САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бэк-энд разработка

Отчет

Домашняя работа №2

Выполнил:

Тарабанов Алексей

Группа K33402

Проверил:
Добряков Д. И.

Санкт-Петербург

2024 г.

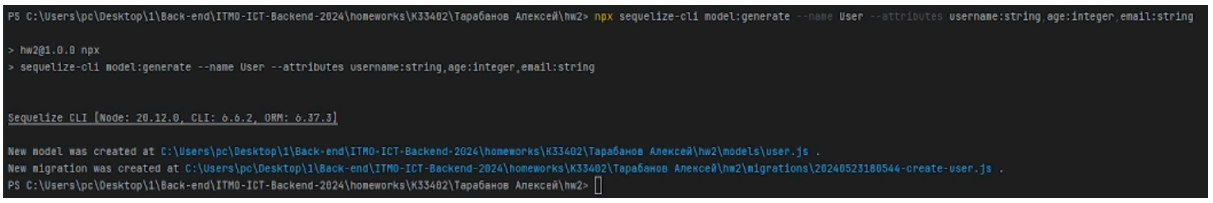
Задача

Задание:

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

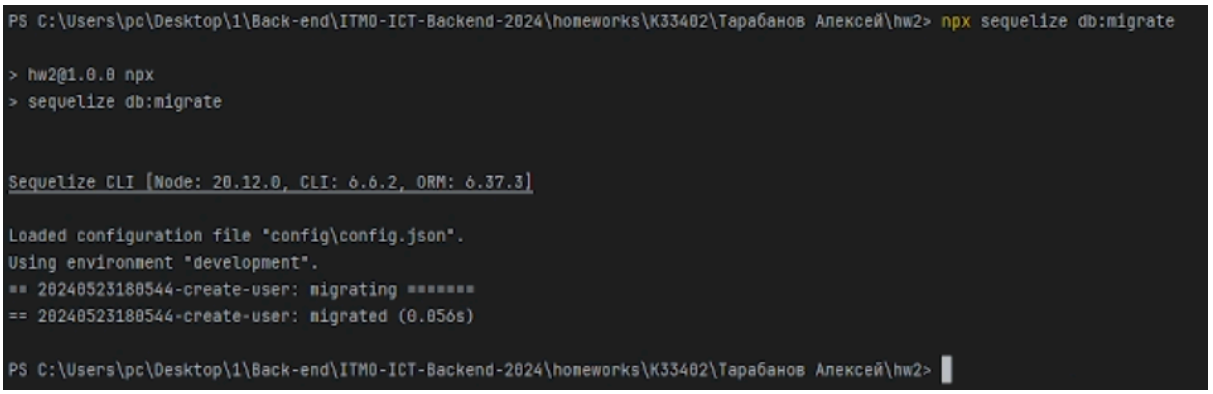
Ход работы

В начале, после инициализации и установки всех необходимых библиотек, я сгенерировал модел при помощи squelize-cli
В качестве параметроввыбрал:
username - Имя пользователя
age - возраст пользователя
email - почта пользователя



Далее я изменил файл конфигурации для использования базы данных MySQL которая находится на выделенном сервере. На нем же я сделал Пользователя и пароль, для обращения к ней

После я мигрировал созданную модель в базу данных



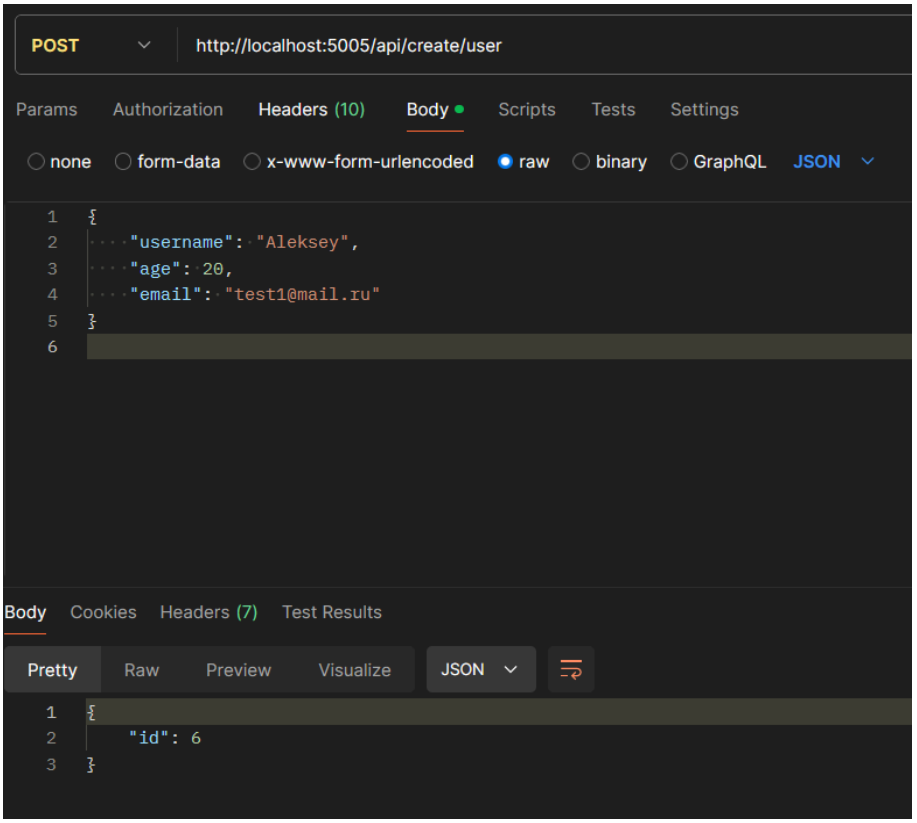
Дальше в коде я прописал эндпоинты ит вынес контроллеры в отдельную папку/файл.

```
app.post( path: '/api/create/user', userController.create_user) // Создать
app.get('/api/get/users', userController.get_users) // Поиск всех
app.get('/api/get/user/:id', userController.get_user_by_id) // Поиск по id
app.post( path: '/api/get/user/by_email', userController.get_user_by_email) // Поиск по почте
app.post( path: '/api/update/user/:id', userController.update_user) // Обновление по id
app.get('/api/delete/user/:id', userController.delete_user) // Удаление по id
```

