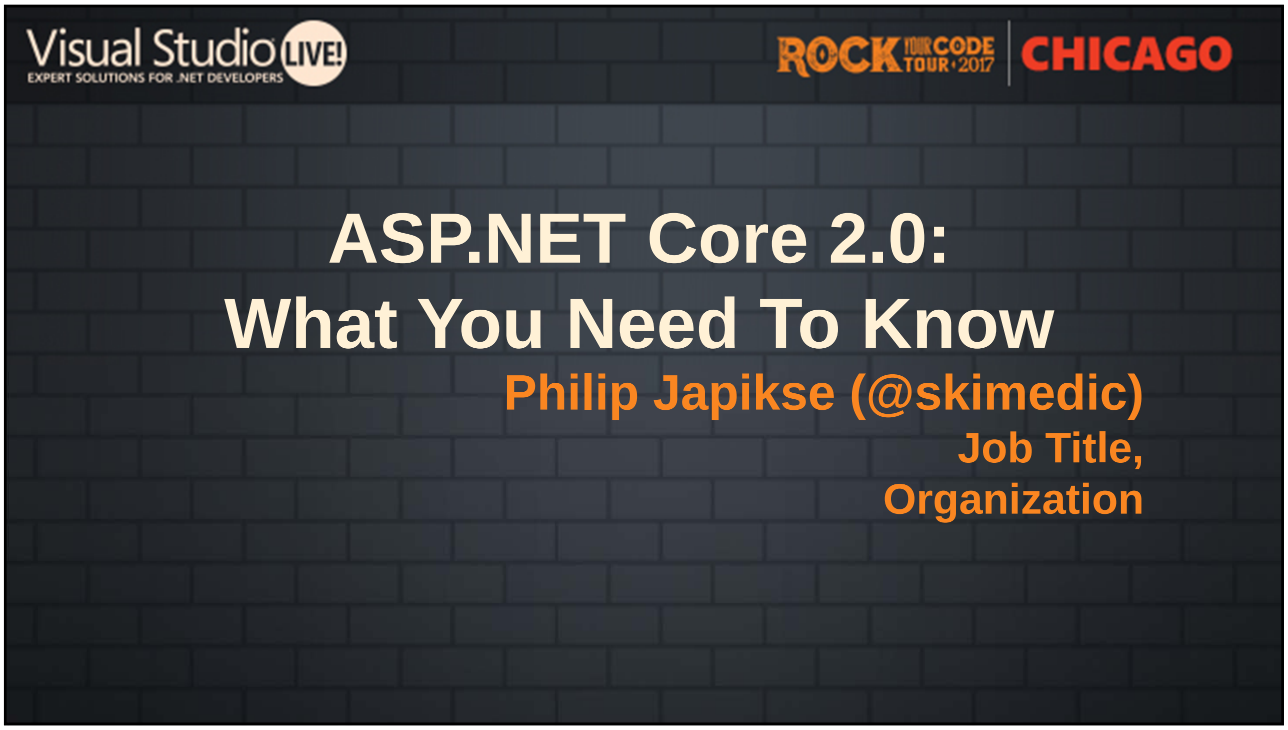


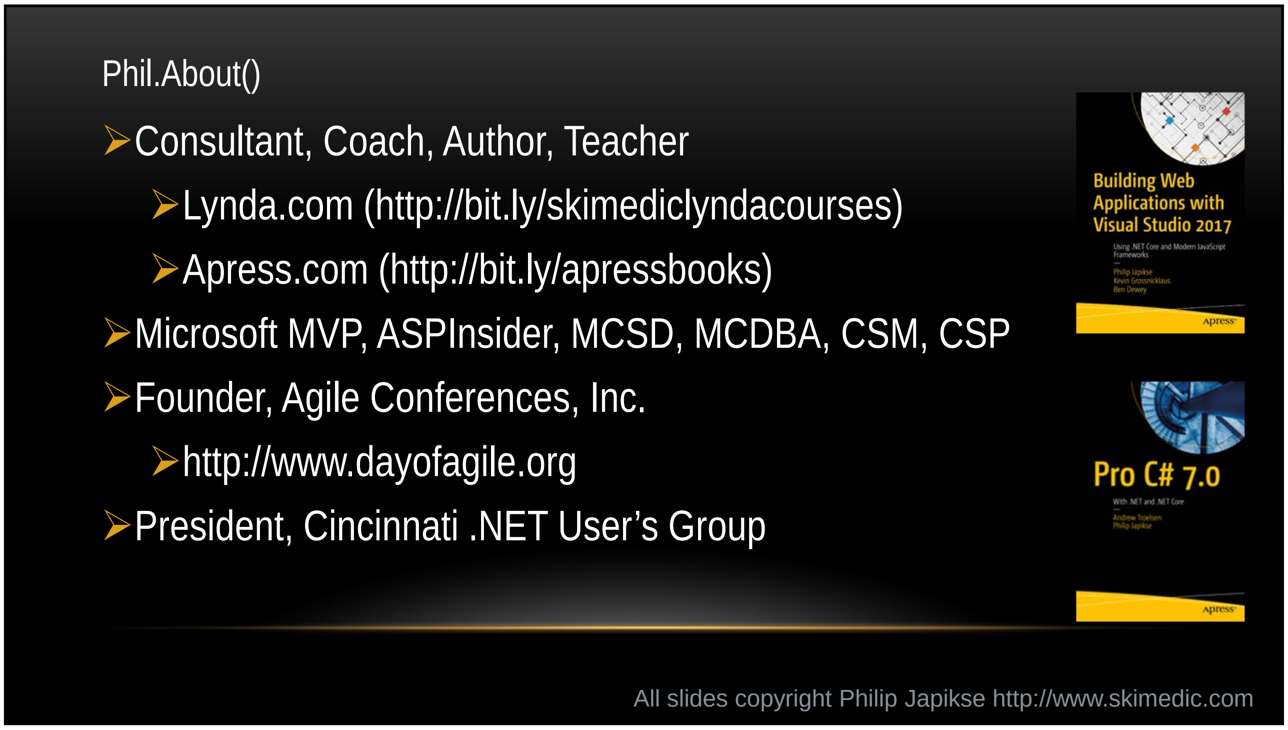
Visual Studio LIVE! EXPERT SOLUTIONS FOR .NET DEVELOPERS — ROCK YOUR CODE TOUR 2017 | CHICAGO

ASP.NET Core 2.0: What You Need To Know

Philip Japikse (@skimedic)

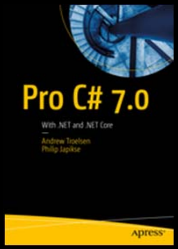
Job Title,

Organization

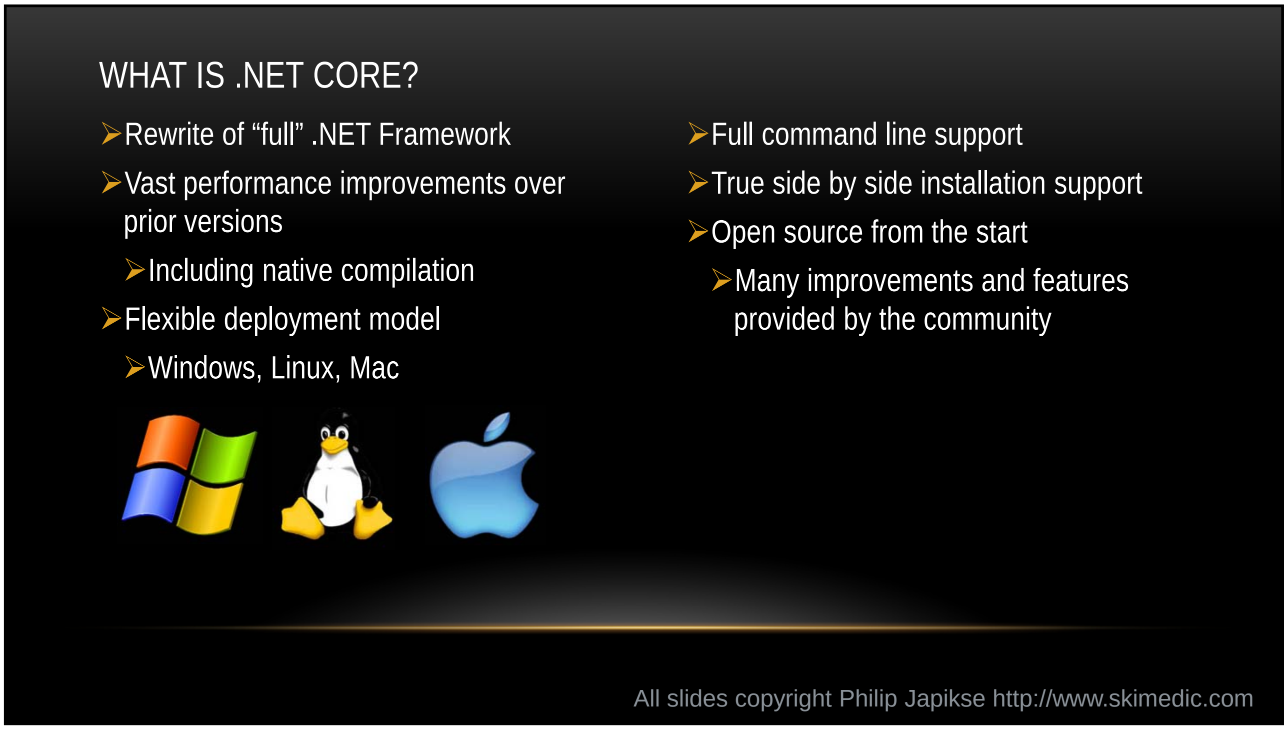


Phil.About()

- Consultant, Coach, Author, Teacher
  - Lynda.com (http://bit.ly/skimediclyndacourses)
  - Apress.com (http://bit.ly/apressbooks)
- Microsoft MVP, ASPInsider, MCSD, MCDBA, CSM, CSP
- Founder, Agile Conferences, Inc.
  - http://www.dayofagile.org
- President, Cincinnati .NET User's Group

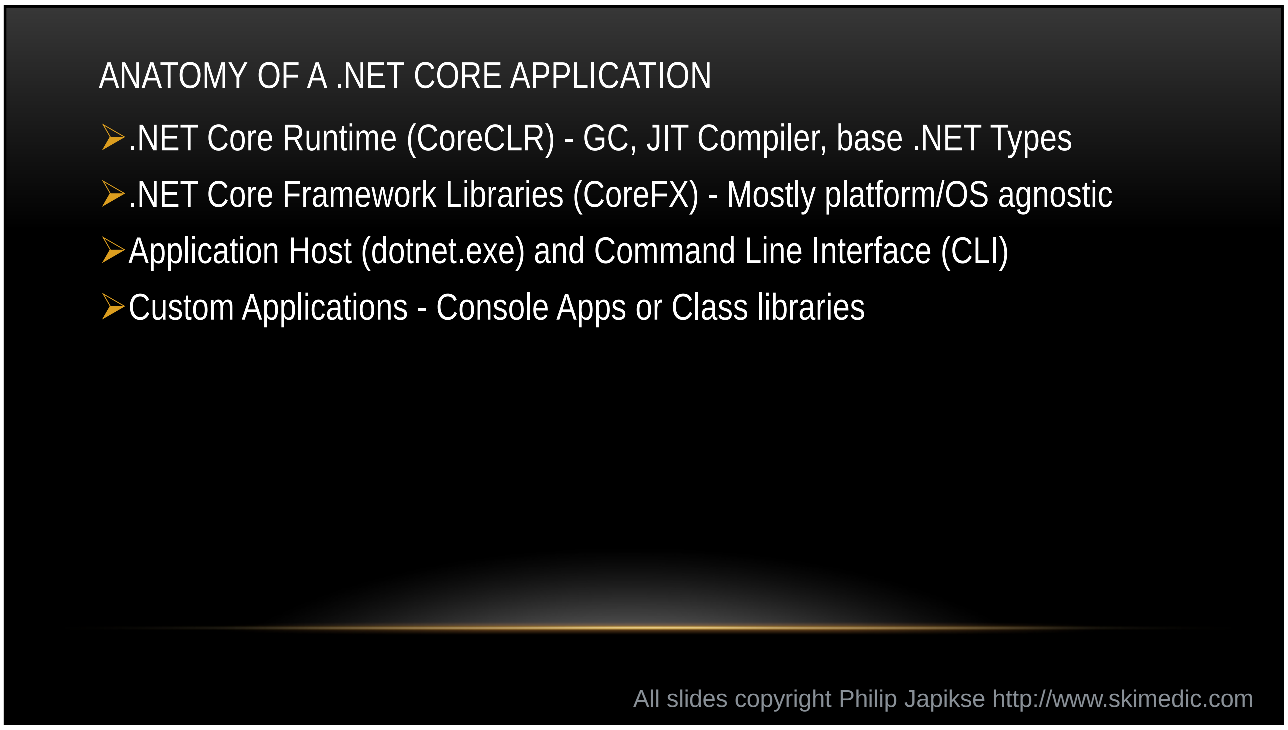


.NET CORE 2.0



WHAT IS .NET CORE?

