САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бэк-энд разработка

Отчет

Лабораторная работа №1

Выполнил:

Шалунов Андрей

Группа К3340

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

1. Файл настройки переменных окружения

Вынес в *config/settings.ts* или *.env* все параметры: порт приложения, доступ к Postgres, пути к сущностям.

2. Инициализация DataSource

В *src/config/data-source.ts* создал *new DataSource({...}) с synchronize: true* для автоматического создания таблиц.

```
import 'reflect-metadata';

import { DataSource } from 'typeorm';

import SETTINGS from './settings';



export const AppDataSource = new DataSource({

    type: 'postgres',

    host: SETTINGS.DB_HOST,

    port: SETTINGS.DB_PORT,

    username: SETTINGS.DB_USER,

    password: SETTINGS.DB_PASSWORD,

    database: SETTINGS.DB_NAME,

    entities: [SETTINGS.DB_ENTITIES],
```

```
    logging: false,

    synchronize: true,

});
```

3. Модель пользователя

В *src/models/user.entity.ts* описал *class User* с полями user_id, name, email, password.

```
import { Entity, PrimaryGeneratedColumn, Column } from 'typeorm';



@Entity({ name: 'users' })

export class User {

    @PrimaryGeneratedColumn()

    user_id!: number

    @Column()

    name!: string;

    @Column({ unique: true })

    email!: string;

    @Column()

    password!: string;

}
```

4. Утилиты для пароля

В *src/utils/hash-password.ts* и *check-password.ts* реализовал хеширование и проверку пароля через bcrypt.

```
import bcrypt from 'bcrypt';

const hashPassword = (password: string): string => {

    return bcrypt.hashSync(password, bcrypt.genSaltSync(8));
```

```
};

export default hashPassword;
```

5. Контроллеры авторизации

В *src/controllers/auth.controller.ts* написал методы *register* (хеширует пароль, сохраняет пользователя) и *login* (проверяет пароль, возвращает JWT по userId + email).

6. Middleware аутентификации

В *src/middlewares/auth.middleware.ts* вынес проверку заголовка Authorization, распарсил JWT через jsonwebtoken, сохранил { userId, email } в request.user.

7. Контроллеры работы с пользователями

В *src/controllers/user.controller.ts* реализовал стандартные CRUD-методы:

- createUser
- getUsers
- getUser,
- getUserMe
- updateUser
- deleteUser.

8. Роутеры

В *src/routes/auth.router.ts* подключил */register* и */login*.

