МИНИСТЕРСТВО ТРАНСПОРТА РФ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «РОССИЙСКИЙ УНИВЕРСИТЕТ ТРАНСПОРТА (МИИТ)»

Кафедра ЦТУТП

**Отчёт**

По лабораторной работе №6  
по дисциплине «Корпоративные информационные системы»

Тема: «Работа с библиотекой MyBatis»

Вариант №13

Выполнил: Веселков М.Ю.

Группа: УИС-411

Преподаватель: доц. Кафедры ЦТУТП

Козьяков П. О.

­

Москва 2024 г.

**СОДЕРЖАНИЕ**

[ПОСТАНОВКА ЗАДАЧИ 3](#_Toc180325218)

[ХОД ВЫПОЛНЕНИЯ РАБОТЫ 4](#_Toc180325219)

[РЕЗУЛЬТАТ РАБОТЫ ПРОГРАММЫ 6](#_Toc180325220)

[КОД ПРОГРАММЫ 7](#_Toc180325221)

[ВЫВОД 11](#_Toc180325222)

# **ПОСТАНОВКА ЗАДАЧИ**

Описать класс Магазин. В БД Postgres создать таблицу «магазины». С помощью MyBatis реализовать операции Добавления, Удаления, Изменения и Выборки данных из таблицы «магазины». Продемонстрировать операции работы с БД.

# **ХОД ВЫПОЛНЕНИЯ РАБОТЫ**

Приложение представляет собой RESTful веб-сервис для управления информацией о магазинах, разработанный с использованием Spring Boot, MyBatis и PostgreSQL. Модель **Shop** определяет структуру данных магазина с полями id, title и adress. Аннотации Lombok (@Data, @NoArgsConstructor, @AllArgsConstructor) используются для автоматической генерации необходимых методов, таких как геттеры и сеттеры.

**Интерфейс ShopDao** определяет методы для взаимодействия с базой данных: добавление, обновление, выборка магазина по ID и удаление. Использование аннотаций MyBatis (@Insert, @Update, @Select, @Delete) позволяет связывать методы с соответствующими SQL-запросами. Дополнительно, XML-файл маппинга предоставляет детальную настройку SQL-запросов и маппинга результатов через resultMap.

Сервисный слой представлен классом **ShopService**, который инкапсулирует бизнес-логику и взаимодействует с ShopDao. Он предоставляет методы для добавления нового магазина, обновления данных, получения информации по ID и удаления магазина по идентификатору.

Контроллер **ShopController** определяет REST API для внешнего взаимодействия. Он обрабатывает HTTP-запросы, такие как POST для добавления нового магазина, PUT для обновления существующего, POST для получения информации о магазине по ID и DELETE для удаления по идентификатору. Каждый из этих методов обращается к соответствующим методам сервиса и возвращает ответ в формате JSON, обернутый в объект ResponseWrapper.

Конфигурация приложения задается в файле application.properties, где устанавливаются параметры подключения к базе данных PostgreSQL, настройки логирования для MyBatis и Spring JDBC, а также порт сервера (8080). Логирование позволяет отслеживать выполняемые SQL-запросы и общее состояние приложения.

При запуске приложения Spring Boot и обращении к API происходят следующие процессы:

1. Клиент отправляет HTTP-запрос к определенному эндпоинту контроллера.
2. Контроллер получает запрос и вызывает соответствующий метод сервиса.
3. Сервис обращается к ShopDao для выполнения необходимой операции с базой данных.
4. ShopDao, используя MyBatis, выполняет SQL-запрос к базе данных PostgreSQL.
5. Результат операции возвращается обратно через слои приложения к клиенту.

Выводом работы приложения являются JSON-ответы на запросы клиентов, содержащие информацию о магазинах. Например, при успешном добавлении нового магазина сервер может вернуть статус 200 OK с подтверждением операции. При запросе информации о магазине клиент получает объект **Shop** с заполненными данными. Если возникает ошибка, например, при попытке обновления несуществующего магазина, сервер вернет соответствующий код ошибки и сообщение.

