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 №7

Выполнил: Шедания В. М.

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

# **ХОД ВЫПОЛНЕНИЯ РАБОТЫ**

Приложение представляет собой RESTful веб-сервис для управления информацией о поездах, разработанный с использованием Spring Boot, MyBatis и PostgreSQL. Модель Train определяет структуру данных поезда с полями id, name, number и isActive. Аннотации Lombok (@Data, @NoArgsConstructor, @AllArgsConstructor) используются для автоматической генерации необходимых методов, таких как геттеры и сеттеры.

Интерфейс TrainMapper определяет методы для взаимодействия с базой данных: добавление, обновление, выборка активных поездов 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 TrainService, который инкапсулирует бизнес-логику и взаимодействует с TrainMapper. Он предоставляет методы для добавления нового поезда, обновления данных, получения списка активных поездов и удаления поезда по идентификатору.

Контроллер TrainController определяет REST API для внешнего взаимодействия. Он обрабатывает HTTP-запросы, такие как POST для добавления нового поезда, PUT для обновления существующего, GET для получения списка активных поездов и DELETE для удаления поезда по ID. Каждый из этих методов обращается к соответствующим методам сервиса.

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 (8081). Логирование настроено таким образом, чтобы выводить отладочную информацию, что облегчает отслеживание выполняемых SQL-запросов и общее состояние приложения.

При запуске приложения Spring Boot и обращении к API происходят следующие процессы: клиент отправляет HTTP-запрос к определенному эндпоинту контроллера; контроллер получает запрос и вызывает соответствующий метод сервиса; сервис обращается к мапперу для выполнения необходимой операции с базой данных; маппер, используя MyBatis, выполняет SQL-запрос к базе данных PostgreSQL; результат операции возвращается обратно через слои приложения к клиенту.

Выводом работы приложения являются JSON-ответы на запросы клиентов, содержащие информацию о поездах. Например, при успешном добавлении нового поезда сервер может вернуть статус 200 OK с подтверждением операции. При запросе списка активных поездов клиент получает массив объектов Train с заполненными данными. Если возникает ошибка, например, при попытке обновления несуществующего поезда, сервер вернет соответствующий код ошибки и сообщение.

