САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бэк-энд разработка

Отчет

Домашняя работа 2: Знакомство с ORM Sequelize

Выполнил:
Ле Хоанг Чыонг

Группа:
K33392

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

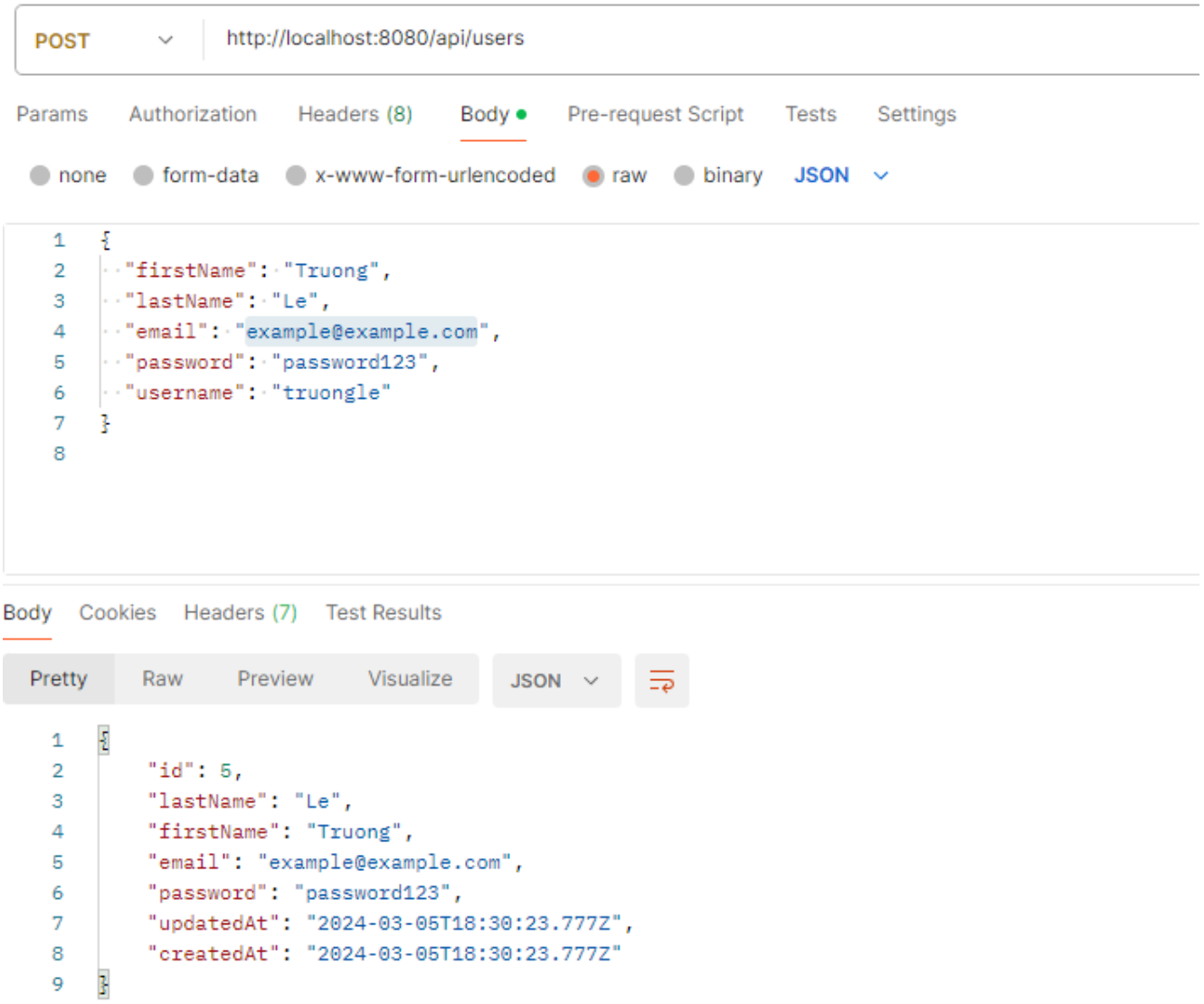
```
exports.deleteUserByIdOrEmail = async (req, res) => {
    try {
        const { idOrEmail } = req.params;
        const user = await db.User.destroy({
            where: {
                [Op.or]: [{ id: idOrEmail }, { email: idOrEmail }]