# **РЕЗУЛЬТАТ РАБОТЫ ПРОГРАММЫ**

Изображение выглядит как снимок экрана, текст, программное обеспечение, Мультимедийное программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 1 – Результат работы программы

# **КОД ПРОГРАММЫ**

package rut.miit.lab6mybatisdb.model;  
  
import com.fasterxml.jackson.annotation.JsonProperty;  
import lombok.AllArgsConstructor;  
import lombok.Data;  
import lombok.NoArgsConstructor;  
  
@Data  
@AllArgsConstructor  
@NoArgsConstructor  
  
public class Shop {  
 private long id;  
  
 @JsonProperty("title")  
 private String title;  
  
 @JsonProperty("adress")  
 private String adress;  
  
 @JsonProperty("phone")  
 private String phone;  
  
}

package rut.miit.lab6mybatisdb.mapper;  
  
import org.apache.ibatis.annotations.\*;  
import rut.miit.lab6mybatisdb.model.Shop;  
  
import java.util.List;  
  
@Mapper  
public interface ShopMapper {  
  
 @Insert("INSERT INTO shops (title, adress, phone) VALUES (#{title}, #{adress}, #{phone})")  
 void insertShop(Shop shop) ;  
  
 @Update("UPDATE shops SET title = #{title}, adress=#{adress}, phone=#{phone} WHERE id = #{id}")  
 void updateShop(Shop shop) ;  
  
 @Select("SELECT \* FROM lab6.shops")  
 @Results({  
 @Result(property = "title", column = "title"),  
 @Result(property = "adress", column = "adress"),  
 @Result(property = "phone", column = "phone")  
 })  
 List<Shop> selectShop();  
  
 @Delete("DELETE FROM shops WHERE id= #{id}")  
 void deleteShop(Long id) ;  
}

package rut.miit.lab6mybatisdb.mapper;  
  
  
import org.apache.ibatis.annotations.Mapper;  
import org.springframework.stereotype.Repository;  
import rut.miit.lab6mybatisdb.dto.ShopRequest;  
import rut.miit.lab6mybatisdb.model.Shop;  
  
import java.util.List;  
  
@Mapper  
public interface ShopDao {  
 int deleteByPrimaryKey(Long id);  
  
 int insert(Shop record);  
  
 int insertSelective(Shop record);  
  
 Shop selectByPrimaryKey(Long id);  
  
 int updateByPrimaryKeySelective(Shop record);  
  
 int updateByPrimaryKey(Shop record);  
  
 List<Shop> selectAll();  
}

<?xml version="1.0" encoding="UTF-8"?>  
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN" "http://mybatis.org/dtd/mybatis-3-mapper.dtd">  
<mapper namespace="rut.miit.lab6mybatisdb.mapper.ShopDao">  
  
 <resultMap id="ShopResultMap" type="rut.miit.lab6mybatisdb.model.Shop">  
 <id column="id" jdbcType="BIGINT" property="id" />  
 <result column="title" jdbcType="VARCHAR" property="title" />  
 <result column="adress" jdbcType="VARCHAR" property="adress" />  
 <result column="phone" jdbcType="VARCHAR" property="phone" />  
 </resultMap>  
  
 <sql id="Shop\_Column\_List">  
 id, title, adress, phone  
 </sql>  
  
 <select id="selectByPrimaryKey" parameterType="java.lang.Long" resultMap="ShopResultMap">  
 select  
 <include refid="Shop\_Column\_List" />  
 from lab6.shops <!-- таблица называется shops -->  
 where id = #{id,jdbcType=BIGINT}  
 </select>  
  
 <insert id="insert" parameterType="rut.miit.lab6mybatisdb.model.Shop">  
 insert into lab6.shops (title, adress, phone) <!-- таблица называется shops -->  
 values (#{title,jdbcType=VARCHAR}, #{adress,jdbcType=VARCHAR}, #{phone,jdbcType=VARCHAR})  
 </insert>  
  
 <update id="updateByPrimaryKey" parameterType="rut.miit.lab6mybatisdb.model.Shop">  
 update lab6.shops <!-- таблица называется shops -->  
 set title = #{title,jdbcType=VARCHAR},  
 adress = #{adress,jdbcType=VARCHAR},  
 phone = #{phone,jdbcType=VARCHAR}  
 where id = #{id,jdbcType=BIGINT}  
 </update>  
  
 <delete id="deleteByPrimaryKey" parameterType="java.lang.Long">  
 delete from lab6.shops where id = #{id,jdbcType=BIGINT} <!-- таблица называется shops -->  
 </delete>  
  
 <select id="selectAll" resultMap="ShopResultMap">  
 select <include refid="Shop\_Column\_List" /> from lab6.shops  
 </select>  
  
</mapper>

package rut.miit.lab6mybatisdb.services;  
  
  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Service;  
import rut.miit.lab6mybatisdb.dto.ShopRequest;  
import rut.miit.lab6mybatisdb.mapper.ShopDao;  
import rut.miit.lab6mybatisdb.model.Shop;  
  
import java.util.List;  
  