Ниже я буду описывать каждый контроллер и его тесты:

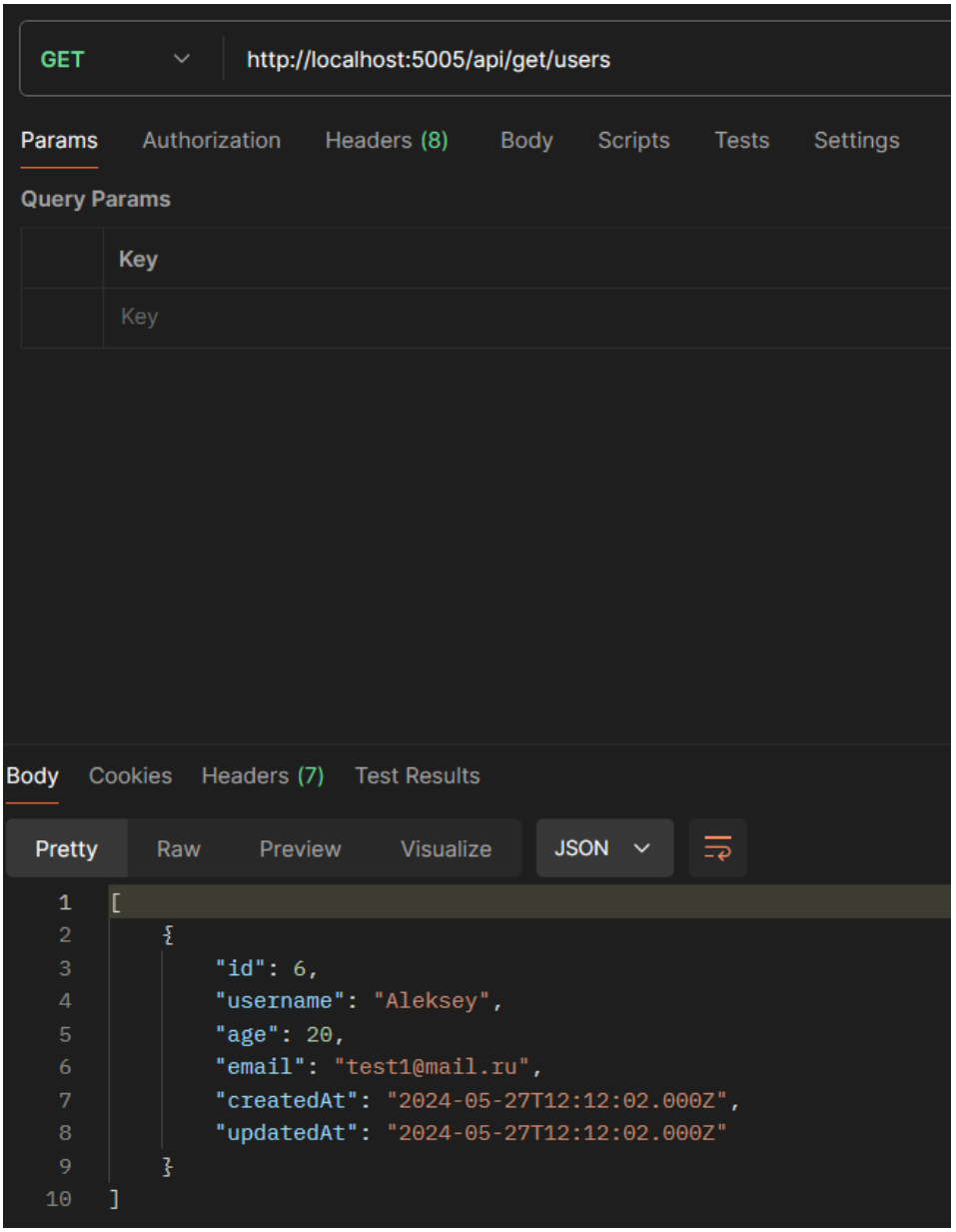
Первый контроллер который я описал, это create_user. Он создает пользователя исходя из полученных данных прикрепленных к post запросу. В качестве ответа, мы возвращаем id пользователя

```
exports.create_user = async (req, res) : Promise<...>  => {
        try {
            const data = await db.User.create(req.body)
            return res.status(200).json({ id: data.id })
        }
        catch (error){
            return res.sendStatus(500)
        }
}
```

