

**САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ
ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО**

Дисциплина: Бэк-энд разработка

Отчет

Домашняя работа №2

Выполнил:

Шляхов Денис Олегович

Группа К33402

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 + Sequelize
- Написать запрос для получения пользователя по id/email

Ход работы

1. npm init
2. npm i express sequelize sqlite3 sequelize-cli body-parser
3. npx sequelize init
4. Инициализирую модель user через cli

```
PS C:\Users\Денис\PycharmProjects\ITMO-ICT-Backend-2024\homeworks\K33402\shLiakhov_denis\hw_2> npx sequelize-cli model:generate --name User --attributes firstName:string,lastName:string,email:string,password:string

Sequelize CLI [Node: 20.11.1, CLI: 6.6.2, ORM: 6.37.1]
New model was created at C:\Users\Денис\PycharmProjects\ITMO-ICT-Backend-2024\homeworks\K33402\shLiakhov_denis\hw_2\models\user.js .
New migration was created at C:\Users\Денис\PycharmProjects\ITMO-ICT-Backend-2024\homeworks\K33402\shLiakhov_denis\hw_2\migrations\20240314194213-create-user.js .
```

5. npx sequelize db:migrate

Сгенерилась такая вот модель

```
1  'use strict';
2  const {
3    Model
4  } = require('sequelize');
5  module.exports = (sequelize, DataTypes) => {
6    1+ usages  ⓘ Puroblast
7    class User extends Model {
8      /**
9       * Helper method for defining associations.
10      * This method is not a part of Sequelize lifecycle.
11      * The `models/index` file will call this method automatically.
12      */
13      1+ usages  ⓘ Puroblast
14      static associate(models) {
15        // define association here
16      }
17    }
18    User.init({ attributes: {
19      firstName: DataTypes.STRING,
20      lastName: DataTypes.STRING,
21      email: DataTypes.STRING,
22      password: DataTypes.STRING
23    }, options: {
24      sequelize,
25      modelName: 'User',
26    }
27  });
28  return User;
29  };
```

6. Реализация CRUD и GET запросов

```

  Puroblast
  app.post(path: "/users", handlers: async (req: Request<P, ResBody, ReqBody, ReqQuery, LocalsObj> , res: Response<ResBody, LocalsObj> ) : Promise<any> => {
    user = await db.User.create(req.body)
    return res.json(user)
  })

  //read
  Puroblast
  app.get('/users', async (req: Request<P, ResBody, ReqBody, ReqQuery, LocalsObj> , res: Response<ResBody, LocalsObj> ) : Promise<any> => {
    const users: Model[] = await db.User.findAll()

    if (users) {
      return res.send(users)
    }

    return res.send({msg: "users are not found"})
  })

  //read id
  Puroblast
  app.get('/users/:id', async (req: Request<P, ResBody, ReqBody, ReqQuery, LocalsObj> , res: Response<ResBody, LocalsObj> ) : Promise<any> => {
    const user: Promise<...> | Promise<...> = await db.User.findByIdPk(req.params.id)

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

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

  Puroblast
  app.get('/users/:email', async (req: Request<P, ResBody, ReqBody, ReqQuery, LocalsObj> , res: Response<ResBody, LocalsObj> ) : Promise<any> => {
    const user: Promise<...> | Promise<...> = await db.User.findByIdPk(req.params.email)

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

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

  //update
  Puroblast
  app.put(path: '/users/:id', handlers: async (req: Request<P, ResBody, ReqBody, ReqQuery, LocalsObj> , res: Response<ResBody, LocalsObj> ) : Promise<any> => {
    const num = await db.User.update(req.body, { where: { id: req.params.id } })

    if (num == 1) {
      return res.send({msg: 'user has been updated'})
    }

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

  //delete
  Puroblast
  app.delete(path: '/users/:id', handlers: async (req: Request<P, ResBody, ReqBody, ReqQuery, LocalsObj> , res: Response<ResBody, LocalsObj> ) : Promise<any> => {
    const user: Promise<...> | Promise<...> = await db.User.findByIdPk(req.params.id)

    if (user) {
      await user.destroy()
    }

    return res.send({msg: "user is deleted"})
  });

```

7. Результаты работы:

The image displays three sequential screenshots of a REST client interface, demonstrating the results of API calls.

