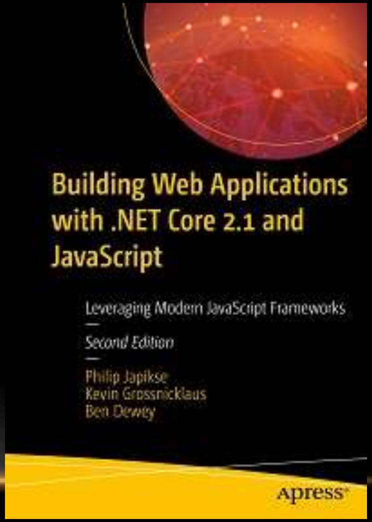
DIVING DEEP INTO ASP.NET CORE 3.X

Philip Japikse (@skimedic)
skimedic@outlook.com
www.skimedic.com/blog
Microsoft MVP, ASPInsider, MCSD, MCDBA, CSM, PSM II, PSD
Director of Consulting, Chief Architect, Author, Speaker

Phil.About()

- Director of Consulting/Chief Architect
- Author: Apress.com (http://bit.ly/apressbooks)
- Speaker: http://www.skimedic.com/blog/page/Abstracts.aspx
- Microsoft MVP, ASPInsider, MCSD, MCDBA, CSM, PSM II, PSD
- Founder, Agile Conferences, Inc.
    - http://www.cincydeliver.org
- President, Cincinnati .NET User's Group



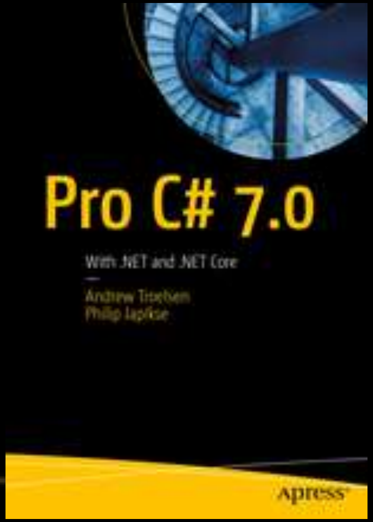
Building Web Applications with .NET Core 2.1 and JavaScript

Leveraging Modern JavaScript Frameworks

Second Edition

Philip Japikse
Kevin Grossnicklaus
Ben Dewey

Apress



Pro C# 7.0

With .NET and .NET Core

Andrew Troelsen
Philip Japikse

Apress

.NET CORE

WHAT IS .NET CORE?

- Rewrite of “full” .NET Framework
- Vast performance improvements over prior versions
  - Including native compilation
- Flexible deployment model
  - Windows, Linux, Mac
- Full command line support
- True side by side installation support
- Open source from the start
  - Many improvements and features provided by the community

DEPLOYMENT

- Deployment models
  - Self contained –includes .NET Core Runtime
  - Portable – expects .NET Core Runtime installed on deployment machine
- Kestrel adds a layer of complexity – see the docs

SUPPORT LIFECYCLES

.NET CORE SUPPORT LIFECYCLES

- Long Term Support (LTS)
  - Only upgraded with critical fixes (patches)
  - Supported for three years after GA release

  or

  - At least one year after the next LTS release.

  - NOTE:
    - 2.1 LTS (support until 8/21/21)
    - 3.1 LTS (support until ~12/03/22)
    - 6.0 will be declared LTS (~Q4/2021)

- Current (STS)
  - Minor releases
  - Upgraded more rapidly
  - Supported for three months after:
    - Next Current or LTS release

  - NOTE:
    - 1.0, 1.1, 2.0, 2.2, 3.0 all End of Lifed
    - 5.0 will be declared current (~Q4/2020)

https://www.microsoft.com/net/core/support

ASP.NET CORE FUNDAMENTALS

ASP.NET CORE

- ASP.NET Core is ASP.NET rebuilt on top of .NET Core
- Single, cross-platform framework for web, services, and microservices
  - WebApi + MVC + Web Pages + Razor Pages = ASP.NET Core
- Takes advantage of .NET Core performance
  - Includes a high performance web server (Kestrel) built on libUV

ASP.NET CORE FEATURES

- Pluggable Middleware - Routing, authentication, static files, etc.
- Full Dependency Injection integration
- Simplified and Improved Configuration System
- Tag Helpers
- View Components

NOTABLE UPDATES IN ASP.NET CORE 2.1

- SignalR
- Razor class libraries
  - Identity as a package or scaffolded
- HTTPS Improvements
  - dotnet dev-certs https --trust
  - On by default
  - Cleaner redirect
- Web API Improvements
  - Enhanced Controllers
- HTTP Client Factory
- Improvements for EU – GDPR
  - Hooks in Identity, cookies, encryption

NOTABLE UPDATES IN ASP.NET CORE 2.2

- Endpoint Routing
  - Interops with middleware better
  - Performance improvements
- Health checks service (Kubernetes)
- HTTP/2 in Kestrel
- IIS Inprocess Hosting
- SignalR Java Client
- Templates updated to Bootstrap 4 and Angular 6
- Performance improvements