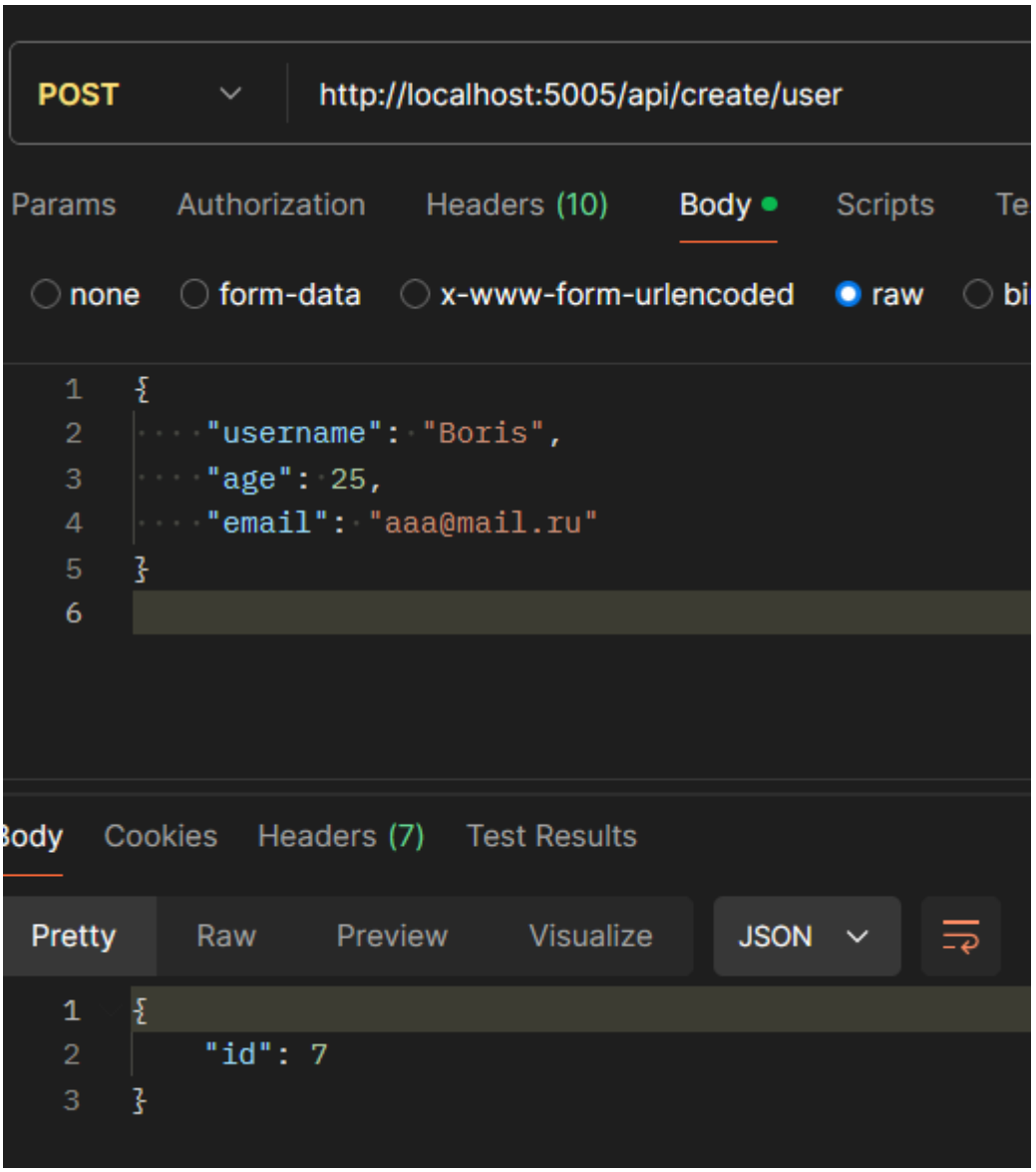


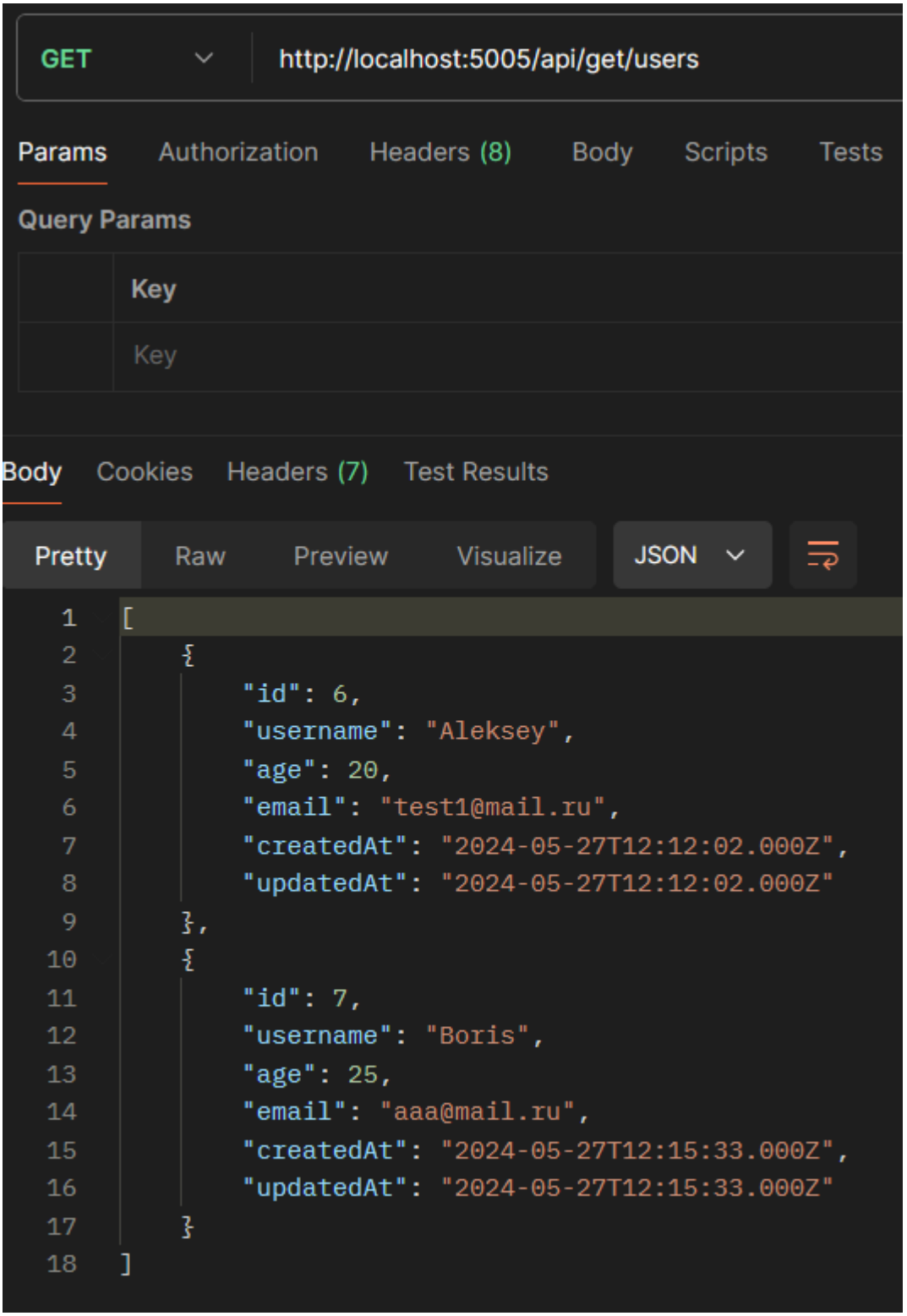
Следующий реализованный контроллер это get_user.
Он позволяет получить всех текущих пользователей:

```
exports.get_users = async (req, res) : Promise<...>  => {
    try {
        const data : Model[]  = await db.User.findAll()
        return res.status(200).send(data)
    }
    catch (error){
        return res.sendStatus(500)
    }
}
```


GET
