

**САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ
ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО**

Дисциплина: Бэк-энд разработка

Отчет

Домашняя работа 2: Знакомство с ORM Sequelize

Выполнила:

Никифорова Кюннэй

Группа К33402

Проверил:

Добряков Д. И.

Санкт-Петербург

2023 г.

Задача

- 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) Продумать свою собственную модель пользователя

Модель пользователя состоит из следующих полей:

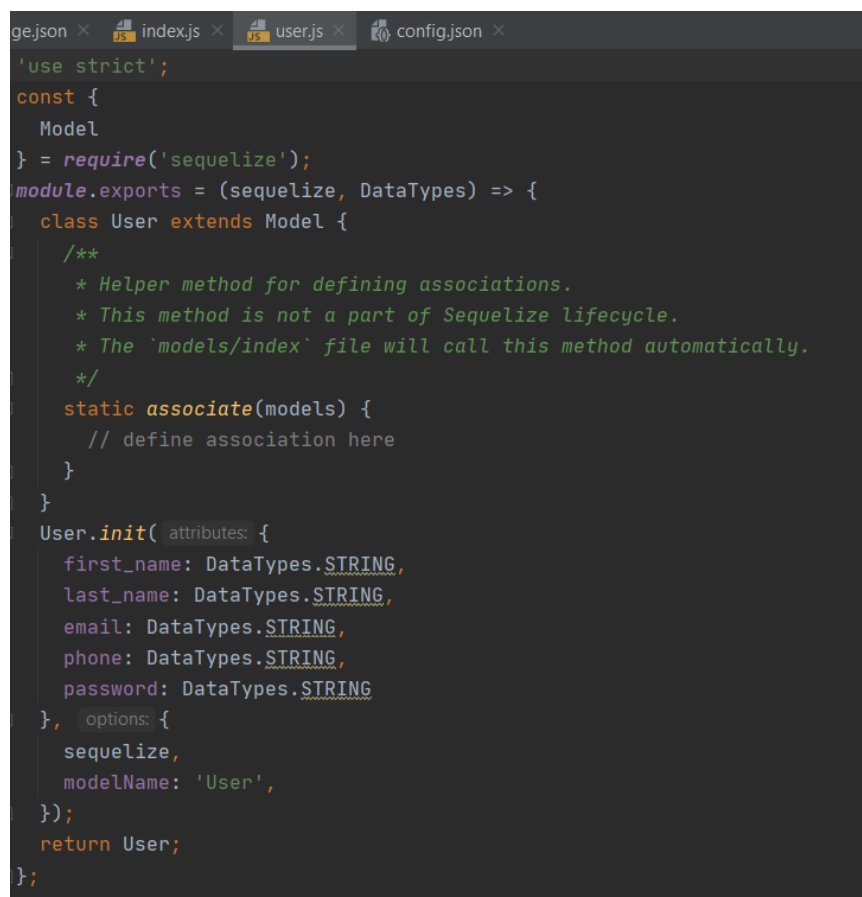
first_name:string, last_name:string, email:string, phone:string, password:string

С помощью *sequelize-cli* создаем модель пользователя и миграции:

```
npx sequelize-cli model:generate --name User --attributes
```

```
first_name:string,last_name:string,email:string,phone:string,password:string
```

```
npx sequelize-cli db:migrate
```

