Tratamento de erros

**📂 Estrutura do Projeto (Atualizada)**

src/main/java/com/supermarket/promows/

├── config/

├── controller/

├── dto/

├── exception/ ← Vamos trabalhar aqui primeiro

├── model/

├── repository/

└── service/

**Passo 1: Crie as Exceções Customizadas (exception/)**

**1.1 DepartmentAlreadyExistsException.java**

package com.supermarket.promows.exception;

public class DepartmentAlreadyExistsException extends RuntimeException {

public DepartmentAlreadyExistsException(String departmentName) {

super("Já existe um departamento com o nome: " + departmentName);

}

}

**1.2 DepartmentNotFoundException.java**

package com.supermarket.promows.exception;

public class DepartmentNotFoundException extends RuntimeException {

public DepartmentNotFoundException(Long id) {

super("Departamento não encontrado com ID: " + id);

}

}

**1.3 ErrorResponse.java (Modelo de resposta de erro)**

package com.supermarket.promows.exception;

import lombok.AllArgsConstructor;

import lombok.Getter;

import lombok.Setter;

import java.time.LocalDateTime;

@Getter

@Setter

@AllArgsConstructor

public class ErrorResponse {

private int status;

private String message;

private LocalDateTime timestamp;

private String path; // Opcional: adicione se quiser mostrar o endpoint

}

**Passo 2: Implemente o GlobalExceptionHandler (exception/)**

**2.1 GlobalExceptionHandler.java**

package com.supermarket.promows.exception;

import jakarta.servlet.http.HttpServletRequest;

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

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

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

import java.time.LocalDateTime;

@ControllerAdvice

public class GlobalExceptionHandler {

@ExceptionHandler(DepartmentAlreadyExistsException.class)

public ResponseEntity<ErrorResponse> handleDepartmentAlreadyExists(

DepartmentAlreadyExistsException ex,

HttpServletRequest request) {

ErrorResponse error = new ErrorResponse(

HttpStatus.CONFLICT.value(),

ex.getMessage(),

LocalDateTime.now(),

request.getRequestURI()

);

return new ResponseEntity<>(error, HttpStatus.CONFLICT);

}

@ExceptionHandler(DepartmentNotFoundException.class)

public ResponseEntity<ErrorResponse> handleDepartmentNotFound(

DepartmentNotFoundException ex,

HttpServletRequest request) {

ErrorResponse error = new ErrorResponse(

HttpStatus.NOT\_FOUND.value(),

ex.getMessage(),

LocalDateTime.now(),

request.getRequestURI()

);

return new ResponseEntity<>(error, HttpStatus.NOT\_FOUND);

}

// Adicione outros handlers conforme necessário

}

**Passo 3: Atualize o DepartmentService (service/)**

**3.1 Modifique seu DepartmentService.java**

package com.supermarket.promows.service;

import com.supermarket.promows.dto.DepartmentDTO;

import com.supermarket.promows.exception.DepartmentAlreadyExistsException;

import com.supermarket.promows.exception.DepartmentNotFoundException;

import com.supermarket.promows.model.Department;

import com.supermarket.promows.repository.DepartmentRepository;

import org.springframework.stereotype.Service;

import org.springframework.transaction.annotation.Transactional;

import java.util.Optional;

@Service

public class DepartmentService {

private final DepartmentRepository departmentRepository;

public DepartmentService(DepartmentRepository departmentRepository) {

this.departmentRepository = departmentRepository;

}

@Transactional

public Department createDepartment(DepartmentDTO departmentDTO) {

Optional<Department> existingDept = departmentRepository

.findByDepartmentName(departmentDTO.getDepartmentName());

if (existingDept.isPresent()) {

throw new DepartmentAlreadyExistsException(departmentDTO.getDepartmentName());

}

return departmentRepository.save(new Department(departmentDTO));

}

public Department getDepartmentById(Long id) {

return departmentRepository.findById(id)

.orElseThrow(() -> new DepartmentNotFoundException(id));

}

// Atualize outros métodos para usar as exceções customizadas

}

**Passo 4: Atualize o DepartmentController (controller/)**

**4.1 Modifique seu DepartmentController.java**

package com.supermarket.promows.controller;

import com.supermarket.promows.dto.DepartmentDTO;

import com.supermarket.promows.model.Department;

import com.supermarket.promows.service.DepartmentService;

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

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

import java.util.List;

@RestController

@RequestMapping("/api/departments")

public class DepartmentController {

private final DepartmentService departmentService;

public DepartmentController(DepartmentService departmentService) {

this.departmentService = departmentService;

}

@PostMapping

public ResponseEntity<Department> createDepartment(@RequestBody DepartmentDTO departmentDTO) {

Department createdDepartment = departmentService.createDepartment(departmentDTO);

return new ResponseEntity<>(createdDepartment, HttpStatus.CREATED);

}

@GetMapping("/{id}")

public ResponseEntity<Department> getDepartmentById(@PathVariable Long id) {

Department department = departmentService.getDepartmentById(id);

return ResponseEntity.ok(department);

}

// Outros endpoints podem permanecer iguais

}

**Passo 5: Teste a Implementação**

Casos de Teste:

Criar departamento duplicado:

Deve retornar HTTP 409 com a mensagem de erro.

Buscar departamento inexistente:

Deve retornar HTTP 404 com a mensagem de erro.

Erro genérico não tratado:

O Spring já retornará HTTP 500 com o formato padronizado.

**Passo 6: Adicione Validações (Opcional)**

**6.1 No DepartmentDTO (dto/)**

package com.supermarket.promows.dto;

import jakarta.validation.constraints.NotBlank;

import lombok.Getter;

import lombok.Setter;

@Getter

@Setter

public class DepartmentDTO {

@NotBlank(message = "O nome do departamento é obrigatório")

private String departmentName;

}

**6.2 Atualize o GlobalExceptionHandler**

@ExceptionHandler(MethodArgumentNotValidException.class)

public ResponseEntity<ErrorResponse> handleValidationErrors(

MethodArgumentNotValidException ex,

HttpServletRequest request) {

List<String> errors = ex.getBindingResult()

.getFieldErrors()

.stream()

.map(error -> error.getField() + ": " + error.getDefaultMessage())

.collect(Collectors.toList());

ErrorResponse error = new ErrorResponse(

HttpStatus.BAD\_REQUEST.value(),

"Erro de validação",

LocalDateTime.now(),

request.getRequestURI()

);

error.setErrors(errors); // Adicione este campo no ErrorResponse

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

}

✅ Pronto! Seu tratamento de erros está completo.

Agora você tem:

Exceções específicas para cada situação

Respostas padronizadas em JSON

Códigos HTTP semânticos

Fácil expansão para novas exceções

Quer que eu mostre como implementar para outra classe (como Product ou Promotion)? Ou precisa de algum ajuste específico?