

**САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ  
ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО**

**Дисциплина:** Бэк-энд разработка

Отчет

Домашняя работа №2: Знакомство с ORM  
Sequelize

Выполнила:

Балдина Дарья

Группа К33401

Проверил:

Добряков Д. И.

Санкт-Петербург

2023 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. Модель пользователя состоит из 5 полей:

- имя
- фамилия
- ник(username)
- email
- пароль

```
'use strict';
const {
  Model
} = require('sequelize');
module.exports = (sequelize, DataTypes) => {
  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.
     */
    static associate(models) {
      // define association here
    }
  }
  User.init({
    firstName: DataTypes.STRING,
    lastName: DataTypes.STRING,
    username: DataTypes.STRING,
    email: DataTypes.STRING,
    password: DataTypes.STRING
  }, {defaultScope: {
    attributes:{exclude:['password']},
  },
  },
  sequelize,
  {modelName: 'User',
  });
  return User;
};
```

## Представления:

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

const app = express()
const port = 3000

app.use(express.json());

app.get('/', (req, res) => {
  res.send('Homework 2')
})

app.get('/users/:id', async (req, res) => {
  const user = await db.User.findByPk(req.params.id)
  if (user) {
    return res.send(user.toJSON())
  }

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

app.get('/users', async (req, res) => {
  const users = await db.User.findAll()
  res.send(users)
})

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

app.put('/users/:id', async (req, res) => {
  try {
    const user = await db.User.update(req.body, {where: {id: req.params.id}});
    res.send({'status_code': 'User updated'});
  } catch (err) {
    res.send(err);
  }
})

app.delete('/users/:id', async (req, res) => {
  const user = await db.User.destroy({ where: { id: req.params.id } })
  if (user) {
    res.send({'status_code': 'User deleted'})
  } else {
    res.status(404).send({'error_code': 'User not found'})
  }
})

app.listen(port, () => {
  console.log(`Example app listening on port ${port}`)
})
```

## 2. Набор из CRUD-методов для работы с пользователями средствами Express + Sequelize

### CREATE (POST):

POST ⌵ http://localhost:3000/users Send ⌵

Params Authorization Headers (9) **Body** ● Pre-request Script Tests Settings Cookies

● none ● form-data ● x-www-form-urlencoded ● raw ● binary ● GraphQL **JSON** ⌵ Beautify

```
1 {
2   "firstName": "Ольга",
3   "lastName": "Максимова",
4   "username": "olga2",
5   "email": "olga@example.com",
6   "password": "1234ivan"
7 }
```

Body Cookies Headers (7) Test Results 🌐 Status: 200 OK Time: 103 ms Size: 448 B Save Response ⌵

Pretty Raw Preview Visualize **JSON** ⌵ 🔍

```
1 {
2   "id": 16,
3   "firstName": "Ольга",
4   "lastName": "Максимова",
5   "username": "olga2",
6   "email": "olga@example.com",
7   "password": "1234ivan",
8   "updatedAt": "2023-03-10T12:21:12.843Z",
9   "createdAt": "2023-03-10T12:21:12.843Z"
10 }
```

### READ (GET):

GET ⌵ http://localhost:3000/users Send ⌵

Params Authorization Headers (7) **Body** ● Pre-request Script Tests Settings Cookies

● none ● form-data ● x-www-form-urlencoded ● raw ● binary ● GraphQL **JSON** ⌵ Beautify

1

Body Cookies Headers (7) Test Results 🌐 Status: 200 OK Time: 24 ms Size: 611 B Save Response ⌵

Pretty Raw Preview Visualize **JSON** ⌵ 🔍

```
1 [
2   {
3     "id": 1,
4     "firstName": "Иван",
5     "lastName": "Иванов",
6     "username": "ivan1",
7     "email": "ivan@example.com",
8     "createdAt": "2023-03-08T09:35:09.148Z",
9     "updatedAt": "2023-03-08T09:35:09.148Z"
10  },
11  {
12    "id": 16,
13    "firstName": "Ольга",
14    "lastName": "Максимова",
15    "username": "olga2",
16    "email": "olga@example.com",
17    "createdAt": "2023-03-10T12:21:12.843Z",
18    "updatedAt": "2023-03-10T12:21:12.843Z"
19  }
20 ]
```

### UPDATE (PUT):

PUT

http://localhost:3000/users/1

Send

Params

Authorization

Headers (9)

Body

Pre-request Script

Tests

Settings

none

form-data

x-www-form-urlencoded

raw

binary

GraphQL

JSON

1

2

3

4

5

6

"firstName": "Иван",

"lastName": "Иванов",

"username": "ivan123",

"email": "ivan@example.com"

Body

Cookies

Headers (7)

Test Results

Status: 200 OK

Time: 129 ms

Size: 265 B

Save Response

Pretty

Raw

Preview

Visualize

JSON

1

2

3

"status\_code": "User updated"

DELETE:

The screenshot shows the REST Client interface with a DELETE request to `http://localhost:3000/users/16`. The request body is a JSON object: `{"firstName": "Иван", "lastName": "Иванов", "username": "ivan123", "email": "ivan@example.com"}`. The response is shown in the bottom panel, indicating a 200 OK status with the message: `{"status_code": "User deleted"}`. The interface includes tabs for Params, Authorization, Headers, Body, Pre-request Script, Tests, and Settings. The Body tab is selected, and the response is displayed in the bottom panel.

### 3. Запрос для получения пользователя по id

The screenshot shows the REST Client interface with a GET request to `http://localhost:3000/users/1`. The request body is a JSON object: `{ "firstName": "Иван", "lastName": "Иванов", "username": "ivan123", "email": "ivan@example.com" }`. The response is a JSON object: `{ "id": 1, "firstName": "Иван", "lastName": "Иванов", "username": "ivan123", "email": "ivan@example.com", "createdAt": "2023-03-08T09:35:09.148Z", "updatedAt": "2023-03-10T13:12:19.814Z" }`. The status is 200 OK, time is 11 ms, and size is 419 B.

## Вывод

В ходе выполнения домашнего задания была придумана и создана собственная модель пользователя средствами ORM Sequelize, а также реализован набор CRUD-методов для работы с моделью юзера с помощью Express.