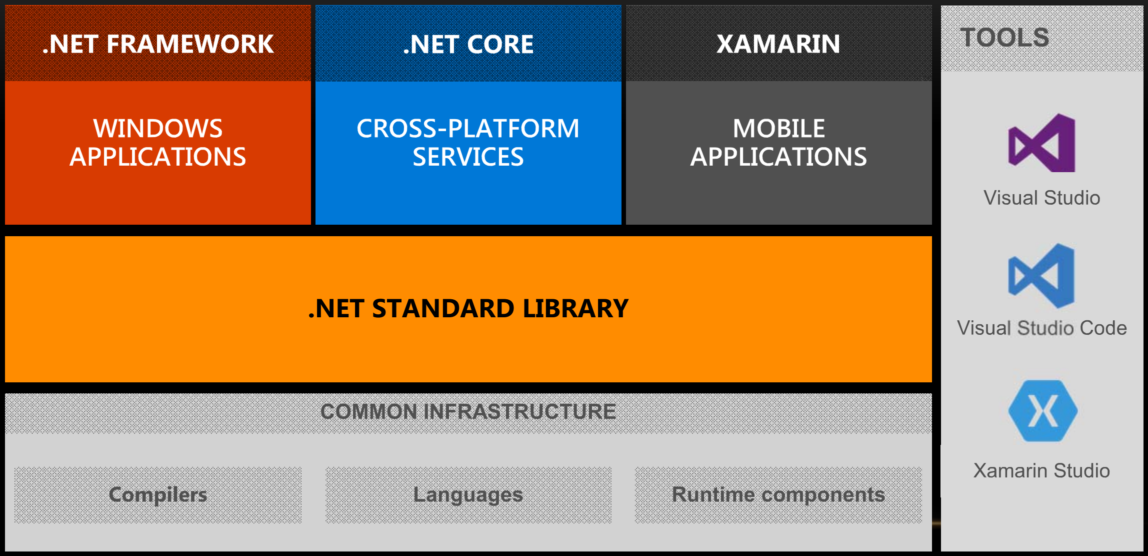
- Rewrite of “full” .NET Framework
- Vast performance improvements over prior versions
  - Including native compilation
- Flexible deployment model
  - Windows, Linux, Mac
- Full command line support
- True side by side installation support
- Open source from the start
  - Many improvements and features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



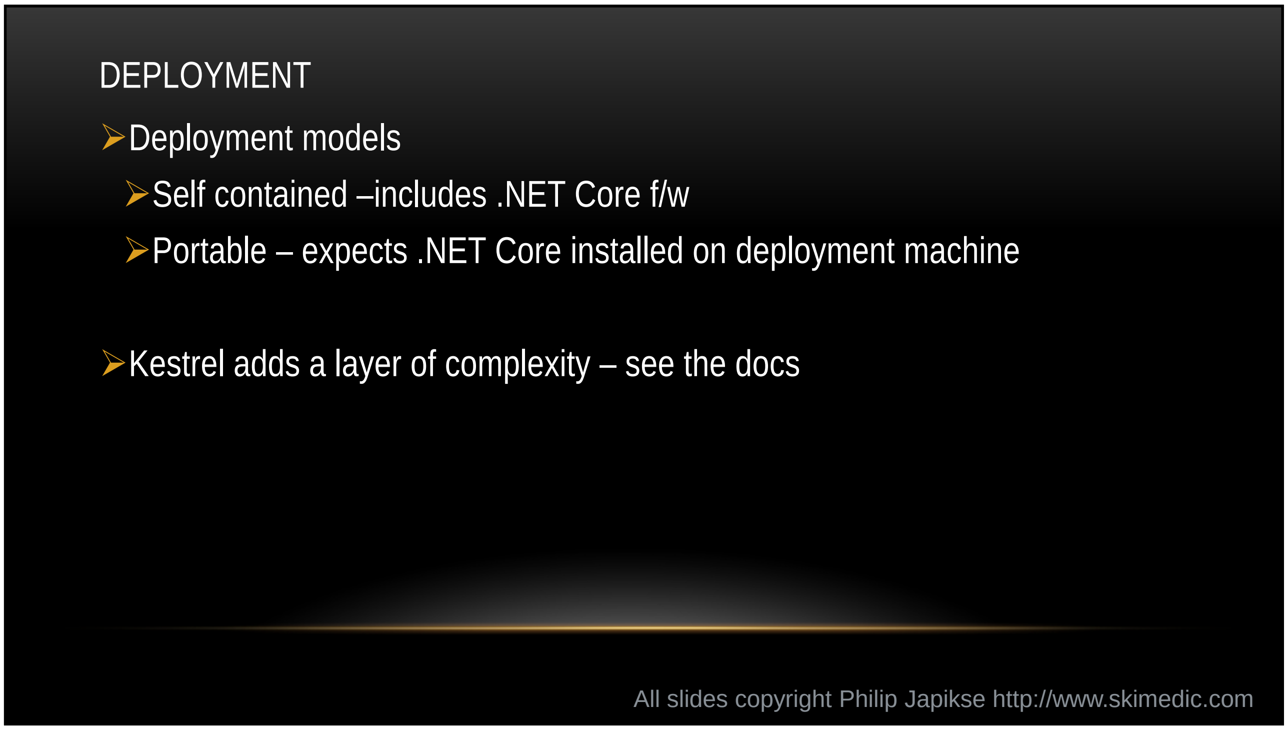
ANATOMY OF A .NET CORE APPLICATION

- .NET Core Runtime (CoreCLR) - GC, JIT Compiler, base .NET Types
- .NET Core Framework Libraries (CoreFX) - Mostly platform/OS agnostic
- Application Host (dotnet.exe) and Command Line Interface (CLI)
- Custom Applications - Console Apps or Class libraries

FULL BCD (BIRTHDAY CAKE DIAGRAM)

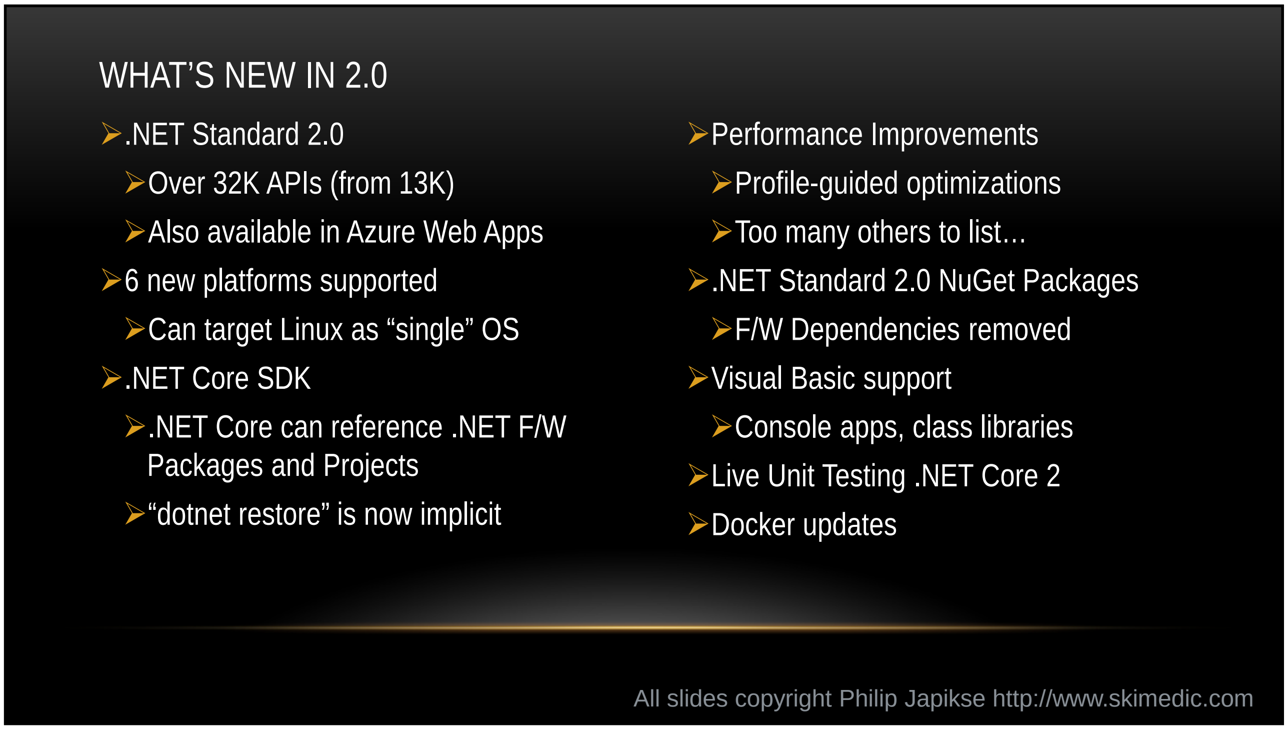


Courtesy of Rowan Miller
https://github.com/rowanmiller/Demo-EFCore



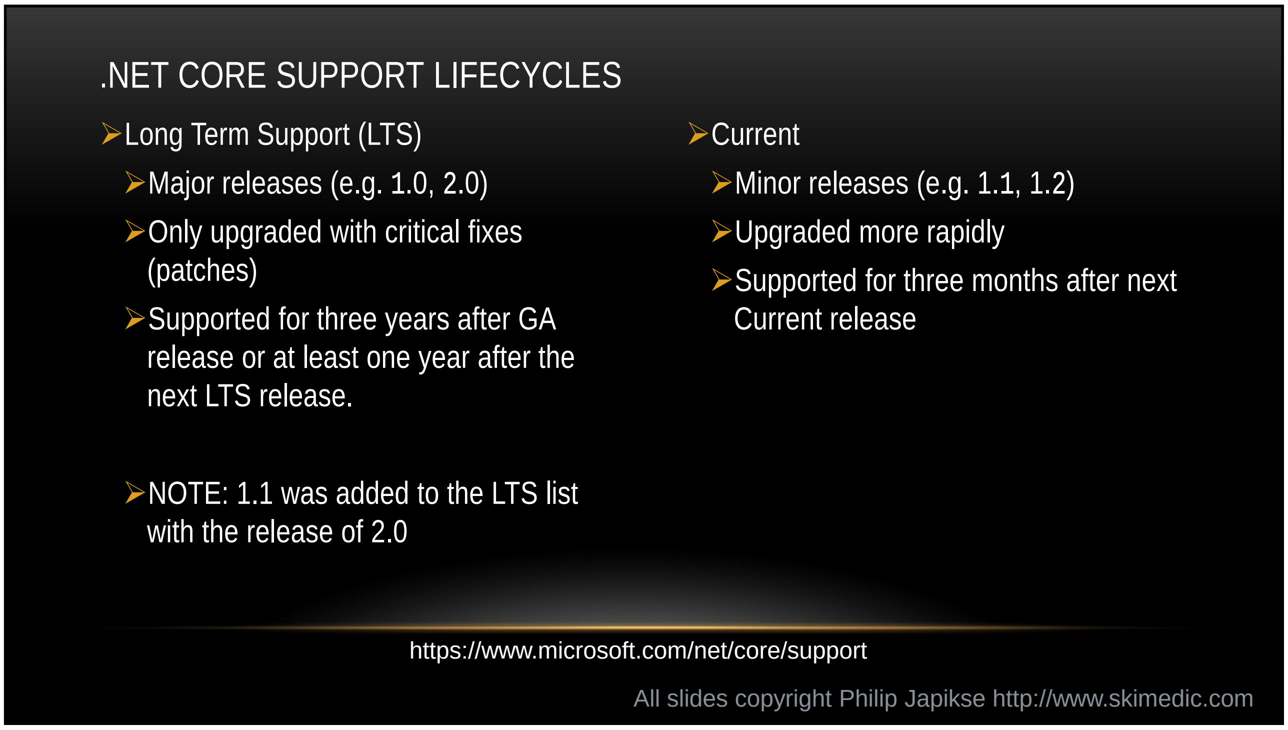
DEPLOYMENT

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



WHAT'S NEW IN 2.0

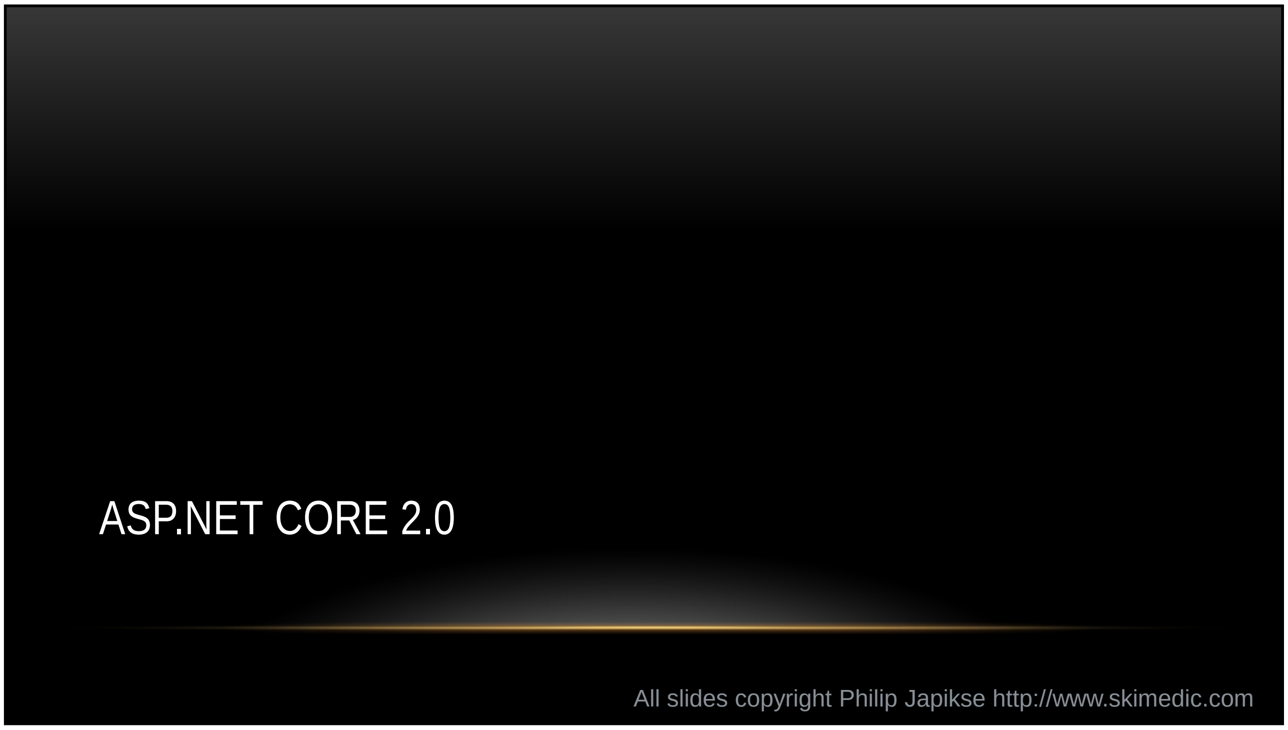
- .NET Standard 2.0
  - Over 32K APIs (from 13K)
  - Also available in Azure Web Apps
- 6 new platforms supported
  - Can target Linux as "single" OS
- .NET Core SDK
  - .NET Core can reference .NET F/W Packages and Projects
  - "dotnet restore" is now implicit
- Performance Improvements
  - Profile-guided optimizations
  - Too many others to list…
- .NET Standard 2.0 NuGet Packages
  - F/W Dependencies removed
- Visual Basic support
  - Console apps, class libraries
- Live Unit Testing .NET Core 2
- Docker updates



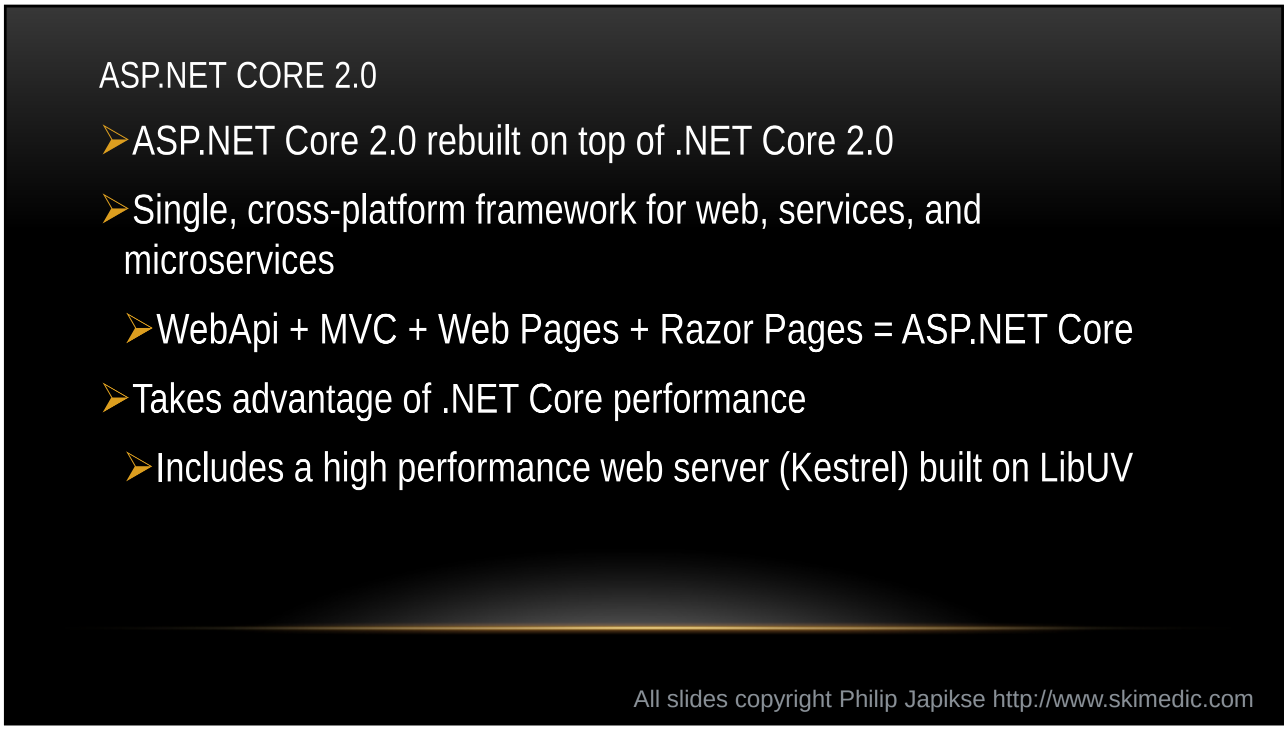
.NET CORE SUPPORT LIFECYCLES

- Long Term Support (LTS)
  - Major releases (e.g. 1.0, 2.0)
  - Only upgraded with critical fixes (patches)
  - Supported for three years after GA release or at least one year after the next LTS release.
  - NOTE: 1.1 was added to the LTS list with the release of 2.0
- Current
  - Minor releases (e.g. 1.1, 1.2)
  - Upgraded more rapidly
  - Supported for three months after next Current release