http://localhost:5005/api/get/users
Params
Authorization
Headers (8)
Body
Scripts
Tests
Settings
Query Params
Key
Key
Body
Cookies
Headers (7)
Test Results
Pretty
Raw
Preview
Visualize
JSON
[
{
"id": 6,
"username": "Aleksey",
"age": 20,
"email": "test1@mail.ru",
"createdAt": "2024-05-27T12:12:02.000Z",
"updatedAt": "2024-05-27T12:12:02.000Z"
}
]

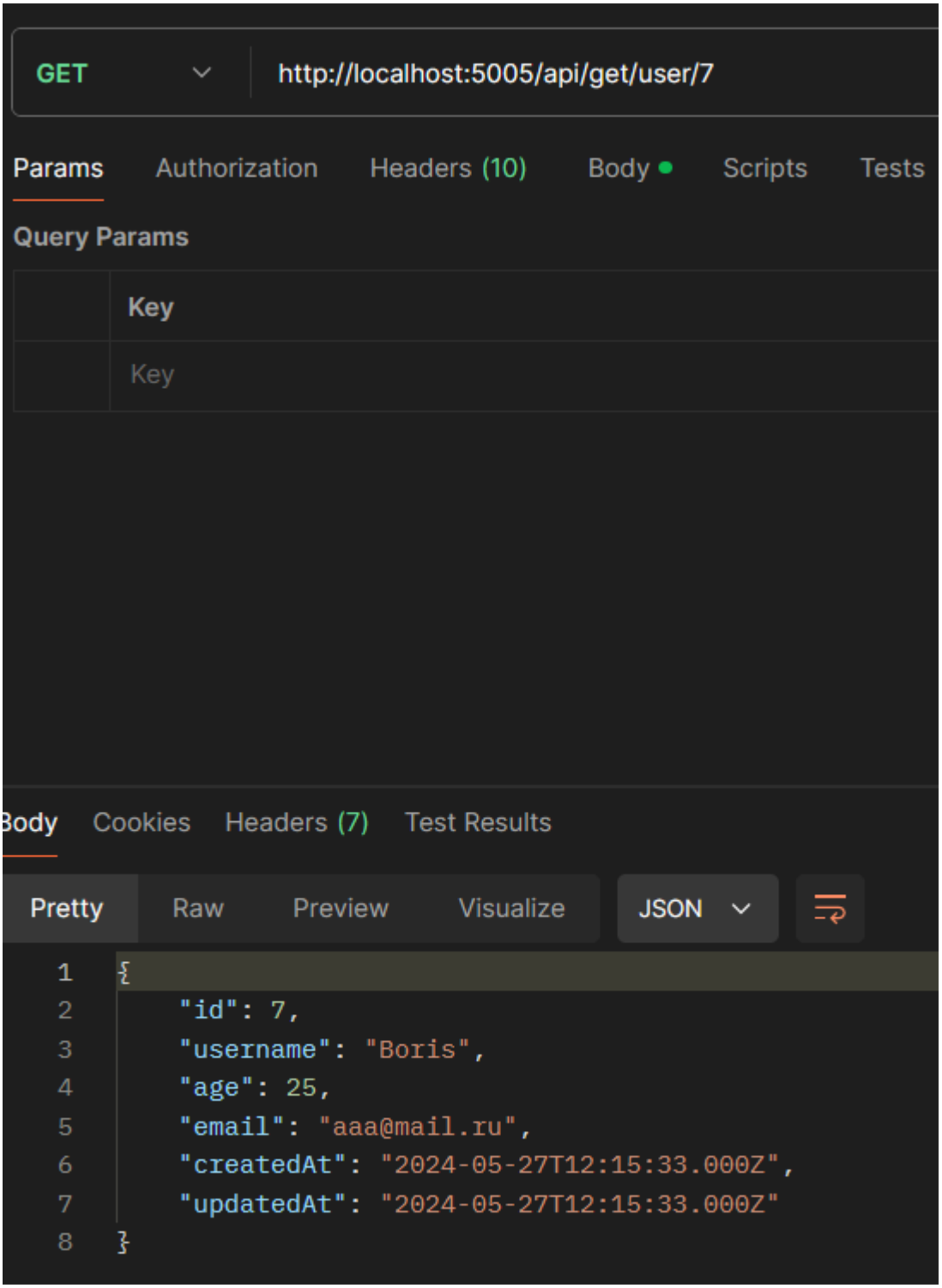
Для примера, создадим еще одного пользователя и выведем всех:




GET
http://localhost:5005/api/get/users
Params
Authorization
Headers (8)
Body
Scripts
Tests
Query Params
Key
Key
Body
Cookies
Headers (7)
Test Results
Pretty
Raw
Preview
Visualize
JSON
[
{
"id": 6,
"username": "Aleksey",
"age": 20,
"email": "test1@mail.ru",
"createdAt": "2024-05-27T12:12:02.000Z",
"updatedAt": "2024-05-27T12:12:02.000Z"
},
{
"id": 7,
"username": "Boris",
"age": 25,
"email": "aaa@mail.ru",
"createdAt": "2024-05-27T12:15:33.000Z",
"updatedAt": "2024-05-27T12:15:33.000Z"
}
]

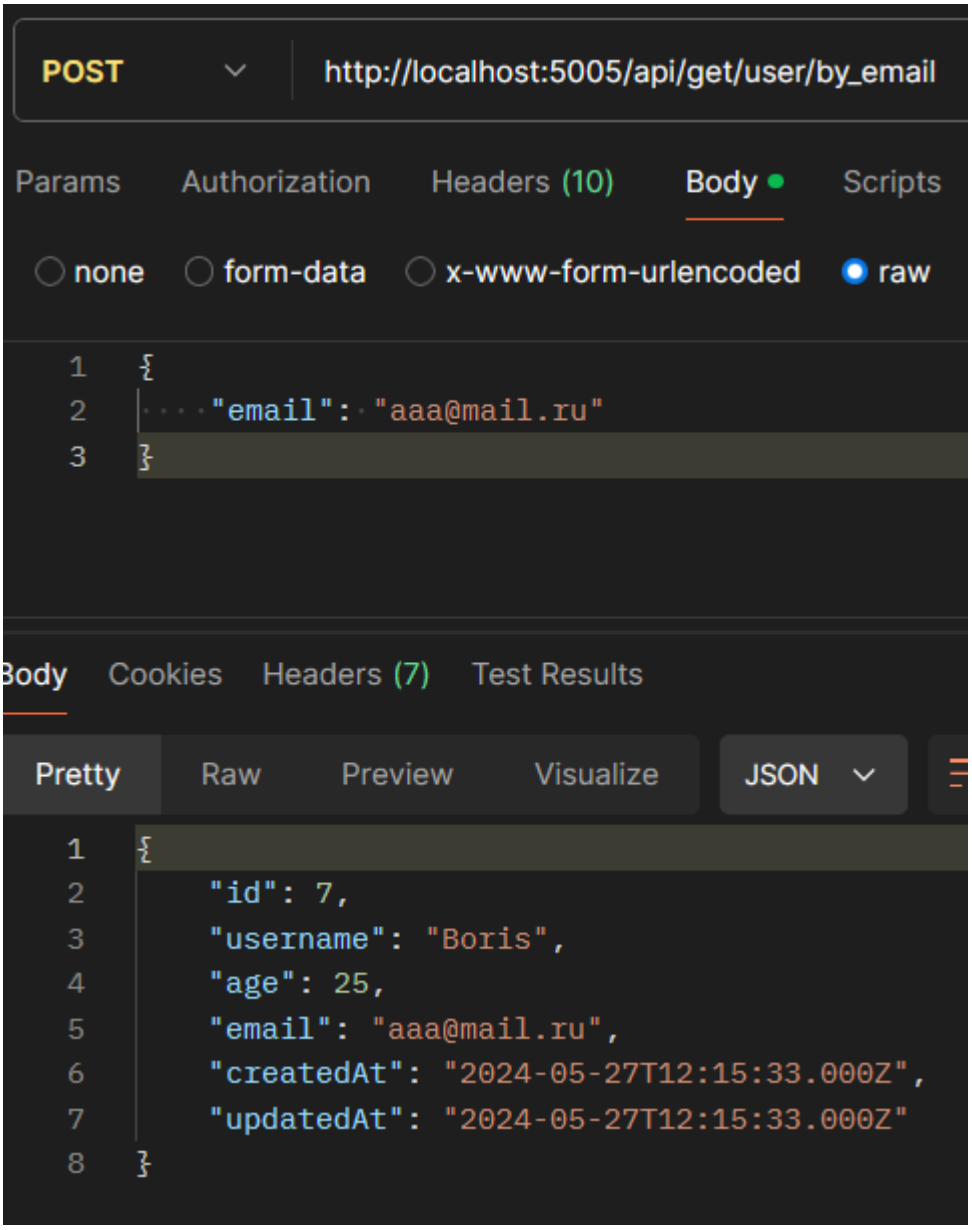
Следующая функция, это Получение определенного пользователя по id.
В данном контроллере id передается в самой ссылке.

```
exports.get_user_by_id = async (req, res) : Promise<...>  => {
    try {
        const user : Promise<...> | Promise<...>  = await db.User.findByPk(req.params.id);
        if (!user) {
            return res.status(404).json({ message: "User not found" });
        }
        return res.status(200).json(user);
    }
    catch (error) {
        return res.status(500).json({ message: error.message });
    }
}
```


