САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бек-энд разработка

Отчет
Домашняя работа №2

Выполнила: Злотникова Карина Александровна

Группа: K33392

Проверил:
Добряков Д. И.

Санкт-Петербург

2024 г.

Задача

Продумать свою собственную модель пользователя. Реализовать набор из CRUD-методов для работы с пользователями средствами Express. Написать запрос для получения пользователя по id/email.

Ход работы

Создадим модель пользователя которая будет содержать id, имя и пароль. Модель будем создавать посредством typeorm.

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

@Entity()
export class User {
   @PrimaryGeneratedColumn()
   id!: number;

   @Column({ unique: true })
   username!: string;

   @Column()
   password!: string;
}
```

В качестве CRUD методов реализуем авторизацию и регистрацию пользователя. Для этого, создадим контроллер юзера и подготовим методы авторизации и регистрации:

```
import { Request, Response } from 'express';
import { getRepository } from 'typeorm';
import bcrypt from 'bcryptjs';
import jwt from 'jsonwebtoken';
import amqp from 'amqplib';
import { User } from '../entities/User';

class UserController {

   static async signUp(req: Request, res: Response) {
       try {
           const userRepository = getRepository(User);
           const hashedPassword = await bcrypt.hash(req.body.password, 8);
```

```
            const newUser = userRepository.create({ username: req.body.username,
password: hashedPassword });
            await userRepository.save(newUser);
            res.status(201).json(newUser);
        } catch (error: any) {
            res.status(400).json({ message: error.message });
        }
    }

    static async signIn(req: Request, res: Response) {
        try {
            const userRepository = getRepository(User);
                  const user = await userRepository.findOne({ where: { username:
req.body.username } });
            if (!user) {
                throw new Error('User not found');
            }
                   const isPasswordValid = await bcrypt.compare(req.body.password,
user.password);
            if (!isPasswordValid) {
                throw new Error('Invalid password');
            }
                 const token = jwt.sign({ userId: user.id },'super_secret_key', {
expiresIn: '1h' });
            res.status(200).json({ token });
        } catch (error: any) {
            res.status(400).json({ message: error.message });
        }
    }

    static verifyToken(token: string) {
        try {
            return jwt.verify(token,'super_secret_key');
        } catch (error) {
            throw new Error('Invalid token');
        }
       }
}