  
@Service  
public class ShopService {  
 @Autowired  
 private ShopDao shopDao;  
  
 public Shop getShopById(Long id) {  
 return shopDao.selectByPrimaryKey(id);  
 }  
  
 public int addShop(Shop shop) {  
 return shopDao.insert(shop);  
 }  
  
 public int updateShop(Shop shop) {  
 return shopDao.updateByPrimaryKey(shop);  
 }  
  
 public int deleteShop(Long id) {  
 return shopDao.deleteByPrimaryKey(id);  
 }  
  
  
 public List<Shop> getAllShops() {  
 return shopDao.selectAll();  
 }  
  
}

package rut.miit.lab6mybatisdb.controllers;  
  
  
import io.swagger.v3.oas.annotations.Operation;  
import lombok.extern.slf4j.Slf4j;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.http.MediaType;  
import org.springframework.web.bind.annotation.\*;  
import rut.miit.lab6mybatisdb.dto.ResponseWrapper;  
import rut.miit.lab6mybatisdb.dto.ShopRequest;  
import rut.miit.lab6mybatisdb.model.Shop;  
import rut.miit.lab6mybatisdb.services.ShopService;  
  
import java.util.List;  
  
@RestController  
@RequestMapping("/shop/op")  
@Slf4j  
public class ShopController {  
 @Autowired  
 private ShopService shopService;  
  
 @Operation(description = "Версия")  
 @GetMapping(value = "/ver", produces = MediaType.*APPLICATION\_JSON\_VALUE*)  
 public ResponseWrapper<String> shopVersion() {  
 ResponseWrapper<String> response = new ResponseWrapper<>();  
  
 response.setSuccess(true);  
 response.setData("Version 1.0.0");  
 return response;  
 }  
  
 @Operation(description = "Получить информацию о магазине по ID")  
 @PostMapping(value = "/id", produces = MediaType.*APPLICATION\_JSON\_VALUE*)  
 public ResponseWrapper<Shop> shopById(@RequestBody ShopRequest request) {  
 ResponseWrapper<Shop> response = new ResponseWrapper<>();  
 response.setSuccess(true);  
 response.setData(shopService.getShopById(request.getShopId()));  
 return response;  
 }  
  
 @Operation(description = "Получить список всех магазинов")  
 @GetMapping(value = "/list", produces = MediaType.*APPLICATION\_JSON\_VALUE*)  
 public ResponseWrapper<List<Shop>> getAllShops() {  
 ResponseWrapper<List<Shop>> response = new ResponseWrapper<>();  
 response.setSuccess(true);  
 response.setData(shopService.getAllShops());  
 return response;  
 }  
  
 @Operation(description = "Добавить новый магазин")  
 @PostMapping(value = "/add", produces = MediaType.*APPLICATION\_JSON\_VALUE*)  
 public ResponseWrapper<Integer> addShop(@RequestBody Shop shop) {  
 ResponseWrapper<Integer> response = new ResponseWrapper<>();  
 response.setSuccess(true);  
 response.setData(shopService.addShop(shop));  
 return response;  
 }  
  
 @Operation(description = "Обновить данные магазина")  
 @PutMapping(value = "/update", produces = MediaType.*APPLICATION\_JSON\_VALUE*)  
 public ResponseWrapper<Integer> updateShop(@RequestBody Shop shop) {  
 ResponseWrapper<Integer> response = new ResponseWrapper<>();  
 response.setSuccess(true);  
 response.setData(shopService.updateShop(shop));  
 return response;  
 }  
  
 @Operation(description = "Удалить магазин по ID")  
 @DeleteMapping(value = "/delete/{id}", produces = MediaType.*APPLICATION\_JSON\_VALUE*)  
 public ResponseWrapper<Integer> deleteShop(@PathVariable Long id) {  
 ResponseWrapper<Integer> response = new ResponseWrapper<>();  
 response.setSuccess(true);  
 response.setData(shopService.deleteShop(id));  
 return response;  
 }  
  
}

spring.application.name=Lab6MybatisDb  
server.port=8080  
  
spring.datasource.url=jdbc:postgresql://localhost:5432/postgres  
spring.datasource.username=postgres  
spring.datasource.password=postgres  
spring.datasource.driver-class-name=org.postgresql.Driver  
logging.level.org.apache.ibatis=DEBUG  
logging.level.rut.miit.lab6mybatisdb.mapper=DEBUG  
  
mybatis.mapper-locations=classpath\*:src/main/resources/rut/miit/lab6mybatisdb/mapper/\*.xml  
  
emd.version=1.0.0

# **ВЫВОД**

В результате, приложение обеспечивает полный цикл операций CRUD (Create, Read, Update, Delete) для управления данными о магазинах, демонстрируя использование современных технологий Java и интеграцию между Spring Boot, MyBatis и PostgreSQL для создания эффективного и масштабируемого веб-сервиса.