First Screenshot: POST Request

- Method:** POST
- URL:** http://127.0.0.1:3000/users
- Body (JSON):**

```
{  "id": 1,  "firstName": "Denis",  "lastName": "Shliakhov",  "email": "asdasd@mail.ru",  "password": "12345",  "updatedAt": "2024-03-14T20:16:493Z",  "createdAt": "2024-03-14T20:16:493Z"}
```
- Status:** 200 OK, Time: 16 ms, Size: 407 B

Second Screenshot: GET Request

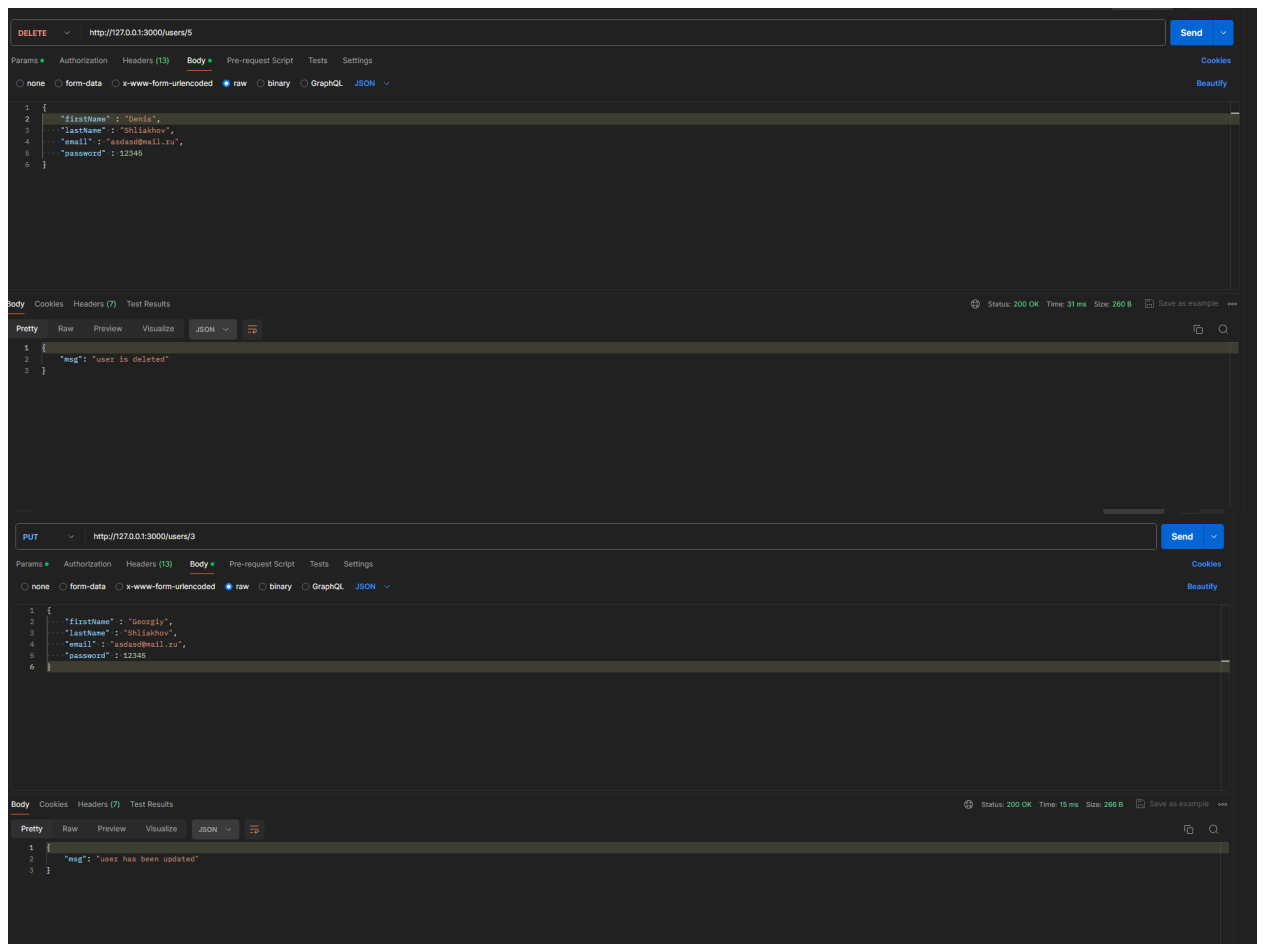
- Method:** GET
- URL:** http://127.0.0.1:3000/users
- Body:** This request does not have a body.
- Status:** 200 OK, Time: 4 ms, Size: 587 B

Third Screenshot: GET Request

- Method:** GET
- URL:** http://127.0.0.1:3000/users/1
- Body:** This request does not have a body.
- Status:** 200 OK, Time: 11 ms, Size: 409 B

Response Body (JSON):

```
{  "id": 1,  "firstName": "Denis",  "lastName": "Shliakhov",  "email": "asdasd@mail.ru",  "password": "12345",  "updatedAt": "2024-03-14T20:16:493Z",  "createdAt": "2024-03-14T20:16:493Z"}
```



Вывод

В ходе данной работы были изучены основы работы с `express` и `sequelize`