

САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ
ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Фронт-энд разработка

Отчет по домашней работе

Знакомство с микрофреймворком Express и ORM Sequelize

Выполнил:

Елистратов В. Д.

Группа К32392

Проверил:

Добряков Д. И.

Санкт-Петербург

2024 г.

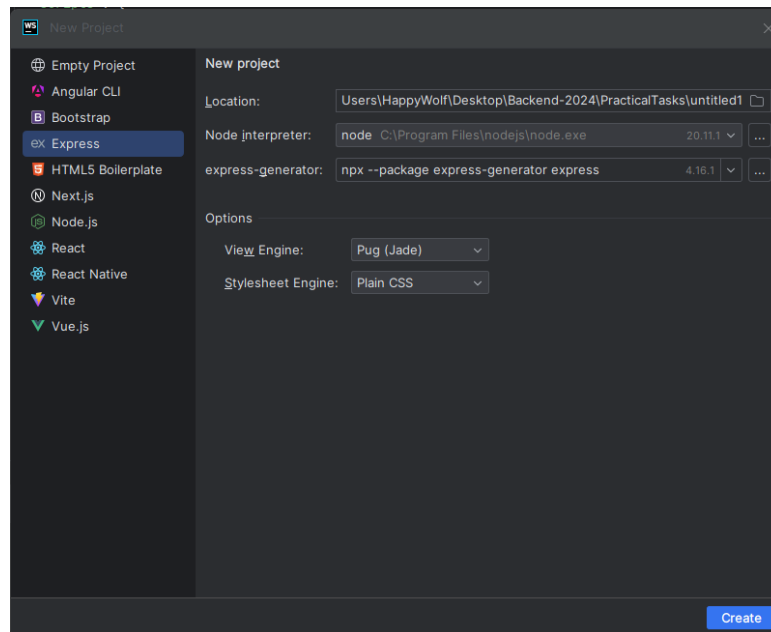
Содержание отчета

1) Постановка задачи	3
2) Создание проекта	3
3) Установка зависимостей	4
4) Инициализация sequelize	4
5) Создание моделей	5
6) Создание Endpoint-ов и примеры работы	6

1) Постановка задачи

- 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

2) Создание проекта



Express установлен по умолчанию

```
package.json x
1  {
2    "name": "untitled",
3    "version": "0.0.0",
4    "private": true,
5    "scripts": {
6      "start": "node ./bin/www"
7    },
8    "dependencies": {
9      "cookie-parser": "~1.4.4",
10     "debug": "~2.6.9",
11     "express": "~4.16.1",
12     "http-errors": "~1.6.3",
13     "morgan": "~1.9.1",
14     "pug": "2.0.0-beta11"
15   }
16 }
17
```

3) Установка зависимостей

`npm install sqlite3`

`npm install sequelize`

`npm install sequelize-cli`

`npm install mysql2`

```
package.json x
1 {
2   "name": "untitled",
3   "version": "0.0.0",
4   "private": true,
5   "scripts": {
6     "start": "node ./bin/www"
7   },
8   "dependencies": {
9     "cookie-parser": "~1.4.4",
10    "debug": "~2.6.9",
11    "express": "~4.16.1",
12    "http-errors": "~1.6.3",
13    "morgan": "~1.9.1",
14    "mysql2": "^3.9.2",
15    "pug": "2.0.0-beta11",
16    "sequelize": "^6.37.1",
17    "sequelize-cli": "^6.6.2",
18    "sqlite3": "^5.1.7"
19  }
20 }
```

4) Инициализация sequelize

`npx sequelize-cli init`

```
PS C:\Users\HappyWolf\Desktop\Backend-2024\PracticalTasks\untitled> npx sequelize-cli init

Sequelize CLI [Node: 20.11.1, CLI: 6.6.2, ORM: 6.37.1]

Created "config\config.json"
Successfully created models folder at "C:\Users\HappyWolf\Desktop\Backend-2024\PracticalTasks\untitled\models".
Successfully created migrations folder at "C:\Users\HappyWolf\Desktop\Backend-2024\PracticalTasks\untitled\migrations".
Successfully created seeders folder at "C:\Users\HappyWolf\Desktop\Backend-2024\PracticalTasks\untitled\seeders".
PS C:\Users\HappyWolf\Desktop\Backend-2024\PracticalTasks\untitled> 
```