NEW IN ASP.NET CORE 3.0/3.1

- Blazor
- gRPC
- HostBuilder replaces WebHostBuilder
- SignalR updates
- C#8
- Partial classes for Razor Components
- Blazor improvements

RUNNING .NET CORE APPLICATIONS

RUNNING ASP.NET CORE APPLICATIONS

- Visual Studio
  - Select IIS or Kestrel
  - Port is controlled by launchSetting.json
- .NET Core CLI
  - ‘dotnet run’
  - Port defaults to 5000/5001
    - Can be changed using HostBuilder

LAUNCHSETTINGS.JSON CONTROLS RUNNING APP

- IIS Settings
  - Sets app URL/SSL Port, auth settings
- Profiles (appear in VS Run command)
  - IIS Express
    - Sets environment variable
  - <AppName>
    - Sets URL, environment variable

CONFIGURING THE WEB SERVER(S)

ASP.NET CORE APPS ARE CONSOLE APPS

- Web server(s) is(are) created in Program Main() method

```
public static IHostBuilder CreateHostBuilder(string[] args) =>
            Host.CreateDefaultBuilder(args)
                .ConfigureWebHostDefaults(webBuilder =>
                {
                    webBuilder.UseStartup<Startup>();
                });
CreateHostBuilder(args).Build().Run();
```

- Configured in Startup.cs

THE STARTUP CLASS

APPLICATION STARTUP

- The Startup class configures services and application pipeline
- Constructor creates configuration builder, configures user secrets
- Configure sets up the HTTP request processing pipeline
- ConfigureServices configures services and the DI container

CONFIGURING THE PIPELINE

- The Configure method sets up how to respond to HTTP requests

```
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
      app.UseExceptionHandler("/Home/Error");
      app.UseStaticFiles();
      app.UseEndPoints(endpoints =>
       {
          endpoints.MapControllers();
          //endpoints.MapControllerRoute(
          //   name: "default",
          //   template: "{controller=Home}/{action=Index}/{id?}");
       });
}
```

CONDITIONAL PIPELINE CONFIGURATION

- Use environment options for conditional pipeline configuration

```
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
        if (env.IsDevelopment() || env.IsEnvironment("Local")
        {
                app.UseDeveloperExceptionPage();
        }
        else
        {
                app.UseExceptionHandler("/Home/Error");
                app.UseHsts();
        }
}
```

CONFIGURING FRAMEWORK SERVICES

- Used to configure any services needed by the application

```
public void ConfigureServices(IServiceCollection services)
{
   // Add framework services.
   services.AddControllersWithViews(config =>
   {
      config.Filters.Add(new SimpleAuthenticationActionFilter());
   })
   .AddJsonOptions(options =>
   {   //Revert to PascalCasing for JSON handling
       options.JsonSerializerOptions.PropertyNamingPolicy = null;
       options.JsonSerializerOptions.WriteIndented = true;
   });
   //Additional services for DI added here (covered later in this presentation)
}
```

CONFIGURING EF CORE CONTEXT POOLING

- New feature in ASP.NET/EF Core 2
- Context must have single public constructor that takes DbContextOptions