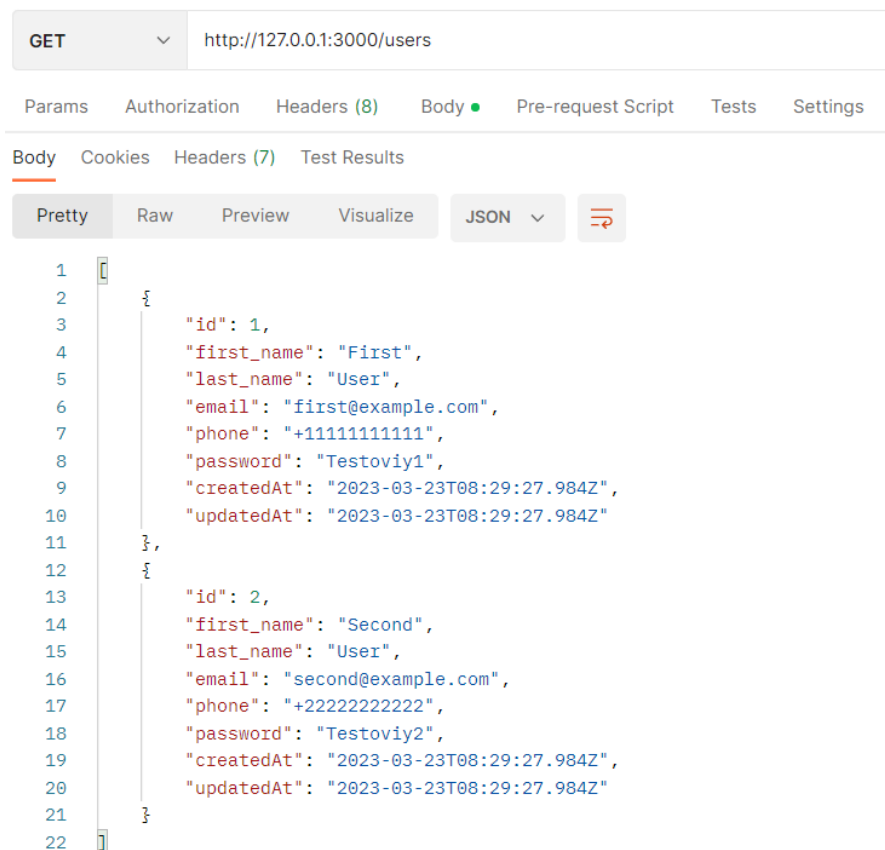


```
ge.json x index.js x user.js x config.json x
'use strict';
const {
  Model
} = require('sequelize');
module.exports = (sequelize, DataTypes) => {
  class User extends Model {
    /**
     * Helper method for defining associations.
     * This method is not a part of Sequelize lifecycle.
     * The `models/index` file will call this method automatically.
     */
    static associate(models) {
      // define association here
    }
  }
  User.init({
    attributes: {
      first_name: DataTypes.STRING,
      last_name: DataTypes.STRING,
      email: DataTypes.STRING,
      phone: DataTypes.STRING,
      password: DataTypes.STRING
    },
    options: {
      sequelize,
      modelName: 'User',
    }
  });
  return User;
};
```

2) Реализовать набор из CRUD-методов для работы с пользователями средствами Express + Sequelize

- GET запрос на получение всех пользователей

```
app.get('/users', async (req : Request<P, ResBody, ReqBody, ReqQuery, LocalsObj> , res : Response<ResBody, LocalsObj> ) => {  
  const users = await db.User.findAll()  
  res.send(users)  
})
```



- POST запрос на добавление нового пользователя

```
app.post( path: '/users', handlers: async (req : Request<P, ResBody, ReqBody, ReqQuery, LocalsObj> , res : Response<ResBody, LocalsObj> ) => {  
  try {  
    const user = await db.User.create(req.body);  
    res.send(user.toJSON());  
  } catch (error) {  
    res.status( code: 400).send( body: {"msg": "Failed to create user"});  
  }  
})
```

POST http://127.0.0.1:3000/users/

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

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

```
1 {
2   "first_name": "Third",
3   "last_name": "User",
4   "email": "third@example.com",
5   "phone": "+3333333333",
6   "password": "Testoviy3"
7 }
```

Body Cookies Headers (7) Test Results

Pretty Raw Preview Visualize JSON

```
1 {
2   "id": 4,
3   "first_name": "Third",
4   "last_name": "User",
5   "email": "third@example.com",
6   "phone": "+3333333333",
7   "password": "Testoviy3",
8   "createdAt": "2023-03-23T09:12:35.522Z",
9   "updatedAt": "2023-03-23T09:14:05.553Z"
10 }
```

- PUT запрос на изменение информации о конкретном пользователе

```
app.put( path: '/users/:id', handlers: async (req : Request<P, ResBody, ReqBody, ReqQuery, LocalsObj> , res : Response<ResBody, LocalsObj> ) => {
  try {
    const user = await db.User.update(req.body, {where: {id: req.params.id}});
    res.send( body: { "msg": "Successfully updated!" });
  } catch (error) {
    res.send(error);
  }
})
```

PUT http://127.0.0.1:3000/users/4

Params Authorization Headers (8) **Body** Pre-request Script

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

```
1 {
2   "first_name": "Nikiforova",
3   "last_name": "Kyunney",
4   "email": "third@example.com",
5   "phone": "+3333333333",
6   "password": "Testoviy3"
7 }
```

