САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бэк-энд разработка

Отчет

Лабораторная работа №2

Выполнил:

Шалунов Андрей

Группа К3340

Проверил:
Добряков Д. И.

Санкт-Петербург

2025 г.

Задача

По выбранному варианту необходимо будет реализовать RESTful API средствами express + typescript (используя ранее написанный boilerplate).

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({...})*.

```
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, phone.

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

import { Property } from './property.entity';

import { Booking  } from './booking.entity';

import { Message  } from './message.entity';

@Entity()

export class User {

    @PrimaryGeneratedColumn()

    user_id!: number;

    @Column()

    name!:string;

    @Column({unique: true})

    email!:string;

    @Column({ nullable: true })

    phone?:string;

    @Column()

    password!:string;



    @OneToMany(() => Property, p => p.owner)

    properties!: Property[];

    @OneToMany(() => Booking, b => b.renter)

    bookings!: Booking[];

    @OneToMany(() => Message, m => m.sender)

    sentMessages!: Message[];
```

```
    @OneToMany(() => Message, m => m.recipient)

    receivedMessages!: Message[];

}
```

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. DTO и валидация

Перешел от интерфейсов к классам с декораторами class-validator в папке src/dto:

- *booking.dto.ts,*
- *message.dto.ts,*
- *photo.dto.ts,*
- *property.dto.ts,*
- *user.dto.ts,*
- *auth.dto.ts (RegisterDto, LoginDto).*

В контроллерах через *plainToInstance() + validateOrReject()* проверяю входящие данные.

6. Middleware

JWT-аутентификация (*auth.middleware.ts*): читает заголовок Authorization, проверяет токен и кладём в req.user.

Глобальный обработчик ошибок (*error.middleware.ts*): ловит все next(err) и форматирует ответ (используя err.status).

7. Сервисы

В *src/services/* описаны классы с static-методами:

- AuthService (register/login с bcrypt и jsonwebtoken)
- BookingService, MessageService, PhotoService, PropertyService, UserService (CRUD-операции через репозитории TypeORM)

8. Контроллеры

   В *src/controllers/* реализовали классы с static async методами.

   Каждый метод:

   - Валидирует DTO.
   - Вызывает сервис.

9. Роутинг

В *src/routes/* для каждого ресурса:

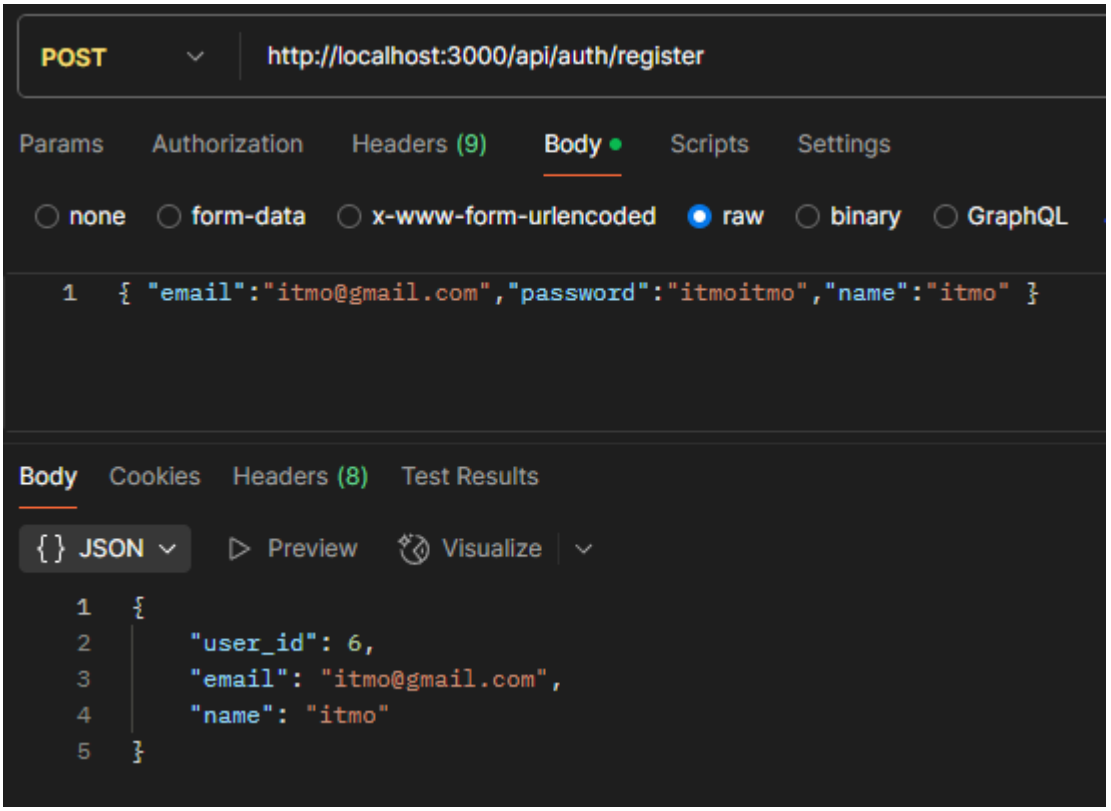
- Подключил authMiddleware к защищённым эндпоинтам.
- В маршрутах передаем прямо методы контроллера