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

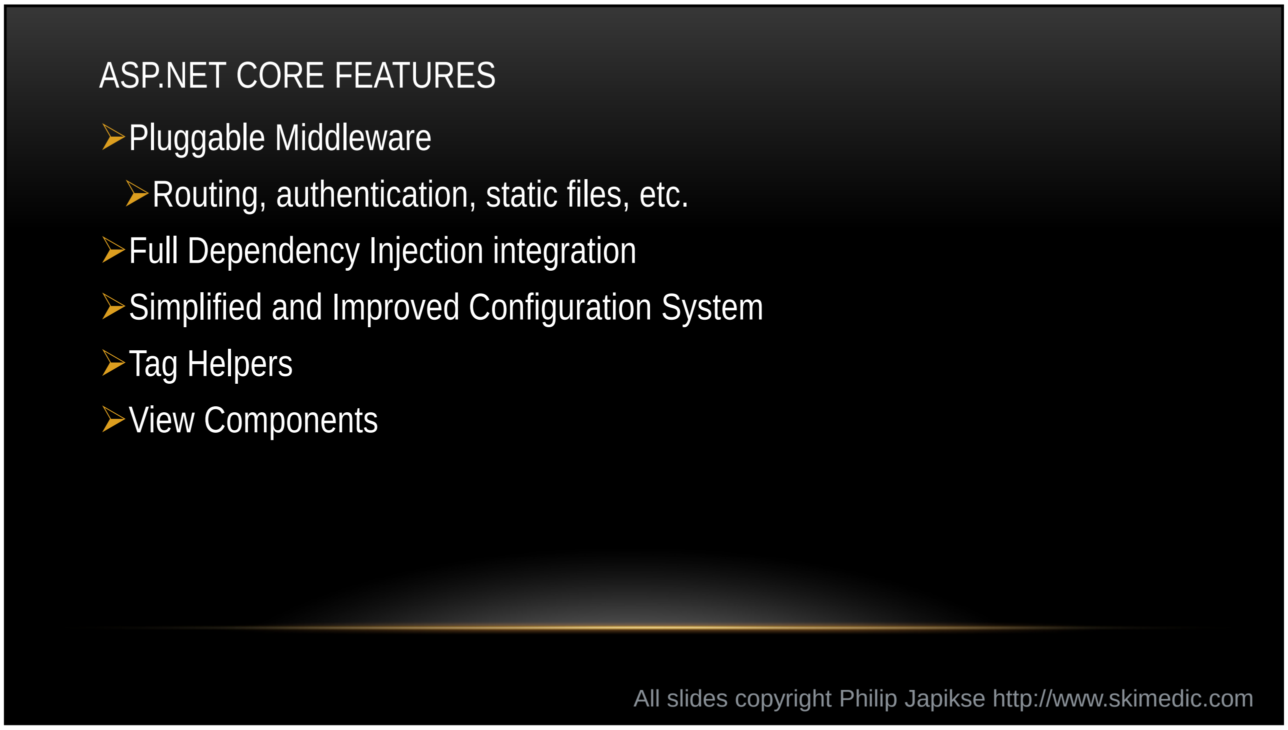


ASP.NET CORE 2.0



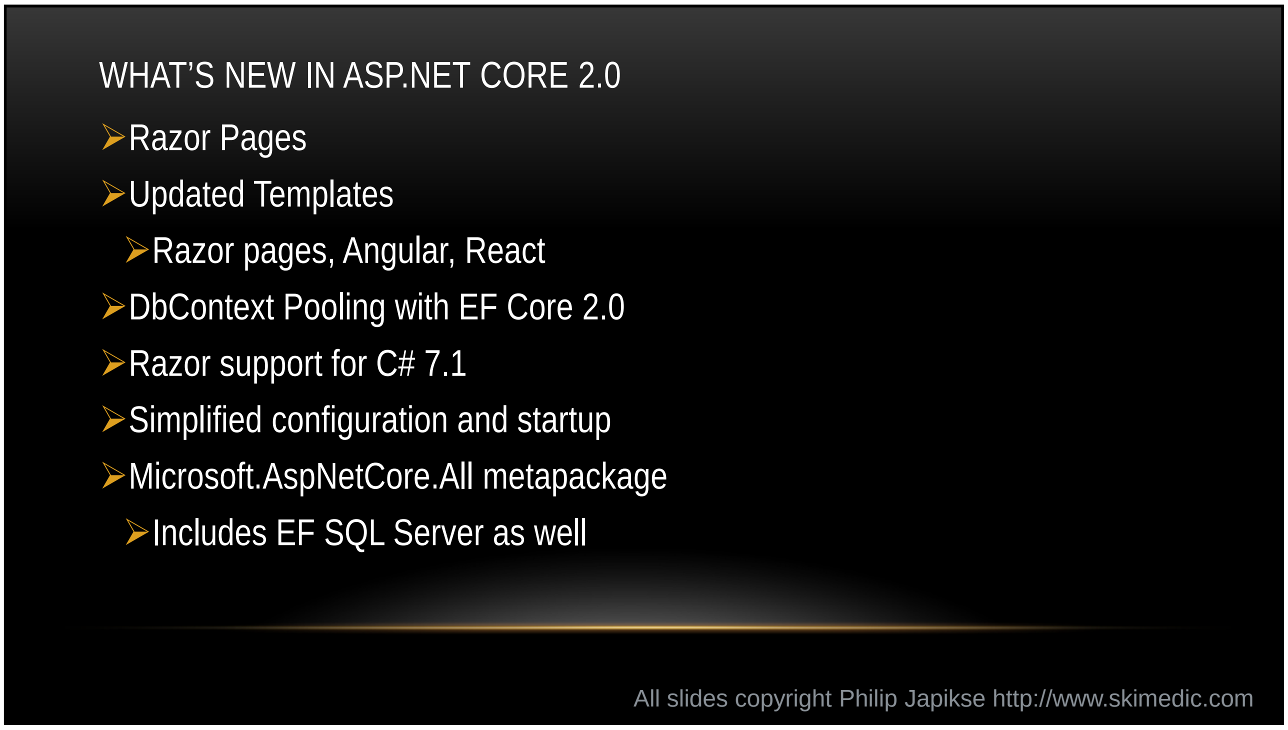
ASP.NET CORE 2.0

- ASP.NET Core 2.0 rebuilt on top of .NET Core 2.0
- 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

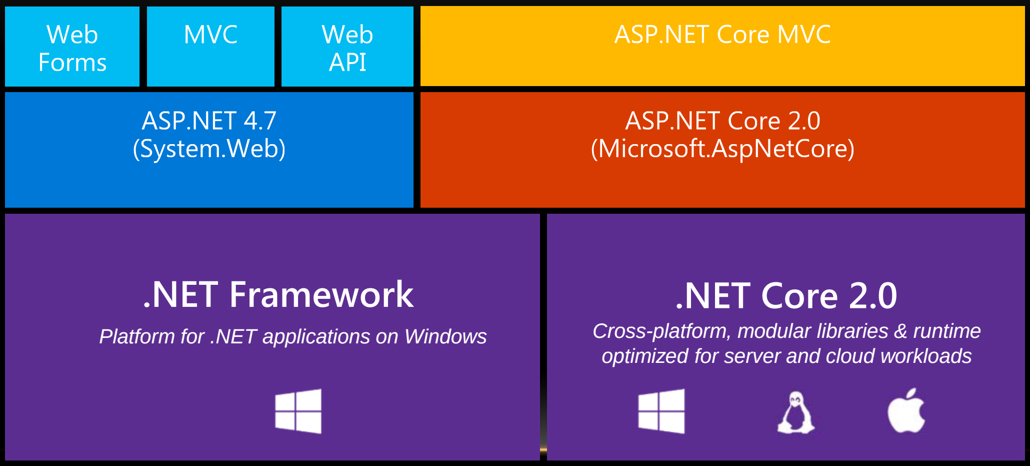
- Pluggable Middleware
  - Routing, authentication, static files, etc.
- Full Dependency Injection integration
- Simplified and Improved Configuration System
- Tag Helpers
- View Components

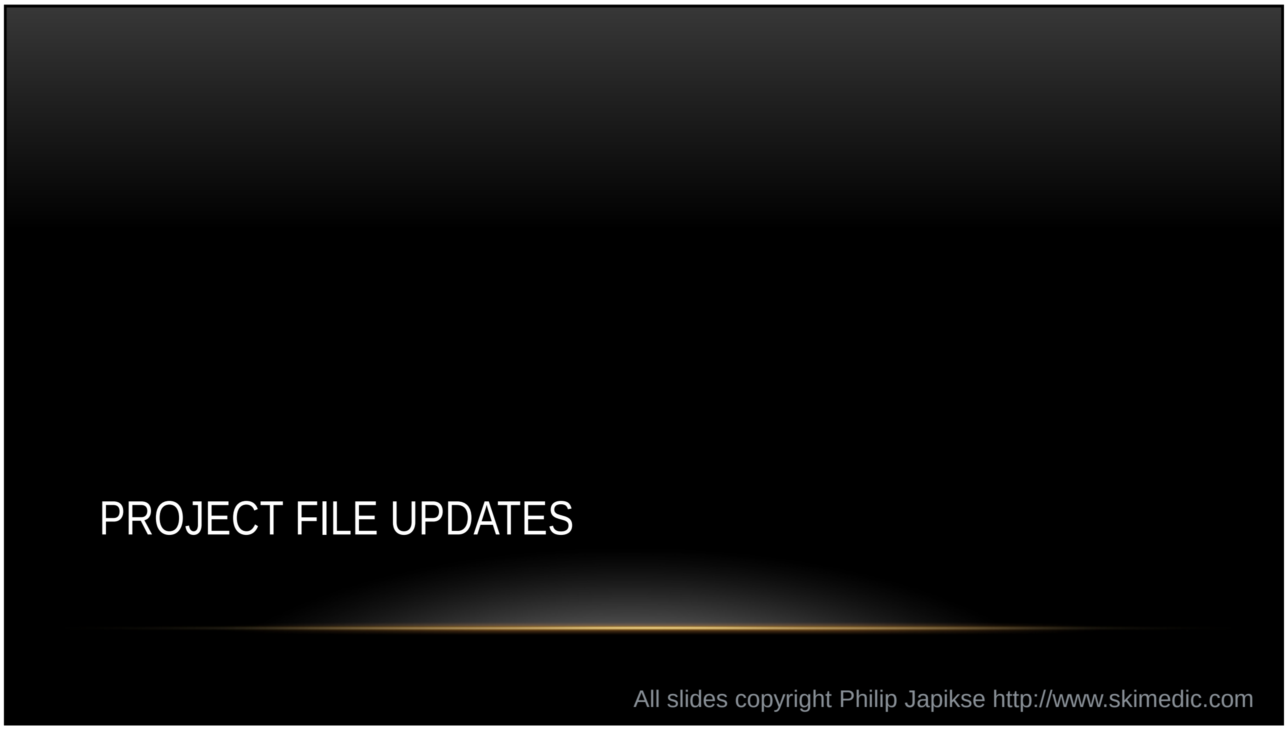


WHAT'S NEW IN ASP.NET CORE 2.0

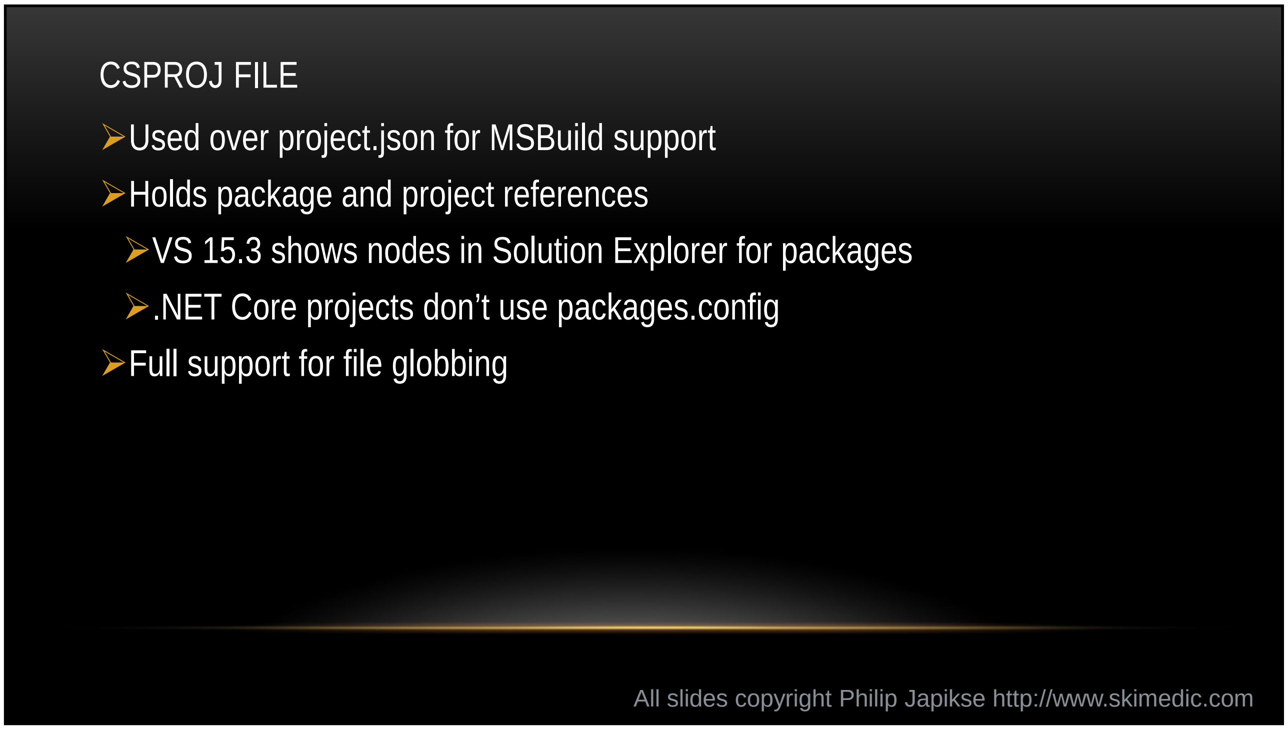
- Razor Pages
- Updated Templates
  - Razor pages, Angular, React
- DbContext Pooling with EF Core 2.0
- Razor support for C# 7.1
- Simplified configuration and startup
- Microsoft.AspNetCore.All metapackage
  - Includes EF SQL Server as well

