

**САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ  
ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО**

**Дисциплина:** Бэк-энд разработка

Отчет

Домашняя работа №2

Выполнил:

Тарабанов Алексей

Группа К33402

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 модель при помощи sequelize-cli

В качестве параметров выбрал:

username - Имя пользователя

age - возраст пользователя

email - почта пользователя

```
PS C:\Users\pc\Desktop\1\Back-end\ITMO-ICT-Backend-2024\homeworks\K33402\Тарабанов Алексей\hw2> npx sequelize-cli model:generate --name User --attributes username:string,age:integer,email:string
> hw2@1.0.0 npx
> sequelize-cli model:generate --name User --attributes username:string,age:integer,email:string

Sequelize CLI [Node: 20.12.0, CLI: 6.6.2, ORM: 6.37.3]

New model was created at C:\Users\pc\Desktop\1\Back-end\ITMO-ICT-Backend-2024\homeworks\K33402\Тарабанов Алексей\hw2\models\user.js .
New migration was created at C:\Users\pc\Desktop\1\Back-end\ITMO-ICT-Backend-2024\homeworks\K33402\Тарабанов Алексей\hw2\migrations\20240523180544-create-user.js .
PS C:\Users\pc\Desktop\1\Back-end\ITMO-ICT-Backend-2024\homeworks\K33402\Тарабанов Алексей\hw2>
```

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

```
config.json x
1  {
2    "development": {
3      "username": "database",
4      "password": "1234",
5      "database": "database_development",
6      "host": "93.100.111.248",
7      "dialect": "mysql"
8    },
9    "test": {
10     "username": "database",
11     "password": "1234",
12     "database": "database_development",
13     "host": "93.100.111.248",
14     "dialect": "mysql"
15   },
16   "production": {
17     "username": "database",
18     "password": "1234",
19     "database": "database_development",
20     "host": "93.100.111.248",
21     "dialect": "mysql"
22   }
23 }
```

После я мигрировал созданную модель в базу данных

```
PS C:\Users\pc\Desktop\1\Back-end\ITM0-ICT-Backend-2024\homeworks\K33402\Тарабанов Алексей\hw2> npx sequelize db:migrate

> hw2@1.0.0 npx
> sequelize db:migrate

Sequelize CLI [Node: 20.12.0, CLI: 6.6.2, ORM: 6.37.3]

Loaded configuration file "config\config.json".
Using environment "development".
== 20240523180544-create-user: migrating =====
== 20240523180544-create-user: migrated (0.056s)

PS C:\Users\pc\Desktop\1\Back-end\ITM0-ICT-Backend-2024\homeworks\K33402\Тарабанов Алексей\hw2>
```