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

import { AuthController  } from '../controllers/auth.controller';

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

const router = Router();

router.post('/register', AuthController.register)

router.post('/login', AuthController.login)
```

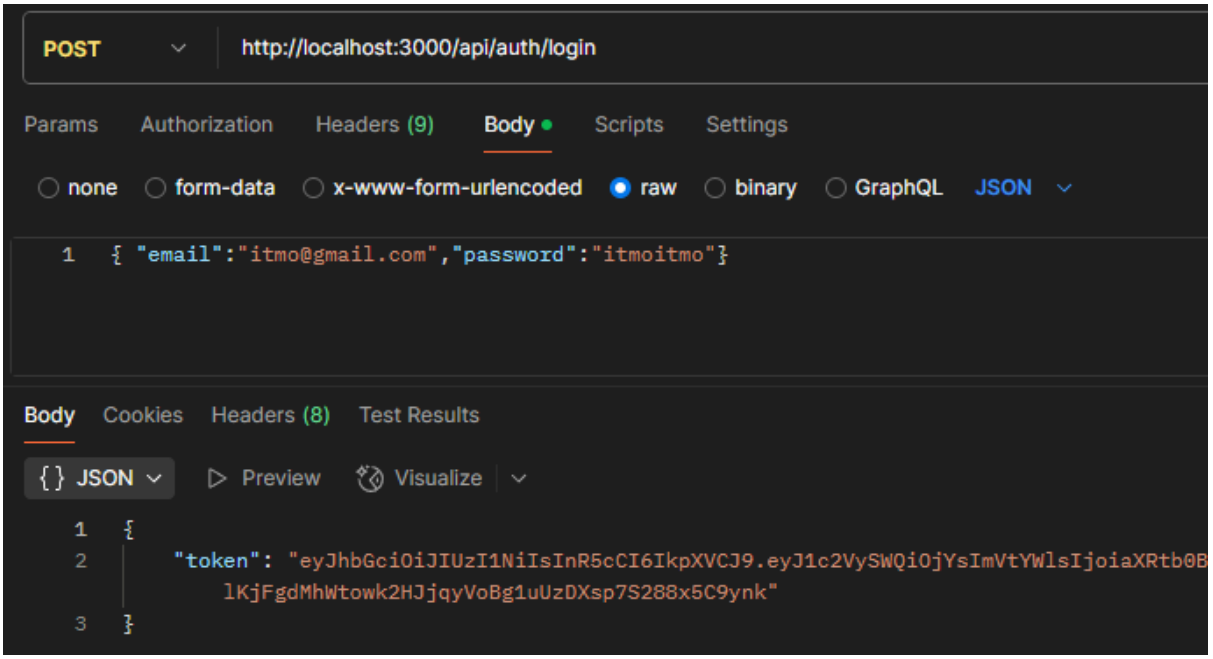
```
export default router;
```

Тестирование с помощью Postman