ASP.NET CORE BCA


Web Forms
MVC
Web API
ASP.NET Core MVC
ASP.NET 4.7 (System.Web)
ASP.NET Core 2.0 (Microsoft.AspNetCore)
.NET Framework
Platform for .NET applications on Windows
.NET Core 2.0
Cross-platform, modular libraries & runtime optimized for server and cloud workloads

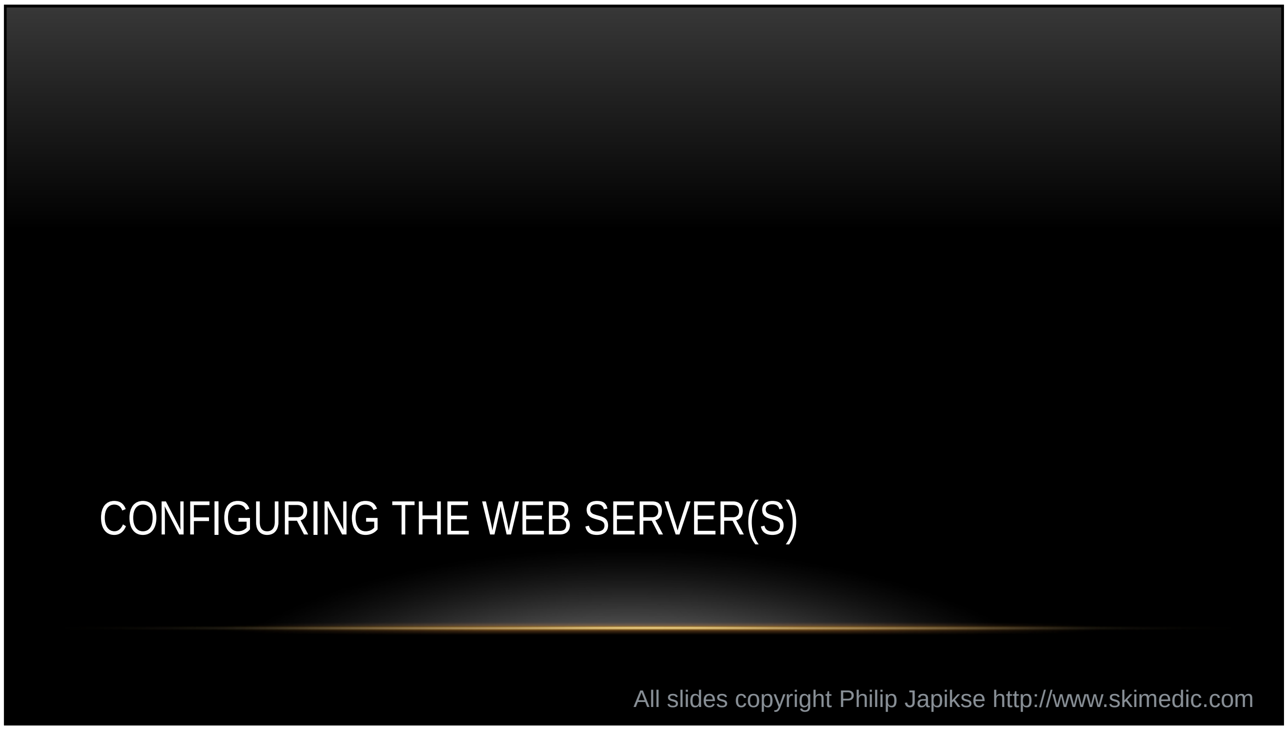


PROJECT FILE UPDATES

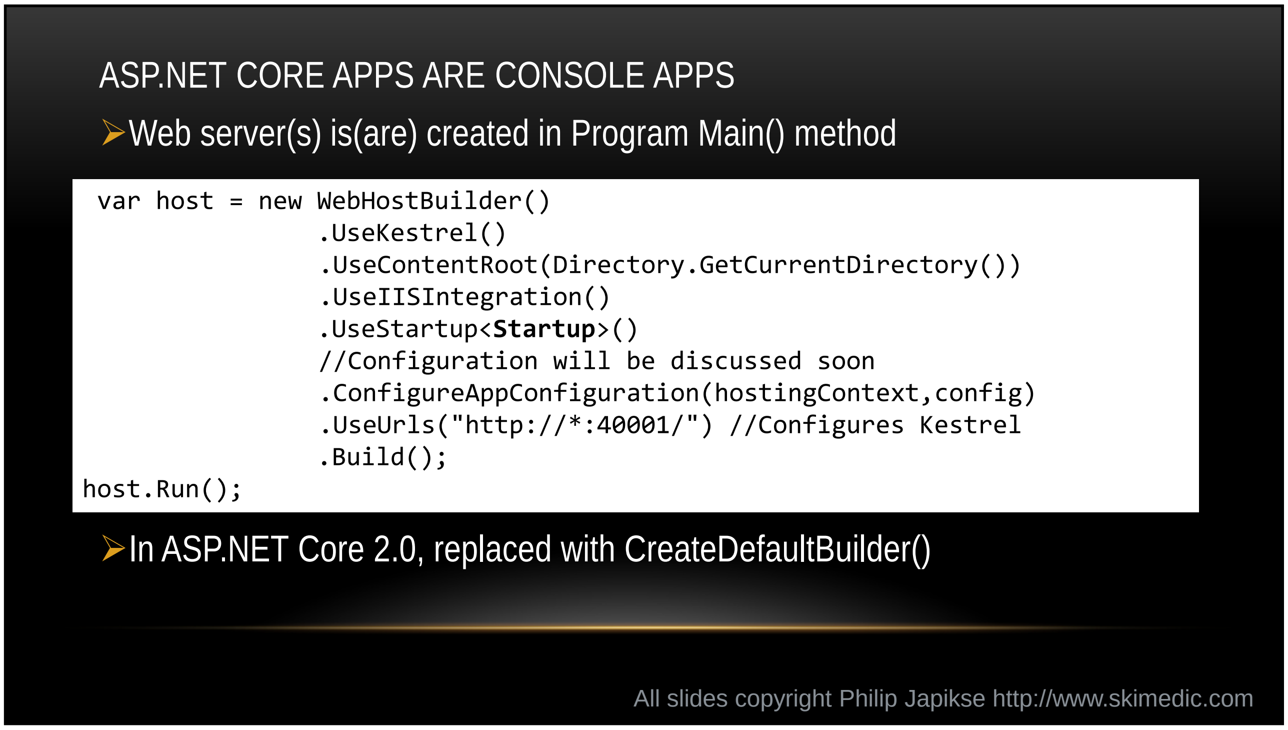


CSPROJ FILE

- Used over project.json for MSBuild support
- Holds package and project references
  - VS 15.3 shows nodes in Solution Explorer for packages
  - .NET Core projects don't use packages.config
- Full support for file globbing



CONFIGURING THE WEB SERVER(S)

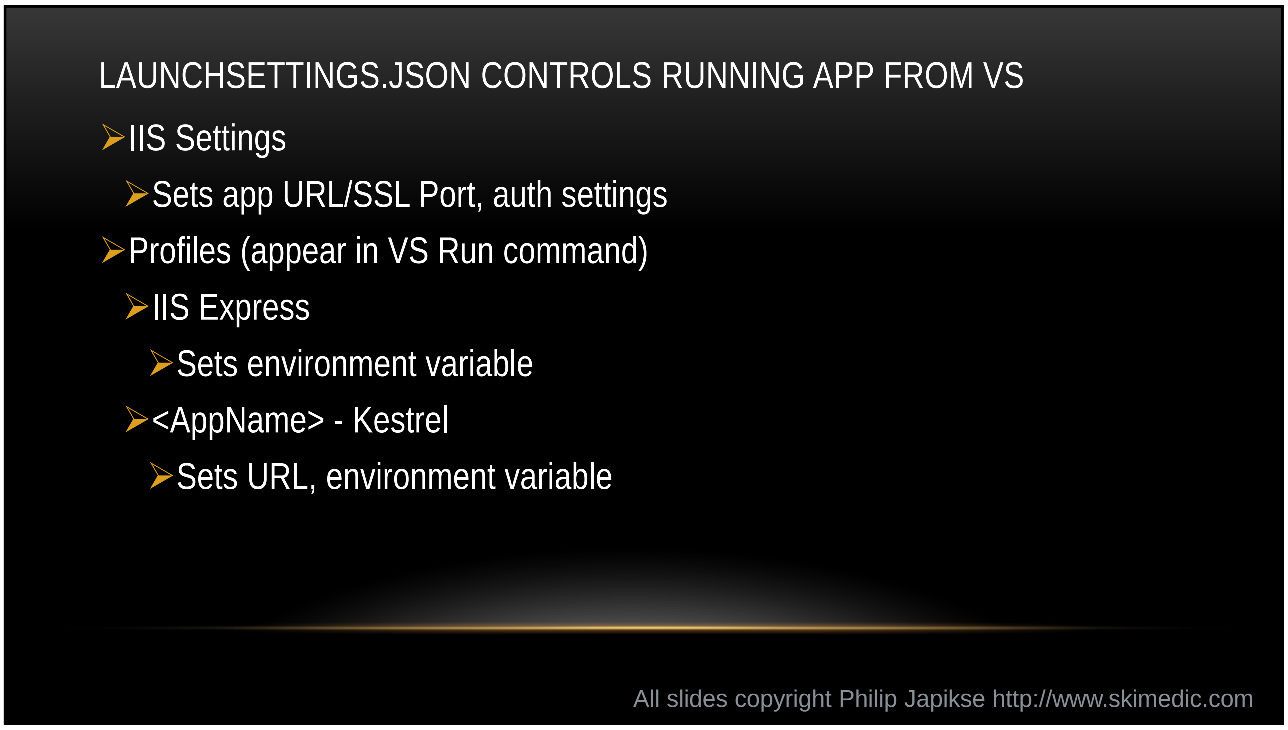


ASP.NET CORE APPS ARE CONSOLE APPS

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

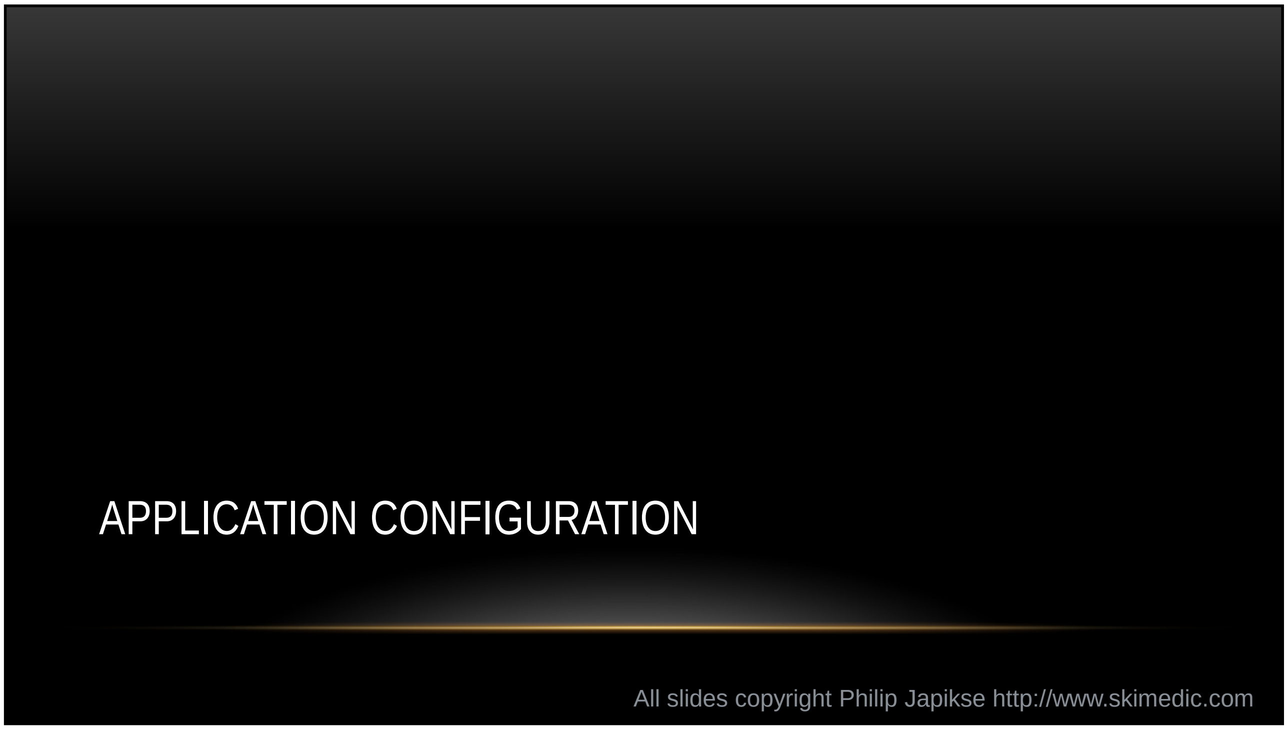
```
 var host = new WebHostBuilder()
                .UseKestrel()
                .UseContentRoot(Directory.GetCurrentDirectory())
                .UseIISIntegration()
                .UseStartup<Startup>()
                //Configuration will be discussed soon
                .ConfigureAppConfiguration(hostingContext,config)
                .UseUrls("http://*:40001/") //Configures Kestrel
                .Build();
host.Run();
```

- In ASP.NET Core 2.0, replaced with CreateDefaultBuilder()

