САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бэк-энд разработка

Отчет

Практическая/Лабораторная работа

Выполнил:

Зайцев Кирилл

Группа К33402

Проверил: Добряков Д. И.

Санкт-Петербург

2024 г.

Задание:

- -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

Этап 1 – настройка среды Webstorm и установка необходимых зависимостей.

С целью сэкономить время и размер отчета ниже будет приведен код файла package.json, со всеми зависимостями:

```
{
   "name": "hw2",
   "version": "0.0.0",
   "description": "",
   "main": "index.js",
   "scripts": {
        "prestart": "echo \"App starting...\"",
        "start": "npx nodemon index.js"
},
   "author": "",
   "license": "ISC",
   "dependencies": {
        "express": "^4.17.3",
        "sequelize": "^6.17.0",
        "sqlite3": "^5.1.7"
},
   "devDependencies": {
        "nodemon": "^2.0.15",
        "sequelize-cli": "^6.4.1"
}
```

Конфигурация sequelize:

```
"development": {
    "username": null,
    "password": null,
    "database": "database_development",
    "host": null,
    "dialect": "sqlite",
    "storage": "db.sqlite"
},
    "test": {
        "username": "root",
        "password": null,
        "database": "database_test",
        "host": "127.0.0.1",
        "dialect": "mysql"
},
    "production": {
        "username": "root",
        "password": null,
        "database": "database_production",
        "host": "127.0.0.1",
        "dialect": "mysql"
}
```

По аналогии с примером, показанным на паре настраиваем загрузку моделей БД в приложение:

```
'use strict';

const fs = require('fs');
const path = require('path');
const Sequelize = require('sequelize');
const basename = path.basename(_filename);
const env = process.env.NODE_ENV || 'development';
const config = require(_dirname + '/../config/config.json')[env];
const db = {};

let sequelize;
if (config.use_env_variable) {
    sequelize = new Sequelize(process.env[config.use_env_variable],
config);
} else {
    sequelize = new Sequelize(config.database, config.username,
config.password, config);
}

fs
    .readdirSync(_dirname)
    .filter(file => {
        return (file.indexOf('.') !== 0) && (file !== basename) &&
    (file.slice(-3) === '.js');
    })
    .forEach(file => {
        const model = require(path.join(_dirname, file))(sequelize,
Sequelize.DataTypes);
        db[model.name] = model;
    });
```

```
Object.keys(db).forEach(modelName => {
    if (db[modelName].associate) {
        db[modelName].associate(db);
    }
});

db.sequelize = sequelize;
db.Sequelize = Sequelize;
module.exports = db;
```

Этап 2 – описание модели пользователя:

```
module.exports = (sequelize, DataTypes) => {
        firstName: DataTypes.STRING,
        lastName: DataTypes.STRING,
        email: DataTypes.STRING,
            type: DataTypes.STRING,
        sequelize,
Index.js:
const express = require('express');
const app = express();
app.use(express.json()); // Для разбора JSON в запросах
```

```
app.get('/users', async (req, res) => {
app.get('/users/:id', async (req, res) => {
        const user = await db.User.findByPk(req.params.id);
app.post('/users', async (req, res) => {
    } catch (error) {
app.put('/users/:id', async (req, res) => {
        await user.update(req.body);
```

```
// DELETE: Удаление пользователя по ID

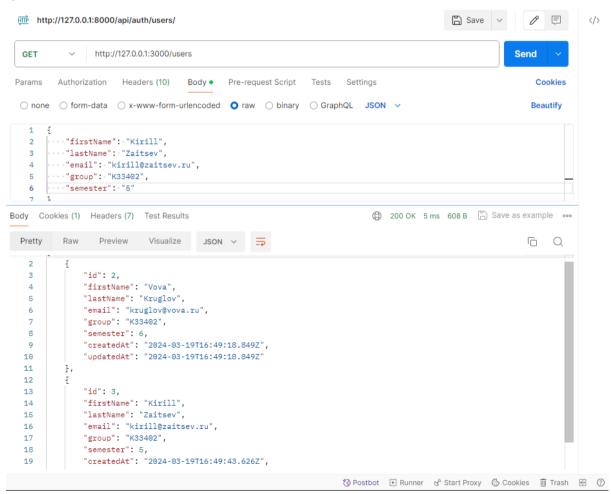
app.delete('/users/:id', async (req, res) => {
    try {
        const user = await db.User.findByPk(req.params.id);
        if (!user) {
            return res.status(404).json({ msg: 'User not found' });
        }
        await user.destroy();
        return res.json({ msg: 'User deleted successfully' });
    } catch (error) {
        return res.status(500).json({ error: error.message });
    }
});

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

Этап 3 – проверка

Скриншоты успешно выполненных запросов:

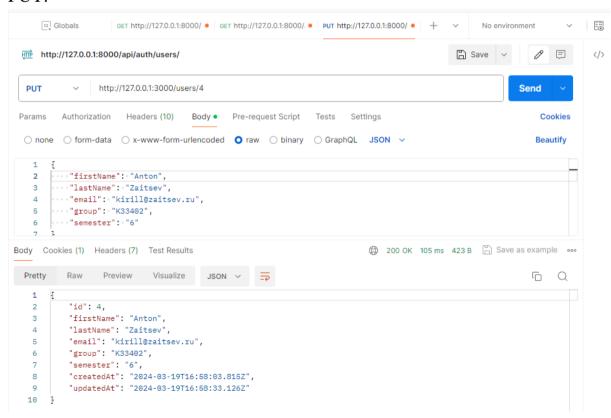
GET:



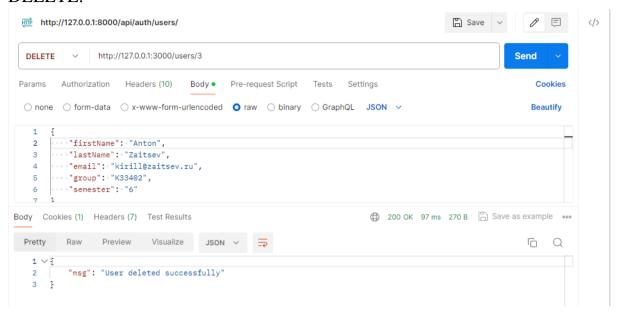
POST:

```
0 = </>
                                                                                       🖺 Save 🗸
 http://127.0.0.1:8000/api/auth/users/
        http://127.0.0.1:3000/users
                                                                                                    Send
 POST
 Params Authorization Headers (10) Body ● Pre-request Script Tests Settings
                                                                                                         Cookies
 ○ none ○ form-data ○ x-www-form-urlencoded ○ raw ○ binary ○ GraphQL JSON ∨
                                                                                                        Beautify
        ···"firstName": "Kirill",
   2
        ···"lastName": "Zaitsev",
   3
        "email": "kirill@zaitsev.ru",
   4
         "group": "K33402",
        ···"semester": "5"
Body Cookies (1) Headers (7) Test Results
                                                                    (201 Created 89 ms 429 B Save as example •••
  Pretty
       Raw Preview Visualize JSON V
                                                                                                       n Q
   1
         "id": 4,
          "firstName": "Kirill",
   3
          "lastName": "Zaitsev",
   4
          "email": "kirill@zaitsev.ru",
   5
          "group": "K33402",
   6
          "semester": "5",
          "updatedAt": "2024-03-19T16:58:03.815Z",
          "createdAt": "2024-03-19T16:58:03.815Z"
  10
```

PUT:



DELETE:



GET после удаления пользователя:

```
🖺 Save 🗸
 http://127.0.0.1:8000/api/auth/users/
                                                                                                                  0 =
                                                                                                                               </>
         v http://127.0.0.1:3000/users
  GET
 Params Authorization Headers (10) Body • Pre-request Script Tests Settings
                                                                                                                    Cookies
  ○ none ○ form-data ○ x-www-form-urlencoded ○ raw ○ binary ○ GraphQL JSON ∨
                                                                                                                   Beautify
          "firstName": "Anton",
    2
          ··"lastName": "Zaitsev",
         email": "kirill@zaitsev.ru",
         ..."group": "K33402",
         ···"semester": "6"
Body Cookies (1) Headers (7) Test Results
                                                                                ② 200 OK 7 ms 607 B Save as example ···
           Raw Preview Visualize JSON V
                                                                                                                  □ Q
               "firstName": "Vova",
    4
                "lastName": "Kruglov",
    5
               "email": "kruglov@vova.ru",
                "group": "K33402",
               "semester": 6,
               "createdAt": "2024-03-19T16:49:18.849Z",
"updatedAt": "2024-03-19T16:49:18.849Z"
  10
  11
  12
               "id": 4,
  13
               "firstName": "Anton",
  14
                "lastName": "Zaitsev",
"email": "kirill@zaitsev.ru",
  15
  16
               "group": "K33402",
  17
  18
                "semester": 6,
  19
                "createdAt": "2024-03-19T16:58:03.815Z",
                "updatedAt": "2024-03-19T16:58:33.126Z"
  20
  21
```

Вывод

В ходе выполнения задания была разработана и реализована модель пользователя, а также созданы CRUD-методы для работы с этой моделью с использованием Express и Sequelize.

На этапе настройки среды разработки и установки необходимых зависимостей был создан файл package.json с указанием всех требуемых модулей для работы с Express, Sequelize и SQLite3.

Далее была настроена конфигурация Sequelize для работы с базой данных SQLite, а также загружены модели в приложение с помощью механизма автоматической загрузки моделей из файлов в директории моделей.

Была описана модель пользователя с указанием полей: firstName, lastName, email, group и semester. Для полей были указаны соответствующие типы данных и правила валидации, такие как проверка формата группы.

Затем были реализованы CRUD-методы для работы с пользователями: создание нового пользователя, получение всех пользователей, получение пользователя по ID, получение пользователя по email, обновление пользователя по ID и удаление пользователя по ID.

Для проверки работы приложения был использован Express, который слушает порт 3000. Приложение успешно обрабатывает запросы к созданию, получению, обновлению и удалению пользователей, обеспечивая стабильную работу с базой данных.

В целом, задание выполнено успешно, и приложение готово к использованию для работы с пользователями в рамках предоставленных функций CRUD.