Министерство образования Республики Беларусь

Учреждение образования

«Белорусский государственный университет информатики и радиоэлектроники»

Кафедра электронных вычислительных машин

Лабораторная работа №4

по дисциплине «Программирование на языках высокого уровня»

«Error logging/handling»

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

гр.250501

Минск 2024

**1. Постановка задачи**

1. Обработать 400 и 500 ошибки.

2. Добавить логирование действий и ошибок (аспекты).

3. Подключить Swagger & CheckStyle. Убрать стилистические ошибки.

**2. Структура проекта**

В проекте используется послойная архитектура из нескольких пакетов,

которые отвечают за определенные функции.

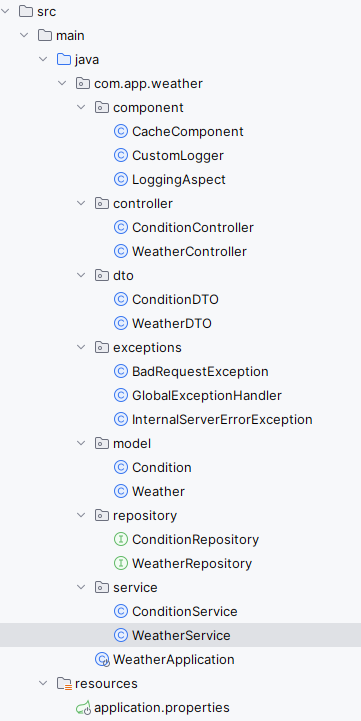


Рисунок 2.1 – Структура проекта

**3. Листинг кода**

Файл CacheComponent.java

package com.app.weather.component;

import org.springframework.stereotype.Component;

import java.util.HashMap;

import java.util.Map;

@Component

public class CacheComponent {

private final Map<String, Object> hashMap = new HashMap<>();

private final CustomLogger customLogger;

public CacheComponent(CustomLogger customLogger) {

this.customLogger = customLogger;

}

public void put(String key, Object value) {

hashMap.put(key, value);

int maxSize = 100;

if (hashMap.size() > maxSize) {

String oldestKey = hashMap.keySet().iterator().next();

hashMap.remove(oldestKey);

}

customLogger.cachePut();

}

public Object get(String key) {

return hashMap.get(key);

}

public void remove(String key) {

hashMap.remove(key);

customLogger.cacheRemove();

}

}

Файл CustomLogger.java

package com.app.weather.component;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.stereotype.Component;

@Component

public class CustomLogger {

private static final Logger LOGGER = LoggerFactory.getLogger(CustomLogger.class);

public void error(String text) {

LOGGER.error(text);

}

public void info(String text) {

LOGGER.info(text);

}

public void cachePut() {

LOGGER.info("Кэш обновлен (put)");

}

public void cacheRemove() {

LOGGER.info("Кэш обновлен (remove)");

}

}

Файл LoggingAspect.java

package com.app.weather.component;

import org.aspectj.lang.JoinPoint;

import org.aspectj.lang.annotation.AfterThrowing;

import org.aspectj.lang.annotation.Aspect;

import org.aspectj.lang.annotation.Before;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.stereotype.Component;

@Aspect

@Component

public class LoggingAspect {

private static final Logger LOGGER = LoggerFactory.getLogger(LoggingAspect.class);

@Before("execution(\* com.app.weather.controller..\*.\*(..))")

public void logBefore(JoinPoint joinPoint) {

String methodName = joinPoint.getSignature().getName();

LOGGER.info("Начало выполнения метода: {}", methodName);

}

@AfterThrowing(pointcut = "execution(\* com.app.weather.controller..\*.\*(..))", throwing = "exception")

public void logAfterThrowing(JoinPoint joinPoint, Throwable exception) {

String methodName = joinPoint.getSignature().getName();

LOGGER.error("Ошибка в методе: {} с сообщением: {}", methodName, exception.getMessage());

}

}

Файл ConditionController.java

package com.app.weather.controller;

import com.app.weather.component.CustomLogger;

import com.app.weather.dto.ConditionDTO;

import com.app.weather.model.Condition;

import com.app.weather.service.ConditionService;

import org.springframework.http.ResponseEntity;