GET
http://localhost:5005/api/get/user/7
Params
Authorization
Headers (10)
Body
Scripts
Tests
Query Params
Key
Key
Body
Cookies
Headers (7)
Test Results
Pretty
Raw
Preview
Visualize
JSON
{
"id": 7,
"username": "Boris",
"age": 25,
"email": "aaa@mail.ru",
"createdAt": "2024-05-27T12:15:33.000Z",
"updatedAt": "2024-05-27T12:15:33.000Z"
}

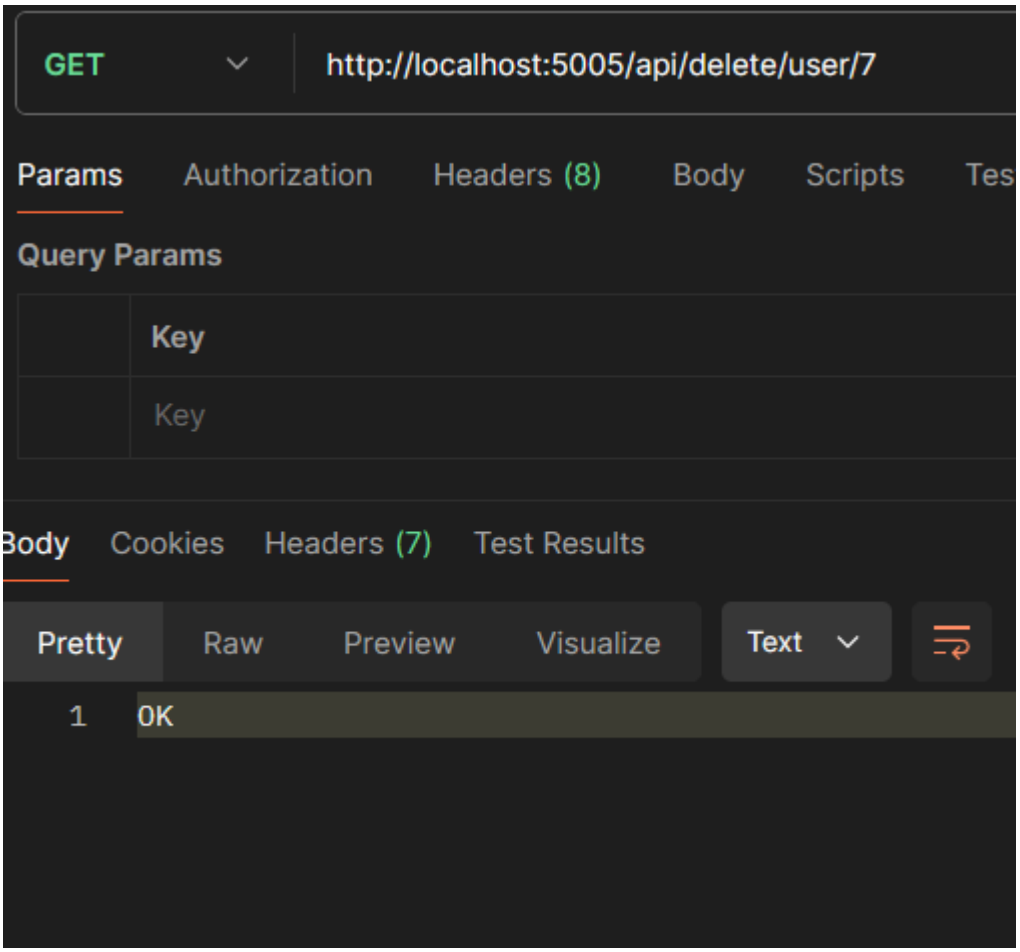
Аналогичный контроллер, который возвращает пользователя по его почте. В данном случае, почта передается приложенным json к запросу:

```
exports.get_user_by_email = async (req, res) : Promise<...>  => {
    try {
        const user : Promise<...> | Promise<...>  = await db.User.findOne({ where: {email: req.body.email} });
        if (!user) {
            return res.status(404).json({ message: "User not found" });
        }
        return res.status(200).json(user);
    }
    catch (error) {
        return res.status(500).json({ message: error.message });
    }
}
```



