Министерство образования Республики Беларусь

Учреждение образования

«Белорусский государственный университет информатики и радиоэлектроники»

Кафедра электронных вычислительных машин

Лабораторная работа №1

по дисциплине «Программирование на языках высокого уровня»

«JPA (Hibernate/Spring Data)»

Выполнил: Снитко Д.А. Проверил: Скиба И.Г.

гр.250501

Минск 2024

**1. Постановка задачи**

Подключить в проект БД (PostgreSQL/MySQL/и т.д.). Реализация связи один ко многим @OneToMany. Реализация связи многие ко многим @ManyToMany. Реализовать CRUD-операции со всеми сущностями.

**2. Структура проекта**

В проекте используется послойная архитектура из нескольких пакетов,

которые отвечают за определенные функции.

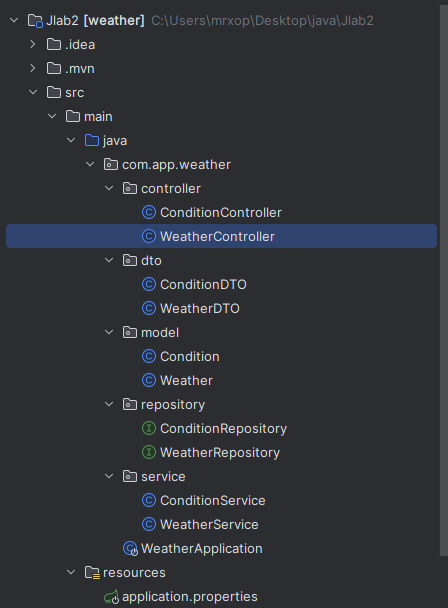


Рисунок 2.1 – Структура проекта

**3. Листинг кода**

Файл ConditionController.java

package com.app.weather.controller;

import com.app.weather.dto.ConditionDTO;

import com.app.weather.model.Condition;

import com.app.weather.model.Weather;

import com.app.weather.repository.ConditionRepository;

import com.app.weather.repository.WeatherRepository;

import com.app.weather.service.ConditionService;

import com.app.weather.service.WeatherService;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.http.ResponseEntity;

import org.springframework.web.bind.annotation.\*;

import java.util.List;

@RestController

@RequestMapping("/conditions")

public class ConditionController {

private final WeatherRepository weatherRepository;

private final ConditionRepository conditionRepository;

private final WeatherService weatherService;

private final ConditionService conditionService;

@Autowired

public ConditionController(WeatherRepository weatherRepository, ConditionRepository conditionRepository,

WeatherService weatherService, ConditionService conditionService) {

this.weatherRepository = weatherRepository;

this.conditionRepository = conditionRepository;

this.weatherService = weatherService;

this.conditionService = conditionService;

}

@PostMapping("/{weatherId}")

public ConditionDTO createCondition(@PathVariable Long weatherId, @RequestBody ConditionDTO conditionDTO) {

Weather weather = weatherService.getWeatherById(weatherId);

Condition newCondition = convertToEntity(conditionDTO);

newCondition.setWeather(weather);

Condition createdCondition = conditionService.createCondition(newCondition);

return convertToDTO(createdCondition);

}

@GetMapping("/{weatherId}")

public ResponseEntity<List<Condition>> getAllConditions(@PathVariable Long weatherId) {

return weatherRepository.findById(weatherId)

.map(weather -> ResponseEntity.ok().body(weather.getConditions()))

.orElse(ResponseEntity.notFound().build());

}

@PutMapping("/{weatherId}")

public ResponseEntity<Void> updateCondition(@PathVariable Long weatherId, @RequestBody Condition conditionDTO) {

Weather weather = weatherRepository.findById(weatherId).orElse(null);

if (weather != null) {

Condition condition = conditionRepository.findByWeather(weather);

condition.setText(conditionDTO.getText());

conditionRepository.save(condition);

return ResponseEntity.ok().build();

}

return ResponseEntity.notFound().build();

}

@DeleteMapping("/{weatherId}")

public ResponseEntity<Void> deleteCondition(@PathVariable Long weatherId) {

Weather weather = weatherRepository.findById(weatherId).orElse(null);

if (weather != null) {

Condition condition = conditionRepository.findByWeather(weather);

weather.removeCondition(condition);

weatherRepository.save(weather);

conditionRepository.delete(condition);

return ResponseEntity.ok().build();

}

return ResponseEntity.notFound().build();

}

private ConditionDTO convertToDTO(Condition condition) {

return new ConditionDTO(condition.getId(), condition.getText(), condition.getWeather().getId());

}

private Condition convertToEntity(ConditionDTO conditionDTO) {

Condition condition = new Condition();

condition.setId(conditionDTO.getId());

condition.setText(conditionDTO.getText());

return condition;

}

}

Файл WeatherController.java

package com.app.weather.controller;

import com.app.weather.dto.WeatherDTO;

import com.app.weather.model.Weather;

import com.app.weather.service.WeatherService;

import org.springframework.web.bind.annotation.\*;

import java.util.List;