            }
        });
        if (!user) {
            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

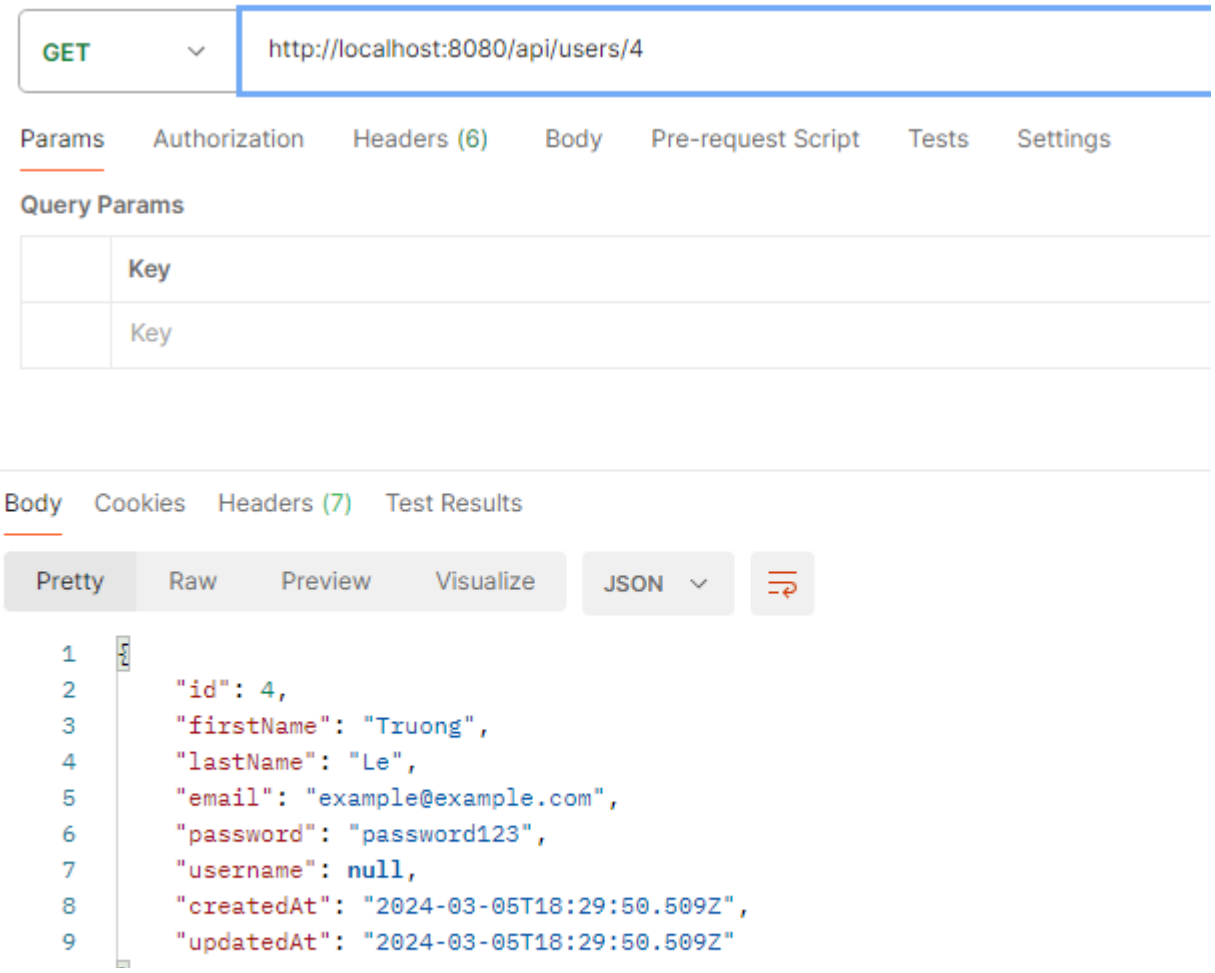
```
exports.updateUserByIdOrEmail = async (req, res) => {
    try {
        const { idOrEmail } = req.params;
        const user = await db.User.update(req.body,{
            where: {
                [Op.or]: [{ id: idOrEmail }, { email: idOrEmail }]
            }
        });
        if (!user) {
  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