Функция удаления пользователя:

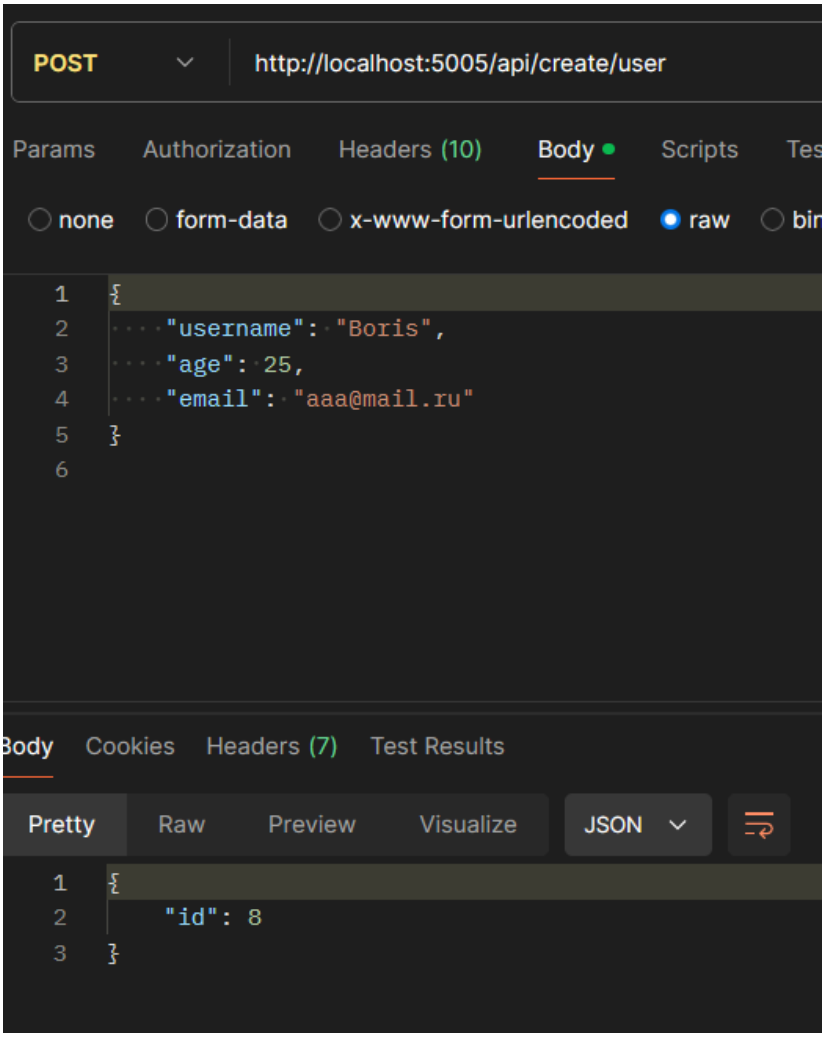
```
exports.delete_user = async (req, res) : Promise<...>  => {
    try {
        const deleted_Count = await db.User.destroy({ where: {id: req.params.id} });
        if (deleted_Count === 0) {
            return res.status(404).json({ message: "User not found" });
        }
        return res.sendStatus(200);
    }
    catch (error) {
        return res.status(500).json({ message: error.message });
    }
}
```