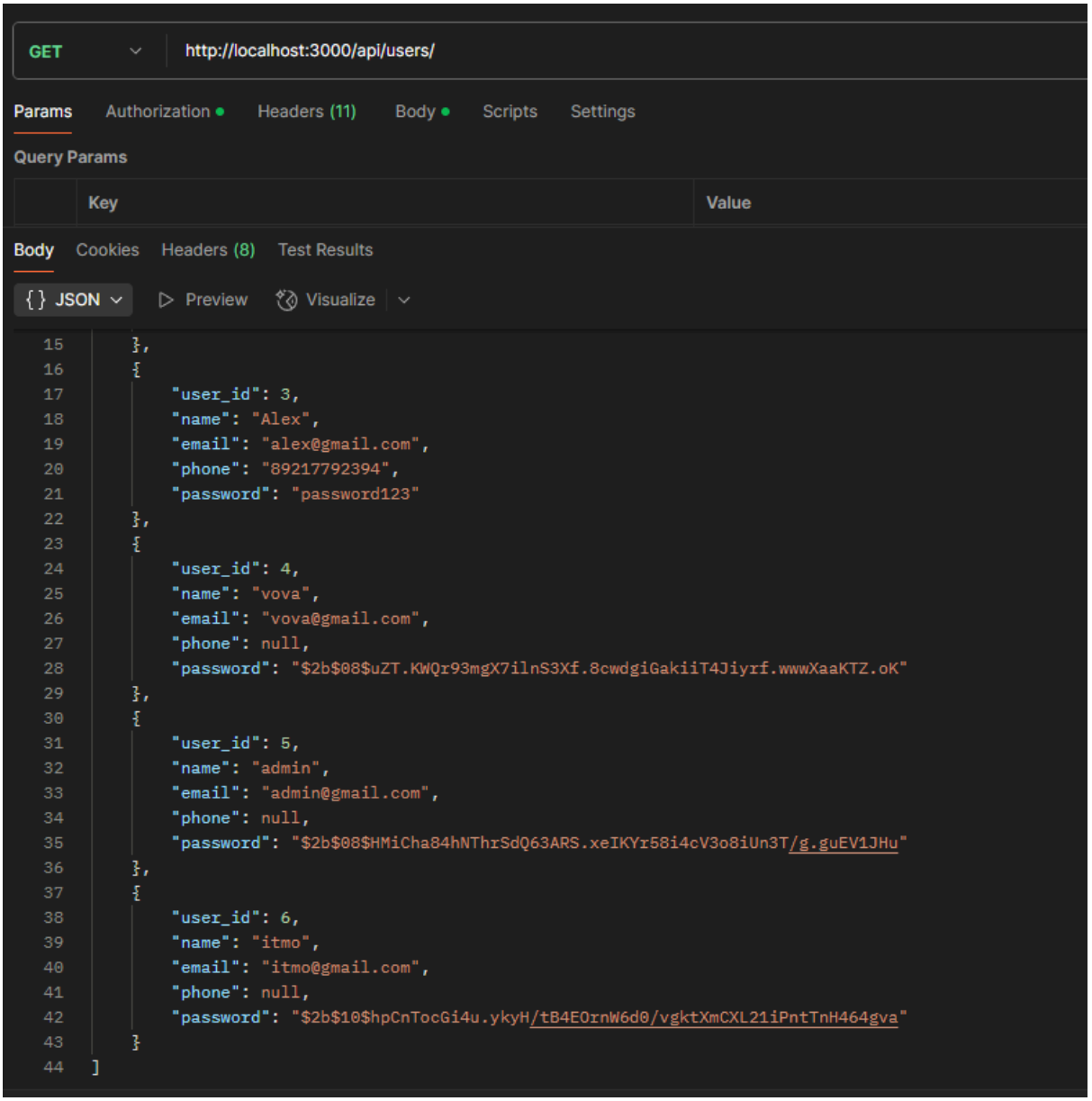
Регистрируем нового пользователя



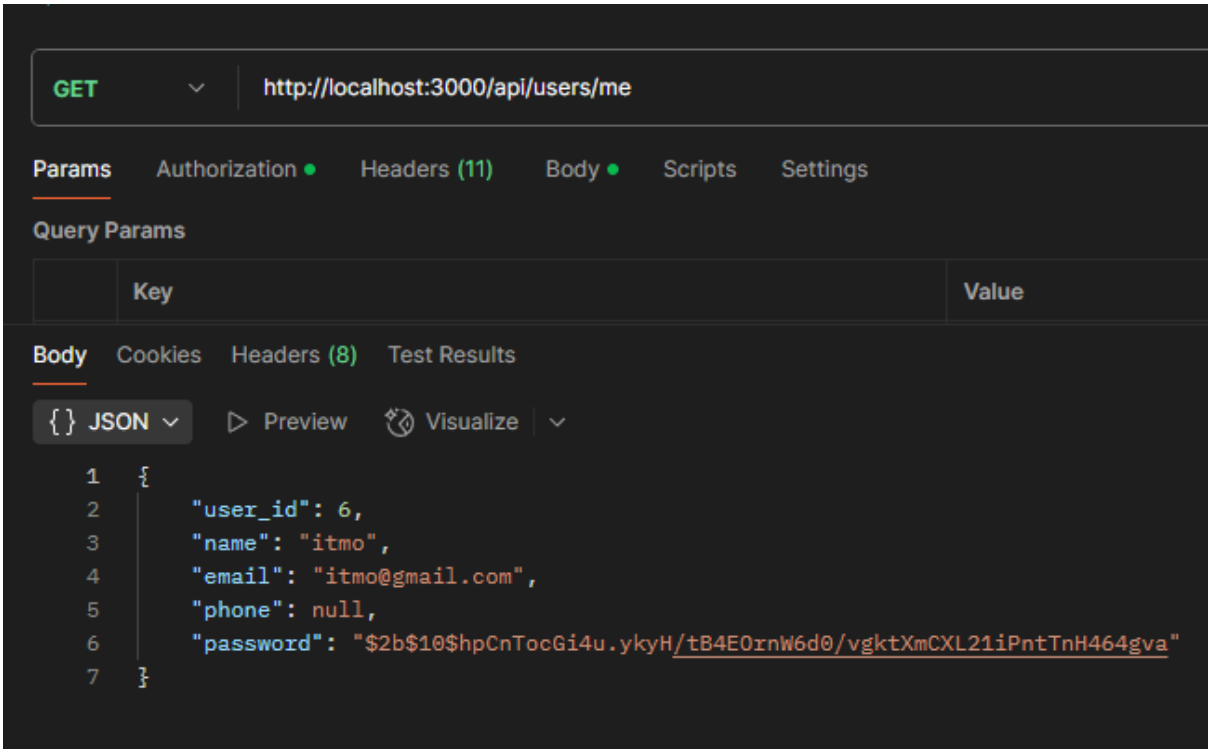
Логинимся и получаем информацию о нашем токене



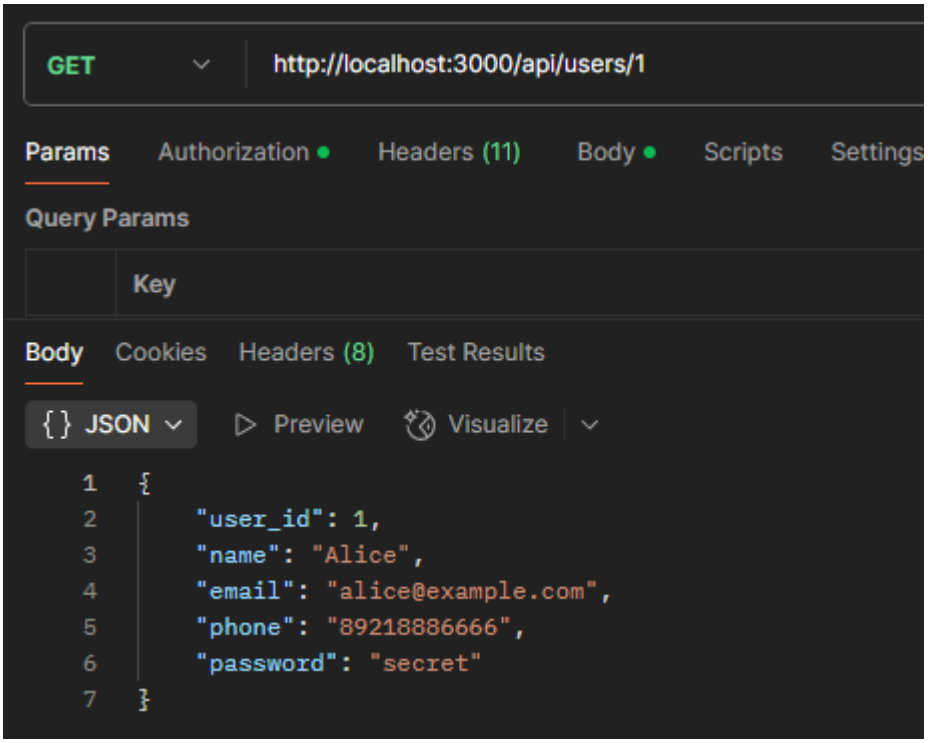
