

**САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ
ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО**

Дисциплина: Backend-Разработка

Отчет

Практическая работа 2

Выполнил:
Бункута Натан

Группа:
К33412

Проверил:
Добрияков Д.И

Санкт-Петербург

2023 г.

Задание:

- Придумать свою собственную модель пользователя
- Реализовать набор из CRUD-методов для работы с пользователями средствами Express + Sequelize
- Написать запрос для получения пользователя по id/email

Выполнение работы

Models

```
models > JS index.js > ...
1  'use strict';
2
3  const fs = require('fs');
4  const path = require('path');
5  const Sequelize = require('sequelize');
6  const basename = path.basename(__filename);
7  const env = process.env.NODE_ENV || 'development';
8  const config = require(__dirname + '/../config/config.json')[env];
9  const db = {};
10
11  let sequelize;
12  if (config.use_env_variable) {
13    | sequelize = new Sequelize(process.env[config.use_env_variable], config);
14  } else {
15    | sequelize = new Sequelize(config.database, config.username, config.password, config);
16  }
17
18  fs
19    .readdirSync(__dirname)
20    .filter(file => {
21      | return (file.indexOf('.') !== 0) && (file !== basename) && (file.slice(-3) === '.js');
22    })
23    .forEach(file => {
24      | const model = require(path.join(__dirname, file))(sequelize, Sequelize.DataTypes);
25      | db[model.name] = model;
26    });
27
28  Object.keys(db).forEach(modelName => {
29    | if (db[modelName].associate) {
30      | db[modelName].associate(db);
31    }
32  });
```

```

models > JS userjs > ...
1  'use strict';
2  const {
3    | Model
4  } = require('sequelize');
5  module.exports = (sequelize, DataTypes) => {
6    class User extends Model {
7      |
8      static associate(models) {
9      |
10     }
11   }
12   User.init({
13     firstName: DataTypes.STRING,
14     lastName: DataTypes.STRING,
15     username: DataTypes.STRING,
16     password: DataTypes.STRING
17   }, {
18     sequelize,
19     modelName: 'User',
20   });
21   return User;
22 };

```

routes

```

1  const express = require('express')
2  const db = require('./models')
3  const bodyParser = require("body-parser");
4
5  const app = express()
6  const port = 5000
7
8  app.use(bodyParser.json())
9
10 app.listen(port, () => {
11   | console.log(`Server listening on port ${port}`)
12 })
13
14 app.get('/', (req, res) => {
15   | res.send('Hello here!')
16 })
17
18 app.post('/add', async (req, res) => {
19   | const user = await db.User.create(req.body)
20   | res.send(user.toJSON())
21 })
22
23 app.get('/get', async (req, res) => {
24   | const users = await db.User.findAll()
25   | res.send(users)
26 })
27

```

```

28 app.get('/users/id/:id', async (req, res) => {
29   const user = await db.User.findByPk(req.params.id)
30   if (user) {
31     res.send(user.toJSON())
32   } else {
33     res.status(404).send({'msg': 'user not found'})
34   }
35 }
36 })
37
38 app.get('/users/username/:username', async (req, res) => {
39   const user = await db.User.findOne({
40     where: {
41       username: req.params.username
42     }
43   })
44   if (user) {
45     res.send(user.toJSON())
46   } else {
47     res.status(404).send({'msg': 'user not found'})
48   }
49 }
50 })
51
52 app.put('/update/:id', async (req, res) => {
53   const user = await db.User.findByPk(req.params.id)
54   if (user) {
55     await db.User.update(req.body, {
56       where: {
57         id: req.params.id
58       }
59     })
60     res.send(user.toJSON())

```

```

61   } else {
62     res.status(404).send({'msg': 'user not found'})
63   }
64 })
65
66 app.delete('/delete/:id', async (req, res) => {
67   const user = await db.User.findByPk(req.params.id)
68   if (user) {
69     user.destroy();
70     res.sendStatus(200).send({'msg': 'user deleted'})
71   }
72   res.status(404).send({'msg': 'user not found'})
73 })
74

```

Add a new user

The screenshot shows a REST client interface with a POST request to `http://localhost:5000/add`. The 'Body' tab is selected, and the request body is a JSON object with the following fields: `firstName`, `lastName`, `username`, and `password`. The response status is `200 OK` with a response time of `2.42 s` and a body size of `406 B`. The response body is displayed in the 'Body' tab, showing the same fields plus `id`, `updatedAt`, and `createdAt`.

```
POST http://localhost:5000/add

{
  "firstName": "Egor",
  "lastName": "Egorevich",
  "username": "Gricha",
  "password": "egor1234"
}
```