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

import java.util.List;

@RestController

@RequestMapping("/conditions")

public class ConditionController {

private final ConditionService conditionService;

private final CustomLogger customLogger;

public ConditionController(ConditionService conditionService, CustomLogger customLogger) {

this.conditionService = conditionService;

this.customLogger = customLogger;

}

@GetMapping

public ResponseEntity<List<ConditionDTO>> getAllConditions() {

customLogger.info("Получение всех condition");

List<Condition> conditions = conditionService.getAllConditions();

List<ConditionDTO> conditionDTOs = conditions.stream()

.map(conditionService::convertToDTO)

.toList();

return ResponseEntity.ok(conditionDTOs);

}

@GetMapping("/{id}")

public ResponseEntity<ConditionDTO> getConditionById(@PathVariable Long id) {

customLogger.info("Получение condition по id: " + id);

Condition condition = conditionService.getConditionById(id);

if (condition == null) {

return ResponseEntity.notFound().build();

}

return ResponseEntity.ok(conditionService.convertToDTO(condition));

}

@PostMapping

public ResponseEntity<ConditionDTO> createCondition(@RequestBody ConditionDTO conditionDTO) {

customLogger.info("Создание условия");

Condition condition = conditionService.convertToEntity(conditionDTO);

Condition savedCondition = conditionService.createCondition(condition);

return ResponseEntity.ok(conditionService.convertToDTO(savedCondition));

}

@PutMapping("/{id}")

public ResponseEntity<ConditionDTO> updateCondition(@PathVariable Long id, @RequestBody ConditionDTO conditionDTO) {

customLogger.info("Обновление condition с id: " + id);

Condition updatedCondition = conditionService.updateCondition(id, conditionDTO);

if (updatedCondition == null) {

return ResponseEntity.notFound().build();

}

return ResponseEntity.ok(conditionService.convertToDTO(updatedCondition));

}

@DeleteMapping("/{id}")

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

customLogger.info("Удаление condition с id: " + id);

if (!conditionService.deleteCondition(id)) {

return ResponseEntity.notFound().build();

}

return ResponseEntity.noContent().build();

}

}

Файл WeatherController.java

package com.app.weather.controller;

import com.app.weather.component.CustomLogger;

import com.app.weather.dto.WeatherDTO;

import com.app.weather.model.Weather;

import com.app.weather.service.WeatherService;

import org.springframework.http.ResponseEntity;

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

import java.util.List;

@RestController

@RequestMapping("/weather")

public class WeatherController {

private final WeatherService weatherService;

private final CustomLogger customLogger;

public WeatherController(WeatherService weatherService, CustomLogger customLogger) {

this.weatherService = weatherService;

this.customLogger = customLogger;

}

@GetMapping

public ResponseEntity<List<WeatherDTO>> getAllWeathers() {

customLogger.info("Получение всех weather");

List<Weather> weathers = weatherService.getAllWeathers();

List<WeatherDTO> weatherDTOs = weathers.stream()

.map(weatherService::convertToDTO)

.toList();

return ResponseEntity.ok(weatherDTOs);

}

@GetMapping("/{id}")

public ResponseEntity<WeatherDTO> getWeatherById(@PathVariable Long id) {

customLogger.info("Получение weather по id: " + id);

Weather weather = weatherService.getWeatherById(id);

if (weather == null) {

return ResponseEntity.notFound().build();

}

return ResponseEntity.ok(weatherService.convertToDTO(weather));

}

//Query

@GetMapping("/citiesT/{temperature}")

public ResponseEntity<List<WeatherDTO>> getWeatherByTemperature(@PathVariable double temperature) {

customLogger.info("Получение weather по температуре: " + temperature);

List<WeatherDTO> weatherDTOs = weatherService.findByTemperature(temperature);

return ResponseEntity.ok(weatherDTOs);