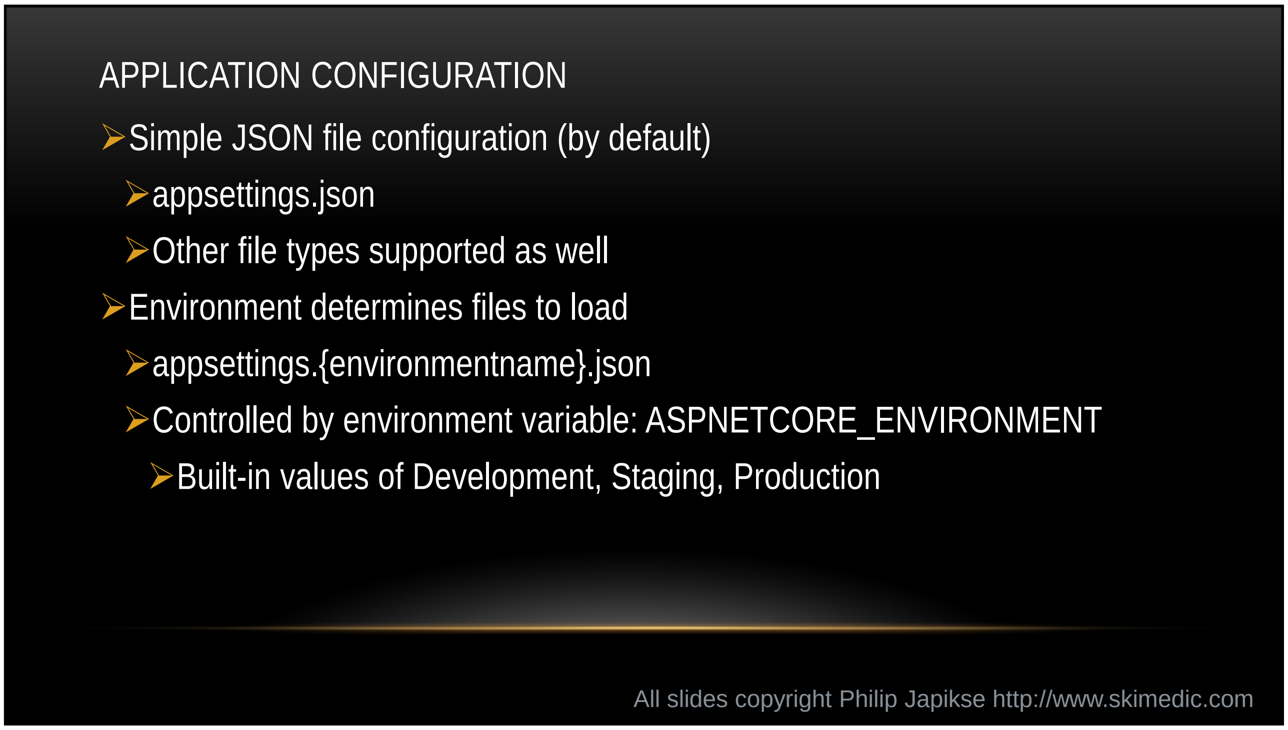


LAUNCHSETTINGS.JSON CONTROLS RUNNING APP FROM VS

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

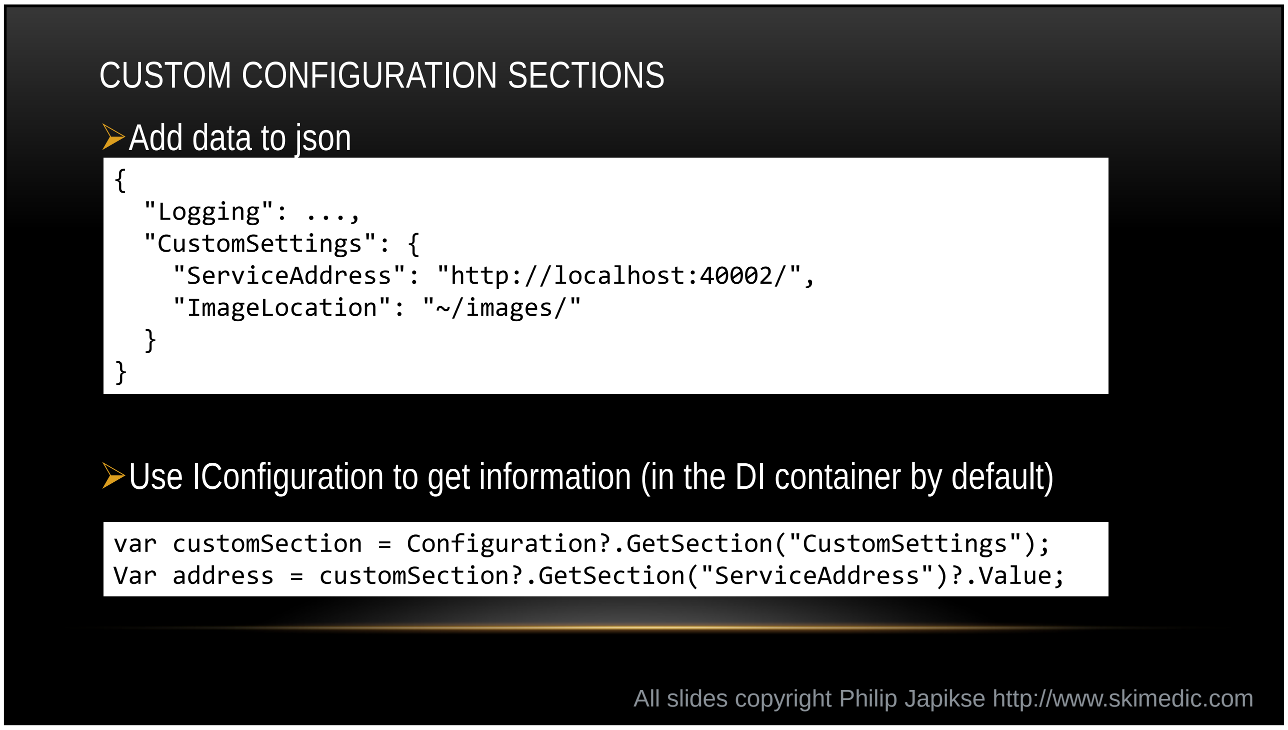


APPLICATION CONFIGURATION



APPLICATION CONFIGURATION

- Simple JSON file configuration (by default)
  - appsettings.json
  - Other file types supported as well
- Environment determines files to load
  - appsettings.{environmentname}.json
  - Controlled by environment variable: ASPNETCORE_ENVIRONMENT
    - Built-in values of Development, Staging, Production



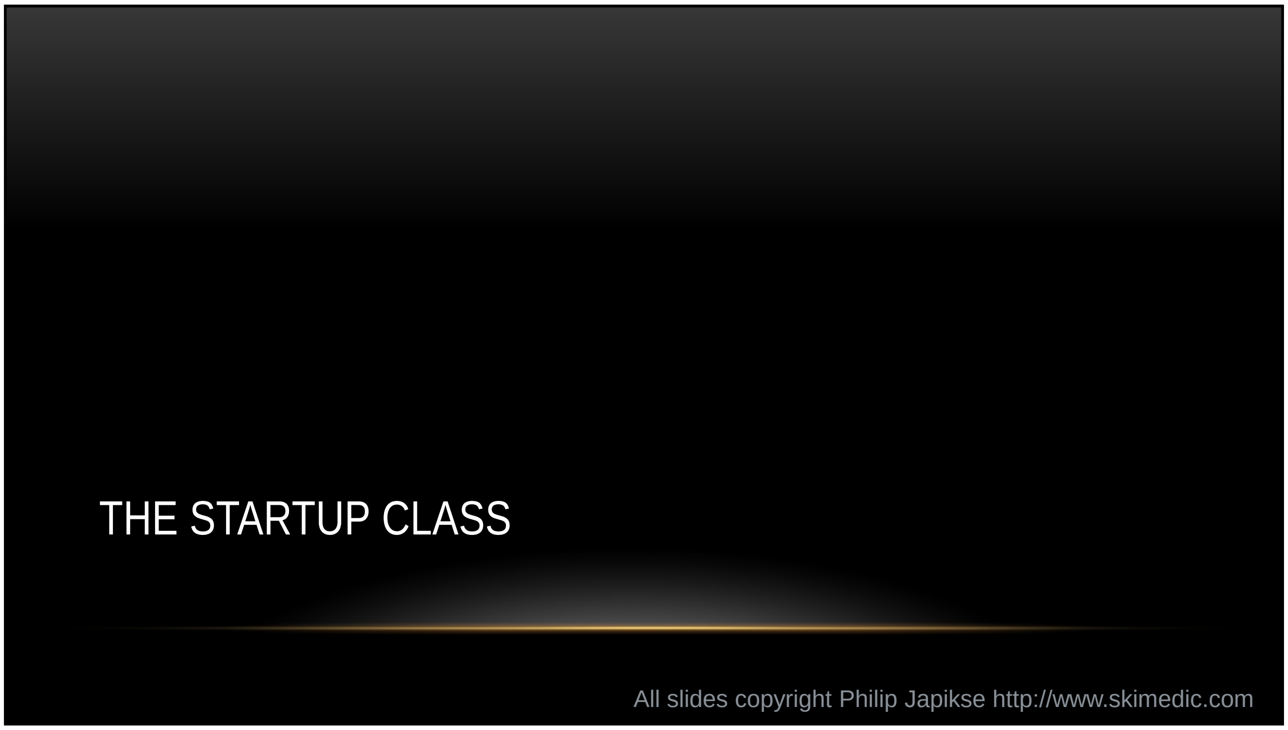
CUSTOM CONFIGURATION SECTIONS

- Add data to json

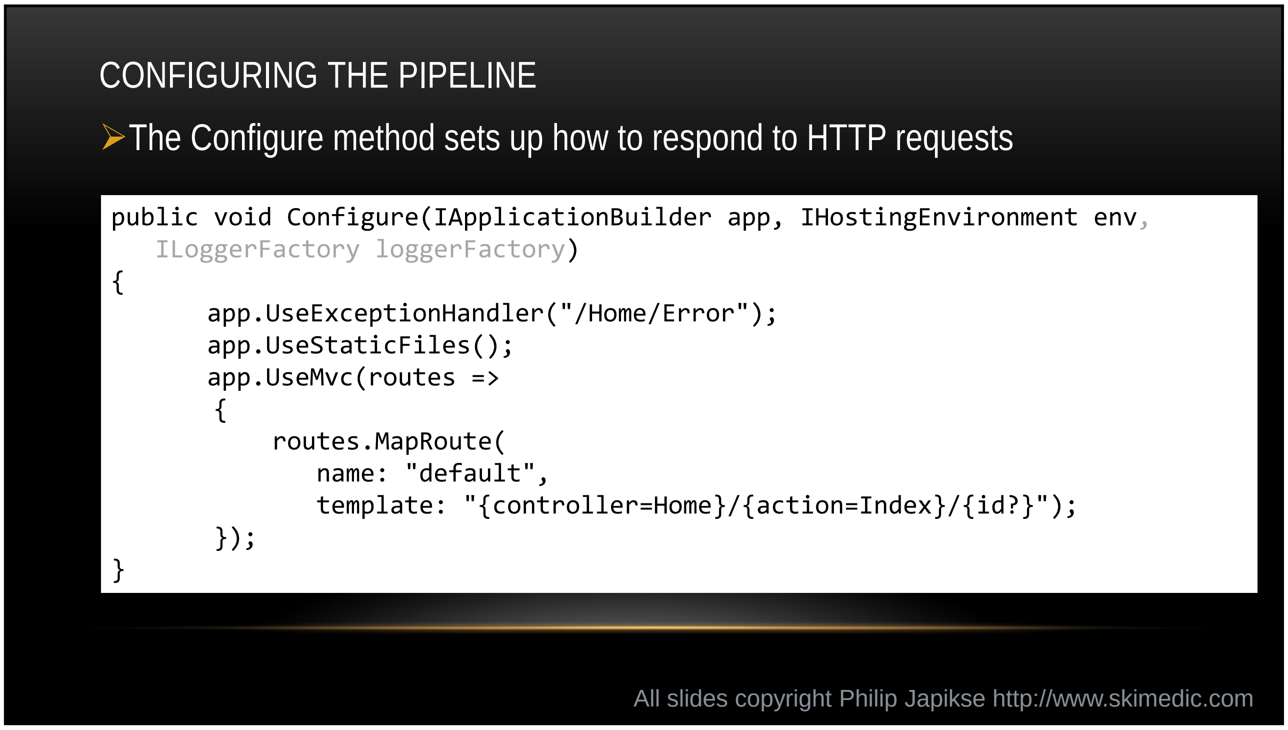
```
{
  "Logging": ...,
  "CustomSettings": {
    "ServiceAddress": "http://localhost:40002/",
    "ImageLocation": "~/images/"
  }
}
```

- Use IConfiguration to get information (in the DI container by default)

```
var customSection = Configuration?.GetSection("CustomSettings");
Var address = customSection?.GetSection("ServiceAddress")?.Value;
```