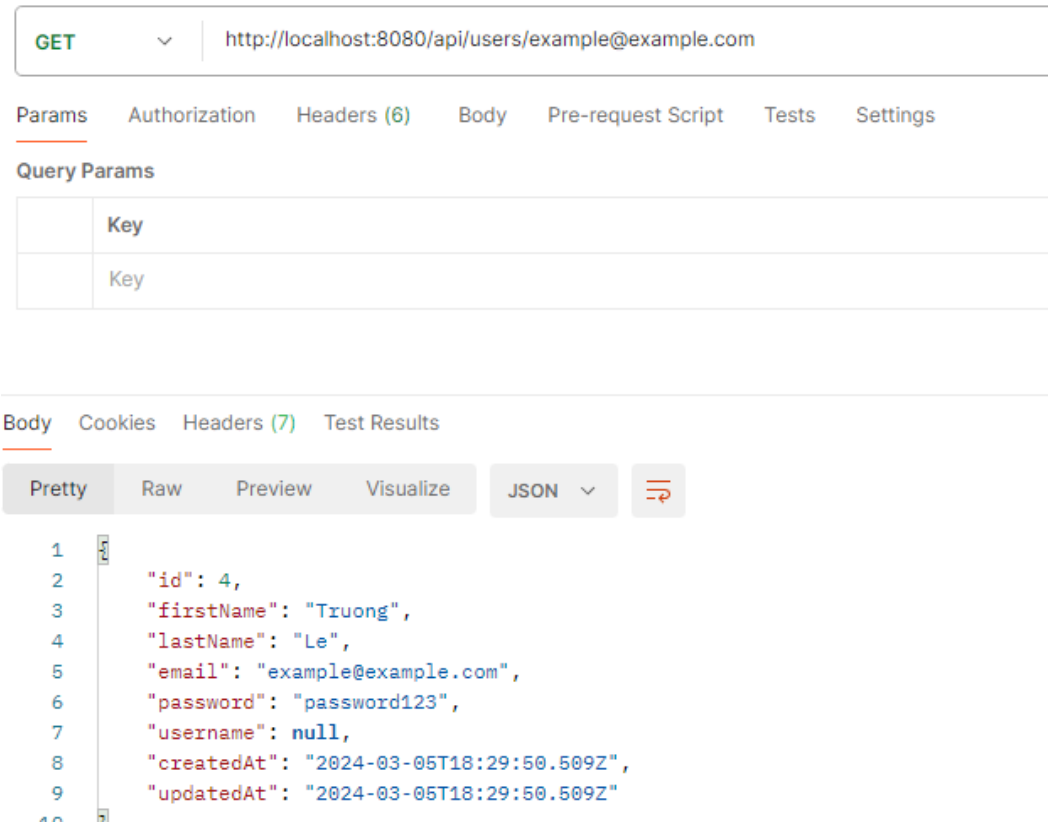
3. Проверка работы CRUD-методов с помощью Postman