}

@GetMapping("/citiesC/{conditionText}")

public ResponseEntity<List<WeatherDTO>> findByConditionText(@PathVariable String conditionText) {

customLogger.info("Получение condition по тексту условия: " + conditionText);

List<WeatherDTO> weathers = weatherService.findByConditionText(conditionText);

return ResponseEntity.ok(weathers);

}

@PostMapping

public ResponseEntity<WeatherDTO> createWeatherWithConditionText(@RequestBody WeatherDTO weatherDTO) {

customLogger.info("Создание weather с condition");

Weather createdWeather = weatherService.createWeatherWithCondition(weatherDTO);

return ResponseEntity.ok(weatherService.convertToDTO(createdWeather));

}

@PutMapping("/{id}")

public ResponseEntity<WeatherDTO> updateWeather(@PathVariable Long id, @RequestBody WeatherDTO weatherDTO) {

customLogger.info("Обновление weather с id: " + id);

Weather weather = weatherService.updateWeather(id, weatherDTO);

if (weather == null) {

return ResponseEntity.notFound().build();

}

return ResponseEntity.ok(weatherService.convertToDTO(weather));

}

@DeleteMapping("/{id}")

public ResponseEntity<Void> deleteWeather(@PathVariable Long id) {

customLogger.info("Удаление weather с id: " + id);

weatherService.deleteWeather(id);

return ResponseEntity.noContent().build();

}

}

Файл ConditionDTO.java

package com.app.weather.dto;

import lombok.\*;

@Getter

@Setter

@AllArgsConstructor

@NoArgsConstructor

public class ConditionDTO {

private Long id;

private String text;

}

Файл ConditionDTO.java

package com.app.weather.dto;

import lombok.\*;

import java.sql.Timestamp;

@Getter

@Setter

@AllArgsConstructor

@NoArgsConstructor

public class WeatherDTO {

private Long id;

private String city;

private Timestamp date;

private double temperature;

private ConditionDTO condition;

}

Файл BadRequestException.java

package com.app.weather.exceptions;

public class BadRequestException extends RuntimeException {

public BadRequestException(String message) {

super(message);

}

}

Файл GlobalExceptionHandler.java

package com.app.weather.exceptions;

import com.app.weather.component.CustomLogger;

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

import org.springframework.web.bind.annotation.ControllerAdvice;

import org.springframework.web.bind.annotation.ExceptionHandler;

@ControllerAdvice

public class GlobalExceptionHandler {

private final CustomLogger customLogger;

public GlobalExceptionHandler(CustomLogger customLogger) {

this.customLogger = customLogger;

}

@ExceptionHandler(BadRequestException.class)

public ResponseEntity<String> handleBadRequestException(BadRequestException e) {

customLogger.error(e.getMessage());

return new ResponseEntity<>(e.getMessage(), HttpStatus.BAD\_REQUEST);

}

@ExceptionHandler(InternalServerErrorException.class)

public ResponseEntity<String> handleInternalServerErrorException(InternalServerErrorException e) {

customLogger.error(e.getMessage());

return new ResponseEntity<>(e.getMessage(), HttpStatus.INTERNAL\_SERVER\_ERROR);

}

}

Файл InternalServerErrorException.java

package com.app.weather.exceptions;

public class InternalServerErrorException extends RuntimeException {

public InternalServerErrorException(String message) {

super(message);

}

}

Файл Condition.java

package com.app.weather.model;

import jakarta.persistence.\*;

import lombok.\*;

import java.util.ArrayList;

import java.util.List;

@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;

// Двунаправленная связь One-to-Many

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

private List<Weather> weathers = new ArrayList<>();

public void addWeather(Weather weather) {

weathers.add(weather);

weather.setCondition(this);

}

}

Файл Weather.java

package com.app.weather.model;

import jakarta.persistence.\*;

import lombok.\*;

import java.sql.Timestamp;

@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;

// Двунаправленная связь Many-to-One

@ManyToOne

@JoinColumn(name = "condition\_id")

private Condition condition;

}

Файл ConditionRepository.java

package com.app.weather.repository;

import com.app.weather.model.Condition;

