

**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:

```

'use strict';
const {
  Model
} = require('sequelize');
module.exports = (sequelize, DataTypes) => {
  class User extends Model {
    static associate(models) {
      // define association here
    }
  }
  User.init({
    firstName: DataTypes.STRING,
    lastName: DataTypes.STRING,
    email: DataTypes.STRING,
    group: {
      type: DataTypes.STRING,
      allowNull: false,
      validate: {
        isGroup(value) {
          // Проверка формата группы: заглавная буква, затем 5
цифр
          if (!/^[A-Z]\d{5}$/.test(value)) {
            throw new Error('Формат группы неверный. Группа
должна начинаться с заглавной буквы, за которой следуют 5 цифр.');
```

Index.js:

```

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

const app = express();
const port = 3000;

app.use(express.json()); // Для разбора JSON в запросах

```

```
// GET: Получение всех пользователей
app.get('/users', async (req, res) => {
  try {
    const users = await db.User.findAll();
    return res.json(users);
  } catch (error) {
    return res.status(500).json({ error: error.message });
  }
});

// GET: Получение пользователя по ID
app.get('/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' });
    }
    return res.json(user);
  } catch (error) {
    return res.status(500).json({ error: error.message });
  }
});

// GET: Получение пользователя по email
app.get('/users/email/:email', async (req, res) => {
  try {
    const user = await db.User.findOne({ where: { email: req.params.email } });
    if (!user) {
      return res.status(404).json({ msg: 'User not found' });
    }
    return res.json(user);
  } catch (error) {
    return res.status(500).json({ error: error.message });
  }
});

// POST: Создание нового пользователя
app.post('/users', async (req, res) => {
  try {
    const newUser = await db.User.create(req.body);
    return res.status(201).json(newUser);
  } catch (error) {
    return res.status(500).json({ error: error.message });
  }
});

// PUT: Обновление пользователя по ID
app.put('/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.update(req.body);
    return res.json(user);
  } catch (error) {
    return res.status(500).json({ error: error.message });
  }
});
```

```
// 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:

The screenshot shows a Postman interface with a GET request to `http://127.0.0.1:3000/users` successfully executed. The response status is 200 OK, with a response time of 5 ms and a body size of 608 B. The response body is displayed in JSON format, showing an array of two user objects.

```
{
  "id": 2,
  "firstName": "Vova",
  "lastName": "Kruglov",
  "email": "kruglov@vova.ru",
  "group": "K33402",
  "semester": 6,
  "createdAt": "2024-03-19T16:49:18.849Z",
  "updatedAt": "2024-03-19T16:49:18.849Z"
},
{
  "id": 3,
  "firstName": "Kirill",
  "lastName": "Zaitsev",
  "email": "kirill@zaitsev.ru",
  "group": "K33402",
  "semester": 5,
  "createdAt": "2024-03-19T16:49:43.626Z",
  "updatedAt": "2024-03-19T16:49:43.626Z"
}
```

POST:

HTTP <http://127.0.0.1:8000/api/auth/users/> Save

POST <http://127.0.0.1:3000/users> Send

Params Authorization Headers (10) **Body** Pre-request Script Tests Settings Cookies

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

```
1 {
2   "firstName": "Kirill",
3   "lastName": "Zaitsev",
4   "email": "kirill@zaitsev.ru",
5   "group": "K33402",
6   "semester": "5"
7 }
```

Body Cookies (1) Headers (7) Test Results 201 Created 89 ms 429 B Save as example

Pretty Raw Preview Visualize **JSON**

```
1 {
2   "id": 4,
3   "firstName": "Kirill",
4   "lastName": "Zaitsev",
5   "email": "kirill@zaitsev.ru",
6   "group": "K33402",
7   "semester": "5",
8   "updatedAt": "2024-03-19T16:58:03.815Z",
9   "createdAt": "2024-03-19T16:58:03.815Z"
10 }
```

PUT:

Globals GET <http://127.0.0.1:8000/> GET <http://127.0.0.1:8000/> **PUT <http://127.0.0.1:8000/>** + No environment

HTTP <http://127.0.0.1:8000/api/auth/users/> Save

PUT <http://127.0.0.1:3000/users/4> Send

Params Authorization Headers (10) **Body** Pre-request Script Tests Settings Cookies

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

```
1 {
2   "firstName": "Anton",
3   "lastName": "Zaitsev",
4   "email": "kirill@zaitsev.ru",
5   "group": "K33402",
6   "semester": "6"
7 }
```

Body Cookies (1) Headers (7) Test Results 200 OK 105 ms 423 B Save as example

Pretty Raw Preview Visualize **JSON**

```
1 {
2   "id": 4,
3   "firstName": "Anton",
4   "lastName": "Zaitsev",
5   "email": "kirill@zaitsev.ru",
6   "group": "K33402",
7   "semester": "6",
8   "createdAt": "2024-03-19T16:58:03.815Z",
9   "updatedAt": "2024-03-19T16:58:33.126Z"
10 }
```

DELETE:

The screenshot shows a REST client interface with a DELETE request to `http://127.0.0.1:3000/users/3`. The request body is a JSON object representing a user. The response is a 200 OK status with a message: "User deleted successfully".

Request:

```
DELETE http://127.0.0.1:3000/users/3
```

Body:

```
{
  "firstName": "Anton",
  "lastName": "Zaitsev",
  "email": "kirill@zaitsev.ru",
  "group": "K33402",
  "semester": "6"
}
```

Response:

```
{
  "msg": "User deleted successfully"
}
```

GET после удаления пользователя:

The screenshot shows a REST client interface with a GET request to `http://127.0.0.1:3000/users`. The response is a 200 OK status with a list of users. The user that was previously deleted is no longer present in the list.

Request:

```
GET http://127.0.0.1:3000/users
```

Response:

```
[
  {
    "firstName": "Vova",
    "lastName": "Kruglov",
    "email": "kruglov@vova.ru",
    "group": "K33402",
    "semester": 6,
    "createdAt": "2024-03-19T16:49:18.849Z",
    "updatedAt": "2024-03-19T16:49:18.849Z"
  },
  {
    "id": 4,
    "firstName": "Anton",
    "lastName": "Zaitsev",
    "email": "kirill@zaitsev.ru",
    "group": "K33402",
    "semester": 6,
    "createdAt": "2024-03-19T16:58:03.815Z",
    "updatedAt": "2024-03-19T16:58:33.126Z"
  }
]
```


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.