5) Создание моделей

`npx sequelize-cli model:generate --name User --attributes firstName:string,lastName:string,email:string,age:integer`

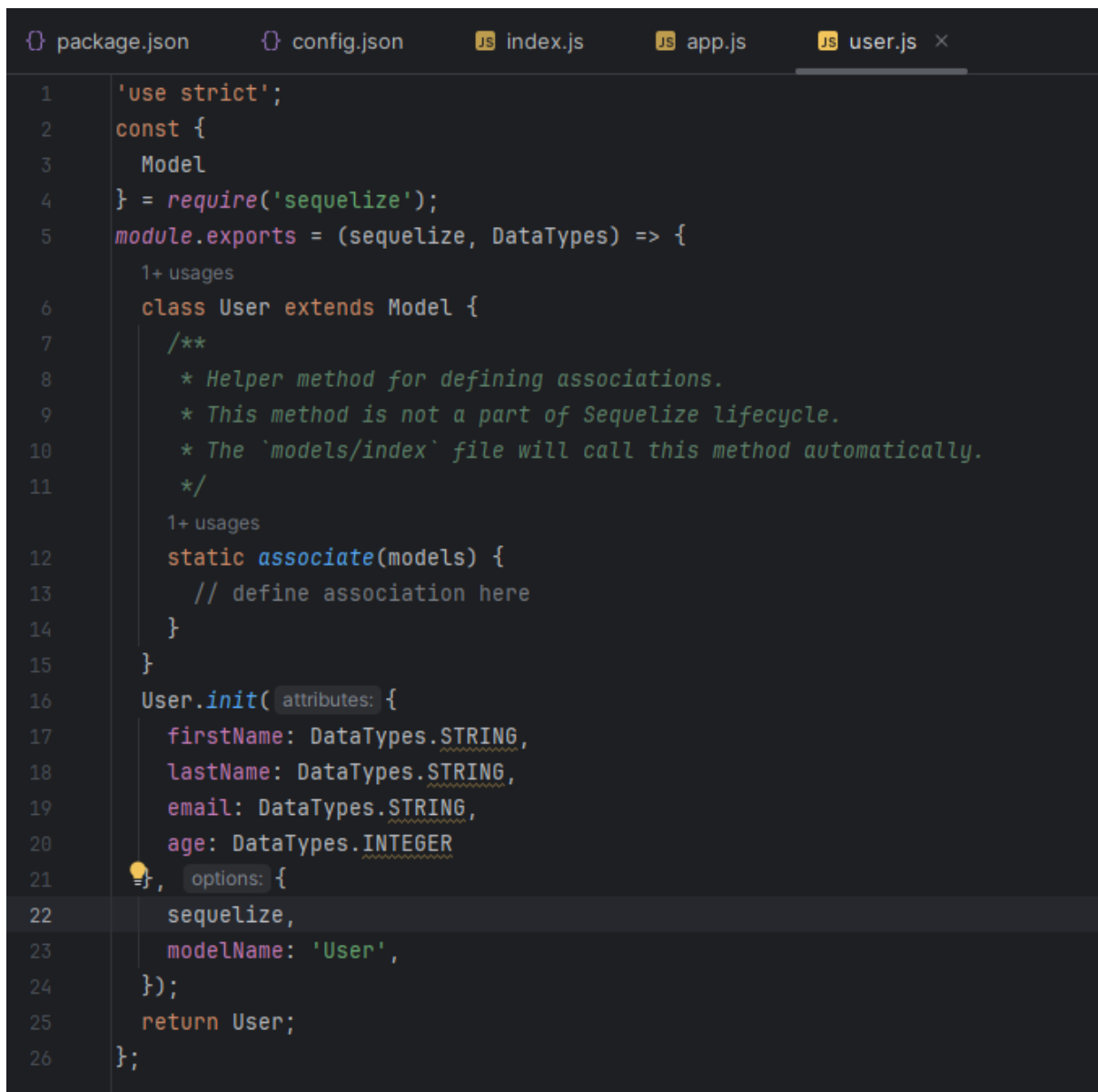
```
PS C:\Users\HappyWolf\Desktop\Backend-2024\PracticalTasks\untitled> npx sequelize-cli model:generate --name User --attributes firstName:string,lastName:string,email:string,age:integer

Sequelize CLI [Node: 20.11.1, CLI: 6.6.2, ORM: 6.37.1]

New model was created at C:\Users\HappyWolf\Desktop\Backend-2024\PracticalTasks\untitled\models\user.js .
New migration was created at C:\Users\HappyWolf\Desktop\Backend-2024\PracticalTasks\untitled\migrations\20240312145130-create-user.js .
PS C:\Users\HappyWolf\Desktop\Backend-2024\PracticalTasks\untitled> 
```