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

import org.springframework.stereotype.Repository;

@Repository

public interface ConditionRepository extends JpaRepository<Condition, Long> {

Condition findByText(String text);

boolean existsByText(String text);

boolean existsByTextAndIdNot(String text, Long id);

}

Файл WeatherRepository.java

package com.app.weather.repository;

import com.app.weather.model.Weather;

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

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

import org.springframework.data.repository.query.Param;

import org.springframework.stereotype.Repository;

import java.util.List;

@Repository

public interface WeatherRepository extends JpaRepository<Weather, Long> {

Weather findByCity(String city);

boolean existsByCity(String city);

boolean existsByCityAndIdNot(String city, Long id);

//JPQL

@Query("SELECT w FROM Weather w WHERE w.temperature = :temperature")

List<Weather> findByTemperature(@Param("temperature") double temperature);

@Query("SELECT w FROM Weather w JOIN w.condition c WHERE c.text = :conditionText")

List<Weather> findByConditionText(@Param("conditionText") String conditionText);

}

Файл ConditionService.java

package com.app.weather.service;

import com.app.weather.component.CacheComponent;

import com.app.weather.component.CustomLogger;

import com.app.weather.dto.ConditionDTO;

import com.app.weather.exceptions.BadRequestException;

import com.app.weather.exceptions.InternalServerErrorException;

import com.app.weather.model.Condition;

import com.app.weather.repository.ConditionRepository;

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

import org.springframework.context.ApplicationContext;

import org.springframework.stereotype.Service;

import org.springframework.transaction.annotation.Transactional;

import java.util.List;

@Service

public class ConditionService {

private static final String notFoundMsg = "Condition not found";

private final ConditionRepository conditionRepository;

private final CacheComponent cache;

private final CustomLogger customLogger;

private String cacheKey;

private final ApplicationContext applicationContext;

@Autowired

public ConditionService(ConditionRepository conditionRepository, CacheComponent cache, CustomLogger customLogger, ApplicationContext applicationContext) {

this.conditionRepository = conditionRepository;

this.cache = cache;

this.customLogger = customLogger;

this.applicationContext = applicationContext;

}

@Transactional

public Condition createCondition(Condition condition) {

customLogger.info("Creating condition");

if (conditionRepository.existsByText(condition.getText())) {

throw new BadRequestException("Condition with this text already exists");

}

Condition savedCondition = conditionRepository.save(condition);

cacheKey = savedCondition.getId().toString();

cache.put(cacheKey, savedCondition);

return savedCondition;

}

@Transactional

public Condition updateCondition(Long id, ConditionDTO conditionDTO) {

customLogger.info("Updating condition with id: " + id);

Condition existingCondition = getConditionService().getConditionById(id);

if (existingCondition == null) {

throw new BadRequestException(notFoundMsg);

}

if (conditionRepository.existsByTextAndIdNot(conditionDTO.getText(), id)) {

throw new BadRequestException("Condition with this text already exists");

}

existingCondition.setText(conditionDTO.getText());

Condition savedCondition = conditionRepository.save(existingCondition);

cacheKey = savedCondition.getId().toString();

cache.put(cacheKey, savedCondition);

return savedCondition;

}

@Transactional

public boolean deleteCondition(Long id) {

customLogger.info("Deleting condition with id: " + id);

if (!conditionRepository.existsById(id)) {

throw new BadRequestException(notFoundMsg);

}

conditionRepository.deleteById(id);

cacheKey = id.toString();

cache.remove(cacheKey);

return true;

}

@Transactional

public Condition getConditionById(Long id) {

customLogger.info("Getting condition by id: {}" + id);

try {

cacheKey = id.toString();

Condition condition = (Condition) cache.get(cacheKey);

if (condition != null) {

return condition;

}

condition = conditionRepository.findById(id).orElse(null);

if (condition != null) {

cache.put(cacheKey, condition);

} else {

throw new BadRequestException(notFoundMsg);

}

return condition;

} catch (Exception e) {

throw new InternalServerErrorException("Failed to get condition by id");