Body Cookies Headers (7) Test Results

Pretty Raw Preview Visualize JSON

```
1 {
2   "msg": "Successfully updated!"
3 }
```

- DELETE запрос на удаление конкретного пользователя

```
app.delete( path: '/users/:id', handlers: async(req : Request<P, ResBody, ReqBody, ReqQuery, LocalsObj> , res : Response<ResBody, LocalsObj> ) => {  
  const user = await db.User.destroy({ where: { id: req.params.id } })  
  if (user) {  
    res.send( body: { "msg": "User deleted" })  
  } else {  
    res.status( code: 404 ).send( body: { "msg": "user not found" })  
  }  
})
```

DELETE ▼ http://127.0.0.1:3000/users/4

Params Authorization Headers (8) Body ● Pre-request Script

Body Cookies Headers (7) Test Results

Pretty Raw Preview Visualize JSON ▼ ≡

```
1 {  
2   "msg": "User deleted"  
3 }
```

3) Написать запрос для получения пользователя по id/email

- GET запрос на получение пользователя по id

```
app.get('/users/:id', async (req : Request<P, ResBody, ReqBody, ReqQuery, LocalsObj> , res : Response<ResBody, LocalsObj> ) => {  
  const user = await db.User.findById(req.params.id)  
  if (user) {  
    return res.send(user.toJSON())  
  }  
  
  return res.send( body: { "msg": "User is not found" })  
})
```

GET ▼ http://127.0.0.1:3000/users/2

Params Authorization Headers (8) Body ● Pre-request Script Tests Set

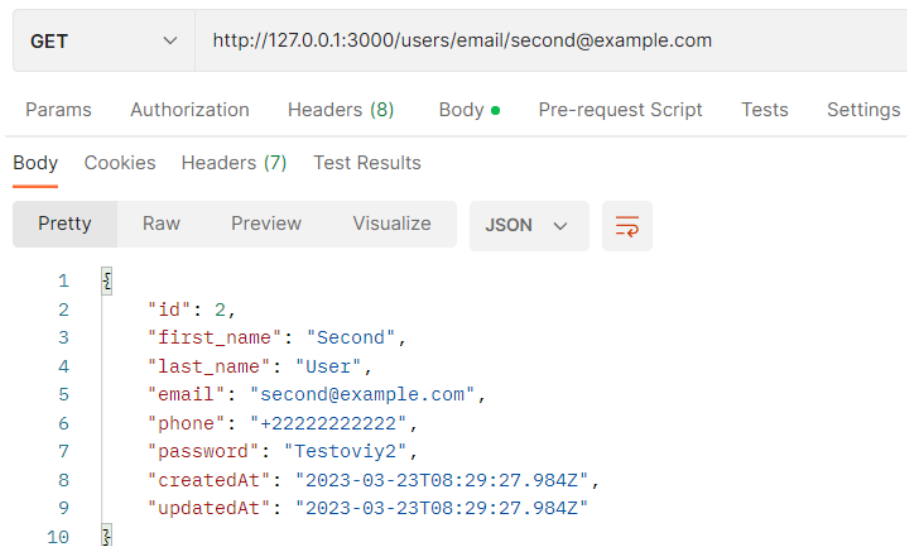
Body Cookies Headers (7) Test Results

Pretty Raw Preview Visualize JSON ▼ ≡

```
1 {  
2   "id": 2,  
3   "first_name": "Second",  
4   "last_name": "User",  
5   "email": "second@example.com",  
6   "phone": "+2222222222",  
7   "password": "Testoviy2",  
8   "createdAt": "2023-03-23T08:29:27.984Z",  
9   "updatedAt": "2023-03-23T08:29:27.984Z"  
10 }
```

- GET запрос на получение пользователя по email

```
app.get('/users/email/:email', async (req : Request<P, ResBody, ReqBody, ReqQuery, LocalsObj> , res : Response<ResBody, LocalsObj> ) => {  
  const user = await db.User.findOne({where: {email: req.params.email}})  
  if (user) {  
    return res.send(user.toJSON())  
  }  
  
  return res.send( body: {"msg": "User is not found by email"})  
})
```



Вывод

В ходе данной работы я познакомилась и научилась работать с ORM Sequelize. Была разработана модель пользователя, подготовлена и запущена миграция средствами `sequelize-cli`. С помощью фреймворка Express был разработан набор CRUD методов для работы с пользователями.