САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Дисциплина: Бек-энд разработка

Отчет

Практическая работа 2: Знакомство с ORM Sequelize

Выполнил:

Скороходова Елена

Группа К33392

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,
      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 = {
     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) => {
     const users = await User.findAll();
     res.json(users);
          ch (error) {
     res.status(500).json({ message: error.message });
  getUserById: async (req, res) => {
  const { id } = req.params;
      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 });
```

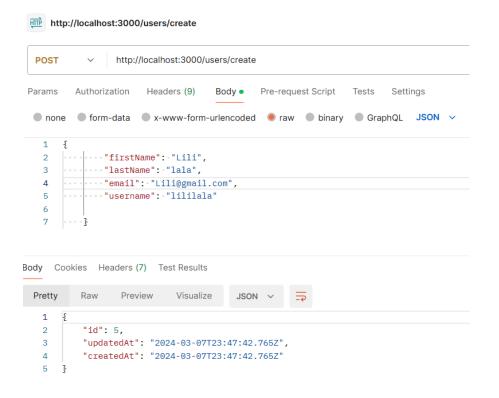
```
updateUser: async (req, res) => {
  const { id } = req.params;
    const [updatedRowsCount, [updatedUser]] = await User.update(req.body,
     where: { id },
      returning: true,
    if (updatedRowsCount === 0) {
     return res.status(404).json({ message: 'User not found' });
    res.json(updatedUser);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

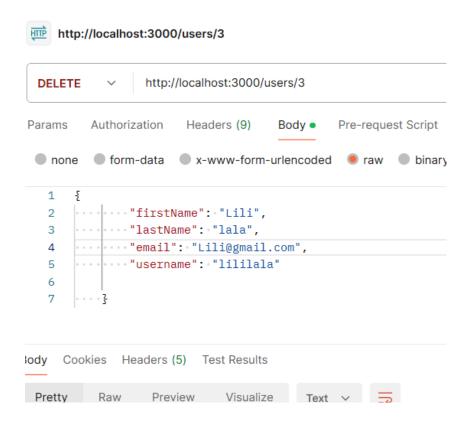
```
getUserById: async (req, res) => {
  const { id } = req.params;
  try {
    const user = await User.findByPk(id);
    if (luser) {
      return res.status(404).json({ message: 'User not found' });
    }
    res.json(user);
    } catch (error) {
    res.status(500).json({ message: error.message });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

getUserByEmail: async (req, res) => {
    const { email } = req.params;
    try {
      const user = await User.findOne({ where: { email } });
      if (luser) {
         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





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.