export default UserController;
```

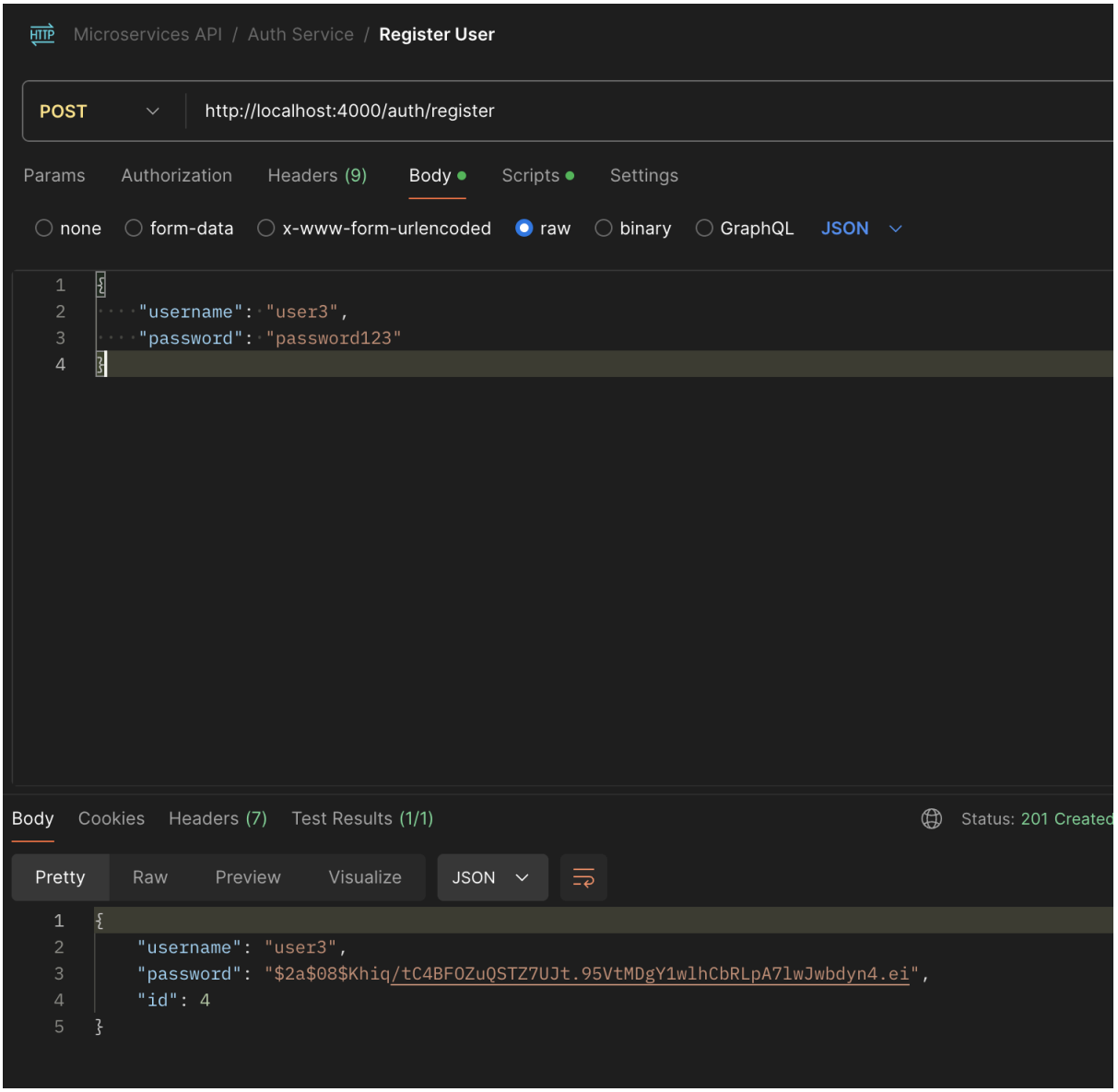
Подключим контроллер пользователя к конкретным эндпоинтам:

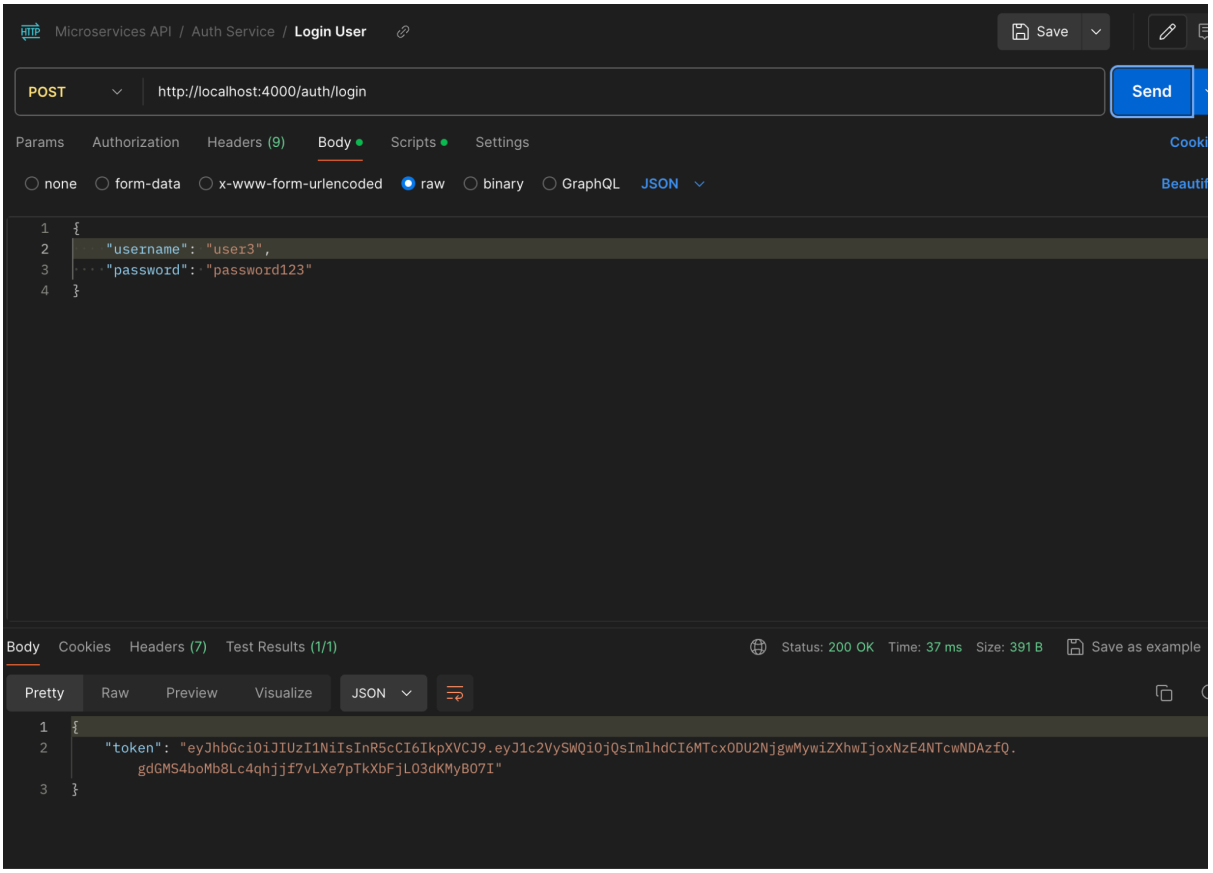
```
import { Router } from 'express';
import UserController from '../controllers/UserController';
```

```
const router = Router();

router.post('/register', UserController.signUp);
router.post('/login', UserController.signIn);

export default router;
```

Таким образом получим метод для регистрации пользователя и метод для авторизации, позволяющий получить jwt токен при вводе правильного логина и пароля.

Протестируем через постман:


Microservices API / Auth Service / Register User
POST
http://localhost:4000/auth/register
Params
Authorization
Headers (9)
Body
Scripts
Settings
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
{
"username": "user3",
"password": "password123"
}
Body
Cookies
Headers (7)
Test Results (1/1)
Status: 201 Created
Pretty
Raw
Preview
Visualize
JSON
{
"username": "user3",
"password": "$2a$08$Khiq/tC4BFOZuQSTZ7UJt.95VtMDgY1wlhCbRLpA7lwJwbdyn4.ei",
"id": 4
}



Вывод

В ходе работы мне удалось реализовать модель пользователя и основные CRUD методы для работы с пользователем в будущем проекте.