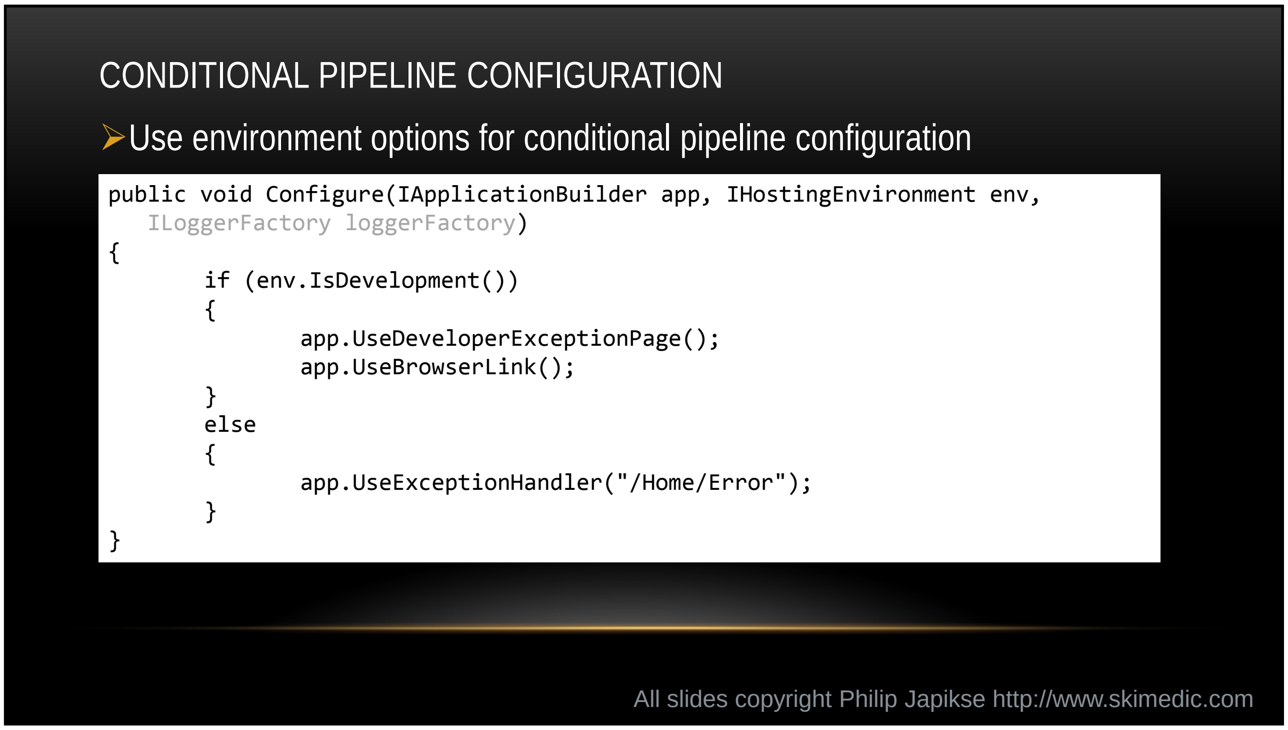
THE STARTUP CLASS



CONFIGURING THE PIPELINE

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

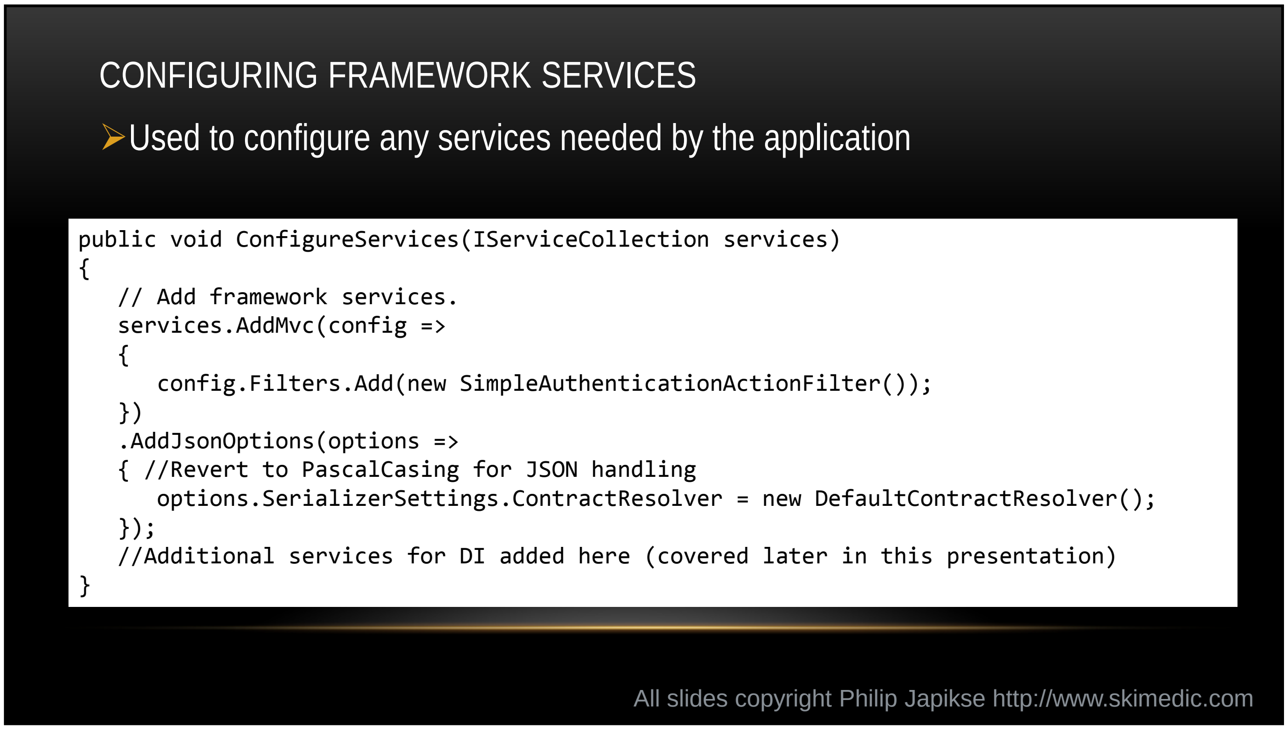
```
public void Configure(IApplicationBuilder app, IHostingEnvironment env,
   ILoggerFactory loggerFactory)
{
       app.UseExceptionHandler("/Home/Error");
       app.UseStaticFiles();
       app.UseMvc(routes =>
       {
           routes.MapRoute(
              name: "default",
              template: "{controller=Home}/{action=Index}/{id?}");
       });
}
```



CONDITIONAL PIPELINE CONFIGURATION

- Use environment options for conditional pipeline configuration

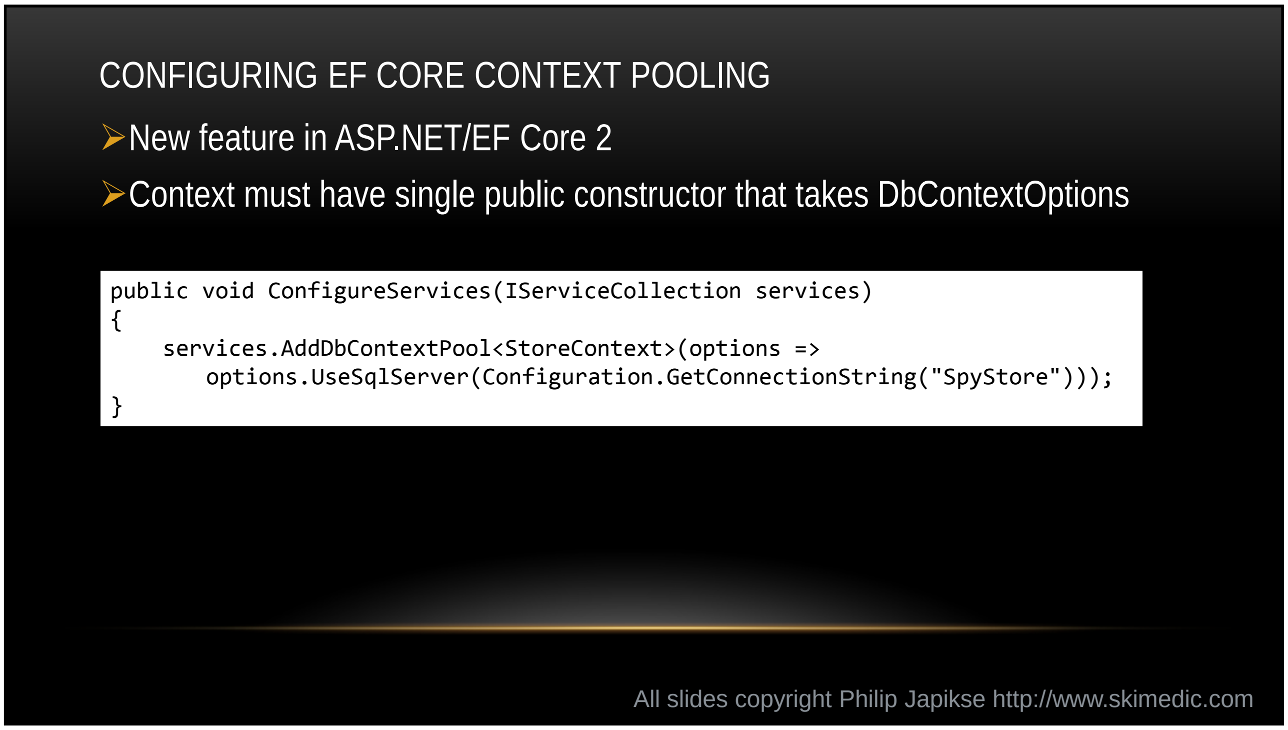
```
public void Configure(IApplicationBuilder app, IHostingEnvironment env,
   ILoggerFactory loggerFactory)
{
       if (env.IsDevelopment())
       {
              app.UseDeveloperExceptionPage();
              app.UseBrowserLink();
       }
       else
       {
              app.UseExceptionHandler("/Home/Error");
       }
}
```



CONFIGURING FRAMEWORK SERVICES

- Used to configure any services needed by the application

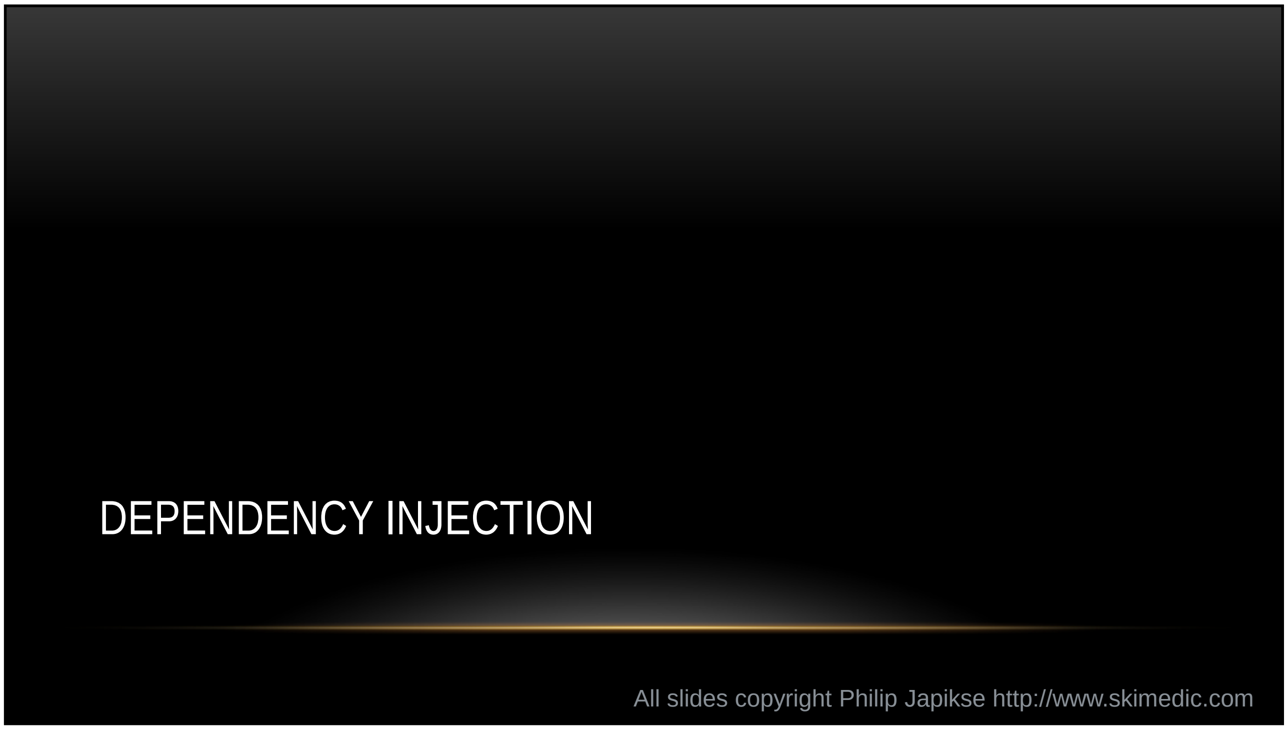
```
public void ConfigureServices(IServiceCollection services)
{
   // Add framework services.
   services.AddMvc(config =>
   {
      config.Filters.Add(new SimpleAuthenticationActionFilter());
   })
   .AddJsonOptions(options =>
   { //Revert to PascalCasing for JSON handling
      options.SerializerSettings.ContractResolver = new DefaultContractResolver();
   });
   //Additional services for DI added here (covered later in this presentation)
}
```



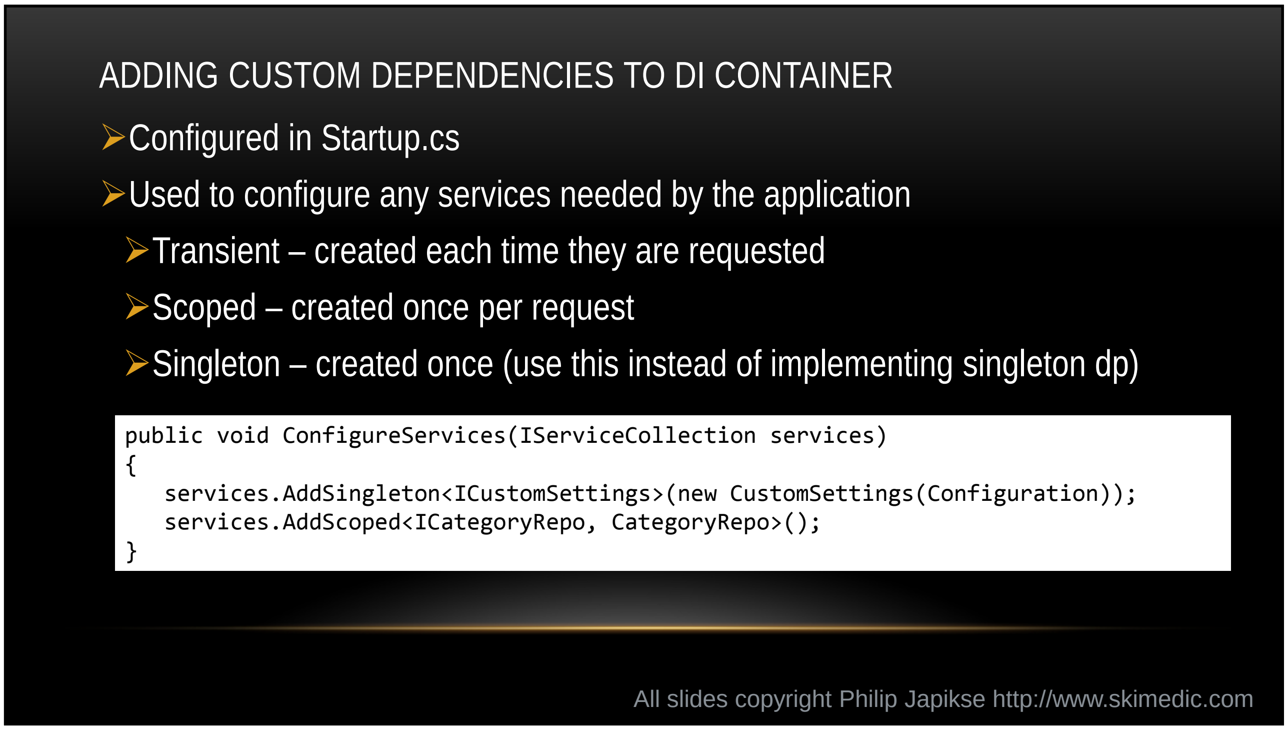
CONFIGURING EF CORE CONTEXT POOLING

- 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")));
}
```



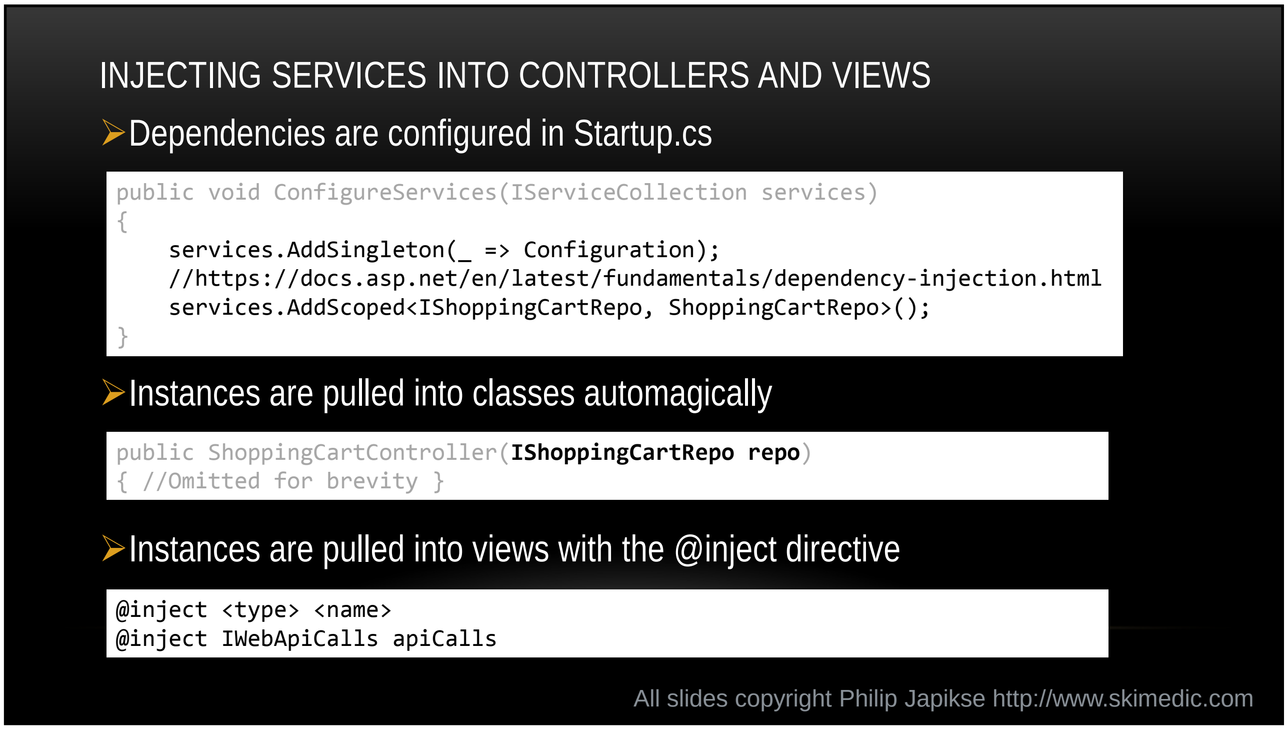
DEPENDENCY INJECTION



ADDING CUSTOM DEPENDENCIES TO DI CONTAINER

- Configured in Startup.cs
- Used to configure any services needed by the application
  - Transient – created each time they are requested
  - Scoped – created once per request
  - Singleton – created once (use this instead of implementing singleton dp)