Body: `200 OK 2.42 s 406 B`

```
{
  "id": 8,
  "firstName": "Egor",
  "lastName": "Egorevich",
  "username": "Gricha",
  "password": "egor1234",
  "updatedAt": "2023-06-13T11:01:23.870Z",
  "createdAt": "2023-06-13T11:01:23.870Z"
}
```

View all users

The screenshot shows a REST client interface with a GET request to `http://localhost:5000/get`. The 'Body' tab is selected, and the response body is a JSON array containing two user objects. The response status is `200 OK` with a response time of `1303 ms` and a body size of `1.03 KB`. The response body is displayed in the 'Body' tab, showing the same fields as the first screenshot, plus `id`, `updatedAt`, and `createdAt`.

```
GET http://localhost:5000/get

[
  {
    "id": 8,
    "firstName": "Egor",
    "lastName": "Egorevich",
    "username": "Gricha",
    "password": "egor1234",
    "updatedAt": "2023-06-13T11:01:23.870Z",
    "createdAt": "2023-06-13T11:01:23.870Z"
  },
  {
    "id": 9,
    "firstName": "Vich",
    "lastName": "vich",
    "username": "vicha",
    "password": "vi1234",
    "updatedAt": "2023-06-13T11:07:02.944Z",
    "createdAt": "2023-06-13T11:07:02.944Z"
  }
]
```

Body: `200 OK 1303 ms 1.03 KB`

Get a user by id

GET <http://localhost:5000/users/id/9>

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

none form-data x-www-form-urlencoded raw binary GraphQL JSON

Body Cookies Headers (7) Test Results 200 OK 1302 ms

Pretty Raw Preview Visualize JSON

```
1 {
2   "id": 9,
3   "firstName": "Vich",
4   "lastName": "vich",
5   "username": "vicha",
6   "password": "vi1234",
7   "createdAt": "2023-06-13T11:07:02.944Z",
8   "updatedAt": "2023-06-13T11:07:02.944Z"
9 }
```

Get a user by username

GET <http://localhost:5000/users/username/Gricha>

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

none form-data x-www-form-urlencoded raw binary GraphQL JSON

Body Cookies Headers (7) Test Results 200 OK 426 ms 406 B

Pretty Raw Preview Visualize JSON

```
1 {
2   "id": 8,
3   "firstName": "Egor",
4   "lastName": "Egorevich",
5   "username": "Gricha",
6   "password": "egor1234",
7   "createdAt": "2023-06-13T11:01:23.870Z",
8   "updatedAt": "2023-06-13T11:01:23.870Z"
9 }
```

Update user's data

The screenshot shows a REST client interface with a PUT request to `http://localhost:5000/update/9`. The 'Body' tab is selected, and the request body is a JSON object: `{ "userName": "Richman", "password": "virich00" }`. The response status is `200 OK`. The 'Body' tab is also selected in the response section, showing a JSON object: `{ "id": 9, "firstName": "Vivien", "lastName": "Richman", "username": "vicha", "password": "virich00", "createdAt": "2023-06-13T11:07:02.944Z", "updatedAt": "2023-06-13T11:16:42.995Z" }`.

```
PUT http://localhost:5000/update/9

{
  "userName": "Richman",
  "password": "virich00"
}
```

Body Cookies Headers (7) Test Results 200 OK 74

Pretty Raw Preview Visualize JSON

```
{
  "id": 9,
  "firstName": "Vivien",
  "lastName": "Richman",
  "username": "vicha",
  "password": "virich00",
  "createdAt": "2023-06-13T11:07:02.944Z",
  "updatedAt": "2023-06-13T11:16:42.995Z"
}
```

Delete a user

The screenshot shows a REST client interface with a DELETE request to `http://localhost:5000/delete/8`. The 'Body' tab is selected, and the response status is `200 OK`. The 'Body' tab is also selected in the response section, showing the text `OK`.

```
DELETE http://localhost:5000/delete/8
```

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

none form-data x-www-form-urlencoded raw binary GraphQL JSON

Body Cookies Headers (7) Test Results 200 OK

Pretty Raw Preview Visualize Text

```
1 OK
```

Вывод:

При выполнении данной домашней работы, я научил работать с основами express и sequelize. Также, сумел создать скрипты и написал запросы для управления базой данных методом CRUD (create, read, update, delete).