`POST /users'`

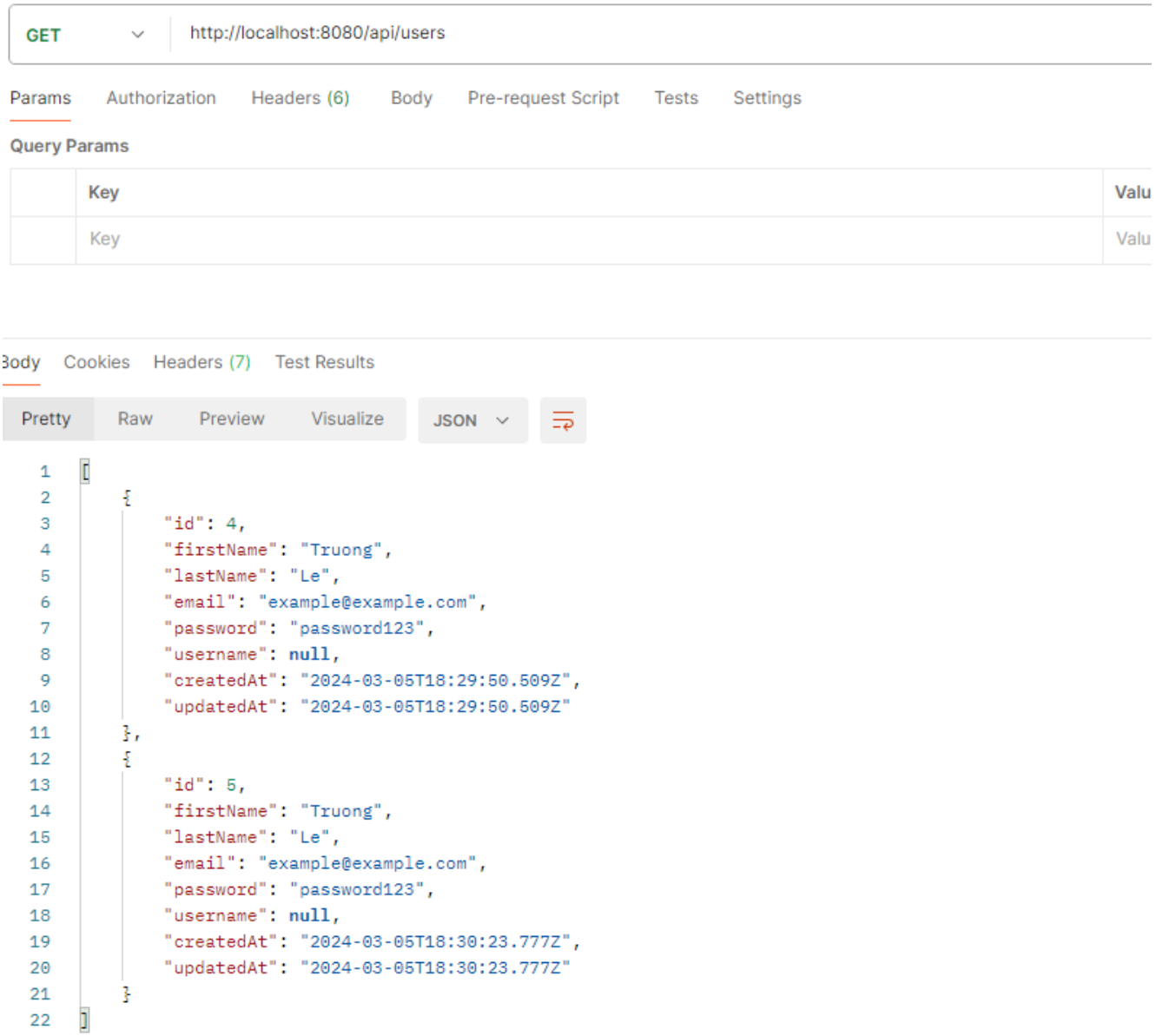


`GET /users/:idOrEmail' | Получить пользователя по ID или электронной почте`

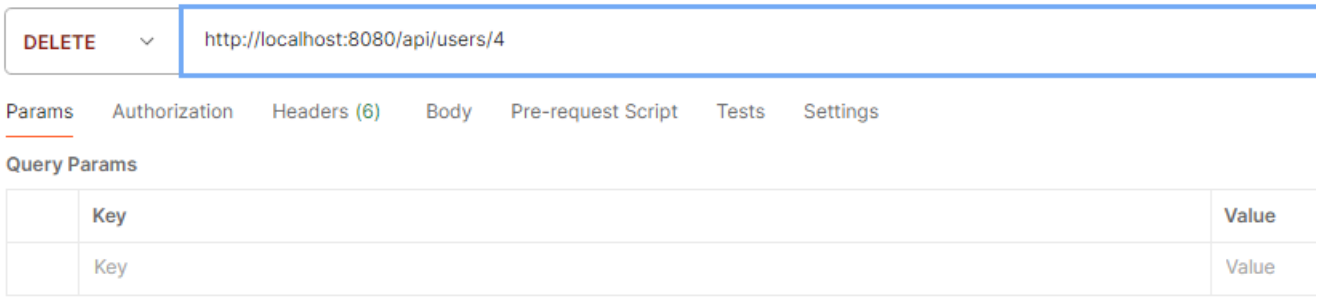


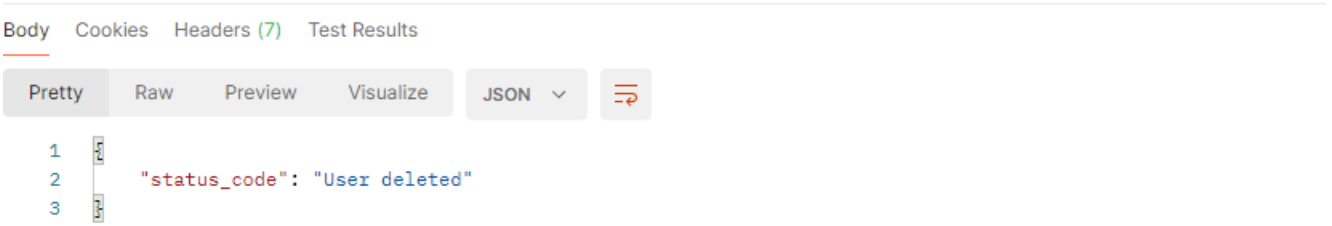

GET
http://localhost:8080/api/users/example@example.com
Params
Authorization
Headers (6)
Body
Pre-request Script
Tests
Settings
Query Params
Key
Key
Body
Cookies
Headers (7)
Test Results
Pretty
Raw
Preview
Visualize
JSON
{
"id": 4,
"firstName": "Truong",
"lastName": "Le",
"email": "example@example.com",
"password": "password123",
"username": null,
"createdAt": "2024-03-05T18:29:50.509Z",
"updatedAt": "2024-03-05T18:29:50.509Z"
}

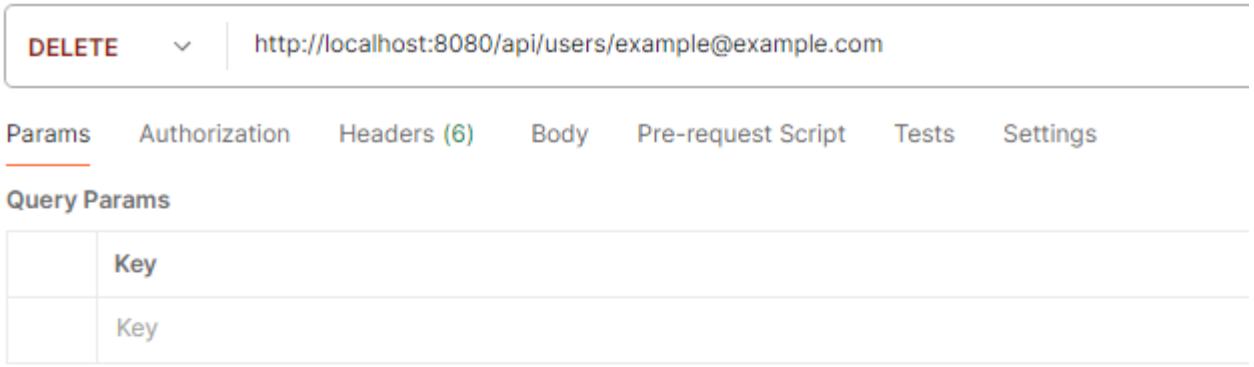
GET /users/


GET
http://localhost:8080/api/users
Params
Authorization
Headers (6)
Body
Pre-request Script
Tests
Settings
Query Params
Key
Valu
Key
Valu
Body
Cookies
Headers (7)
Test Results
Pretty
Raw
Preview
Visualize
JSON
[
{
"id": 4,
"firstName": "Truong",
"lastName": "Le",
"email": "example@example.com",
"password": "password123",
"username": null,
"createdAt": "2024-03-05T18:29:50.509Z",
"updatedAt": "2024-03-05T18:29:50.509Z"
},
{
"id": 5,
"firstName": "Truong",
"lastName": "Le",
"email": "example@example.com",
"password": "password123",
"username": null,
"createdAt": "2024-03-05T18:30:23.777Z",
"updatedAt": "2024-03-05T18:30:23.777Z"
}
]

DELETE /users/:idOrEmail'