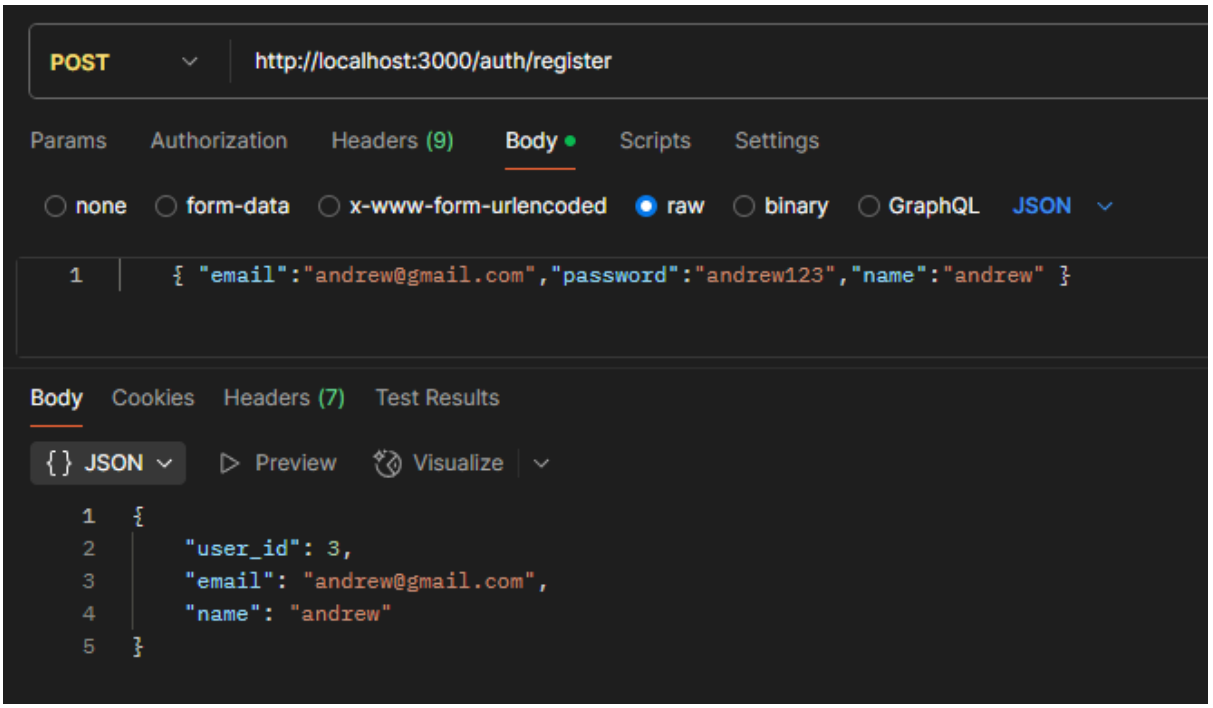
В *src/routes/user.router.ts* организовал маршруты */users*.

```
import { Router } from "express";

import {createUser, getUsers, getUser, getUserMe, updateUser,
deleteUser } from '../controllers/user.controller';

import { authMiddleware } from '../middlewares/auth.middleware';

const router = Router();

router.post('/', createUser)

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

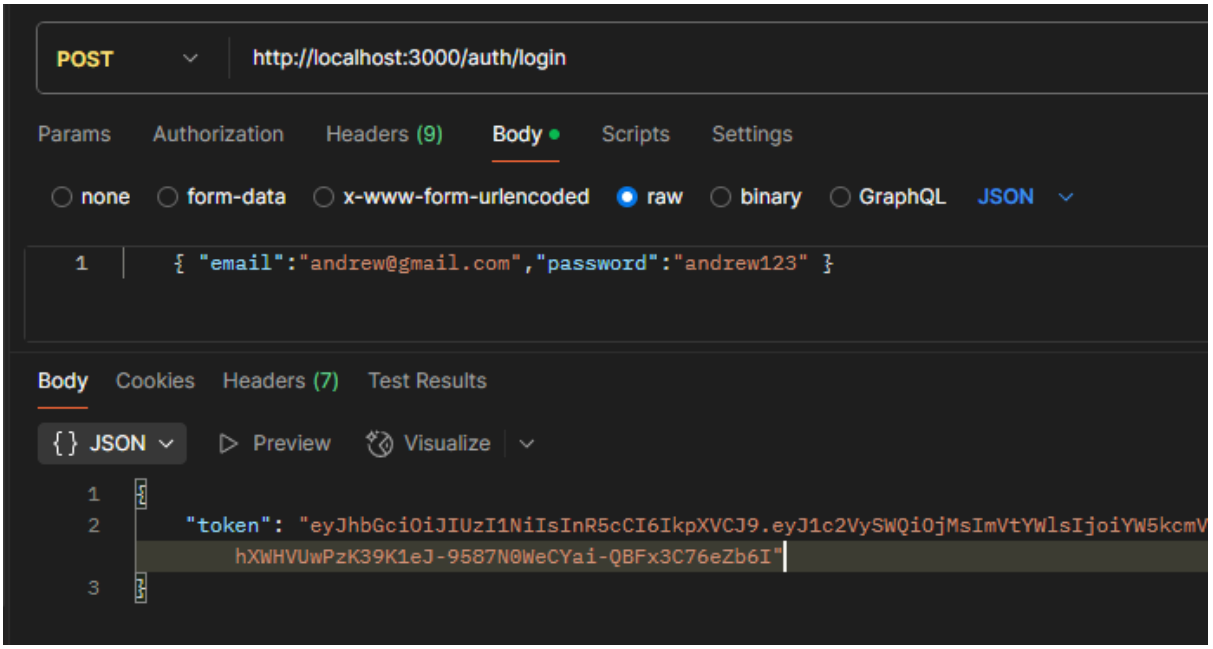
```
router.get('/me', authMiddleware, getUserMe);

router.get('/:id', getUser);

router.put('/:id', updateUser);

router.delete('/:id', deleteUser);