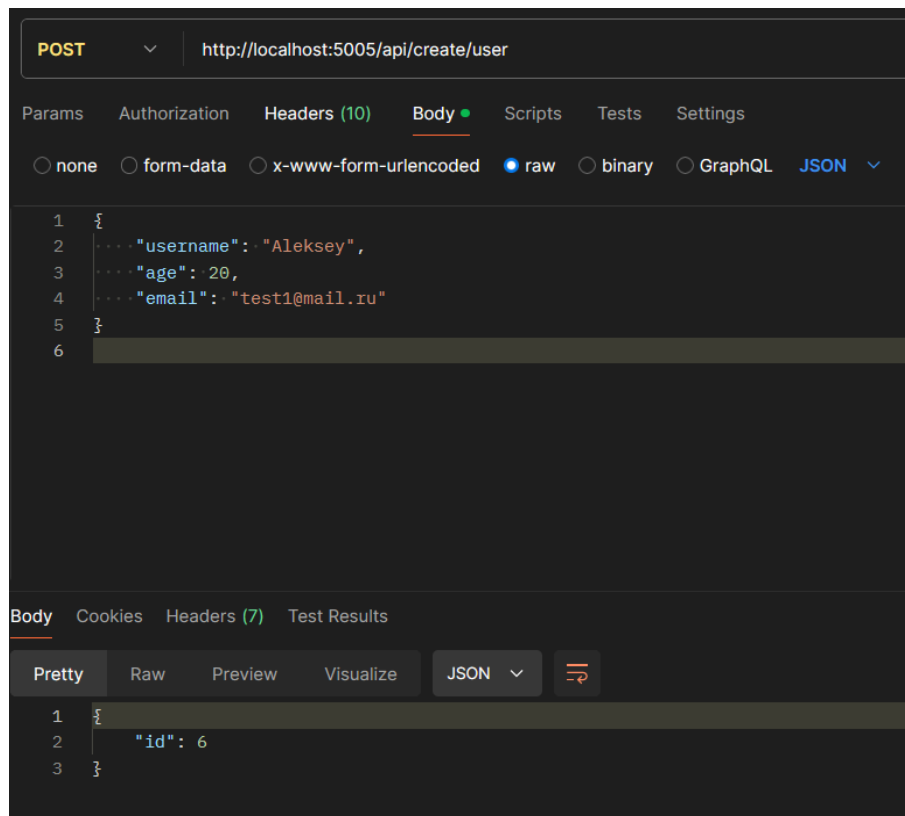
Дальше в коде я прописал эндпоинты ит вынес контроллеры в отдельную папку/файл.

```
9 app.post(path: '/api/create/user', userController.create_user) // Создать
10 app.get('/api/get/users', userController.get_users) // Поиск всех
11 app.get('/api/get/user/:id', userController.get_user_by_id) // Поиск по id
12 app.post(path: '/api/get/user/by_email', userController.get_user_by_email) // Поиск по почте
13 app.post(path: '/api/update/user/:id', userController.update_user) // Обновление по id
14 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 ▼

↺

1

[

2

{

3

"id": 6,

4

"username": "Aleksey",

5

"age": 20,

6

"email": "test1@mail.ru",

7

"createdAt": "2024-05-27T12:12:02.000Z",

8

"updatedAt": "2024-05-27T12:12:02.000Z"

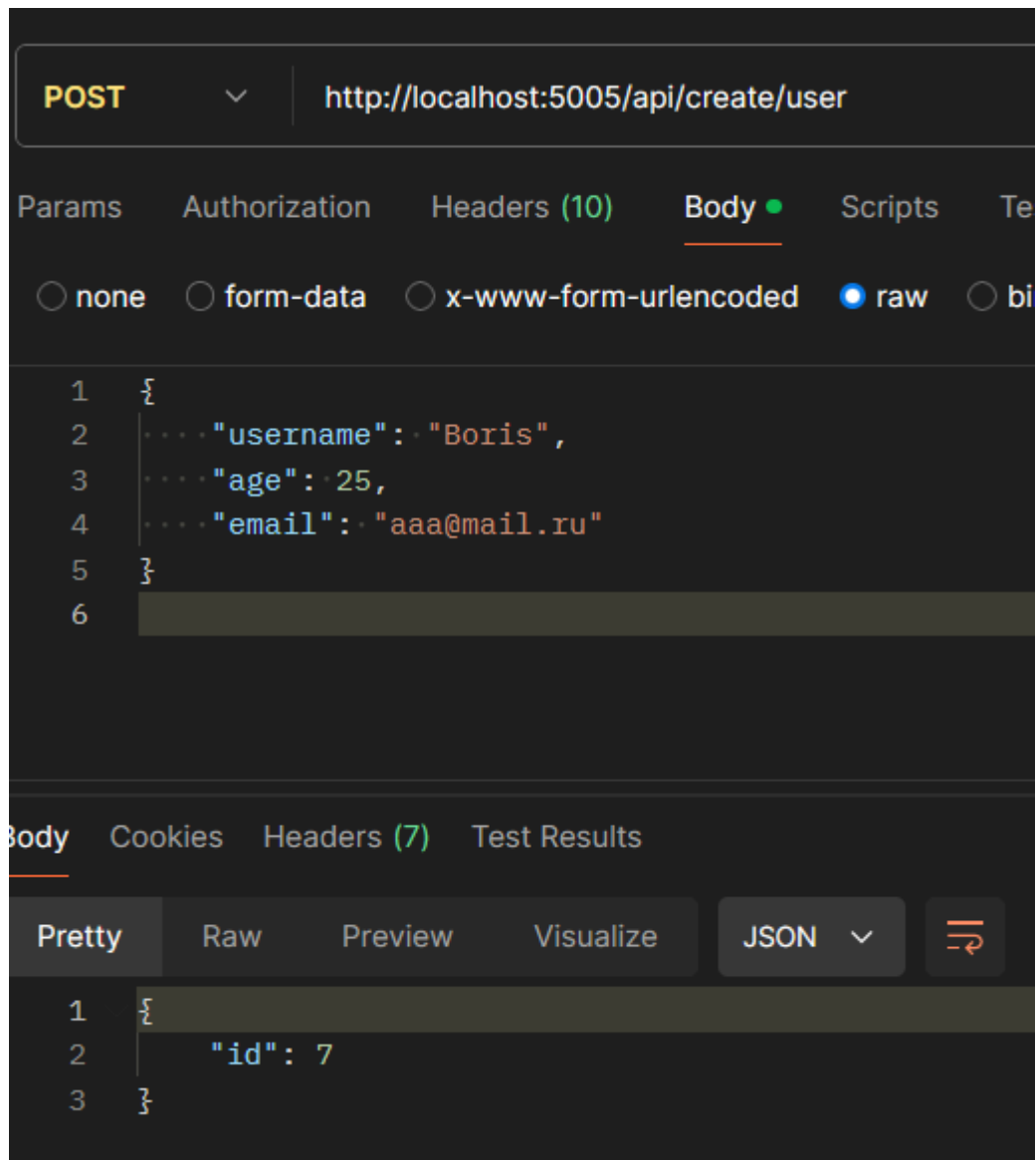
9

}

10

]

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



