

**САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ  
ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО**

**Дисциплина:** Бэк-энд разработка

Отчет

Домашнее задание №2  
«Знакомство с ORM Sequelize»

Выполнила:  
Коник Анастасия

Группа:  
К33402

Проверил:  
Добряков Д. И.

Санкт-Петербург

2023 г.

## Задача

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

## Ход работы

Инициализируем проект и качаем необходимые инструменты, сохранив их в файл конфигурации.

С помощью sequelize-cli создаем модель пользователя и миграции:

```
npx sequelize-cli model:generate --name User --attributes  
firstName:string,lastName:string,email:string,password:string,img_url:string,u  
sername:string
```

```
npx sequelize-cli db:migrate
```

```
'use strict';  
const {  
  Model  
} = require('sequelize');  
module.exports = (sequelize, DataTypes) => {  
  2 usages new *  
  class User extends Model {  
    /**  
     * Helper method for defining associations.  
     * This method is not a part of Sequelize lifecycle.  
     * The 'models/index' file will call this method automatically.  
     */  
    2 usages new *  
    static associate(models) {  
      // define association here  
    }  
  }  
  User.init({ attributes: {  
    firstName: DataTypes.STRING,  
    lastName: DataTypes.STRING,  
    email: DataTypes.STRING,  
    password: DataTypes.STRING,  
    img_url: DataTypes.STRING,  
    username: DataTypes.STRING  
  }, options: {  
    sequelize,  
    modelName: 'User',  
  }}, {  
    return User;  
  });  
};
```

Файл index.js с нужными запросами:

```
const express = require('express')
const db = require('./models')

const app = express()
const port = 3000

app.use(express.json())
new *
app.get('/', (req :... , res : Response<ResBody, LocalsObj> ) => {
  res.send( body: 'Homework 2')
})

new *
app.get('/users/:id', async (req :... , res : Response<ResBody, LocalsObj> ) => {
  const user = await db.User.findById(req.params.id)

  if (user) {
    return res.send(user.toJSON())
  }

  return res.send( body: {"msg": "user is not found"})
})

new *
app.get('/users', async (req :... , res : Response<ResBody, LocalsObj> ) => {
  const users = await db.User.findAll()
  res.send(users)
})
```

```

app.post( path: '/users', handlers: async (req :... , res : Response<ResBody, LocalsObj> ) => {
  try {
    console.log(req.body)
    const user = await db.User.create(req.body);
    res.send(user.toJSON());
  } catch (e) {
    res.status( code: 400).send( body: {"detail": "Failed to create user"});
  }
})

new *
app.put( path: '/users/:id', handlers: async (req :... , res : Response<ResBody, LocalsObj> ) => {
  try {
    console.log(req.body)
    const user = await db.User.update(req.body, {where: {id: req.params.id}});
    res.send( body: {'status_code': 'User updated'});
  } catch (err) {
    res.send(err);
  }
})

new *
app.delete( path: '/users/:id', handlers: async (req :... , res : Response<ResBody, LocalsObj> ) => {
  const user = await db.User.destroy({where: {id: req.params.id}})
  if (user) {
    res.send( body: {'status_code': 'User deleted'})
  } else {
    res.status( code: 404).send( body: {'error_code': 'User not found'})
  }
})

new *
app.listen(port, hostname: () => {
  console.log('Example app listening on port ${port}')
})

```

Файлик для создания пользователя:

```

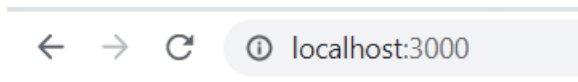
const db = require('./models')

1 usage new *
async function main() {
  await db.User.create({
    firstName: 'Anastasia',
    lastName: 'Konik',
    email: 'konik.ftl@mail.ru',
    password: '123konik123',
    username: 'anastasiakonik'
  })
}

main()