Проведем миграцию

`npx sequelize db:migrate`



```
package.json  config.json  JS index.js  JS app.js  JS user.js x

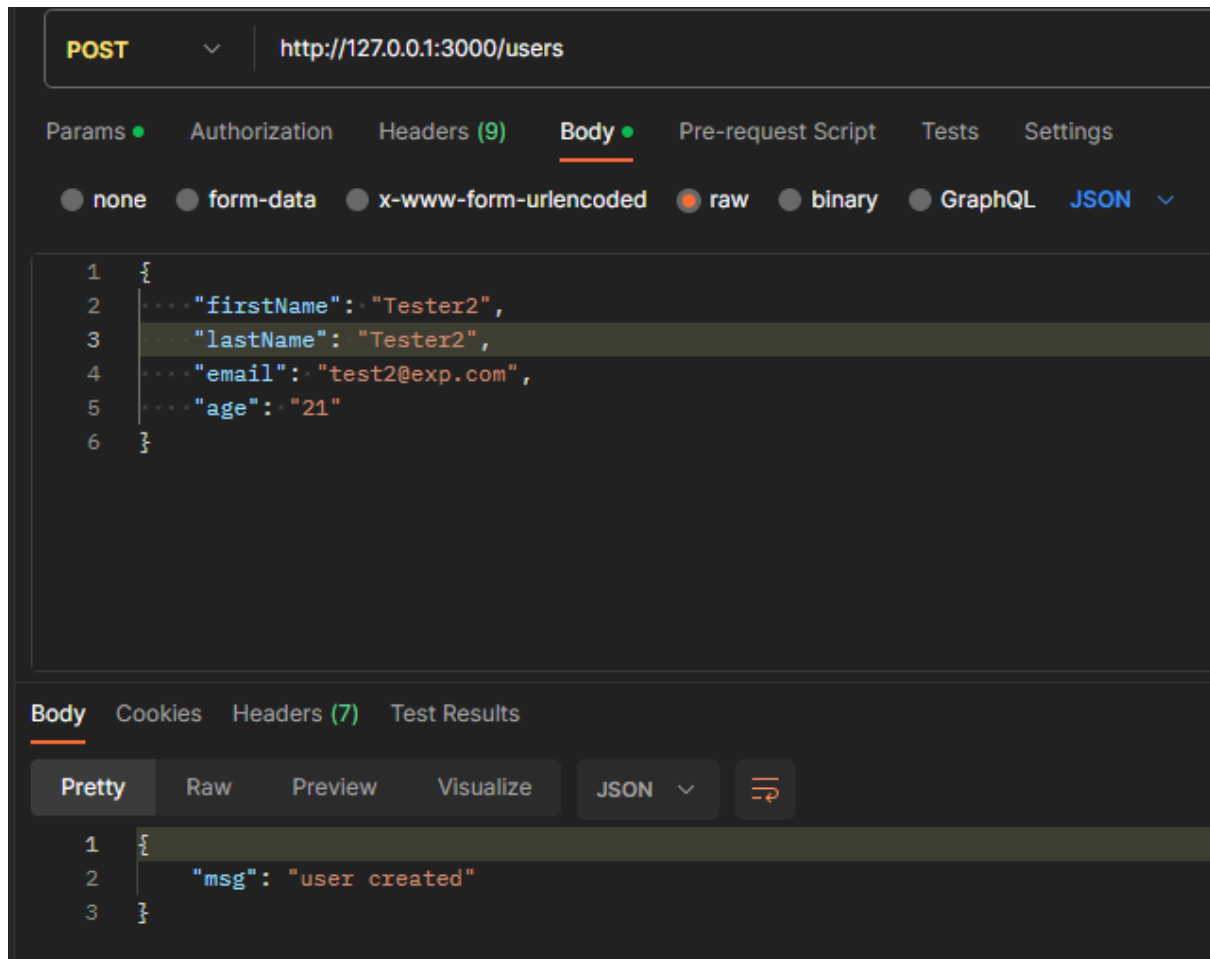
1  'use strict';
2  const {
3    Model
4  } = require('sequelize');
5  module.exports = (sequelize, DataTypes) => {
6    1+ usages
7    class User extends Model {
8      /**
9       * Helper method for defining associations.
10     * This method is not a part of Sequelize lifecycle.
11     * The `models/index` file will call this method automatically.
12     */
13     1+ usages
14     static associate(models) {
15       // define association here
16     }
17   }
18   User.init({ attributes: {
19     firstName: DataTypes.STRING,
20     lastName: DataTypes.STRING,
21     email: DataTypes.STRING,
22     age: DataTypes.INTEGER
23   }, options: {
24     sequelize,
25     modelName: 'User',
26   });
27   return User;
28 };