```
public void ConfigureServices(IServiceCollection services)
{
   services.AddSingleton<ICustomSettings>(new CustomSettings(Configuration));
   services.AddScoped<ICategoryRepo, CategoryRepo>();
}
```



INJECTING SERVICES INTO CONTROLLERS AND VIEWS

- Dependencies are configured in Startup.cs

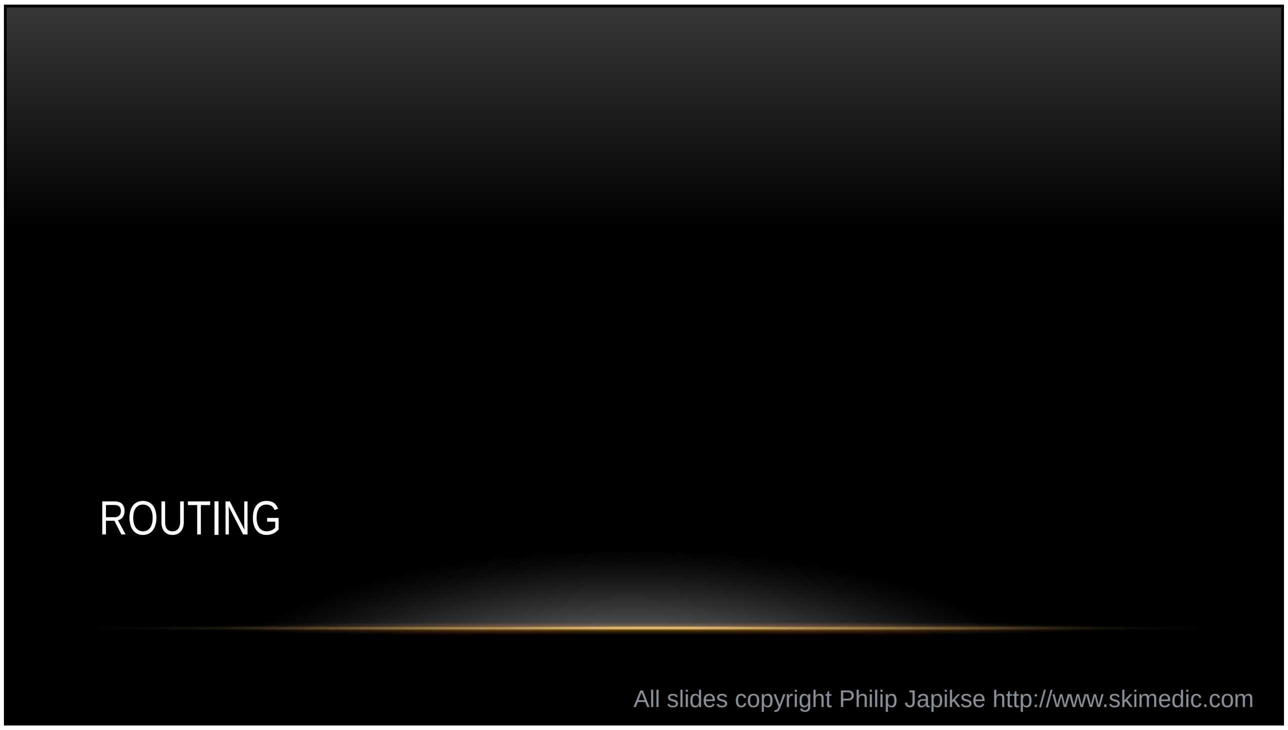
```
public void ConfigureServices(IServiceCollection services)
{
    services.AddSingleton(_ => Configuration);
    //https://docs.asp.net/en/latest/fundamentals/dependency-injection.html
    services.AddScoped<IShoppingCartRepo, ShoppingCartRepo>();
}
```

- Instances are pulled into classes automagically

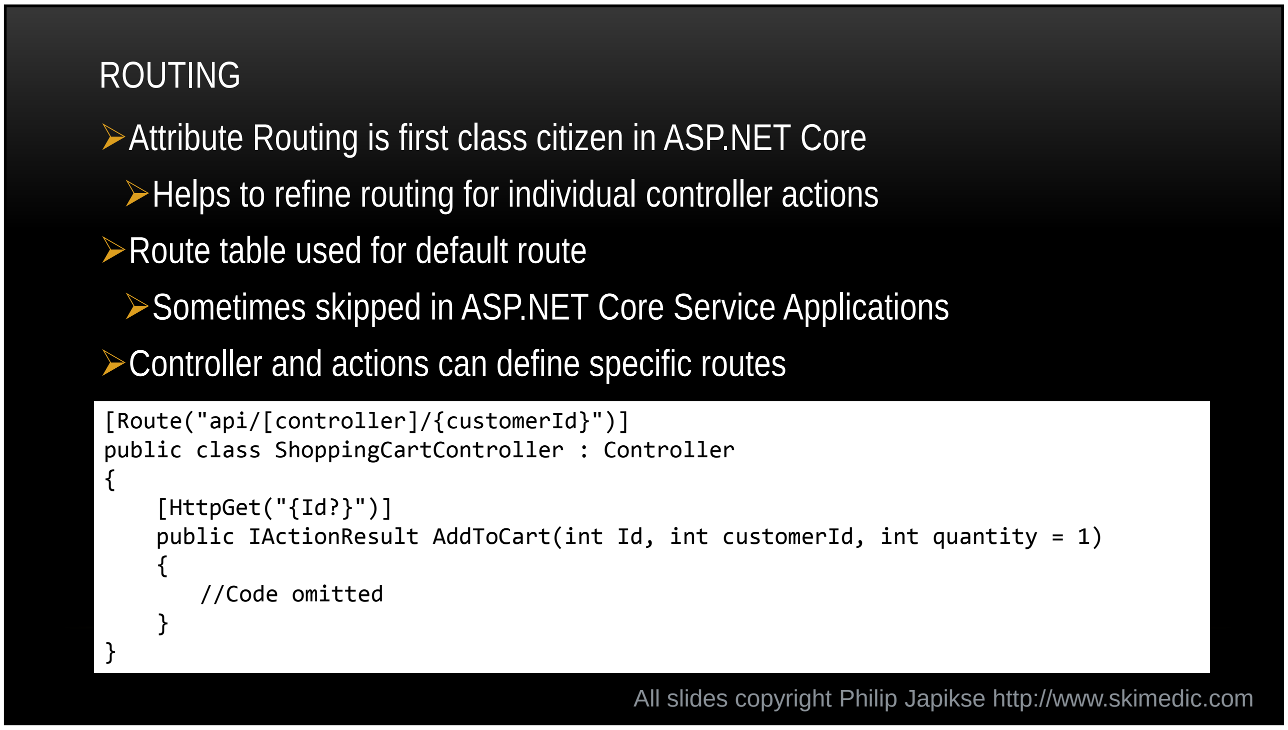
```
public ShoppingCartController(IShoppingCartRepo repo)
{ //Omitted for brevity }
```

- Instances are pulled into views with the @inject directive

```
@inject <type> <name>
@inject IWebApiCalls apiCalls
```



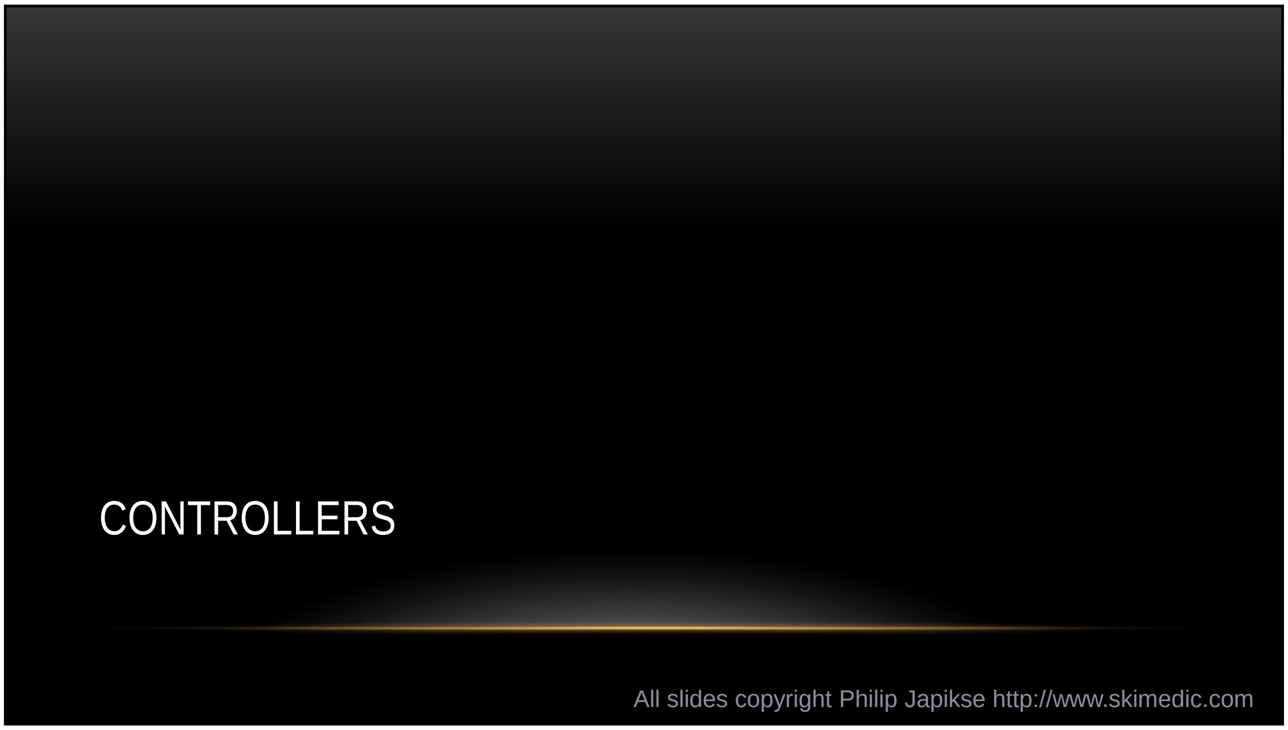
ROUTING



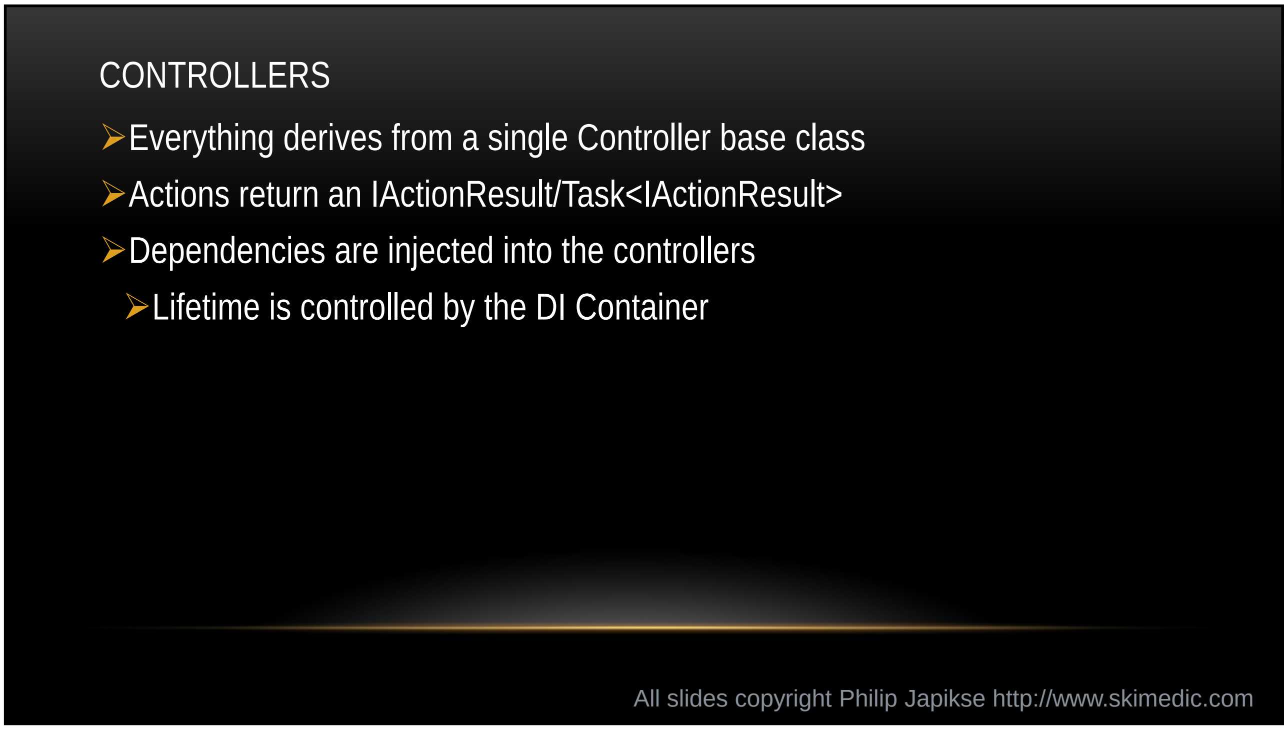
ROUTING

- Attribute Routing is first class citizen in ASP.NET Core
  - Helps to refine routing for individual controller actions
- Route table used for default route
  - Sometimes skipped in ASP.NET Core Service Applications
- Controller and actions can define specific routes