```

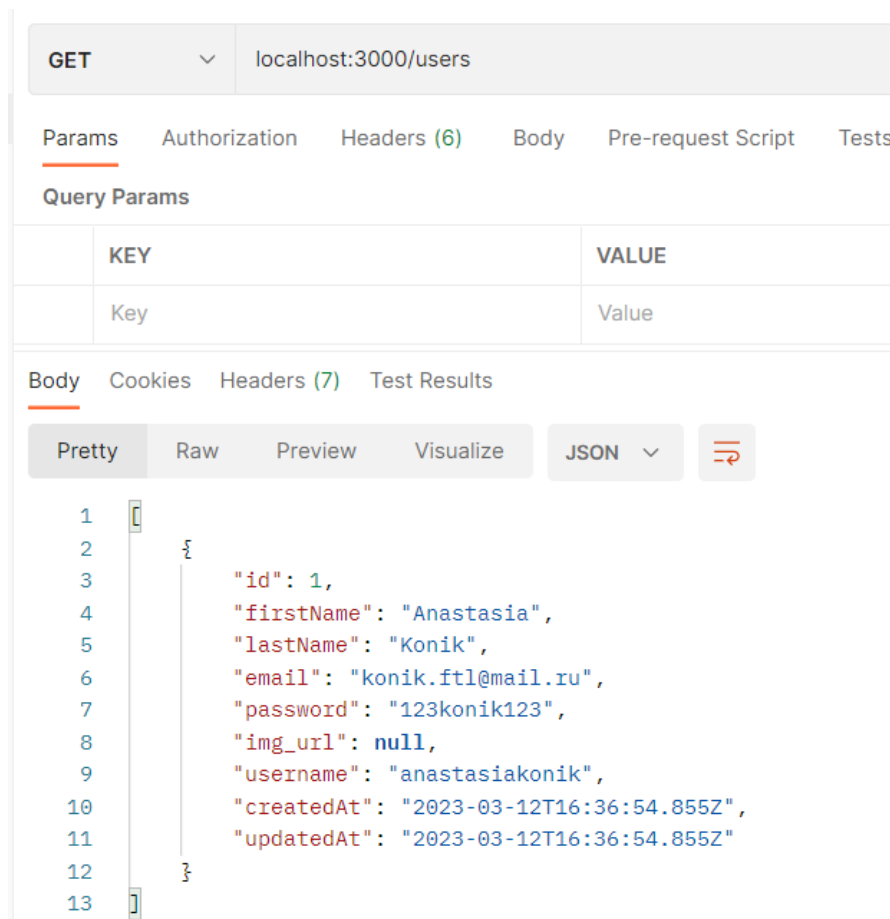
Проверка работоспособности сервера:



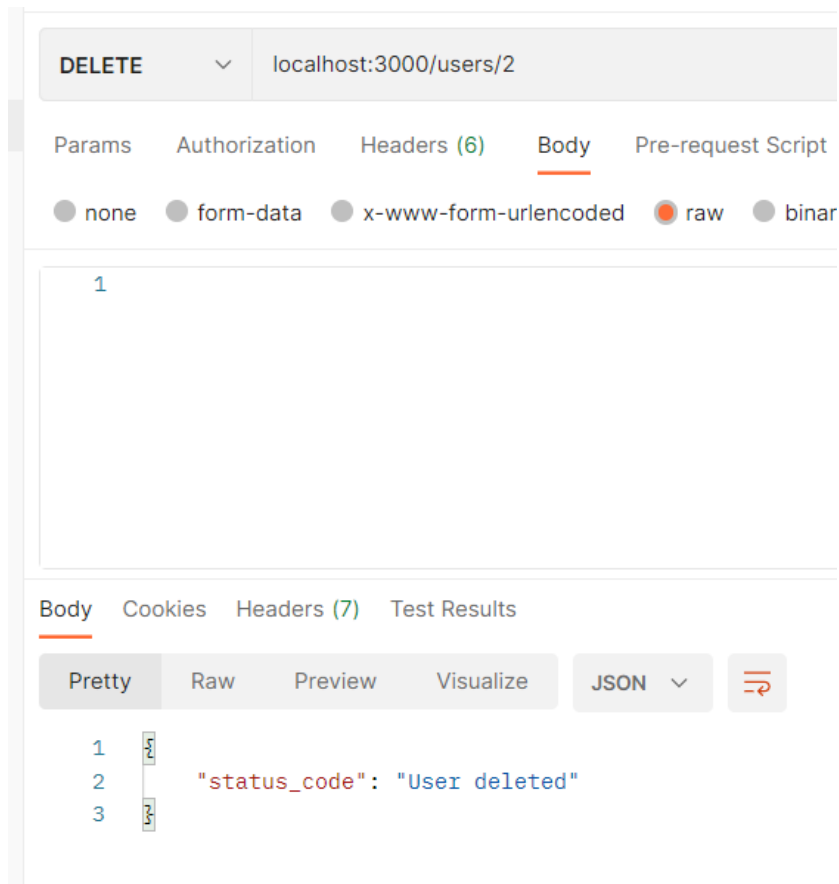
Homework 2

Запросы в postman:

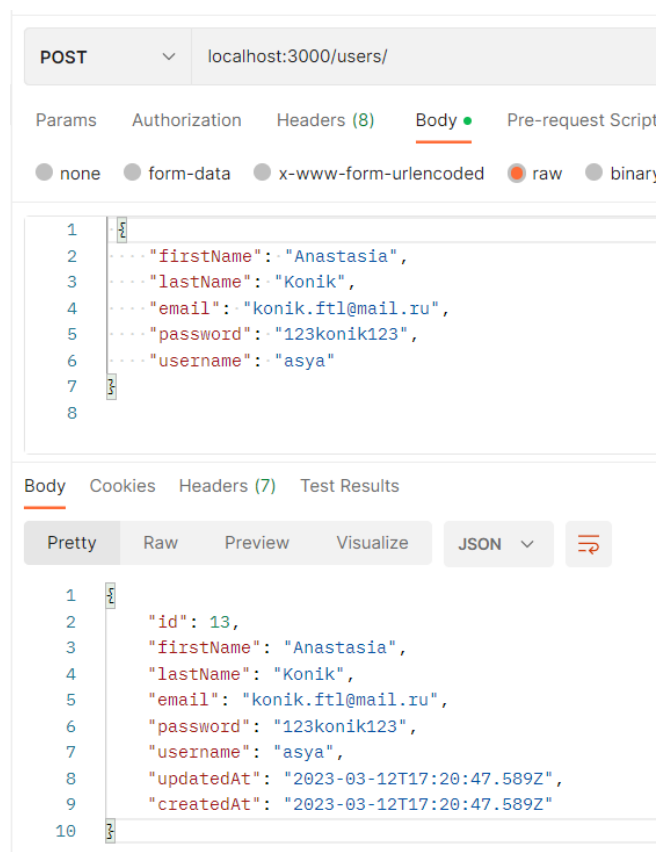
– получение пользователей:



– удаление пользователя по id:



– добавление пользователя в бд:



– изменение информации о пользователе:

The screenshot shows a REST client interface. At the top, the method is **PUT** and the URL is `localhost:3000/users/1`. Below this, there are tabs for **Params**, **Authorization**, **Headers (9)**, and **Body** (which is selected). Under the **Body** tab, there are radio buttons for **none**, **form-data**, and **x-www-form-urlencoded**, with **x-www-form-urlencoded** being selected. The body content is a JSON object:

```
1 {  
2   "firstName": "Anastasia",  
3   "lastName": "Konik",  
4   "email": "konik.ftl@mail.ru",  
5   "password": "123konik123",  
6   "img_url": null,  
7   "username": "asya",  
8 }  
9
```

Below the body tab, there are tabs for **Body** (selected), **Cookies**, **Headers (7)**, and **Test Results**. Under the **Body** tab, there are buttons for **Pretty**, **Raw**, **Preview**, **Visualize**, and **JSON**. The **Pretty** button is selected, and the response is displayed in a formatted JSON:

```
1 {  
2   "status_code": "User updated"  
3 }
```

## Вывод:

В ходе работы я познакомилась с ORM Sequelize и созданием с ее помощью моделей, а также написала запросы к бд с помощью Express.