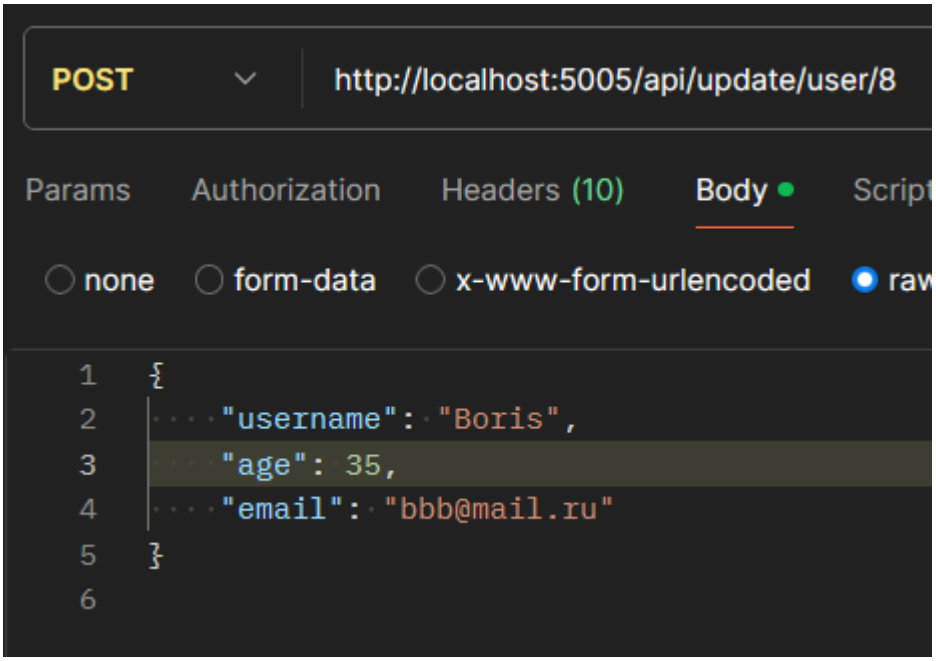


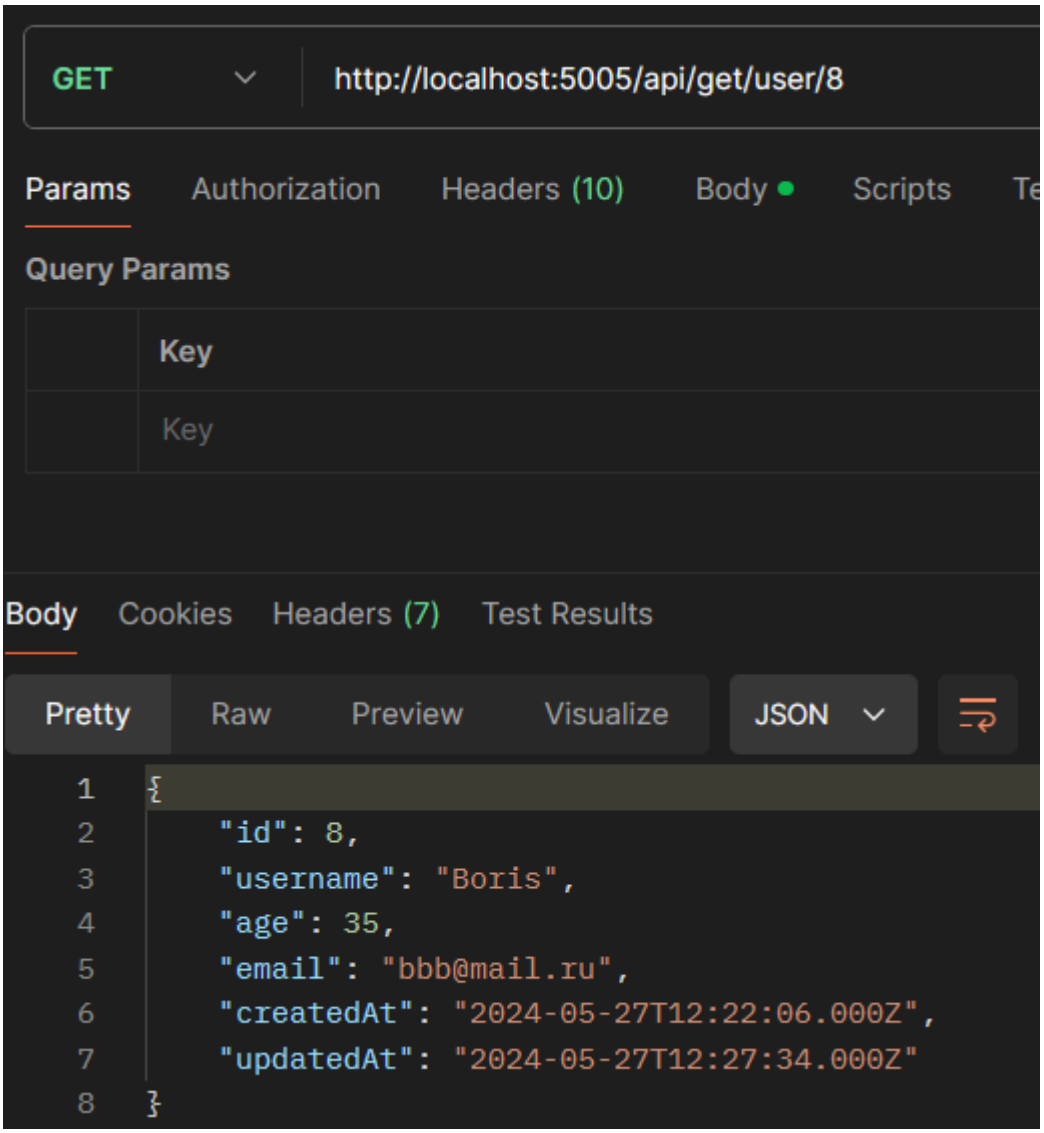
Функция обновления данных пользователя:

```
exports.update_user = async (req, res) : Promise<...>  => {
    try {
        const [updated_Count, [updated_user]] = await db.User.update(req.body, {
            where: {id: req.params.id},
            returning: true,
        });
        if (updated_Count === 0) {
            return res.status(404).json({ message: "User not found" });
        }
        return res.sendStatus(200);
    }
    catch (error) {
        return res.status(500).json({ message: error.message });
    }
}
```

Создадим еще одного бориса:




POST
http://localhost:5005/api/update/user/8
Params
Authorization
Headers (10)
Body
Script
none
form-data
x-www-form-urlencoded
raw
{
"username": "Boris",
"age": 35,
"email": "bbb@mail.ru"
}


GET
http://localhost:5005/api/get/user/8
Params
Authorization
Headers (10)
Body
Scripts
Te
Query Params
Key
Key
Body
Cookies
Headers (7)
Test Results
Pretty
Raw
Preview
Visualize
JSON
{
"id": 8,
"username": "Boris",
"age": 35,
"email": "bbb@mail.ru",
"createdAt": "2024-05-27T12:22:06.000Z",
"updatedAt": "2024-05-27T12:27:34.000Z"
}

Вывод

В данной работе была выполнена задача реализовать базовый HTTP-сервер, обрабатывающий CRUD операции с использованием express и sequilize библиотек, а также базы данных MySql