@RestController

@RequestMapping("/weather")

public class WeatherController {

private final WeatherService weatherService;

public WeatherController(WeatherService weatherService) {

this.weatherService = weatherService;

}

@GetMapping("/{city}")

public WeatherDTO getWeather(@PathVariable String city) {

Weather weather = weatherService.getWeather(city);

return weatherService.convertToDTO(weather);

}

@PostMapping("/{city}")

public WeatherDTO addWeather(@PathVariable String city, @RequestBody WeatherDTO weatherDTO) {

Weather weather = weatherService.convertToEntity(weatherDTO);

Weather createdWeather = weatherService.createWeather(city, weather);

return weatherService.convertToDTO(createdWeather);

}

@GetMapping("/all")

public List<WeatherDTO> getAllWeather() {

List<Weather> allWeather = weatherService.getAllWeather();

return allWeather.stream()

.map(weatherService::convertToDTO)

.toList();

}

@DeleteMapping("/{city}")

public void deleteWeatherByCity(@PathVariable String city) {

weatherService.deleteWeatherByCity(city);

}

@PutMapping("/{city}")

public WeatherDTO updateWeather(@PathVariable String city, @RequestBody WeatherDTO weatherDTO) {

Weather weather = weatherService.convertToEntity(weatherDTO);

Weather updatedWeather = weatherService.updateWeather(city, weather);

return weatherService.convertToDTO(updatedWeather);

}

}

Файл ConditionDTO.java

package com.app.weather.dto;

import lombok.AllArgsConstructor;

import lombok.Getter;

import lombok.NoArgsConstructor;

import lombok.Setter;

@Getter

@Setter

@AllArgsConstructor

@NoArgsConstructor

public class ConditionDTO {

private Long id;

private String text;

private Long weatherId;

}

Файл WeatherDTO.java

package com.app.weather.dto;

import lombok.AllArgsConstructor;

import lombok.Getter;

import lombok.NoArgsConstructor;

import lombok.Setter;

import java.sql.Timestamp;

import java.util.List;

@Getter

@Setter

@AllArgsConstructor

@NoArgsConstructor

public class WeatherDTO {

private Long id;

private String city;

private Timestamp date;

private double temperature;

private List<ConditionDTO> conditions;

public WeatherDTO(Long id, String city, Timestamp date, double temperature) {

}

}

Файл Сondition.java

package com.app.weather.model;

import com.fasterxml.jackson.annotation.JsonIgnore;

import jakarta.persistence.\*;

import lombok.AllArgsConstructor;

import lombok.Getter;

import lombok.NoArgsConstructor;

import lombok.Setter;

@Entity

@Table(name = "condition")

@Getter

@Setter

@AllArgsConstructor

@NoArgsConstructor

public class Condition {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

@Column(name = "text")

private String text;

@ManyToOne

@JoinColumn(name = "weatherId")

@JsonIgnore

private Weather weather;

public Long getWeatherId() {

return weather.getId();

}

}

Файл Weather.java

package com.app.weather.model;

import jakarta.persistence.\*;

import lombok.AllArgsConstructor;

import lombok.Getter;

import lombok.NoArgsConstructor;

import lombok.Setter;

import java.sql.Timestamp;

import java.util.ArrayList;

import java.util.List;

@Entity

@Table(name = "weather")

@Getter

@Setter

@AllArgsConstructor

@NoArgsConstructor

public class Weather {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

@Column(name = "date")

private Timestamp date;

@Column(name = "city")

private String city;

@Column(name = "temperature")

private double temperature;

@OneToMany(mappedBy = "weather", cascade = CascadeType.ALL, orphanRemoval = true)

private List<Condition> conditions = new ArrayList<>();

public void addCondition(Condition condition) {

conditions.add(condition);

condition.setWeather(this);

}

public void removeCondition(Condition condition) {

conditions.remove(condition);

condition.setWeather(null);

}

}

Файл ConditionRepository.java

package com.app.weather.repository;

import com.app.weather.model.Condition;

import com.app.weather.model.Weather;

import org.springframework.data.jpa.repository.JpaRepository;

import org.springframework.stereotype.Repository;

@Repository

public interface ConditionRepository extends JpaRepository<Condition, Long> {

Condition findByWeather(Weather weather);

}

Файл WeatherRepository.java

package com.app.weather.repository;

import com.app.weather.model.Weather;

import org.springframework.data.jpa.repository.JpaRepository;

import org.springframework.stereotype.Repository;

@Repository

public interface WeatherRepository extends JpaRepository<Weather, Long> {

Weather findByCity(String city);

}

Файл ConditionService.java

package com.app.weather.service;

import com.app.weather.dto.ConditionDTO;

import com.app.weather.model.Condition;

import com.app.weather.model.Weather;

import com.app.weather.repository.ConditionRepository;

import com.app.weather.repository.WeatherRepository;

import org.springframework.stereotype.Service;

