

**САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ
ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО**

Дисциплина: Бек-энд разработка

Отчет

Практическая работа 2: Знакомство с ORM Sequelize

Выполнил:

Скороходова Елена

Группа
K33392

Проверил:

Добряков Д. И.

Санкт-Петербург

2024 г.

Задачи

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

Ход работы

Собственная модель пользователя:

firstName, lastName, email, username

Для создания модели используем команду

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

```
'use strict';  
const { Model } = require('sequelize');  
  
module.exports = (sequelize, DataTypes) => {  
  class User extends Model {  
    static associate(models) {  
      // define association here  
    }  
  }  
  User.init({  
    firstName: DataTypes.STRING,  
    lastName: DataTypes.STRING,  
    email: DataTypes.STRING,  
    username: DataTypes.STRING  
  }, {  
    sequelize,  
    modelName: 'User',  
  });  
  return User;  
};
```

Реализация CRUD-методов

Напишем маршруты в routes/users.js

```

const express = require('express');
const router = express.Router();
const UserController = require('../controllers/userControl');

// создание
router.post('/create', UserController.createUser);

// получение всех
router.get('/', UserController.getAllUsers);

// получение по ID
router.get('/:id', UserController.getUserById);

// обновление
router.put('/:id', UserController.updateUser);

// удаление
router.delete('/:id', UserController.deleteUser);

// по email
router.get('/email/:email', UserController.getUserByEmail);

module.exports = router;

```

Далее реализуем контроллеры controllers/userControl.js

```

const db = require('../models');
const User = db.User;

const UserController = {

  createUser: async (req, res) => {
    try {
      const newUser = await User.create(req.body);
      res.status(201).json(newUser);
    } catch (error) {
      res.status(400).json({ message: error.message });
    }
  },

  getAllUsers: async (req, res) => {
    try {
      const users = await User.findAll();
      res.json(users);
    } catch (error) {
      res.status(500).json({ message: error.message });
    }
  },

  getUserById: async (req, res) => {
    const { id } = req.params;
    try {
      const user = await User.findByPk(id);
      if (!user) {
        return res.status(404).json({ message: 'User not found' });
      }
      res.json(user);
    } catch (error) {
      res.status(500).json({ message: error.message });
    }
  },
};

```

```

updateUser: async (req, res) => {
  const { id } = req.params;
  try {
    const [updatedRowCount, [updatedUser]] = await User.update(req.body,
      where: { id },
      returning: true,
    );
    if (updatedRowCount === 0) {
      return res.status(404).json({ message: 'User not found' });
    }
    res.json(updatedUser);
  } catch (error) {
    res.status(400).json({ message: error.message });
  }
},

deleteUser: async (req, res) => {
  const { id } = req.params;
  try {
    const deletedRowCount = await User.destroy({ where: { id } });
    if (deletedRowCount === 0) {
      return res.status(404).json({ message: 'User not found' });
    }
    res.sendStatus(204);
  } catch (error) {
    res.status(500).json({ message: error.message });
  }
},

getUserById: async (req, res) => {
  const { id } = req.params;
  try {
    const user = await User.findByPk(id);

```

```

getUserById: async (req, res) => {
  const { id } = req.params;
  try {
    const user = await User.findByPk(id);
    if (!user) {
      return res.status(404).json({ message: 'User not found' });
    }
    res.json(user);
  } catch (error) {
    res.status(500).json({ message: error.message });
  }
},


getUserByEmail: async (req, res) => {
  const { email } = req.params;
  try {
    const user = await User.findOne({ where: { email } });
    if (!user) {
      return res.status(404).json({ message: 'User not found' });
    }
    res.json(user);
  } catch (error) {
    res.status(500).json({ message: error.message });
  }
}

];

module.exports = UserController;

```

Проверка в Postman

 http://localhost:3000/users/create

POST http://localhost:3000/users/create

Params Authorization Headers (9) **Body** Pre-request Script Tests Settings


● none ● form-data ● x-www-form-urlencoded ● **raw** ● binary ● GraphQL **JSON** ▾

```

1  {
2    ... "firstName": "Lili",
3    ... "lastName": "lala",
4    ... "email": "Lili@gmail.com",
5    ... "username": "lililala"
6  }
7  ... }

```

Body Cookies Headers (7) Test Results

Pretty Raw Preview Visualize JSON ▾ 

```

1  {
2    "id": 5,
3    "updatedAt": "2024-03-07T23:47:42.765Z",
4    "createdAt": "2024-03-07T23:47:42.765Z"
5  }

```

HTTP <http://localhost:3000/users/3>


DELETE ▼ <http://localhost:3000/users/3>

Params Authorization Headers (9) **Body** ● Pre-request Script

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

```
1  {
2    "firstName": "Lili",
3    "lastName": "lala",
4    "email": "Lili@gmail.com",
5    "username": "lililala"
6  }
7
```

body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize Text ▼ 

Вывод

В домашнем задании номер два познакомились с ORM Sequelize, удалось создать собственную модель пользователя и реализовать набор из CRUD-методов для работы с пользователями средствами Express + Sequelize.