```
1  [  
2    {  
3      "id": 6,  
4      "username": "Aleksey",  
5      "age": 20,  
6      "email": "test1@mail.ru",  
7      "createdAt": "2024-05-27T12:12:02.000Z",  
8      "updatedAt": "2024-05-27T12:12:02.000Z"  
9    },  
10   {  
11     "id": 7,  
12     "username": "Boris",  
13     "age": 25,  
14     "email": "aaa@mail.ru",  
15     "createdAt": "2024-05-27T12:15:33.000Z",  
16     "updatedAt": "2024-05-27T12:15:33.000Z"  
17   }  
18 ]
```



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



```
1 {  
2   "id": 7,  
3   "username": "Boris",  
4   "age": 25,  
5   "email": "aaa@mail.ru",  
6   "createdAt": "2024-05-27T12:15:33.000Z",  
7   "updatedAt": "2024-05-27T12:15:33.000Z"  
8 }
```

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 });  
  }  
}
```

The screenshot displays a REST client interface. At the top, a POST request is configured to `http://localhost:5005/api/get/user/by_email`. The 'Body' tab is selected, showing a raw JSON payload: `{ "email": "aaa@mail.ru" }`. Below this, the 'Test Results' section shows the response body in a 'Pretty' JSON format: `{ "id": 7, "username": "Boris", "age": 25, "email": "aaa@mail.ru", "createdAt": "2024-05-27T12:15:33.000Z", "updatedAt": "2024-05-27T12:15:33.000Z" }`.

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 });  
  }  
}
```

The screenshot shows a REST client interface with the following details:

- Method:** GET
- URL:** http://localhost:5005/api/delete/user/7
- Params:** Authorization, Headers (8), Body, Scripts, Test
- Query Params:** A table with two rows, each containing a 'Key' header and an empty value field.
- Body:** Pretty, Raw, Preview, Visualize, Text (selected), and a refresh icon.
- Response:** 1 OK

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 });
  }
}
```