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddDbContextPool<StoreContext>(options =>
        options.UseSqlServer(Configuration.GetConnectionString("SpyStore")));
}
```

APPLICATION CONFIGURATION

ENVIRONMENTAL AWARENESS

- ASP.NET Core uses ASPNETCORE_ENVIRONMENT variable
  - Development, Staging, Production are built-in environments
- Environment is used throughout ASP.NET Core
  - Determining which configuration files to load
  - Execution paths based on the environment (using IWebHostEnvironment)
  - Environment Tag Helper
- Simplifies deployment and testing

APPLICATION CONFIGURATION

- Applications are configured using:
  - Simple JSON (or other file types)
  - Command line arguments
  - Environment variables
  - In memory .NET objects, Encrypted user store, Custom providers

- Configuration values are set in the order received
- User Secrets are loaded last
- Environment determines which additional file(s) to load
  - appsettings.<environment>.json

APPLICATION CONFIGURATION

- Custom classes can represent configuration values
  - Can bind to entire configuration or individual sections with services.Configure<T>
  - Can injected using DI (via IOptions[type]<T>)

DEPENDENCY INJECTION

BUILT-IN DEPENDENCY INJECTION

- Items added to the services container in Startup.cs
- Services are accessed through:
  - Constructor injection
  - Method injection (with [FromServices])
  - View injection (with @inject)
- Can also retrieve services through:
  - ApplicationServices (for non-controller classes)
  - HttpContext.RequestServices (for controllers)
  - Injection is the preferred mechanism

REGISTER CUSTOM SERVICES

- Custom services can be registered as well:
  - Transient: Created each time they are requested
  - Scoped: Created once per HTTP request
  - Singleton: Max of one instance per application

IHTTPCLIENTFACTORY

IHTTPCLIENTFACTORY

- ASP.NET Core 2.1 includes a new IHttpClientFactory service that makes it easier to configure and consume instances of HttpClient in apps.
- The factory:
  - Makes registering of instances of HttpClient per named client more intuitive.
  - Implements a Polly handler that allows Polly policies to be used for Retry, CircuitBreakers, etc.
  - Handles pooling and lifetime management of HttpClient

ROUTING

ROUTING

- Attribute Routing is first class citizen in ASP.NET Core
  - Helps to refine routing for individual controller actions
- Controller and actions can define specific routes
- If routing added to Controller, it’s inherited by Actions

- Note: Traditional routing still exists

CONTROLLERS

CONTROLLERS AND ACTIONS

- All derive from single Controller class (derived from ControllerBase)
  - Controller, AsyncController, APIController all rolled into one
- API non-HttpGet methods must specify HTTP Verb
  - No long based on name of method
- All return IActionResult (or Task<IActionResult>)

- Many helper methods built into base Controller for returning HttpStatusCodes
  - NoContent (204), OK (200), BadRequest (400), etc.

WEB API IMPROVEMENTS

- Inherit from ControllerBase
- Add ApiController Attribute
  - Enables REST-specific behavior for controllers
    - Automatic 400 responses on model validation errors
    - Binding source parameter inference
    - Multipart/form-data inference

MANAGING CLIENT SIDE LIBRARIES

MANAGING CLIENT SIDE LIBRARIES

- Library Manager built into VS 2017 15.8+ and VS2019
  - https://github.com/aspnet/LibraryManager
- Available as a global dotnet tool
- Libraries are managed in libman.json
  - Cdnjs is default library source (also unpkg or file system)
  - Can be configure per package or globally
- Another great tool by Mads Kristensen

BUNDLING AND MINIFICATION

BUNDLING AND MINIFICATION

- JavaScript and CSS files should be bundled and minified for performance
- WebOptimizer is the ASP.NET Core solution
  - https://github.com/ligershark/WebOptimizer
  - Does more than just bundle/minify
  - Another great tool by Mads Kristensen
- Updated for .NET Core 3 (check the version before installing)

add app.UseWebOptimizer() to the Configure method anywhere before app.UseStaticFiles

ASP.NET CORE WEB OPTIMIZER

- "ASP.NET Core middleware for bundling and minification of CSS and JavaScript files at runtime. With full server-side and client-side caching to ensure high performance. No complicated build process and no hassle."
- Installation:
  - Add package LigerShark.WebOptimizer.Core
  - Update _ViewStart.cshtml
    - @addTagHelper *, WebOptimizer.Core
  - Add app.UseWebOptimizer() to the Configure method
    - Must be called before app.UseStaticFiles()
  - Add services.AddWebOptimizer() to the ConfigureServices method

VIEWS AND LAYOUTS

LAYOUTS AND VIEWS

- ViewStart sets default
  - Can be configured per view
- RenderBody renders the view
- Sections add more control (required || optional)
  - RenderSection/IgnoreSection
- Partials don’t use a layout
- Razor code mixes with layout
- Tag Helpers keep you in the mark up

VIEW COMPONENTS

VIEW COMPONENTS

- View Components combine server side code with partial views
  - Used to render a chunk of the response
  - Used instead of ChildActions/PartialViews
- Common Uses:
  - Dynamically created menus
  - Login panel
  - Shopping cart

LIMITATIONS

- Can’t serve as a client-side end point
- Don’t use model binding
- Don’t participate in controller lifecycle
- Must locate partial view in:

```
Views/<controller_name>/Components/<view_component_name>/<view_name>
Views/Shared/Components/<view_component_name>/<view_name>
Pages/Shared/Components/<View Component Name>/<View Name>
```

INVOKING VIEW COMPONENTS

- Invoke from a view (or layout) using lower-kebob-casing:
  - <vc:view-component-name model="@Model">
- Invoke from a controller action method:
  - return ViewComponent("<name>“, <anonymous type with parameters>);

TAG HELPERS

TAG HELPERS

- Enable server-side code to participate in rendering HTML elements in Razor views
- Reduces the transition between code and markup
  - Tag Helpers Attach to HTML elements
  - HTML Helpers are invoked as methods
- Fully supported with IntelliSense

- Can also create custom tag helpers

EU GENERAL DATA PROTECTION REGULATION (GDPR)

GDPR SUPPORT

- Extension points and stubbed markup for privacy and cookie use policy.
- Cookie consent feature asks (and tracks) consent
  - Without consent, non-essential cookies aren't sent to the browser.
  - Cookies can be marked as essential.
  - TempData and Session cookies aren't functional when tracking is disabled.
- The Identity manage page provides a link to download and delete user data.

- ASP.NET Core 3.1 templates removed starter code

Contact Me

skimedic@outlook.com
www.skimedic.com/blog
www.twitter.com/skimedic

http://bit.ly/skimediclyndacourses
http://bit.ly/apressbooks

www.hallwayconversations.com

Questions?

Thank You!

Find the code at: https://github.com/skimedic/presentations/tree/master/DOTNETCORE/ASP.NETCore