Написать свой boilerplate на express + TypeORM + typescript.

Должно быть явное разделение на:

- модели
- контроллеры
- роуты

Ход работы

Структура проекта получилась со следующим разделением:

- Модели
- Контроллеры
- Сервисы
- Роуты
- Мидлвари
- Хендлеры ошибок
- Конфиги (настройки из переменных окружения + источник данных)

В качестве примера были реализованы эндпоинты /api/users/register и /api/users/login, использующие JWT для авторизации пользователя.

Реализация их роутов:

```
const router = Router();
const userController = new UserController();

router.post(
    '/register',
    validationMiddleware(RegisterUserDto),
    async (req: Request, res: Response, next: NextFunction) => {
        const { email, password } = req.body as RegisterUserDto;

        try {
            const result = await userController.register({ email, password });
            res.status(201).json(result);
        } catch (error) {
            if (error instanceof HttpError) {
                res.status(error.statusCode).json({
                    message: error.message,
                    ...(error.details && { errors: error.details }),
                });
            } else {
                next(error);
            }
        }
    }
);
```

```
router.post(
    '/login',
    validationMiddleware(LoginUserDto),
    async (req: Request, res: Response, next: NextFunction) => {
        const { email, password } = req.body as LoginUserDto;

        try {
            const result = await userController.login({ email, password });
            res.status(200).json(result);
        } catch (error) {
            if (error instanceof HttpError) {
                res.status(error.statusCode).json({
                    message: error.message,
                    ...(error.details && { errors: error.details }),
                });
            } else {
                next(error);
            }
        }
    }
);
```

Реализация их контроллера:

```
export class UserController {
    private authService = new AuthService();
    private userRepository = myDataSource.getRepository(User);

    async register(userData: { email?: string; password?: string }): Promise<RegisterSuccessResponse> {
        const { email, password } = userData;

        if (!email || !password) {
            throw new BadRequestError('Email and password are required');
        }

        const existingUser = await this.userRepository.findOne({ where: { email } });
        if (existingUser) {
            throw new ConflictError('Email already in use');
        }

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

        const user = new User();
        user.email = email;
        user.password = hashedPassword;

        const validationErrors: ClassValidatorError[] = await validate(user);
        if (validationErrors.length > 0) {
            const mappedErrors = validationErrors.map(err => ({
                property: err.property,
                constraints: err.constraints,
                value: err.value,
            }));
            throw new ValidationError(mappedErrors);
        }
```

```
    try {
        const newUser = await this.userRepository.save(user);
        const token = this.authService.generateToken(newUser);

        const userOutput: UserOutput = {
            id: newUser.id,
            email: newUser.email,
            createdAt: newUser.createdAt,
            updatedAt: newUser.updatedAt,
        };

        return {
            message: 'User registered successfully',
            user: userOutput,
            token,
            tokenType: settings.auth.TOKEN_TYPE,
        };
    } catch (error) {
        console.error('Persistence or token generation error during user registration:', error);
        throw new InternalServerError('Failed to register user due to an internal server issue.');
    }
}
```

```
async login(credentials: { email?: string; password?: string }): Promise<LoginSuccessResponse> {
    const { email, password } = credentials;

    if (!email || !password) {
        throw new BadRequestError('Email and password are required for login.');
    }

    const user = await this.userRepository.findOne({ where: { email } });
    if (!user) {
        throw new UnauthorizedError('Invalid email or password.');
    }

    const isPasswordMatch = await user.comparePassword(password);
    if (!isPasswordMatch) {
        throw new UnauthorizedError('Invalid email or password.');
    }

    try {
        const token = this.authService.generateToken(user);
        const userOutput: UserOutput = {
            id: user.id,
            email: user.email,
            createdAt: user.createdAt,
            updatedAt: user.updatedAt,
        };

        return {
            message: 'Login successful',
            user: userOutput,
```

Модель пользователя:

```
@Entity('users')
export class User {
    @PrimaryGeneratedColumn('uuid')
    id!: string;

    @Column({ unique: true, length: 255 })
    @IsEmail()
    email!: string;

    @Column({ length: 255 })
    @MinLength(8)
    password!: string;

    @CreateDateColumn()
    createdAt!: Date;

    @UpdateDateColumn()
    updatedAt!: Date;

    async comparePassword(attempt: string): Promise<boolean> {
        return bcrypt.compare(attempt, this.password);
    }
}
```

Вывод

В ходе данной лабораторной работы был разработан boilerplate-проект с использованием технологий Express, TypeORM и TypeScript. Было проведено разделение кода на модули для удобства его расширения в дальнейшем.