

**САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ  
ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО**

**Дисциплина:** Бэк-энд разработка

Отчет

Домашняя работа №2: Знакомство с ORM  
Sequelize

Выполнила:

Бабан Виктория

Группа К33412

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. Модель пользователя состоит из 6 полей:

- фамилия (lastName)
- имя (firstName)
- отчество (patronymic)
- email
- пароль (password)
- номер телефона (phone)

```
'use strict';
const {
  Model
} = require('sequelize');
module.exports = (sequelize, DataTypes) => {
  class User extends Model {
    /** Helper method for defining associations. ...*/
    static associate(models) {...}
  }
  User.init({ attributes: {
    lastName: DataTypes.STRING,
    firstName: DataTypes.STRING,
    patronymic: DataTypes.STRING,
    email: DataTypes.STRING,
    password: DataTypes.STRING,
    phone: DataTypes.STRING
  }}, { options: {
    defaultScope: {
      attributes: {exclude: ['password']},
    },
    sequelize,
    modelName: 'User',
    tableName: 'Users',
  });
  return User;
};
```

## 2. Были реализованы следующие CRUD-методы:

- просмотр 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)  
})
```

GET http://127.0.0.1:3000/users/

Params Authorization Headers (9) Body Pre-request Script

Body Cookies Headers (7) Test Results

Pretty Raw Preview Visualize JSON

```
1 [
2   {
3     "id": 1,
4     "lastName": "Test1",
5     "firstName": "User1",
6     "patronymic": "-",
7     "email": "user1@example.com",
8     "phone": "+0123456789",
9     "createdAt": "2023-03-13T15:21:40.573Z",
10    "updatedAt": "2023-03-13T15:21:40.573Z"
11  },
12  {
13    "id": 2,
14    "lastName": "Test2",
15    "firstName": "User2",
16    "patronymic": "user_2",
17    "email": "user2@example.com",
18    "phone": "+9876543210",
19    "createdAt": "2023-03-13T15:25:12.318Z",
20    "updatedAt": "2023-03-13T15:27:36.533Z"
21  }
22 ]
```

- добавление 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: {"detail": "Failed to create user"});  
  }  
})
```

POST http://127.0.0.1:3000/users/

Params Authorization Headers (9) Body Pre-request Script

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

1 {  
2 "lastName": "Test3",  
3 "firstName": "User3",  
4 "patronymic": "user\_3",  
5 "email": "user3@example.com",  
6 "phone": "+9876543211"  
7 }

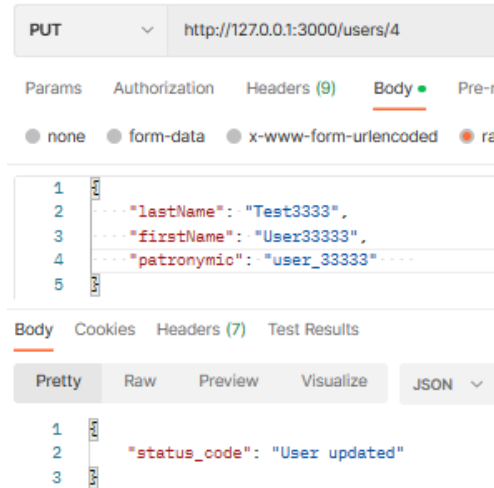
Body Cookies Headers (7) Test Results

Pretty Raw Preview Visualize JSON

```
1 {
2   "id": 4,
3   "lastName": "Test3",
4   "firstName": "User3",
5   "patronymic": "user_3",
6   "email": "user3@example.com",
7   "phone": "+9876543211",
8   "updatedAt": "2023-03-13T16:07:19.536Z",
9   "createdAt": "2023-03-13T16:07:19.536Z"
10 }
```

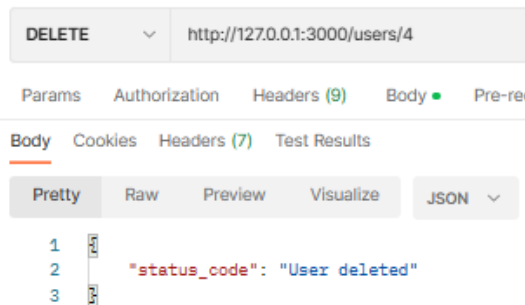
- изменение данных о пользователе

```
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: {'status_code': 'User updated'});
  } catch (error) {
    res.send(error);
  }
})
```



- удаление 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: {'status_code': 'User deleted'})
  } else {
    res.status( code: 404 ).send( body: {'error_code': 'User is not found'})
  }
})
```

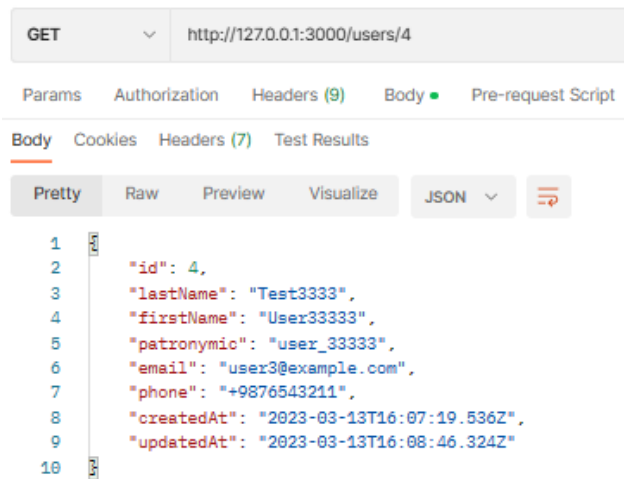


### 3. Написаны запросы для получения пользователя по id и email:

- 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())
  }

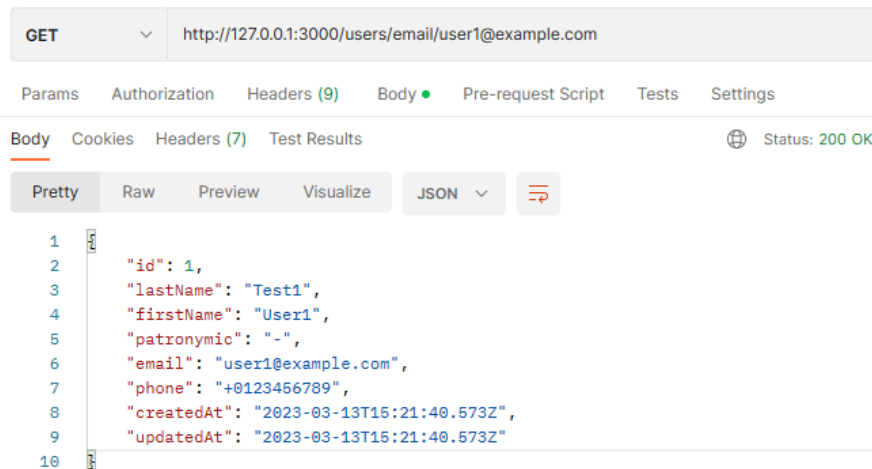
  return res.send( body: {\"msg\": \"User is not found\"})
})
```



- 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 результате выполнения домашнего задания мною была придумана и создана собственная модель пользователя средствами ORM Sequelize, а также реализован набор CRUD-методов для работы с данной моделью с помощью Express.