export default router;
```

Тестирование с помощью Postman

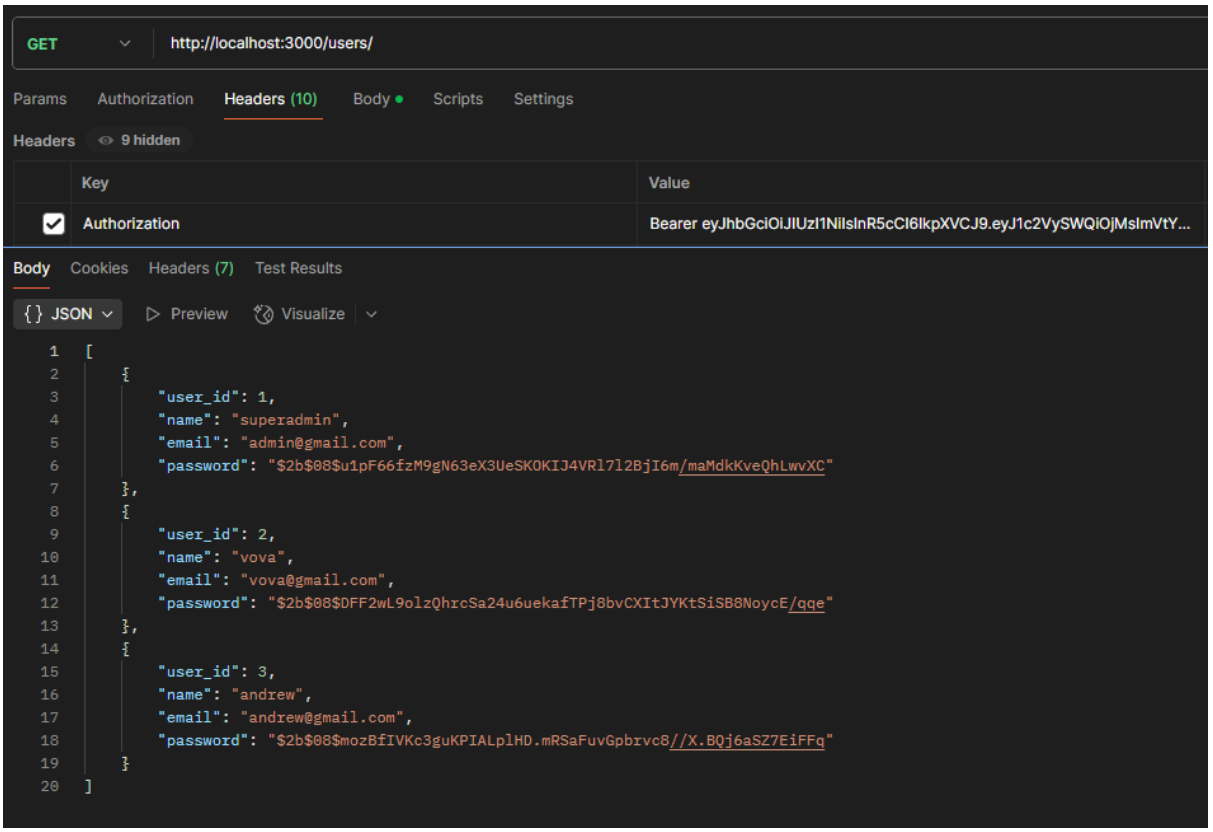
Регистрируем нового пользователя



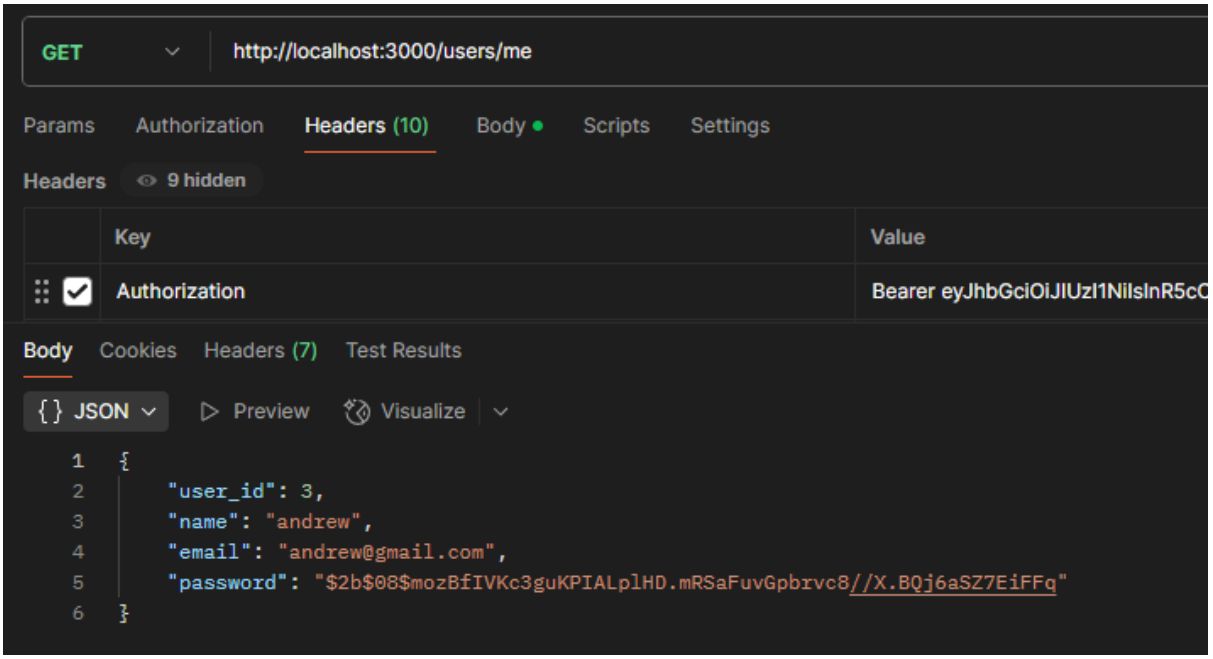
Логинимся и получаем информацию о нашем токене



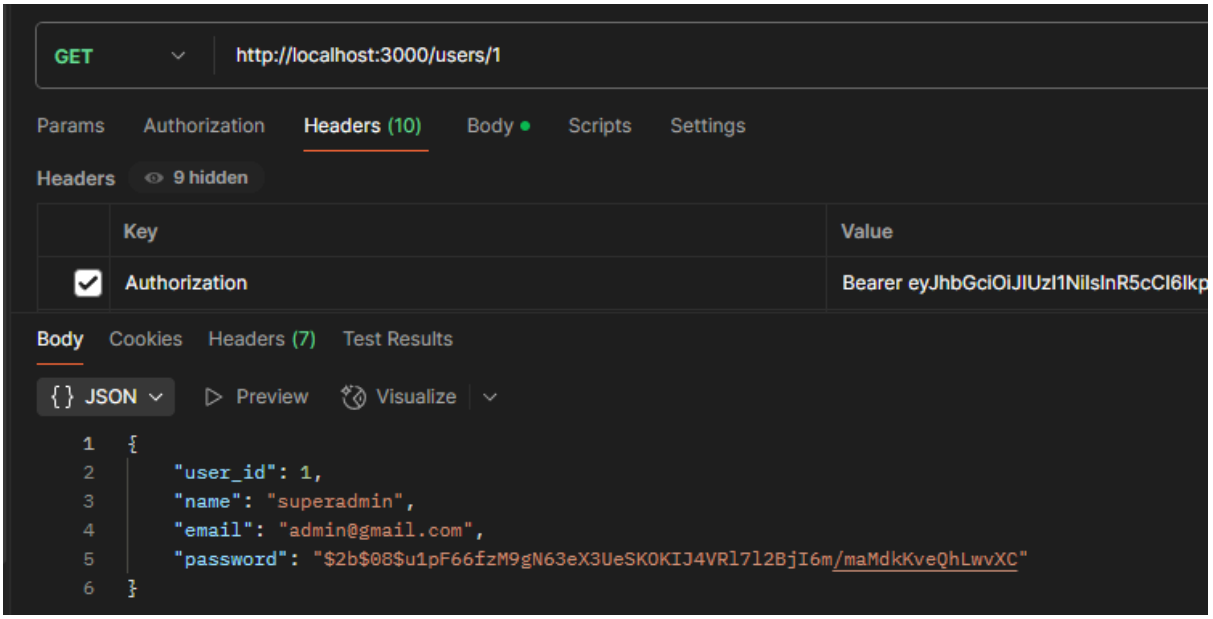
Теперь выведем информацию о всех юзерах (до этого уже создавалось двое)



