**САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ**

**ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО**

**Дисциплина:** Бек-энд разработка

Отчет

Лабораторная работа 1: Typescript: основы языка

Выполнила:

Пронина Александра

Группа

К33392

Проверил:

Добряков Д. И.

Санкт-Петербург

2024 г.

Цель работы:

Целью данной лабораторной работы является разработка RESTful API для управления пользователями с использованием технологий Node.js, Express.js и Sequelize ORM.

Используемый стек технологии:

Node.js - среда выполнения JavaScript на сервере.

Express.js - фреймворк для создания веб-приложений на Node.js.

Sequelize - ORM (Object-Relational Mapping) для работы с базами данных в Node.js.

Ход работы:

Настройка проекта:

В начале работы был создан проект с использованием Node.js. Были установлены необходимые зависимости, такие как Express, Sequelize, и другие. Проект был настроен с использованием TypeScript для типизации кода.

Настройка базы данных:

Для работы с базой данных была выбрана SQLite, так как она легко интегрируется в проекты Node.js и не требует отдельного сервера баз данных. Был создан файл конфигурации для Sequelize, где указаны параметры подключения к базе данных.

Вот так выглядит код: import { Sequelize } from 'sequelize-typescript';  
import { User } from '../models/User'; *// Путь к модели User*class SQLRepo {  
  
 repo: Sequelize | null = null;  
  
 constructor() {  
 this.connectDb();  
 }  
  
 connectDb(): any {  
 this.repo = new Sequelize({  
 database: 'db',  
 dialect: 'sqlite',  
 username: 'root',  
 password: '',  
 storage: ':memory:',  
 models: [User], *// Добавляем модель User в массив моделей* });  
  
 this.repo.authenticate()  
 .then(() => {  
 console.log("Database connected ...");  
 })  
 .catch((error: any) => {  
 console.log("Database connection error: " + error);  
 });  
 }  
}  
  
export { SQLRepo };

Создание модели пользователей:

Была создана модель User, описывающая структуру данных пользователей. В модели были определены поля id, name, email и password, а также их типы и ограничения.

+ контроллер:

import { Request, Response } from 'express';  
import { UserService } from '../services/UserService';  
  
class UserController {  
 static async *createUser*(req: Request, res: Response): Promise<void> {  
 const { name, email, password } = req.body;  
  
 try {  
 const newUser = await UserService.*createUser*(name, email, password);  
 res.status(201).json(newUser);  
 } catch (error: any) {  
 console.error(error);  
 res.status(500).json({ message: 'Internal server error' });  
 }  
 }  
  
 static async *getUserById*(req: Request, res: Response): Promise<void> {  
 const id = *Number*(req.params.id);  
  
 try {  
 const user = await UserService.*getUserById*(id);  
 if (user) {  
 res.json(user);  
 } else {  
 res.status(404).json({ message: 'User not found' });  
 }  
 } catch (error: any) {  
 console.error(error);  
 res.status(500).json({ message: 'Internal server error' });  
 }  
 }  
  
 static async *deleteUser*(req: Request, res: Response): Promise<void> {  
 const id = *Number*(req.params.id);  
  
 try {  
 const deleted = await UserService.*deleteUser*(id);  
 if (deleted) {  
 res.status(204).end();  
 } else {  
 res.status(404).json({ message: 'User not found' });  
 }  
 } catch (error: any) {  
 console.error(error);  
 res.status(500).json({ message: 'Internal server error' });  
 }  
 }  
}  
  
export { UserController };

Разработка репозитория пользователей:

Был разработан репозиторий UserRepository, отвечающий за выполнение операций с данными пользователей (создание, чтение, удаление) в базе данных. Репозиторий использует модель User для взаимодействия с базой данных.

import { User } from '../models/User';  
  
class UserRepository {  
 static async *createUser*(name: string, email: string, password: string): Promise<User> {  
 try {  
 const newUser = await User.*create*({ name, email, password });  
 return newUser;  
 } catch (error: any) {  
 throw new *Error*('Error creating user: ' + error.message);  
 }  
 }  
  
 static async *getUserById*(id: number): Promise<User | null> {  
 try {  
 const user = await User.*findByPk*(id);  
 return user;  
 } catch (error: any) {  
 throw new *Error*('Error getting user by id: ' + error.message);  
 }  
 }  
  
 static async *deleteUser*(id: number): Promise<boolean> {  
 try {  
 const deletedCount = await User.*destroy*({ where: { id } });  
 return deletedCount > 0;  
 } catch (error: any) {  
 throw new *Error*('Error deleting user: ' + error.message);  
 }  
 }  
}  
  
export { UserRepository };

Разработка сервиса пользователей:

Был разработан сервис UserService, предоставляющий бизнес-логику для управления пользователями. Сервис использует репозиторий UserRepository для выполнения операций с данными пользователей.

Настройка маршрутов:

Были настроены маршруты Express для обработки запросов от клиента. Для управления пользователями были созданы маршруты POST /users (создание пользователя), GET /users/:id (получение пользователя по ID) и DELETE /users/:id (удаление пользователя).

import express from 'express';  
import { UserController } from '../controllers/User';  
  
const router = *express*.*Router*();  
  
router.post('/', UserController.*createUser*);  
router.get('/:id', UserController.*getUserById*);  
router.delete('/:id', UserController.*deleteUser*);  
  
export { router as userRoutes };

Донастройка синхронизации: Изображение выглядит как текст, Шрифт, снимок экрана

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, Шрифт, линия

Автоматически созданное описание

Тест на Node.js:

Изображение выглядит как текст, снимок экрана, программное обеспечение, Значок на компьютере

Автоматически созданное описание

**Выводы:**

В результате выполнения лабораторной работы было разработано RESTful API для управления пользователями с использованием Node.js, Express и Sequelize. Полученный API обладает функциональностью создания, чтения, обновления и удаления пользователей, что позволяет эффективно управлять данными пользователей через HTTP запросы. Процесс разработки позволил понять основы работы с Node.js, фреймворком Express и ORM Sequelize, а также научиться создавать веб-приложения и API на их основе.