

**САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ  
УНИВЕРСИТЕТ ИТМО**

**Дисциплина:** Бэк-энд разработка

**Отчет**

**Домашняя работа 2: Знакомство с ORM Sequelize**

**Выполнил:**  
Ле Хоанг Чьонг

**Группа:**  
К33392

**Проверил:**  
Добряков Д. И.

Санкт-Петербург  
2024 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/email

## Ход работы

1. Собственная модель пользователя выглядит следующим образом:

firstName (Имя)  
lastName (Фамилия)  
email (Почта)  
password (Пароль)  
username (Логин)

```
'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,
    email: DataTypes.STRING,
    password: DataTypes.STRING,
    username: DataTypes.STRING
  }, {
    sequelize,
    modelName: 'User',
  });
  return User;
};
```

## 2. Реализация CRUD-методов средствами Express + Sequelize

POST /users' | Создать нового пользователя

```
exports.createUser = async (req, res) => {
  try {
    const { name, email, password } = req.body;
    const newUser = await db.User.create({ name, email, password });
    res.status(201).json(newUser);
  } catch (error) {
    res.status(400).json({ message: error.message });
  }
};
```

GET /users/:idOrEmail' | Получить пользователя по ID или электронной почте

```
exports.getUserByIdOrEmail = async (req, res) => {
  try {
    const { idOrEmail } = req.params;
    const user = await db.User.findOne({
      where: {
        [Op.or]: [{ id: idOrEmail }, { email: idOrEmail }]
      }
    });
    if (!user) {
      return res.status(404).json({ message: 'User not found' });
    }
    res.json(user);
  } catch (error) {
    res.status(500).json({ message: error.message });
  }
};
```

GET /users/ | Получить всех пользователей

```
exports.getAllUser = async (req, res) => {
  try {
    const user = await db.User.findAll();
    if (!user) {
      return res.status(404).json({ message: 'User not found' });
    }
    res.json(user);
  } catch (error) {
    res.status(500).json({ message: error.message });
  }
};
```

DELETE /users/:idOrEmail' | Удалить пользователя по ID или электронной почте

```
exports.deleteUserByIdOrEmail = async (req, res) => {
  try {
    const { idOrEmail } = req.params;
    const user = await db.User.destroy({
      where: {
        [Op.or]: [{ id: idOrEmail }, { email: idOrEmail }]
      }
    });
    if (!user) {
      return res.status(404).json({ message: 'User not found' });
    }
    res.send({ 'status_code': 'User deleted' })
  } catch (error) {
    res.status(500).json({ message: error.message });
  }
};
```

PUT /users/:idOrEmail' | Обновить пользователя по ID или электронной почте

```
exports.updateUserByIdOrEmail = async (req, res) => {
  try {
    const { idOrEmail } = req.params;
    const user = await db.User.update(req.body, {
      where: {
        [Op.or]: [{ id: idOrEmail }, { email: idOrEmail }]
      }
    });
    if (!user) {
      return res.status(404).json({ message: 'User not found' });
    }
    res.send({ 'status_code': 'User updated' })
  } catch (error) {
    res.status(500).json({ message: error.message });
  }
};
```

### 3. Проверка работы CRUD-методов с помощью Postman

POST /users'

The screenshot displays the Postman interface for a POST request. The top bar shows the method 'POST' and the URL 'http://localhost:8080/api/users'. Below this, the 'Body' tab is selected, showing a JSON payload. The bottom section shows the 'Test Results' tab with the response body formatted as JSON.

**Request:**

```
1 {
2   "firstName": "Truong",
3   "lastName": "Le",
4   "email": "example@example.com",
5   "password": "password123",
6   "username": "truongle"
7 }
8
```

**Response:**

```
1 {
2   "id": 5,
3   "lastName": "Le",
4   "firstName": "Truong",
5   "email": "example@example.com",
6   "password": "password123",
7   "updatedAt": "2024-03-05T18:30:23.777Z",
8   "createdAt": "2024-03-05T18:30:23.777Z"
9 }
```

