

**САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ
ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО**

Дисциплина: Бэк-энд разработка

Отчет

Практическая/Лабораторная работа

Выполнил:
Кривцов Павел
Группа К33402

Проверил:
Добряков Д. И.

Санкт-Петербург

2023 г.

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.

Ход работы

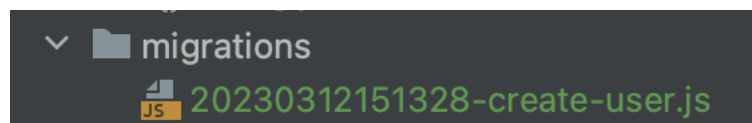
1. Создадим собственную модель пользователя. Она будет содержать в себе следующие поля: username, email, password. Для этого воспользуемся *sequelize-cli* командой:

```
npx sequelize-cli model:generate --name User --attributes username:string,email:string,password:string
```

```
pavel HW2 (hw-2) >> npx sequelize-cli model:generate --name User --attributes username:string,email:string,password:string
Sequelize CLI [Node: 18.13.0, CLI: 6.6.0, ORM: 6.29.3]

New model was created at /Users/pavel/study/ITMO-ICT-Backend-2023/homeworks/K33402/Krivtsov_Pavel/HW2/models/user.js .
New migration was created at /Users/pavel/study/ITMO-ICT-Backend-2023/homeworks/K33402/Krivtsov_Pavel/HW2/migrations/20230312151328-create-user.js .
pavel HW2 (hw-2) >> _
```

После чего в файловой системе проекта появятся новые миграция и модель:



```

'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({
    attributes: {
      username: DataTypes.STRING,
      email: DataTypes.STRING,
      password: DataTypes.STRING
    },
    options: {
      defaultScope: {
        attributes: {exclude: ['password']},
      },
    },
    sequelize,
    modelName: 'User',
  });
  return User;
};

```

Применим появившуюся миграцию с помощью команды `prx sequelize-cli db:migrate`, после чего в файловой системе появится файл базы данных.

2. Реализуем CRUD методы для работы с пользователем в роутере `users.js`, который будем подключать к приложению по пути `“/users”` в запросе:

```

const usersRouter = require('./routes/users');
app.use('/users', usersRouter);

```

- Create

```
// CREATE user
router.post( path: '/', handlers: async (req, res) => {
  console.log(req.body)
  try {
    const user = await db.User.create(req.body);
    res.status( code: 200 ).send(user.toJSON());
  } catch (err) {
    res.status( code: 400 ).send( body: {"msg": err});
  }
});
```

The screenshot shows a REST client interface with a POST request to `http://127.0.0.1:3000/users/`. The request body is a JSON object: `{ "username": "User", "email": "user@gmail.com", "password": "qwerty123" }`. The response status is `200 OK` with a response time of `11 ms` and a size of `389 B`. The response body is a JSON object: `{ "id": 19, "username": "User", "email": "user@gmail.com", "password": "qwerty123", "updatedAt": "2023-03-13T11:23:39.495Z", "createdAt": "2023-03-13T11:23:39.495Z" }`.

POST `http://127.0.0.1:3000/users/` [Send](#)

Params Authorization Headers (8) **Body** Pre-request Script Tests Settings [Cookies](#)

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL [JSON](#) [Beautify](#)

```
1 {
2   "username": "User",
3   "email": "user@gmail.com",
4   "password": "qwerty123"
5 }
```

Body Cookies Headers (7) Test Results [Save Response](#) `Status: 200 OK Time: 11 ms Size: 389 B`

Pretty Raw Preview Visualize [JSON](#)

```
1 {
2   "id": 19,
3   "username": "User",
4   "email": "user@gmail.com",
5   "password": "qwerty123",
6   "updatedAt": "2023-03-13T11:23:39.495Z",
7   "createdAt": "2023-03-13T11:23:39.495Z"
8 }
```

