Unit Testing ASP.NET Core Middleware, Filters and Service Registrations



Kevin Dockx Architect

@KevinDockx https://www.kevindockx.com



Coming Up



Unit testing middleware

Unit testing ASP.NET Core filters

Unit testing service registrations

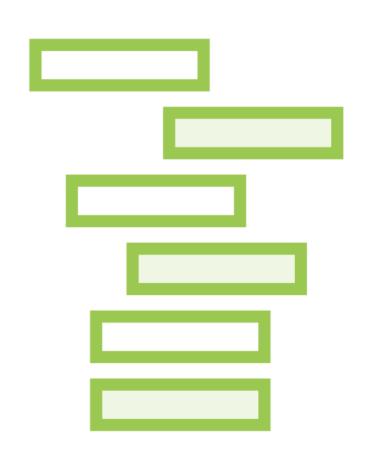


Unit Testing Middleware

Test custom middleware, not built-in middleware



Unit Testing Middleware







Mostly though, a unit test is advisable for middleware testing



Unit Testing Middleware

Typical concerns when unit testing middleware:

- Mock the HttpContext (or use DefaultHttpContext)
- Handle the RequestDelegate

Demo



Unit testing middleware



ASP.NET Core Filter

A filter allows code to run before or after specific stages in the request processing pipeline



Unit Testing ASP.NET Core Filters

Custom filters often handle cross-cutting concerns:

- Error handling
- Caching

Filters can be used to avoid code duplication



Unit Testing ASP.NET Core Filters

Filters run in the ASP.NET Core action invocation pipeline

Unit Testing ASP.NET Core Filters









Exception filter



Result filter

Unit Testing ASP.NET Core Filters

Action filters:

- Run immediately before and after an action method is called
- Can change the arguments passed into an action
- Can change the result returned from the action

Demo



Unit testing ASP.NET Core filters



Unit Testing Service Registrations

Services are registered on ASP.NET Core's included IoC container

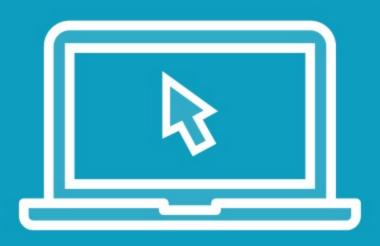
- These registrations can be unit tested

Unit Testing Service Registrations

Approach:

- Create an IServiceCollection
- Register the services on it
- Build an IServiceProvider
- Verify whether the services were registered

Demo



Unit testing service registrations

Summary



Challenges with middleware and filter testing are related to test isolation

Test service registrations by building an IServiceProvider and testing whether you can get a service instance

Up Next:

Integrating Unit Tests in Your Development and Release Flows