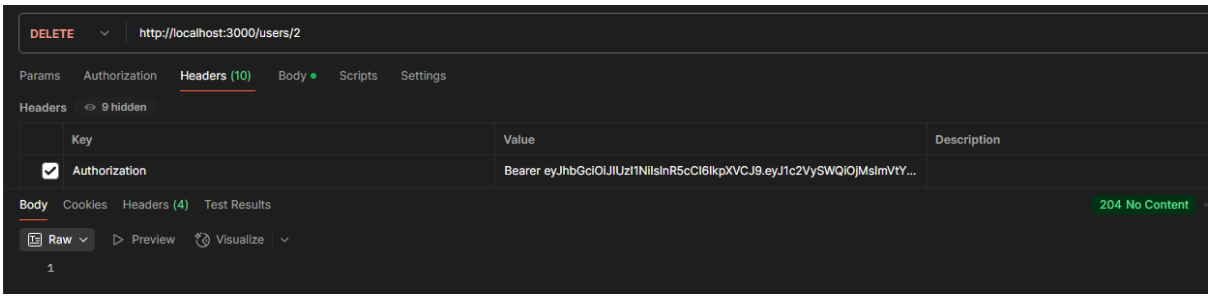
Выведем информацию о себе



Узнаем информацию о первом юзере



Удалим второго юзера



Вывод

Получилось написать boilerplate на Express + TypeORM + TypeScript с модульной структурой, в которой чётко разделены сущности, контроллеры и маршруты. Реализована регистрация и вход через JWT.