САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бэк-энд разработка

Отчет

ЛР 1 - реализация boilerplate

Выполнил:

Сергеев Виктор

K3341

Проверил:
Добряков Д. И.

Санкт-Петербург

2025 г.

Задача

Требуется реализовать boilerplate на Express + TypeORM + typescript

Должно быть разделение на:

- модели
- контроллеры
- роуты

Ход работы

Данный проект будет реализован по архитектуре MVCS. Текущая файловая структура проекта изображена на рисунке 1

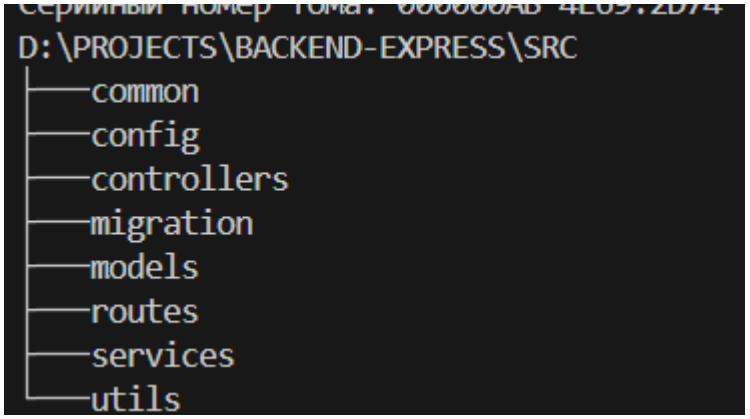

Рисунок 1 - файловая структура проекта

Помимо этого, в директории src находятся файлы index.ts - содержит инициализацию приложения - и data-source.ts - содержит инициализацию настроек базы данных. Назначение каждой директории следующее:

- common - содержит дженерик классы частей проекта для их успешного переиспользования
- config - содержит конфигурационные параметры проекта
- controllers - содержит контроллеры моделей
- migration - содержит файлы миграции базы данных
- models - содержит описания моделей
- routes - содержит роутеры для каждой модели
- services - содержит сервисы моделей
- utils - вспомогательные классы и функции

В этой структуре сервис отвечает за обращения к бд, контроллер для принятие данных, их валидации и взаимодействие с сервисом.

Для сервисов были реализованы дженерик классы BaseService и BaseCRUDService, от которых наследуются все дальнейшие сервисы. Это удобно, поскольку для большинства моделей надо сделать простой набор из CRUD-методов, и благодаря наследованию можно просто создать дочерний класс с нужной моделью, чтобы получить уже готовую реализацию. На рисунках 2 и 3 соответственно представлен их код.

Рисунок 2 - файл BaseService.ts

Рисунок 3 - файл BaseCRUDService.ts

Далее были реализованы дженерики контроллеров: BaseController и BaseCRUDController. Их использование по аналогичной причине, что и для сервисов, также удобно поскольку позволяет переиспользовать код в разных контекстах. Они представлены на рисунках 4 и 5.