```

6) Создание Endpoint-ов и примеры работы

Создание пользователя

```
app.post('/users', async (req, res) => {  
  await db.User.create(req.body).then(() => {  
    res.send({"msg": 'user created'})  
  })  
})
```



Отображение всех пользователей

```
app.get('/users', async (req, res) => {
  const users = await db.User.findAll()
  if (!users) {
    return res.send({"msg": "users is not found"})
  }

  return res.send(users)
})
```

The screenshot shows a REST client interface with a GET request to `http://127.0.0.1:3000/users`. The response is displayed in the 'Body' tab, showing a JSON array of two user objects. The 'Query Params' section is empty, and the 'Headers' section shows 7 headers. The 'Body' tab is set to 'Pretty' view.

Query Params

Key	Value
Key	Value

Body

Pretty Raw Preview Visualize JSON

```
1 [
2   {
3     "id": 2,
4     "firstName": "Tester",
5     "lastName": "Tester",
6     "email": "test@exp.com",
7     "age": 20,
8     "createdAt": "2024-03-12T14:09:13.139Z",
9     "updatedAt": "2024-03-12T14:09:13.139Z"
10  },
11  {
12    "id": 3,
13    "firstName": "Tester2",
14    "lastName": "Tester2",
15    "email": "test2@exp.com",
16    "age": 21,
17    "createdAt": "2024-03-12T14:59:21.499Z",
18    "updatedAt": "2024-03-12T14:59:21.499Z"
19  }
20 ]
```

Поиск пользователя по ID

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

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

The screenshot shows a REST client interface with the following details:

- Method:** GET
- URL:** http://127.0.0.1:3000/users/getId/2
- Params:** The 'Query Params' section is empty, with placeholder text 'Key' in two rows.
- Body:** The response is displayed in the 'Body' tab, formatted as JSON. The response object contains the following fields:
 - id: 2
 - firstName: "Tester"
 - lastName: "Tester"
 - email: "test@exp.com"
 - age: 20
 - createdAt: "2024-03-12T14:09:13.139Z"
 - updatedAt: "2024-03-12T14:09:13.139Z"

Поиск пользователя по Email

```
app.get('/users/email/:email', async (req, res) => {
  const user = await db.User.findOne(
    {
      where: {
        email: req.params.email
      }
    }
  )
  if (user) {
    return res.send(user)
  }

  return res.send({"msg": req.body.email})
})
```

HTTP **http://127.0.0.1:3000/users/email/test2@exp.com**

GET **http://127.0.0.1:3000/users/email/test2@exp.com**

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

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

Body Cookies Headers (7) Test Results

Pretty Raw Preview Visualize JSON

```
1 {
2   "id": 3,
3   "firstName": "Tester2",
4   "lastName": "Tester2",
5   "email": "test2@exp.com",
6   "age": 21,
7   "createdAt": "2024-03-12T14:59:21.499Z",
8   "updatedAt": "2024-03-12T14:59:21.499Z"
9 }
```

Удаление пользователя по ID

```
app.delete('/users/:id', async (req, res) => {
  const user = await db.User.findByPk(req.params.id)
  if (user) {
    await user.destroy()
    return res.send({"msg": "user deleted"})
  }

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

