Automating Standard Entries

Erik Dahl

PRINCIPAL ARCHITECT

@dahlsailrunner   knowyourtoolset.com

Overview

Exception Handling and Logging
- Special in development?
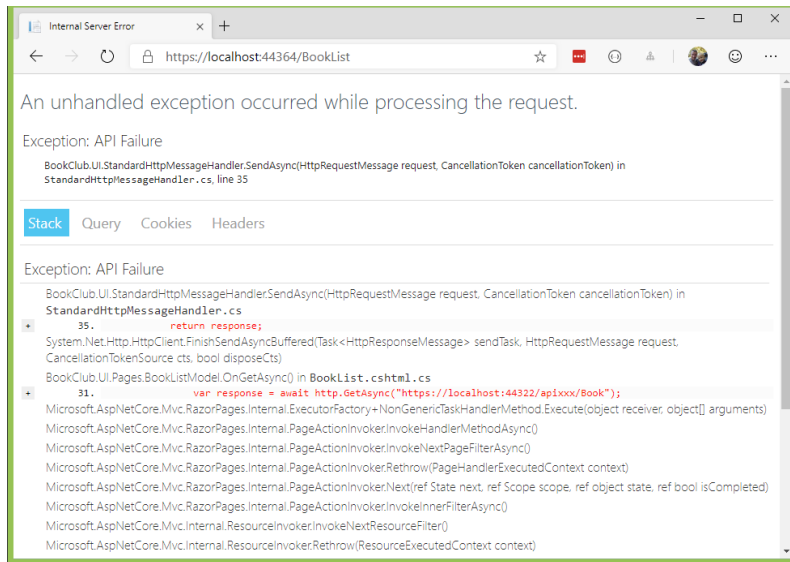- Global handling, shielding, and logging

Filters
- Can be applied more globally

Attributes and Base Classes
- Can be more selectively applied

Exception Handling

Do not use the DeveloperExceptionPage

- **Best**: Crutch in development; app is different in non-dev

- **Worst**: Expose sensitive information

Shield error details

- Provide some kind of "ID"

- Log all details (mask sensitive)

Global handling covers most scenarios

- Different between UI and API apps

Demo

Remove Developer Exception page in UI

Be aware of `Exception.ToString()`
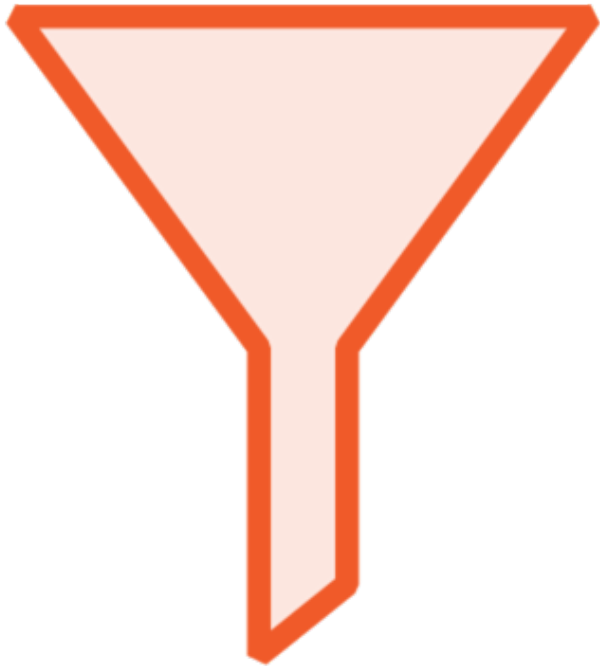
Update the Error page
- `RequestId`
- `IExceptionHandlerPathFeature`

Define custom middleware for API
- Return custom error response with ID
- Log all details
- Update UI to read/log error response

Filters

For Razor pages:

- `IPageFilter` and `IAsyncPageFilter`

For MVC controllers:

- `IActionFilter` and `IAsyncActionFilter`

In both cases:

- Don't implement both sync and async

- Sync versions contain before and after methods

Demo

Move API middleware to Infrastructure

Add Page Filter for UI
- Track performance of handler methods

Add Action Filter for API
- Track performance of API calls

Attributes and Base Classes

ActionFilterAttribute

- Has before / after methods
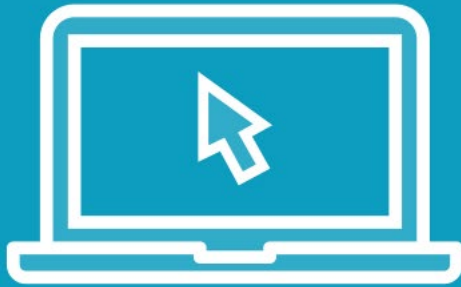
- Controllers and Actions

ResultFilterAttribute

- Only after "your code" has executed

- Controllers, Actions, Page Models

Custom base classes

- Override methods you want

- Apply as globally or as selectively as you want

Demo

Use attribute to track performance for API

- Apply to controllers or actions

Update BookList Page Model to override methods

- Create a base class to use more consistently

Summary

Global exception handling & logging

Affected global behavior via filters

Applied behavior selectively with attributes

Override filter methods in Razor pages
- Created base class

Next: Content in log entries