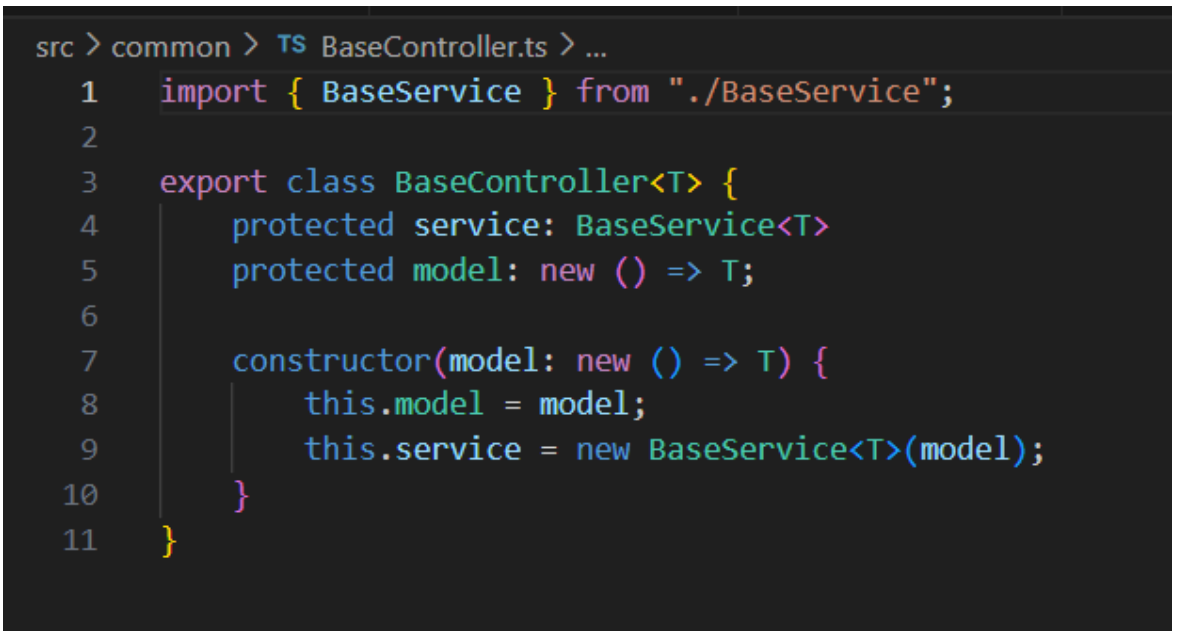

Рисунок 4 - файл BaseController.ts

Рисунок 5 - файл BaseCRUDController.ts

Дальше, идёт инициализация конфигурационных параметров проекта. Это происходит в файлы config/settings.ts. Параметры подтягиваются либо из .env файла, либо берётся стандартное значение.

Наконец, была реализована возможность регистрации и авторизации на сервере. Для этого был реализован контроллер AuthController, который

содержит методы register и login, отвечающие за соответствующие функции. Они представлены на рисунках 6 и 7

```
register = async (request: Request, response: Response) => {
    const errors = new ValidationErrors();
    await validateRegister(request.body, errors, this.service);
    if (errors.isError) {
        response.status(422).json({errors: errors.errors});
        return;
    }

    const hash = hashPassword(request.body.password)
    const dto = {
        "username": request.body.username,
        "password": hash
    }
    try {
        const entity = await this.service.createEntity(dto);
        response.status(201).json(entity);
    } catch (error) {
        response.status(400).json({error: error.message});
    }
}
```

Рисунок 6 - метод register из AuthController

```
login = async (request: Request, response: Response) => {
    const errors = new ValidationErrors();
    validateLogin(request.body, errors);
    if (errors.isError) {
        response.status(422).json({errors: errors.errors});
        return;
    }

    const dto = {
        "username": request.body.username,
        "password": request.body.password
    };
    const user = await this.service.getEntityByUsername(dto.username);
    if (!user) {
        response.status(400).json({message: "Username or password incorrect"})
        return;
    }

    if (!checkPassword(dto.password, user.password)) {
        response.status(400).json({message: "Username or password incorrect"})
        return;
    }

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

    response.json({"access_token": accessToken});
}
```

Рисунок 7 - метод login из AuthController

Остаётся проверить работоспособность проекта. Через роутеры контроллеры были подключены к приложению. После этого приложение было поднято локально. С помощью Burp Suite была проведена проверка работоспособности доступного API. На рисунках 8 и 9 представлена проверка методов, реализованных в AuthController, а на рисунках 10-12 - методов контроллера UserController.

Request

Pretty Raw Hex

```
POST /api/auth/register HTTP/1.1
Host: 127.0.0.1:3000
Content-Type: application/json
Content-Length: 83

{
    "username":"username5",
    "password":"password123",
    "password_confirm":"password123"
}
```

Response

Pretty Raw Hex Render

```
HTTP/1.1 201 Created
X-Powered-By: Express
Content-Type: application/json; charset=utf-8
Content-Length: 175
ETag: W/"af-nZ7qPiAdq//IUGt8i4NsU04vp+c"
Date: Mon, 28 Apr 2025 13:42:08 GMT
Connection: keep-alive
Keep-Alive: timeout=5

{
    "username":"username5",
    "password":
    "$2b$12$fr3a6mv2qsCR3.CEdMtJUu3Z8xRrfhXGHJ57d0
    5N0TKSyEaTAPZEG",
    "avatar_url":null,
    "bio":null,
    "id":21,
    "created_at":"2025-04-28T10:42:08.850Z"
}
```

Рисунок 9 - регистрация пользователя

Request

Pretty Raw Hex

```
POST /api/auth/login HTTP/1.1
Host: 127.0.0.1:3000
Content-Type: application/json
Content-Length: 49

{
    "username":"username5",
    "password":"password123"
}
```

Response

Pretty Raw Hex Render

```
HTTP/1.1 200 OK
X-Powered-By: Express
Content-Type: application/json; charset=utf-8
Content-Length: 164
ETag: W/"a4-hKTjG5eihPpGCr3bnQM3glpnGGY"
Date: Mon, 28 Apr 2025 13:43:24 GMT
Connection: keep-alive
Keep-Alive: timeout=5

{
    "access_token":
    "eyJhbGci0iJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c
    2VyX2lkIjoyMSwiaWF0IjoxNzQ1ODQ30DA0LCJleHAi
    0jE3NDU4NTE0MDR9.Daj2jK3P5ml4P2OYZYddLYZlkG
    vEk-qXpdnl8ju4dYI"
}
```