@Service

public class ConditionService {

private final ConditionRepository conditionRepository;

private final WeatherRepository weatherRepository;

public ConditionService(ConditionRepository conditionRepository, WeatherRepository weatherRepository) {

this.conditionRepository = conditionRepository;

this.weatherRepository = weatherRepository;

}

public Condition createCondition(Condition conditionDTO) {

Weather weather = weatherRepository.findById(conditionDTO.getWeatherId()).orElse(null);

if (weather != null) {

Condition existingCondition = conditionRepository.findByWeather(weather);

if (existingCondition != null) {

existingCondition.setText(conditionDTO.getText());

return conditionRepository.save(existingCondition);

} else {

Condition newCondition = new Condition();

newCondition.setText(conditionDTO.getText());

newCondition.setWeather(weather);

return conditionRepository.save(newCondition);

}

}

return null;

}

public Condition updateCondition(Long conditionId, ConditionDTO updatedConditionDTO) {

return conditionRepository.findById(conditionId).map(condition -> {

Weather weather = weatherRepository.findById(updatedConditionDTO.getWeatherId()).orElse(null);

condition.setText(updatedConditionDTO.getText());

condition.setWeather(weather);

return conditionRepository.save(condition);

}).orElse(null);

}

public boolean deleteCondition(Long conditionId) {

return conditionRepository.findById(conditionId).map(condition -> {

conditionRepository.delete(condition);

return true;

}).orElse(false);

}

}

Файл WeatherService.java

package com.app.weather.service;

import com.app.weather.dto.ConditionDTO;

import com.app.weather.dto.WeatherDTO;

import com.app.weather.model.Condition;

import com.app.weather.model.Weather;

import com.app.weather.repository.WeatherRepository;

import org.springframework.stereotype.Service;

import java.sql.Timestamp;

import java.time.LocalDateTime;

import java.util.List;

@Service

public class WeatherService {

private final WeatherRepository weatherRepository;

public WeatherService(WeatherRepository weatherRepository) {

this.weatherRepository = weatherRepository;

}

public List<Weather> getAllWeather() {

return weatherRepository.findAll();

}

public void deleteWeatherByCity(String city) {

Weather weather = weatherRepository.findByCity(city);

if (weather != null) {

weatherRepository.delete(weather);

}

}

public Weather getWeatherById(Long weatherId) {

return weatherRepository.findById(weatherId).orElse(null);

}

public Weather updateWeather(String city, Weather updatedWeather) {

Weather existingWeather = weatherRepository.findByCity(city);

existingWeather.setTemperature(updatedWeather.getTemperature());

existingWeather.setDate(Timestamp.valueOf(LocalDateTime.now()));

return weatherRepository.save(existingWeather);

}

public Weather createWeather(String city, Weather weather) {

Weather existingWeather = weatherRepository.findByCity(city);

if (existingWeather != null) {

existingWeather.setTemperature(weather.getTemperature());

existingWeather.setDate(new Timestamp(System.currentTimeMillis()));

return weatherRepository.save(existingWeather);

} else {

weather.setCity(city);

weather.setDate(new Timestamp(System.currentTimeMillis()));

return weatherRepository.save(weather);

}

}

public Weather getWeather(String city) {

return weatherRepository.findByCity(city);

}

public WeatherDTO convertToDTO(Weather weather) {

WeatherDTO dto = new WeatherDTO();

dto.setId(weather.getId());

dto.setCity(weather.getCity());

dto.setDate(weather.getDate());

dto.setTemperature(weather.getTemperature());

List<ConditionDTO> conditionDTOs = weather.getConditions().stream()

.map(this::convertConditionToDTO)

.toList();

dto.setConditions(conditionDTOs);

return dto;

}

public ConditionDTO convertConditionToDTO(Condition condition) {

return new ConditionDTO(condition.getId(), condition.getText(), condition.getWeatherId());

}

public Weather convertToEntity(WeatherDTO weatherDTO) {

Weather weather = new Weather();

weather.setId(weatherDTO.getId());

weather.setCity(weatherDTO.getCity());

weather.setTemperature(weatherDTO.getTemperature());

return weather;

}

}

Файл application.properties содержащий реализацию подключения базы данных к проекту

spring:

spring.datasource.url=jdbc:postgresql://localhost:5432/weather\_db

spring.datasource.username=postgres

spring.datasource.password=1102

spring.datasource.driver-class-name=org.postgresql.Driver

spring.jpa.hibernate.ddl-auto=update

**4. Результат программы**

При GET запросе http://localhost:8080/conditions/1  
Ответ (информация о condition по id weather):

[

{

"id": 5,

"text": "Солнечно",

"weatherId": 2

}

]

**5. Заключение**

В результате работы была подключена база данных PostgreSQL 16 и реализованы основные операции CRUD (Create, Read, Update, Delete) для работы с данными.