}

}

@Transactional

public List<Condition> getAllConditions() {

customLogger.info("Getting all conditions");

try {

List<Condition> conditions = conditionRepository.findAll();

conditions.forEach(condition -> {

cacheKey = condition.getId().toString();

cache.put(cacheKey, condition);

});

return conditions;

} catch (Exception e) {

throw new InternalServerErrorException("Failed to get all conditions");

}

}

public Condition convertToEntity(ConditionDTO conditionDTO) {

customLogger.info("Converting ConditionDTO to Condition");

Condition condition = new Condition();

condition.setText(conditionDTO.getText());

return condition;

}

public ConditionDTO convertToDTO(Condition condition) {

customLogger.info("Converting Condition to ConditionDTO");

if (condition == null) {

return null;

}

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

}

@Transactional

public Condition getConditionByText(String text) {

customLogger.info("Getting condition by text: " + text);

try {

Condition condition = conditionRepository.findByText(text);

if (condition != null) {

cacheKey = condition.getId().toString();

cache.put(cacheKey, condition);

}

return condition;

} catch (Exception e) {

throw new InternalServerErrorException("Failed to get condition by text");

}

}

private ConditionService getConditionService() {

return applicationContext.getBean(ConditionService.class);

}

}}

Файл WeatherService.java

package com.app.weather.service;

import com.app.weather.component.CacheComponent;

import com.app.weather.component.CustomLogger;

import com.app.weather.dto.ConditionDTO;

import com.app.weather.dto.WeatherDTO;

import com.app.weather.exceptions.BadRequestException;

import com.app.weather.exceptions.InternalServerErrorException;

import com.app.weather.model.Condition;

import com.app.weather.model.Weather;

import com.app.weather.repository.WeatherRepository;

import org.springframework.context.ApplicationContext;

import java.sql.Timestamp;

import java.util.List;

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

import org.springframework.stereotype.Service;

import org.springframework.transaction.annotation.Transactional;

@Service

public class WeatherService {

private static final String NOT\_FOUND\_MSG = "Weather not found";

private final WeatherRepository weatherRepository;

private final ConditionService conditionService;

private final CacheComponent cache;

private final CustomLogger customLogger;

private String cacheKey;

private final ApplicationContext applicationContext;

@Autowired

public WeatherService(WeatherRepository weatherRepository, ConditionService conditionService, CacheComponent cache, CustomLogger customLogger, ApplicationContext applicationContext) {

this.weatherRepository = weatherRepository;

this.conditionService = conditionService;

this.cache = cache;

this.customLogger = customLogger;

this.applicationContext = applicationContext;

}

@Transactional

public Weather createWeatherWithCondition(WeatherDTO weatherDTO) {

customLogger.info("Creating weather with condition");

if (weatherRepository.existsByCity(weatherDTO.getCity())) {

throw new BadRequestException("Weather for this city already exists");

}

try {

Weather weather = convertToEntity(weatherDTO);

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

cacheKey = weather.getCity();

// Проверяем, существует ли уже погода для этого города

Weather existingWeather = (Weather) cache.get(cacheKey);

if (existingWeather != null) {

return existingWeather;

}

Weather weatherFromDb = weatherRepository.findByCity(weather.getCity());

if (weatherFromDb != null) {

cache.put(cacheKey, weatherFromDb);

return weatherFromDb;

}

// Проверяем, существует ли условие, если нет, то создаем его

Condition condition = conditionService.getConditionByText(weatherDTO.getCondition().getText());

if (condition == null) {

condition = conditionService.convertToEntity(weatherDTO.getCondition());

condition = conditionService.createCondition(condition); // сохраняем объект Condition в базе данных

}

// Устанавливаем связь между погодой и условием

weather.setCondition(condition);

condition.addWeather(weather);

// Создаем новую погоду

Weather savedWeather = weatherRepository.save(weather);

cache.put(cacheKey, savedWeather);

return savedWeather;

} catch (Exception e) {

throw new InternalServerErrorException("Failed to create weather with condition");