# **РЕЗУЛЬТАТ РАБОТЫ ПРОГРАММЫ**

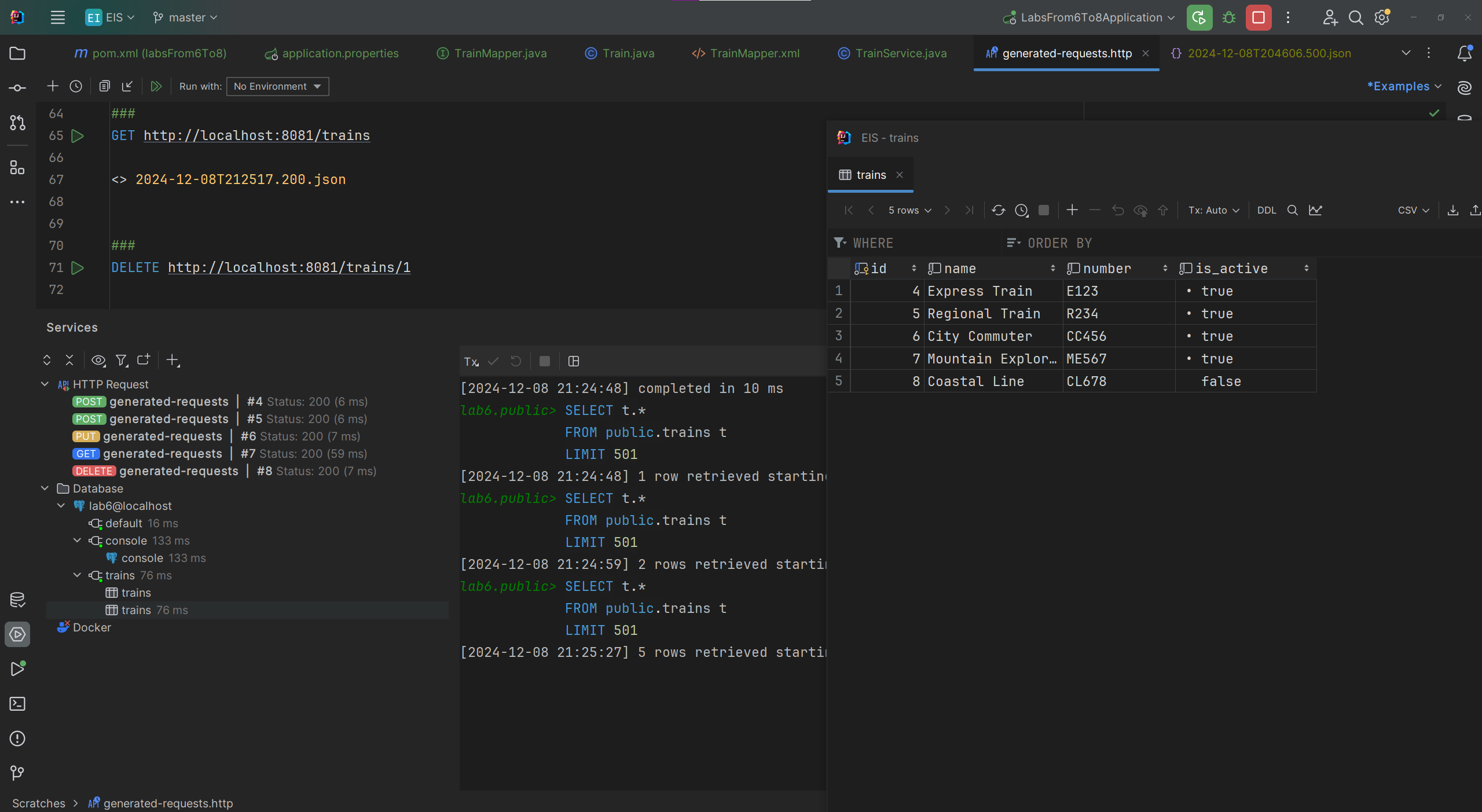


Рисунок 1 – Результат работы программы

# **КОД ПРОГРАММЫ**

package com.labsfrom6to8.Model;  
  
import lombok.AllArgsConstructor;  
import lombok.Data;  
import lombok.NoArgsConstructor;  
  
import com.fasterxml.jackson.annotation.JsonProperty;  
  
@Data  
@NoArgsConstructor  
@AllArgsConstructor  
public class Train {  
  
 private Long id;  
  
 @JsonProperty("name")  
 private String name;  
  
 @JsonProperty("number")  
 private String number;  
  
 @JsonProperty("is\_active")  
 private boolean isActive;  
  
}

package com.labsfrom6to8.Mapper;  
  
import com.labsfrom6to8.Model.Train;  
import org.apache.ibatis.annotations.\*;  
  
import java.util.List;  
  
@Mapper  
public interface TrainMapper {  
  
 @Insert("INSERT INTO trains (name, number, is\_active) VALUES (#{name}, #{number}, #{isActive})")  
 void insertTrain(Train train);  
  
 @Update("UPDATE trains SET name = #{name}, number = #{number}, is\_active = #{isActive} WHERE id = #{id}")  
 void updateTrain(Train train);  
  
 @Select("SELECT \* FROM trains WHERE is\_active = TRUE")  
 @Results({  
 @Result(property = "name", column = "name"),  
 @Result(property = "number", column = "number"),  
 @Result(property = "isActive", column = "is\_active")  
 })  
 List<Train> selectActiveTrains();  
  
 @Delete("DELETE FROM trains WHERE id = #{id}")  
 void deleteTrain(Long id);  
}

<?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="com.labsfrom6to8.Mapper.TrainMapper">  
  
 <!-- Вставка новой записи -->  
 <insert id="insertTrain" parameterType="com.labsfrom6to8.Model.Train">  
 INSERT INTO trains (name, number, is\_active)  
 VALUES (#{name}, #{number}, #{isActive})  
 </insert>  
  