- Read

```
// GET users listing
router.get( path: '/', handlers: async (req, res) => {
  const users = await db.User.findAll()
  res.status( code: 200 ).send(users)
});
```

GET ▼ http://127.0.0.1:3000/users/ Send ▼

Params Authorization Headers (6) Body Pre-request Script Tests Settings Cookies

Query Params

KEY	VALUE	DESCRIPTION	...	Bulk Edit
Key	Value	Description		

Body Cookies Headers (7) Test Results ⊕ Status: 200 OK Time: 5 ms Size: 635 B Save Response ▼

Pretty Raw Preview Visualize JSON ▼ 🔍

```

1  {
2    {
3      "id": 18,
4      "username": "User4",
5      "email": "user4@gmail.com",
6      "createdAt": "2023-03-13T11:23:10.871Z",
7      "updatedAt": "2023-03-13T11:23:10.871Z"
8    },
9    {
10     "id": 19,
11     "username": "User",
12     "email": "user@gmail.com",
13     "createdAt": "2023-03-13T11:23:39.495Z",
14     "updatedAt": "2023-03-13T11:23:39.495Z"
15   },
16   {
17     "id": 20,
18     "username": "User1",
19     "email": "user1@gmail.com",
20     "createdAt": "2023-03-13T13:55:17.142Z",
21     "updatedAt": "2023-03-13T13:55:17.142Z"
22   }
23 }

```

- Update

```

// UPDATE user
router.put( path: '/:id', handlers: async (req, res) => {
  try {
    const user = await db.User.update(req.body, { where: { id: req.params.id } });
    res.status( code: 200 ).send( body: { "msg": "User has been successfully updated" });
  } catch (err) {
    res.status( code: 400 ).send( body: { "msg": err });
  }
})

```

PUT http://127.0.0.1:3000/users/18 Send

Params Authorization Headers (8) Body Pre-request Script Tests Settings Cookies

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

```

1  {
2    "username": "User2",
3    "email": "user2@gmail.com",
4    "password": "qwerty123"
5  }

```

Body Cookies Headers (7) Test Results Status: 200 OK Time: 49 ms Size: 279 B Save Response

Pretty Raw Preview Visualize JSON

```

1  {
2    "msg": "User has been successfully updated"
3  }

```

- Delete

```

// DELETE user
router.delete( path:('/:id'), handlers: async (req, res) => {
  const user = await db.User.destroy({ where: { id: req.params.id } })
  if (user) {
    res.status( code: 204 ).send()
  } else {
    res.status( code: 404 ).send( body: { 'msg': 'User not found' })
  }
})

```

DELETE http://127.0.0.1:3000/users/17 Send

Params Authorization Headers (8) Body Pre-request Script Tests Settings Cookies

Query Params

KEY	VALUE	DESCRIPTION	...	Bulk Edit
Key	Value	Description		

Body Cookies Headers (4) Test Results Status: 204 No Content Time: 6 ms Size: 134 B Save Response

Pretty Raw Preview Visualize Text

```

1

```

3. Напишем запрос для получения пользователя по id

```
// GET user by id
router.get( path:('/:id'), handlers: async (req, res) => {
  const user = await db.User.findByPk(req.params.id)
  if (user) {
    return res.status( code: 200 ).send(user.toJSON())
  }

  return res.status( code: 404 ).send( body: {"msg": "user not found"})
})
```

The screenshot shows a REST client interface with a GET request to `http://127.0.0.1:3000/users/20`. The response is a JSON object with the following fields:

```
{
  "id": 20,
  "username": "User1",
  "email": "user1@gmail.com",
  "createdAt": "2023-03-13T13:55:17.142Z",
  "updatedAt": "2023-03-13T13:55:17.142Z"
}
```

The status is 200 OK, Time: 20 ms, Size: 368 B.

Выводы

В ходе выполнения работы была создана модель пользователя и реализованы CRUD-методы для работы с ней с использованием ORM Sequelize и Express.