# Centralized Exception Handling Middleware - Design Document

## Overview

This document details the design of a centralized exception handling middleware for an ASP.NET Core application. The middleware intercepts exceptions during request processing, categorizes them, applies retry policies for system exceptions, logs structured information, and returns standardized error responses to clients.

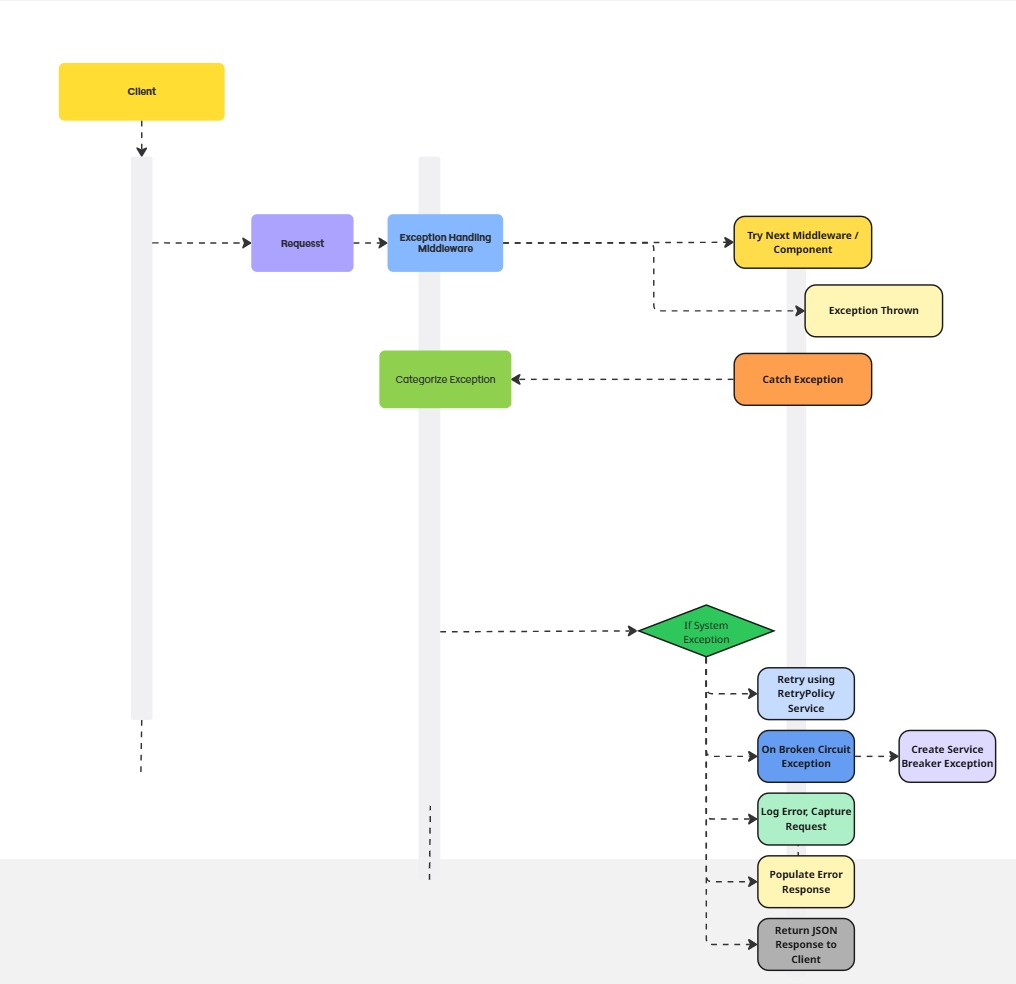
### Architecture Diagram

**A diagram of a company

AI-generated content may be incorrect.**

### Middleware Flow (Sequence Diagram)

#### Sequence on Exception:

****

### Component Responsibilities

#### ExceptionHandlingMiddleware

* Captures all unhandled exceptions in the request pipeline.
* Differentiates between custom application exceptions and unexpected exceptions.
* Applies retry and circuit breaker policies on SystemException.
* Logs structured error data and publishes metrics.
* Returns an ErrorResponse object with contextual data.

#### RetryPolicyService

* Defines retry behavior using exponential backoff.
* Uses Polly to implement retry and circuit breaker patterns.
* Retry 5 times with exponential delay (50ms to 800ms).
* Circuit breaker opens after 5 failures and stays open for 500ms.
* Supports both Func<Task> and Func<Task<T>>.

#### MetricsService

* Tracks error count by total, category, and error code.
* Logs structured metrics per error event with correlation ID.
* Maintains timestamped queues for error rate calculation.
* Periodically cleans up old error timestamps.
* Provides summaries including top error codes and category breakdown.

#### ErrorResponse

* Encapsulates all client-facing error data: code, message, suggestions, documentation URL, etc.

#### RequestContext

* Extracts and stores metadata about the HTTP request (IP, method, path, headers, user).

### Exception Classification

| **Exception Type** | **Code Range** | **Description** |
| --- | --- | --- |
| ConfigException | 1000-1999 | Configuration-related issues |
| DataException | 2000-2999 | Database and persistence errors |
| LogicalException | 3000-3999 | Application logic bugs |
| SystemException | 4000-4999 | Infrastructure/system failures |
| CircuitBreakerException | 4500 | Circuit breaker triggered |

### Retry and Circuit Breaker Logic

* Retry applied to SystemException via RetryPolicyService.
* Uses Polly's exponential backoff policy.
* Retry delay grows: 50ms, 100ms, 200ms, 400ms, 800ms.
* After 5 failures, BrokenCircuitException is thrown.
* Circuit breaker remains open for 500ms and logs open/close transitions.
* After 500ms, it enters a half-open state where one trial request is allowed:
  + If the trial succeeds, the circuit resets to closed (normal operation).
  + If the trial fails, the circuit returns to open state, restarting the break duration timer.

### Logging and Observability

* Structured logging using ILogger.BeginScope with Correlation ID.
* Includes context like HTTP method, path, user, IP, error category/code.
* MetricsService used to increment counters and log summaries.

#### Error Response Format

{

"correlationId": "abc123",

"errorCode": 4001,

"errorCategory": "System Issue",

"httpStatusCode": 500,

"message": "System resource unavailable",

"detailedMessage": "System resource unavailable (Error Code: 4001)",

"timestamp": "2025-05-02T20:15:30Z",

"documentationUrl": "https://docs.example.com/errors/system-issue/4001",

"suggestions": [

"The system is experiencing issues. Please try again later."

],

"requestContext": {

"correlationId": "abc123",

"httpMethod": "GET",

"path": "/api/test/system-error",

...

}

}

### Configuration

* Logging level configured in appsettings.Development.json:

{

"Logging": {

"LogLevel": {

"Default": "Information",

"Microsoft.AspNetCore": "Warning"

}

}

}

### Deployment Notes

* Add the middleware early in the pipeline (before UseRouting).
* Ensure RetryPolicyService and MetricsService are registered in DI.
* Ensure Polly library is included for retry/circuit-breaker support.

### Future Enhancements

* Plug into external monitoring tools (e.g., Application Insights).
* Support for localization of error messages.
* Auto-reporting to incident management systems.
* Rate limiting for repeat exception responses.
* Dashboard UI to visualize metrics from MetricsService.

### Conclusion

This middleware provides robust and extensible centralized error handling for .NET APIs, including observability, structured responses, and automatic retry logic for transient system failures.