 <!-- Обновление записи -->  
 <update id="updateTrain" parameterType="com.labsfrom6to8.Model.Train">  
 UPDATE trains  
 SET name = #{name},  
 number = #{number},  
 is\_active = #{isActive}  
 WHERE id = #{id}  
 </update>  
  
 <!-- Результат маппинга -->  
 <resultMap id="TrainResultMap" type="com.labsfrom6to8.Model.Train">  
 <result property="id" column="id"/>  
 <result property="name" column="name"/>  
 <result property="number" column="number"/>  
 <result property="isActive" column="is\_active"/>  
 </resultMap>  
  
 <!-- Выбор всех активных поездов -->  
 <select id="selectActiveTrains" resultMap="TrainResultMap">  
 SELECT \* FROM trains WHERE is\_active = TRUE  
 </select>  
  
 <!-- Удаление записи -->  
 <delete id="deleteTrain" parameterType="Long">  
 DELETE FROM trains  
 WHERE id = #{id}  
 </delete>  
</mapper>

package com.labsfrom6to8.Service;  
  
import com.labsfrom6to8.Mapper.TrainMapper;  
import com.labsfrom6to8.Model.Train;  
import org.springframework.stereotype.Service;  
  
import java.util.List;  
  
@Service  
public class TrainService {  
  
 private final TrainMapper trainMapper;  
  
 // ctor  
 public TrainService(TrainMapper trainMapper) {  
 this.trainMapper = trainMapper;  
 }  
  
 // Добавление нового поезда  
 public void addTrain(Train train) {  
 trainMapper.insertTrain(train);  
 }  
  
 // Обновление данных поезда  
 public void updateTrain(Train train) {  
 trainMapper.updateTrain(train);  
 }  
  
 // Получение списка активных поездов  
 public List<Train> getActiveTrains() {  
 return trainMapper.selectActiveTrains();  
 }  
  
 // Удаление поезда  
 public void deleteTrainById(Long id) {  
 trainMapper.deleteTrain(id);  
 }  
}

package com.labsfrom6to8.Controller;  
  
import com.labsfrom6to8.Model.Train;  
import com.labsfrom6to8.Service.TrainService;  
import org.springframework.web.bind.annotation.\*;  
  
import java.util.List;  
  
@RestController  
@RequestMapping("/trains")  
public class TrainController {  
  
 private final TrainService trainService;  
  
 // ctor  
 public TrainController(TrainService trainService) {  
 this.trainService = trainService;  
 }  
  
 // Добавление нового поезда  
 @PostMapping  
 public void addTrain(@RequestBody Train train) {  
 trainService.addTrain(train);  
 }  
  
 // Обновление данных поезда  
 @PutMapping  
 public void updateTrain(@RequestBody Train train) {  
 trainService.updateTrain(train);  
 }  
  
 // Получение списка активных поездов  
 @GetMapping  
 public List<Train> getActiveTrains() {  
 return trainService.getActiveTrains();  
 }  
  
 // Удаление поезда по ID  
 @DeleteMapping("/{id}")  
 public void deleteTrain(@PathVariable Long id) {  
 trainService.deleteTrainById(id);  
 }  
}

spring.application.name=labsFrom6To8  
  
spring.datasource.url=jdbc:postgresql://localhost:5432/lab6  
spring.datasource.username=postgres  
spring.datasource.password=123  
spring.datasource.driver-class-name=org.postgresql.Driver  
  
logging.level.org.mybatis=DEBUG  
logging.level.org.springframework.jdbc.core=DEBUG  
logging.level.org.springframework.jdbc.datasource=DEBUG  
logging.level.org.apache.ibatis.logging.stdout.StdOutImpl=TRACE  
mybatis.configuration.log-impl=org.apache.ibatis.logging.stdout.StdOutImpl  
  
spring.mvc.view.prefix=  
spring.mvc.view.suffix=  
  
server.port=8081

# **ВЫВОД**

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