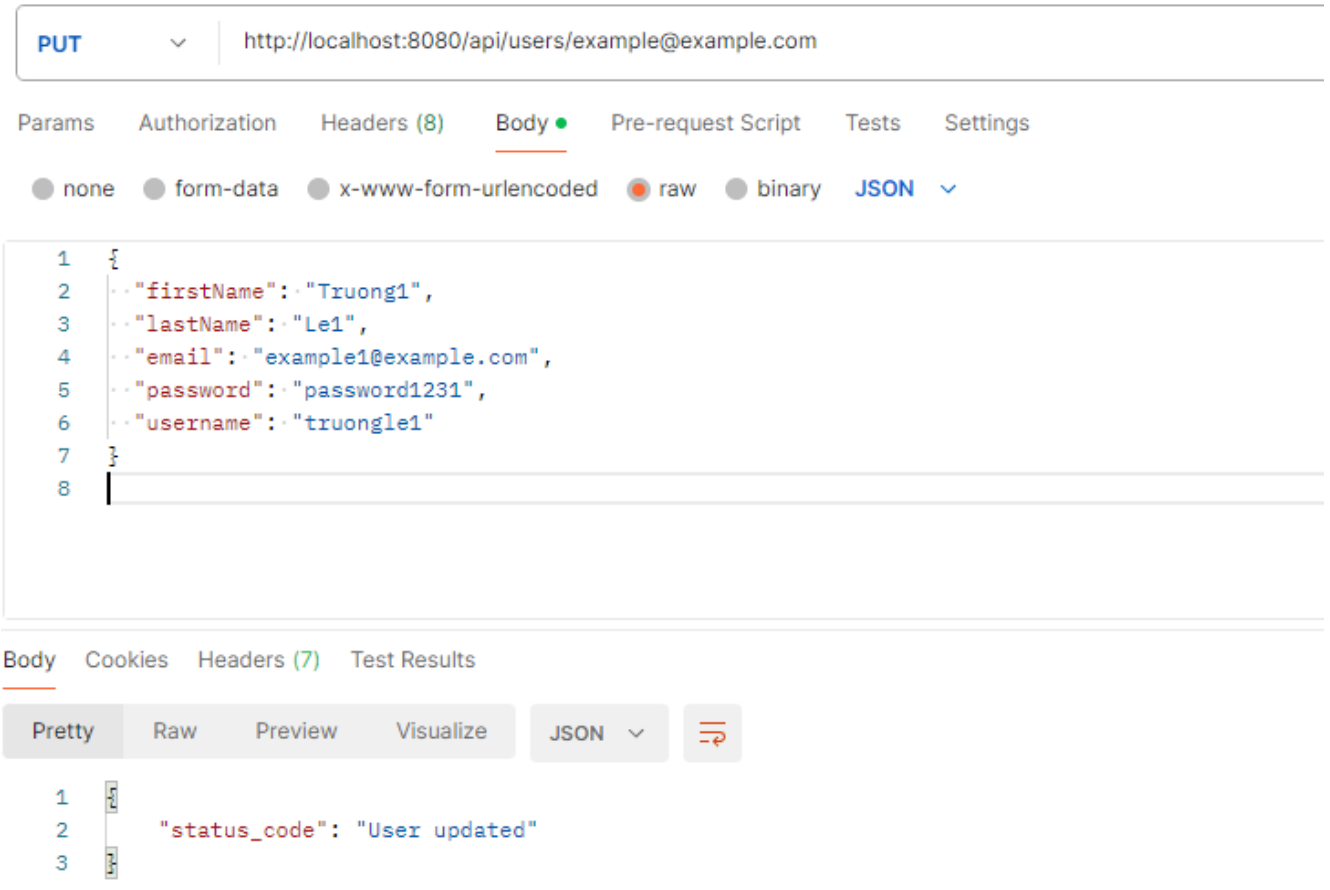


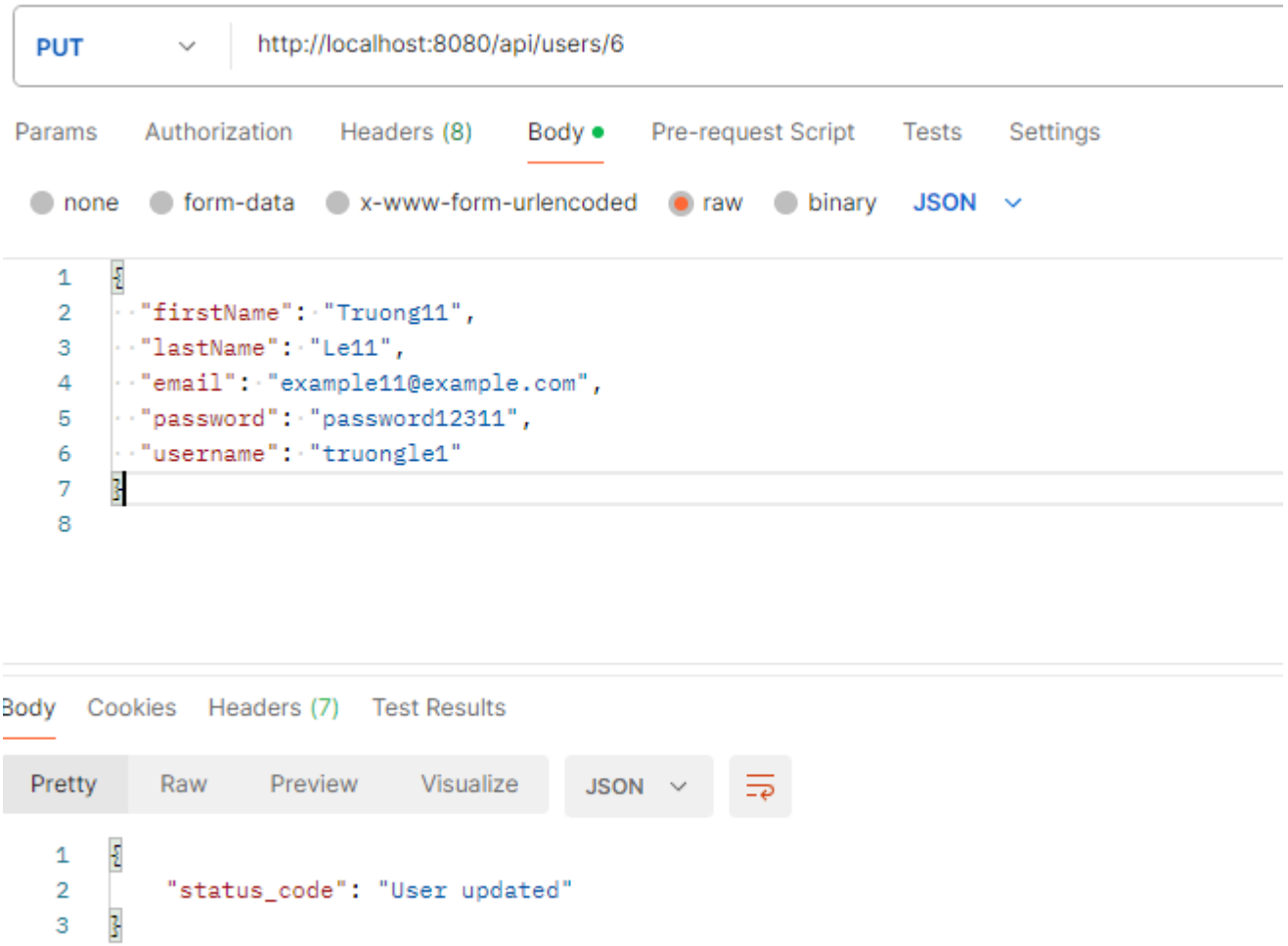



DELETE
http://localhost:8080/api/users/example@example.com
Params
Authorization
Headers (6)
Body
Pre-request Script
Tests
Settings
Query Params
Key
Key


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/6
Params
Authorization
Headers (8)
Body
Pre-request Script
Tests
Settings
none
form-data
x-www-form-urlencoded
raw
binary
JSON
{
"firstName": "Truong11",
"lastName": "Le11",
"email": "example11@example.com",
"password": "password12311",
"username": "truongle1"
}
Body
Cookies
Headers (7)
Test Results
Pretty
Raw
Preview
Visualize
JSON
{
"status_code": "User updated"
}

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.