}

}

@Transactional

public Weather updateWeather(Long id, WeatherDTO weatherDTO) {

customLogger.info("Updating weather with id: " + id);

Weather existingWeather = getWeatherService().getWeatherById(id);

if (existingWeather == null) {

throw new BadRequestException(NOT\_FOUND\_MSG);

}

if (weatherRepository.existsByCityAndIdNot(weatherDTO.getCity(), id)) {

throw new BadRequestException("Weather for this city already exists");

}

try {

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

existingWeather.setTemperature(weatherDTO.getTemperature());

cacheKey = existingWeather.getCity();

// Проверяем, существует ли уже погода для этого города

Weather weatherByCity = weatherRepository.findByCity(weatherDTO.getCity());

if (weatherByCity != null && !weatherByCity.getId().equals(id)) {

return weatherByCity;

}

// Проверяем, существует ли условие, если нет, то создаем его

Condition condition = conditionService.getConditionByText(weatherDTO.getCondition().getText());

if (condition == null) {

condition = conditionService.convertToEntity(weatherDTO.getCondition());

}

existingWeather.setCondition(condition);

Weather savedWeather = weatherRepository.save(existingWeather);

cache.put(cacheKey, savedWeather);

return savedWeather;

} catch (Exception e) {

throw new InternalServerErrorException("Failed to update weather");

}

}

@Transactional

public void deleteWeather(Long id) {

customLogger.info("Deleting weather with id: {}" + id);

Weather weather = getWeatherService().getWeatherById(id);

if (weather == null) {

throw new BadRequestException(NOT\_FOUND\_MSG);

}

weatherRepository.delete(weather);

cacheKey = weather.getCity();

cache.remove(cacheKey);

}

@Transactional

public Weather getWeatherById(Long id) {

customLogger.info("Getting weather by id: " + id);

try {

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

if (weather == null) {

throw new BadRequestException(NOT\_FOUND\_MSG);

}

cacheKey = weather.getCity();

cache.put(cacheKey, weather);

return weather;

} catch (Exception e) {

throw new InternalServerErrorException("Failed to get weather by id");

}

}

@Transactional

public List<Weather> getAllWeathers() {

customLogger.info("Getting all weathers");

try {

List<Weather> weathers = weatherRepository.findAll();

weathers.forEach(weather -> {

cacheKey = weather.getCity();

cache.put(cacheKey, weather);

});

return weathers;

} catch (Exception e) {

throw new InternalServerErrorException("Failed to get all weathers");

}

}

public WeatherDTO convertToDTO(Weather weather) {

customLogger.info("Converting Weather to WeatherDTO");

WeatherDTO dto = new WeatherDTO();

dto.setId(weather.getId());

dto.setCity(weather.getCity());

dto.setDate(weather.getDate());

dto.setTemperature(weather.getTemperature());

ConditionDTO conditionDTO = conditionService.convertToDTO(weather.getCondition());

dto.setCondition(conditionDTO);

return dto;

}

private Weather convertToEntity(WeatherDTO weatherDTO) {

customLogger.info("Converting WeatherDTO to Weather");

Weather weather = new Weather();

weather.setCity(weatherDTO.getCity());

weather.setDate(weatherDTO.getDate());

weather.setTemperature(weatherDTO.getTemperature());

// Создаем объект Condition на основе conditionText

Condition condition = new Condition();

condition.setText(weatherDTO.getCondition().getText());

weather.setCondition(condition);

return weather;

}

public List<WeatherDTO> findByTemperature(double temperature) {

customLogger.info("Getting weathers by temperature: " + temperature);

try {

List<Weather> weathers = weatherRepository.findByTemperature(temperature);

return weathers.stream()

.map(this::convertToDTO)

.toList();

} catch (Exception e) {

throw new InternalServerErrorException("Failed to find weathers by temperature");

}

}

public List<WeatherDTO> findByConditionText(String conditionText) {

customLogger.info("Getting weathers by condition text: " + conditionText);

try {

List<Weather> weathers = weatherRepository.findByConditionText(conditionText);

return weathers.stream()

.map(this::convertToDTO)

.toList();