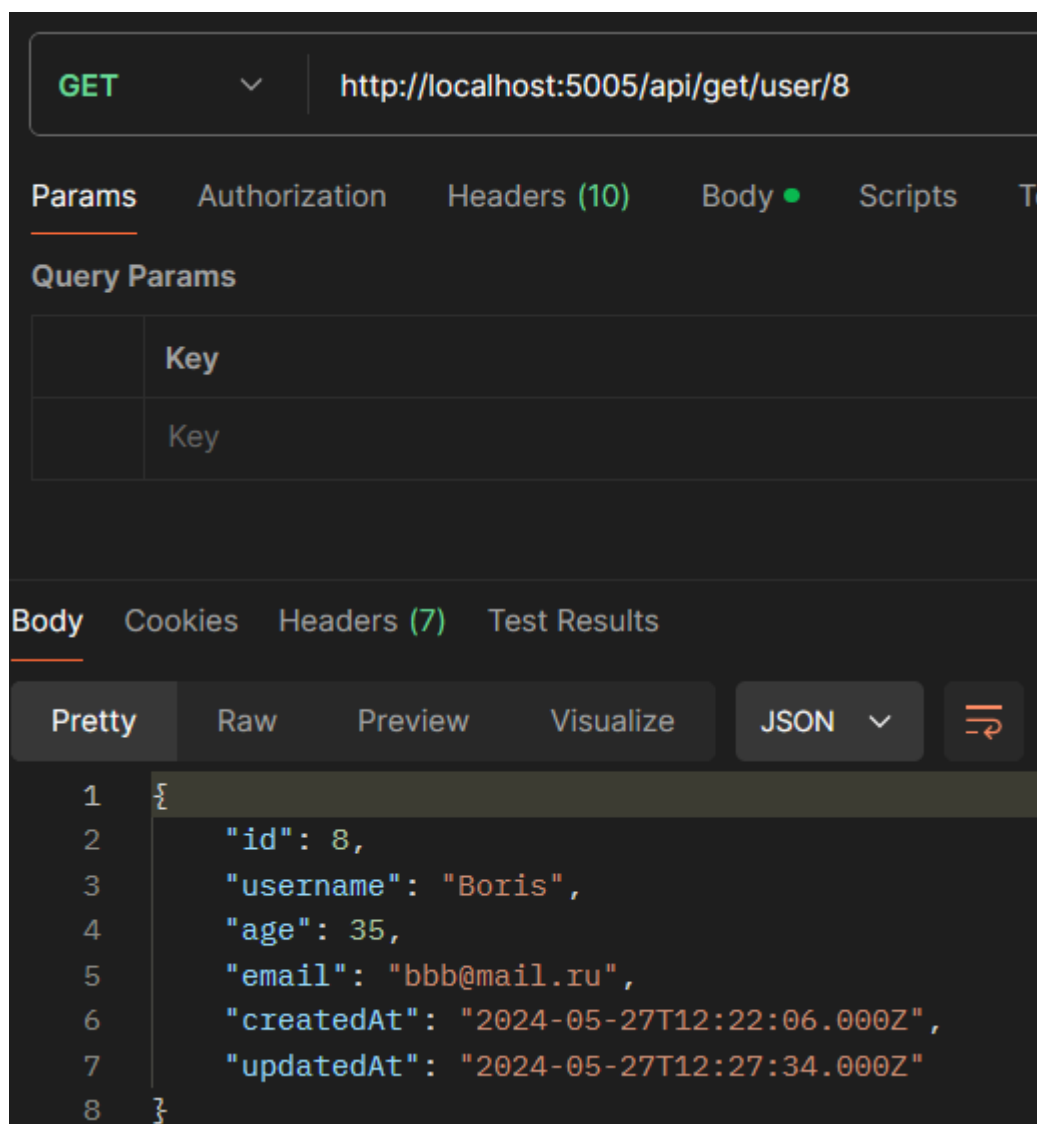
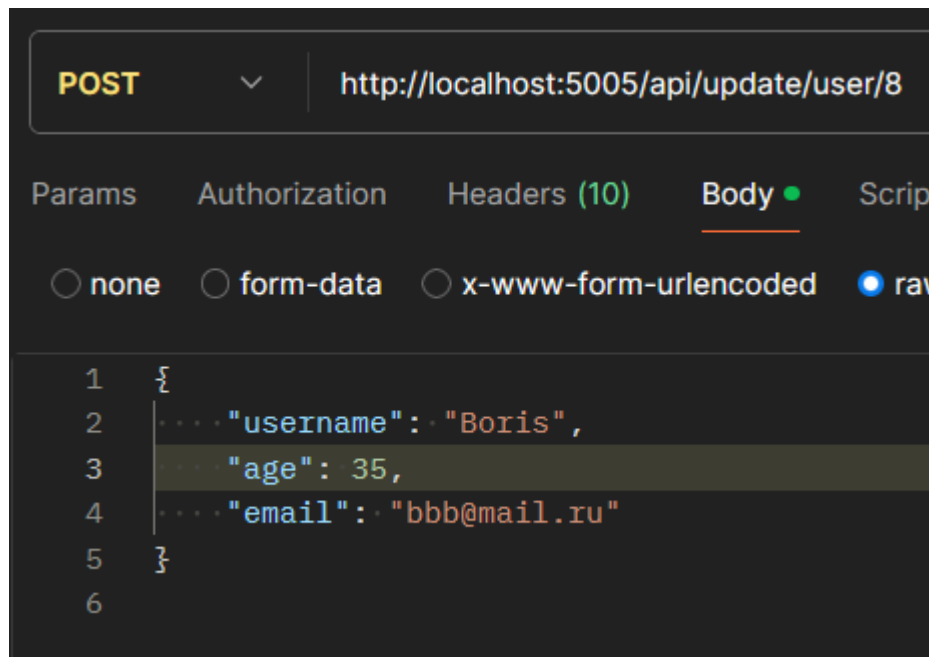
Создадим еще одного бориса:

The screenshot shows a REST client interface with a POST request to `http://localhost:5005/api/create/user`. The request body is a JSON object with the following fields:

```
{
  "username": "Boris",
  "age": 25,
  "email": "aaa@mail.ru"
}
```

The response body is a JSON object with the following field:

```
{
  "id": 8
}
```



## **Вывод**

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