Теперь выведем информацию о всех юзерах



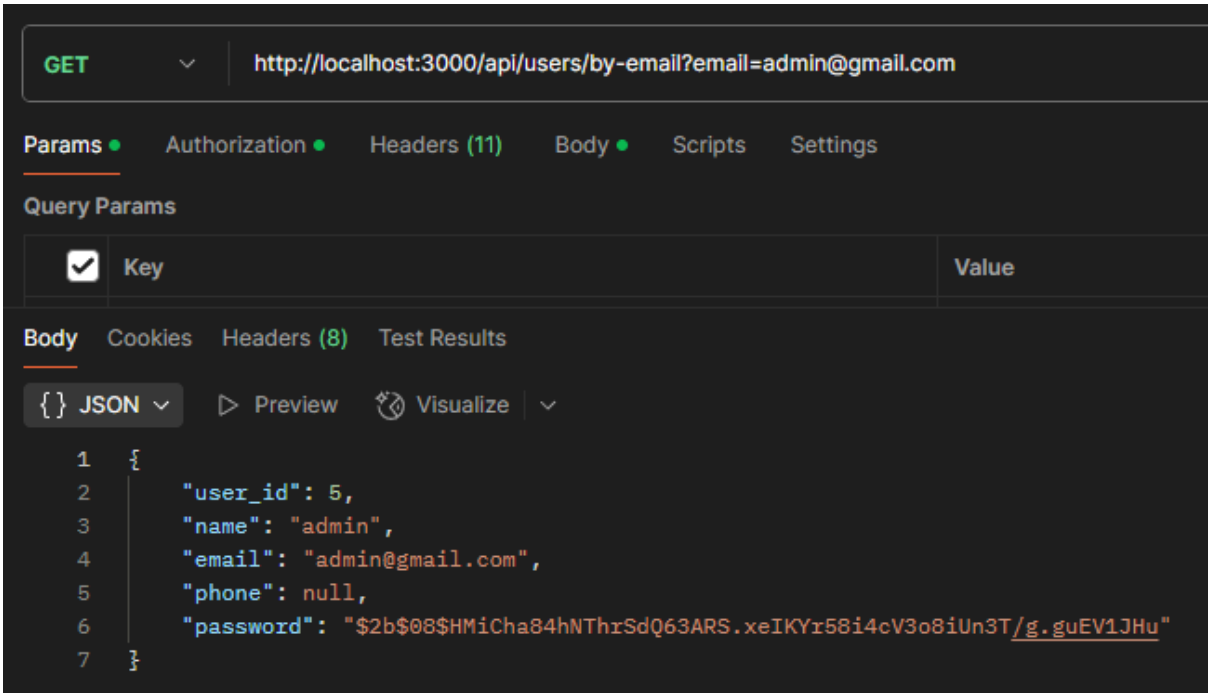
Выведем информацию о себе



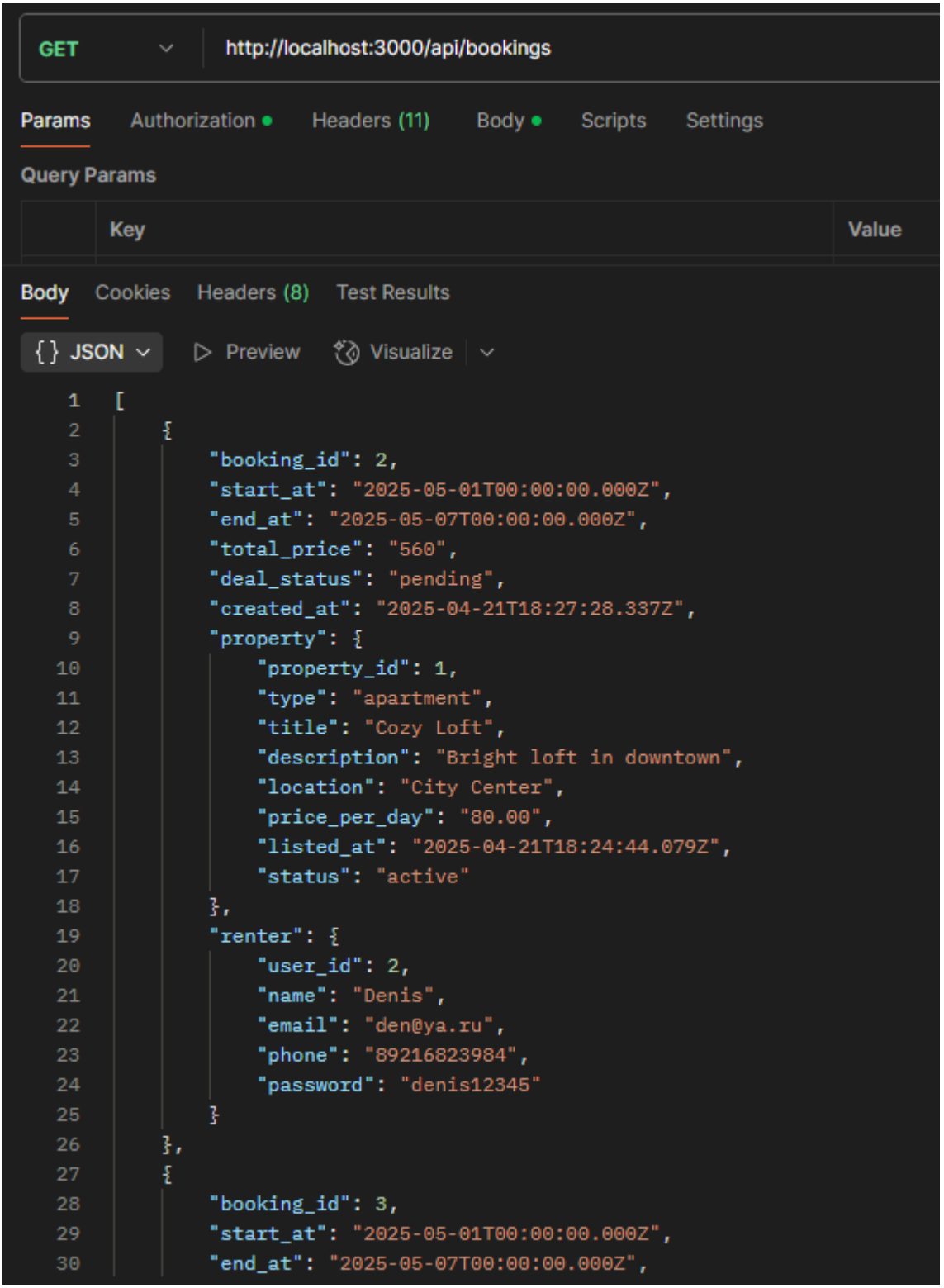
Узнаем информацию о первом юзере



Теперь получим пользователя по email



Выведем информацию о всех бронях



Вывод

Получилось написать RESTful API на Express + TypeScript с JWT-аутентификацией, валидацией DTO. Весь код структурирован по слоям: модели, сервисы, контроллеры, роутеры.