Рисунок 10 - авторизация пользователя

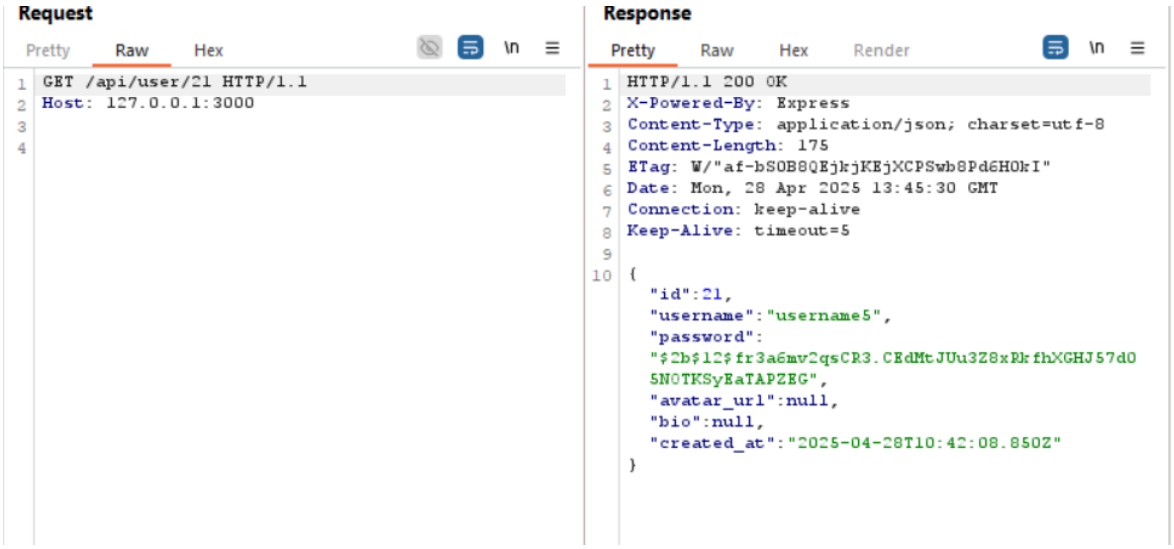

Рисунок 11 - получение данных о пользователе

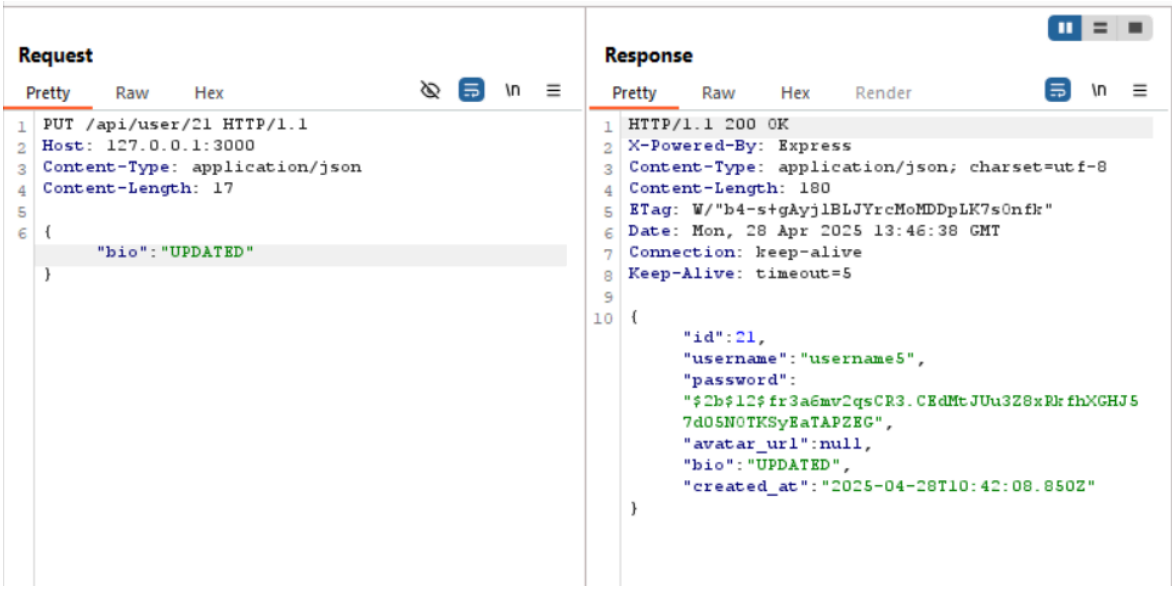

Рисунок 12 - изменение пользователя

Рисунок 13 - удаление пользователя

Вывод

В процессе работы был реализован boilerplate веб-приложения на стеке express + typeorm + typescript для более удобной дальнейшей реализации и поддержки. Были реализованы дженерик классы сервисов и контроллеров, реализован функционал аутентификации и проверены на работоспособность.