GET /users/:idOrEmail' | Получить пользователя по ID или электронной почте

GET

▼

http://localhost:8080/api/users/4

ParamsAuthorizationHeaders (6)BodyPre-request ScriptTestsSettings

Query Params

	Key
	Key

BodyCookiesHeaders (7)Test Results

PrettyRawPreviewVisualizeJSON▼☰

```
1  {
2    "id": 4,
3    "firstName": "Truong",
4    "lastName": "Le",
5    "email": "example@example.com",
6    "password": "password123",
7    "username": null,
8    "createdAt": "2024-03-05T18:29:50.509Z",
9    "updatedAt": "2024-03-05T18:29:50.509Z"
```

GET

⌵

http://localhost:8080/api/users/example@example.com

Params

Authorization

Headers (6)

Body

Pre-request Script

Tests

Settings

Query Params

	Key
	Key

Body

Cookies

Headers (7)

Test Results

Pretty

Raw

Preview

Visualize

JSON ⌵

```
1  {
2    "id": 4,
3    "firstName": "Truong",
4    "lastName": "Le",
5    "email": "example@example.com",
6    "password": "password123",
7    "username": null,
8    "createdAt": "2024-03-05T18:29:50.509Z",
9    "updatedAt": "2024-03-05T18:29:50.509Z"
10 }
```



GET /users/

GET

⌵

http://localhost:8080/api/users

ParamsAuthorizationHeaders (6)BodyPre-request ScriptTestsSettings

Query Params

	Key	Value
	Key	Value

BodyCookiesHeaders (7)Test Results

PrettyRawPreviewVisualizeJSON⌵⋮

```
1  [
2    {
3      "id": 4,
4      "firstName": "Truong",
5      "lastName": "Le",
6      "email": "example@example.com",
7      "password": "password123",
8      "username": null,
9      "createdAt": "2024-03-05T18:29:50.509Z",
10     "updatedAt": "2024-03-05T18:29:50.509Z"
11   },
12   {
13     "id": 5,
14     "firstName": "Truong",
15     "lastName": "Le",
16     "email": "example@example.com",
17     "password": "password123",
18     "username": null,
19     "createdAt": "2024-03-05T18:30:23.777Z",
20     "updatedAt": "2024-03-05T18:30:23.777Z"
21   }
22 ]
```

## DELETE /users/:idOrEmail'

DELETE



http://localhost:8080/api/users/4

Params

Authorization

Headers (6)

Body

Pre-request Script

Tests

Settings

### Query Params

	Key	Value
	Key	Value

Body

Cookies

Headers (7)

Test Results

Pretty

Raw

Preview

Visualize

JSON



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

DELETE

⌵

http://localhost:8080/api/users/example@example.com

Params

Authorization

Headers (6)

Body

Pre-request Script

Tests

Settings

Query Params

	Key
	Key

Body

Cookies

Headers (7)

Test Results

Pretty

Raw

Preview

Visualize

JSON

⌵



1

2

3

```
{
  "status_code": "User deleted"
}
```

PUT /users/:idOrEmail'

PUT



http://localhost:8080/api/users/example@example.com

Params

Authorization

Headers (8)

Body ●

Pre-request Script

Tests

Settings

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary JSON ▾

```
1 {  
2   · "firstName": "Truong1",  
3   · "lastName": "Le1",  
4   · "email": "example1@example.com",  
5   · "password": "password1231",  
6   · "username": "truongle1"  
7 }  
8 |
```

Body

Cookies

Headers (7)

Test Results

Pretty

Raw

Preview

Visualize

JSON ▾



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

PUT



http://localhost:8080/api/users/6

Params Authorization Headers (8) Body ● Pre-request Script Tests Settings

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary **JSON** ▼

```
1 {
2   "firstName": "Truong11",
3   "lastName": "Le11",
4   "email": "example11@example.com",
5   "password": "password12311",
6   "username": "truongle1"
7 }
8
```

Body Cookies Headers (7) Test Results

Pretty Raw Preview Visualize JSON ▼

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

## **Вывод**

В ходе домашней работы была придумана собственная модель пользователя с помощью ORM Sequelize. Также реализованы CRUD-методы для работы с пользователем средствами Express и Sequelize. Была добавлена возможность поиска пользователя по id.