} catch (Exception e) {

throw new InternalServerErrorException("Failed to find weathers by condition text");

}

}

private WeatherService getWeatherService() {

return applicationContext.getBean(WeatherService.class);

}

}

**4. Результат программы**

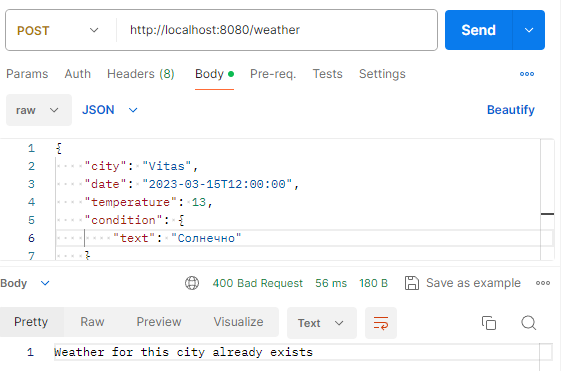


Рисунок 1.1 – обработка ошибки 400

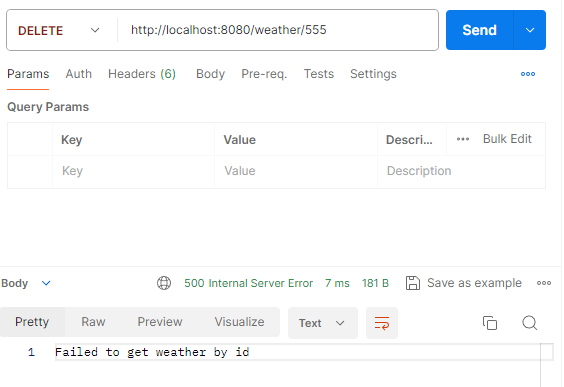


Рисунок 1.2 – обработка ошибки 500

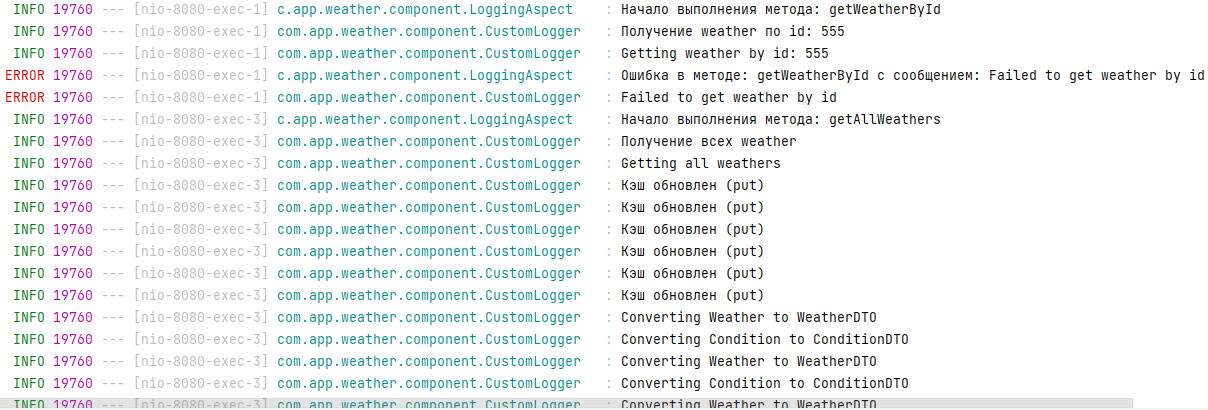


Рисунок 1.3 – логирование действий и ошибок

**5. Заключение**

В результате работы обработаны ошибки 400 и 500, добавлены соответствующие обработчики в контроллер и настроены сообщения об ошибках. Добавлено логирование действий и ошибок с использованием аспектов. Для этого были созданы соответствующие аспекты и настроены необходимые параметры логирования. Подключен Swagger для документирования API. Были устранены стилистические ошибки с помощью CheckStyle. Для этого были настроены необходимые параметры в файле конфигурации CheckStyle и устранены все обнаруженные ошибки.