```
[Route("api/[controller]/{customerId}")]
public class ShoppingCartController : Controller
{
    [HttpGet("{Id?}")]
    public IActionResult AddToCart(int Id, int customerId, int quantity = 1)
    {
       //Code omitted
    }
}
```

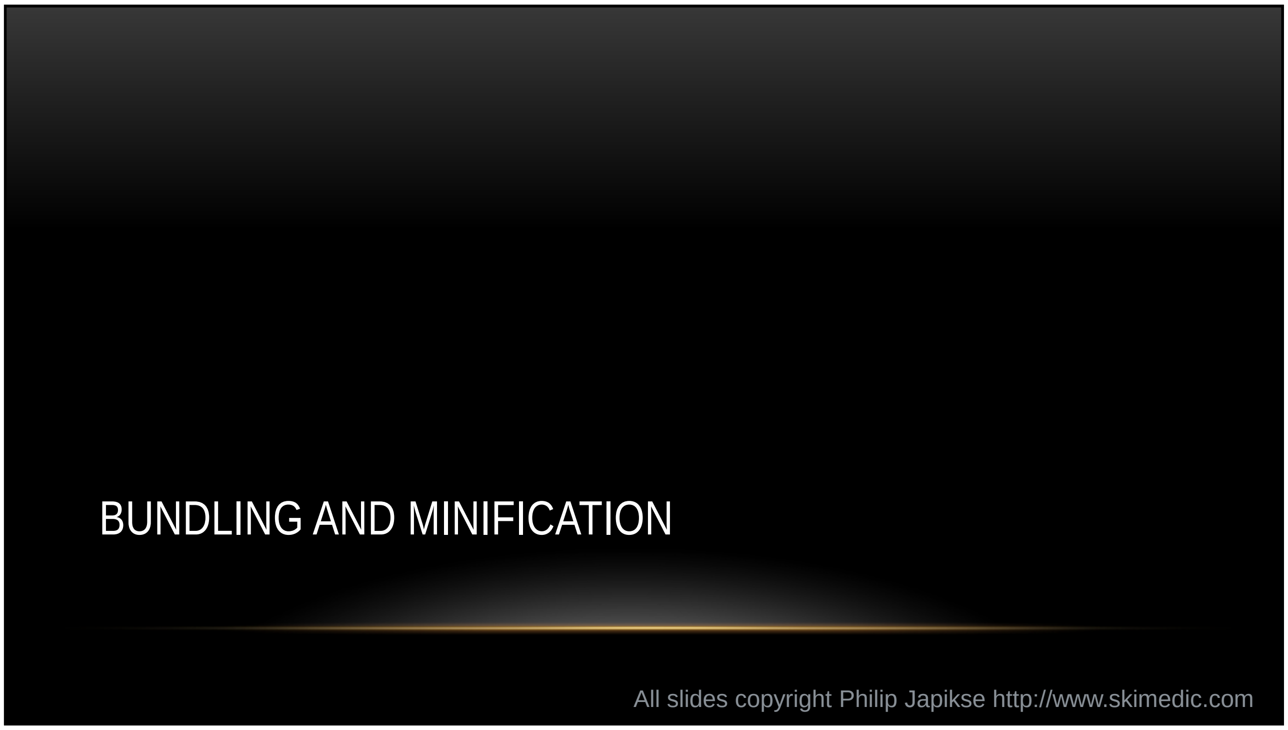


CONTROLLERS

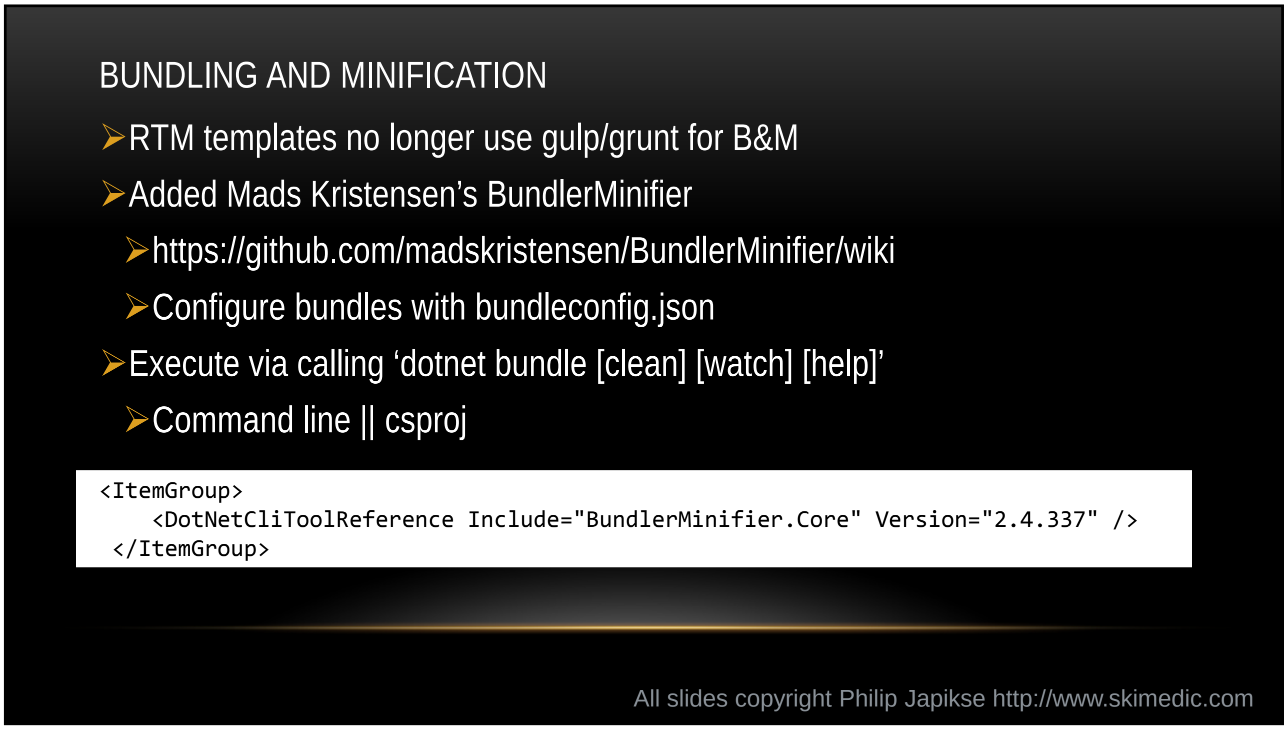


CONTROLLERS

- Everything derives from a single Controller base class
- Actions return an IActionResult/Task<IActionResult>
- Dependencies are injected into the controllers
  - Lifetime is controlled by the DI Container



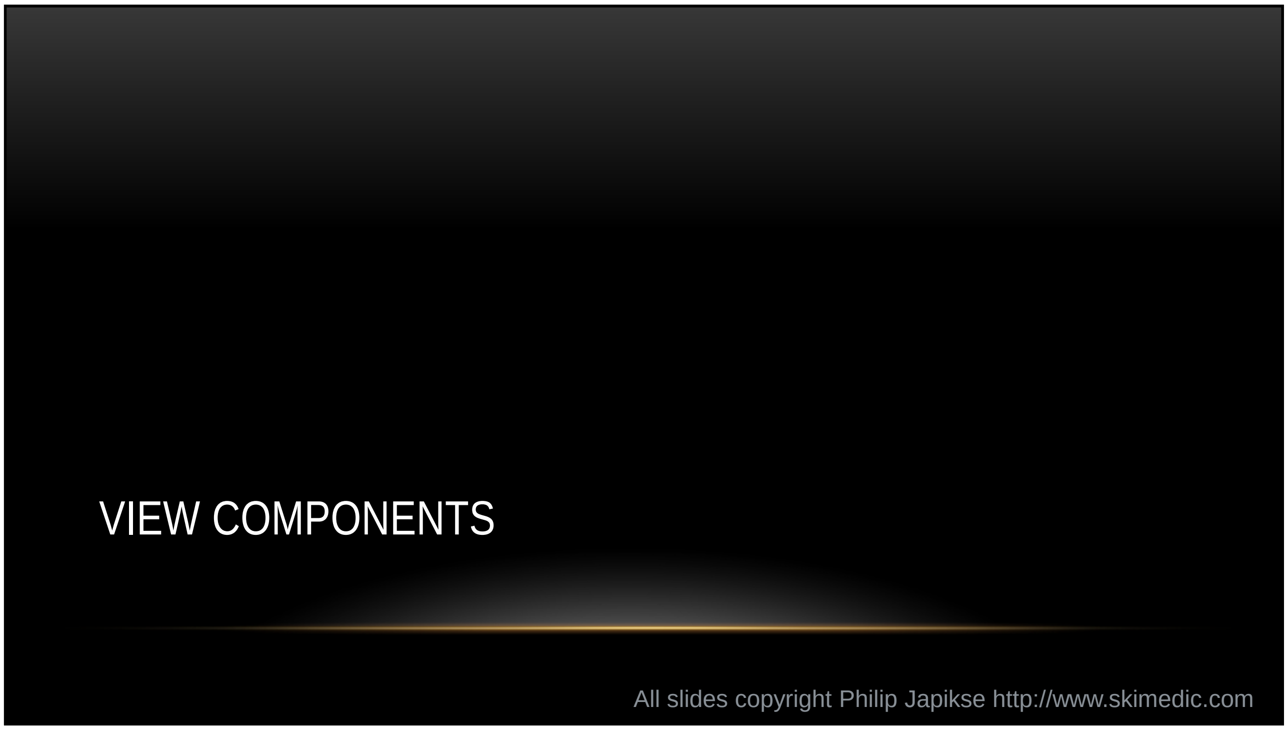
BUNDLING AND MINIFICATION



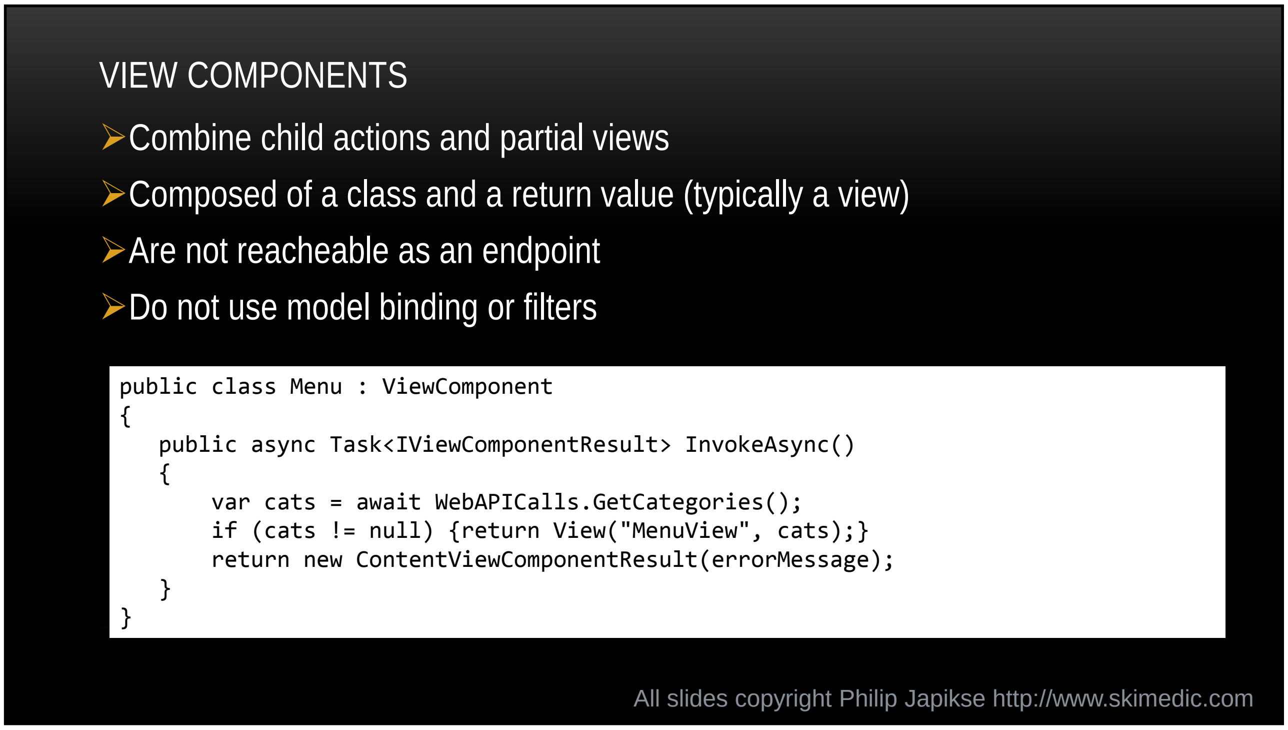
BUNDLING AND MINIFICATION

- RTM templates no longer use gulp/grunt for B&M
- Added Mads Kristensen's BundlerMinifier
  - https://github.com/madskristensen/BundlerMinifier/wiki
  - Configure bundles with bundleconfig.json
- Execute via calling 'dotnet bundle [clean] [watch] [help]'
  - Command line || csproj

```
<ItemGroup>
    <DotNetCliToolReference Include="BundlerMinifier.Core" Version="2.4.337" />
 </ItemGroup>
```



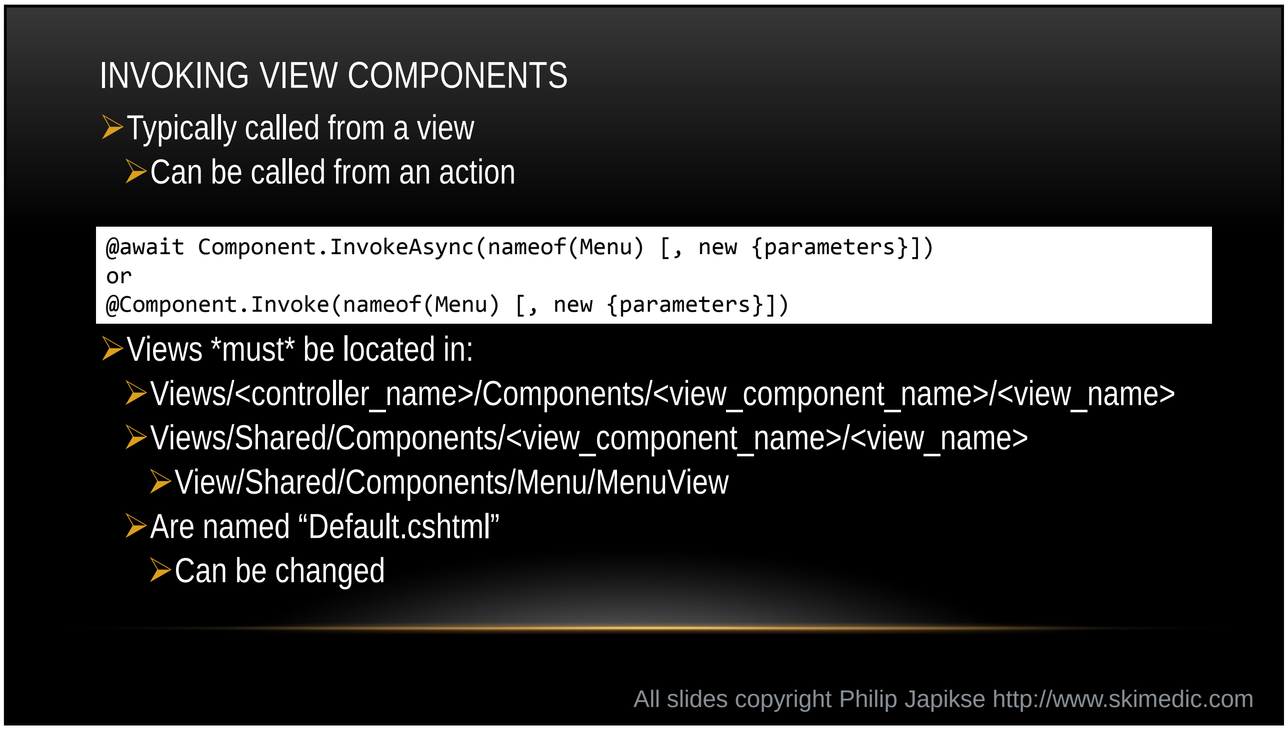
VIEW COMPONENTS



VIEW COMPONENTS

- Combine child actions and partial views
- Composed of a class and a return value (typically a view)
- Are not reacheable as an endpoint
- Do not use model binding or filters

```
public class Menu : ViewComponent
{
   public async Task<IViewComponentResult> InvokeAsync()
   {
       var cats = await WebAPICalls.GetCategories();
       if (cats != null) {return View("MenuView", cats);}
       return new ContentViewComponentResult(errorMessage);
   }
}
```

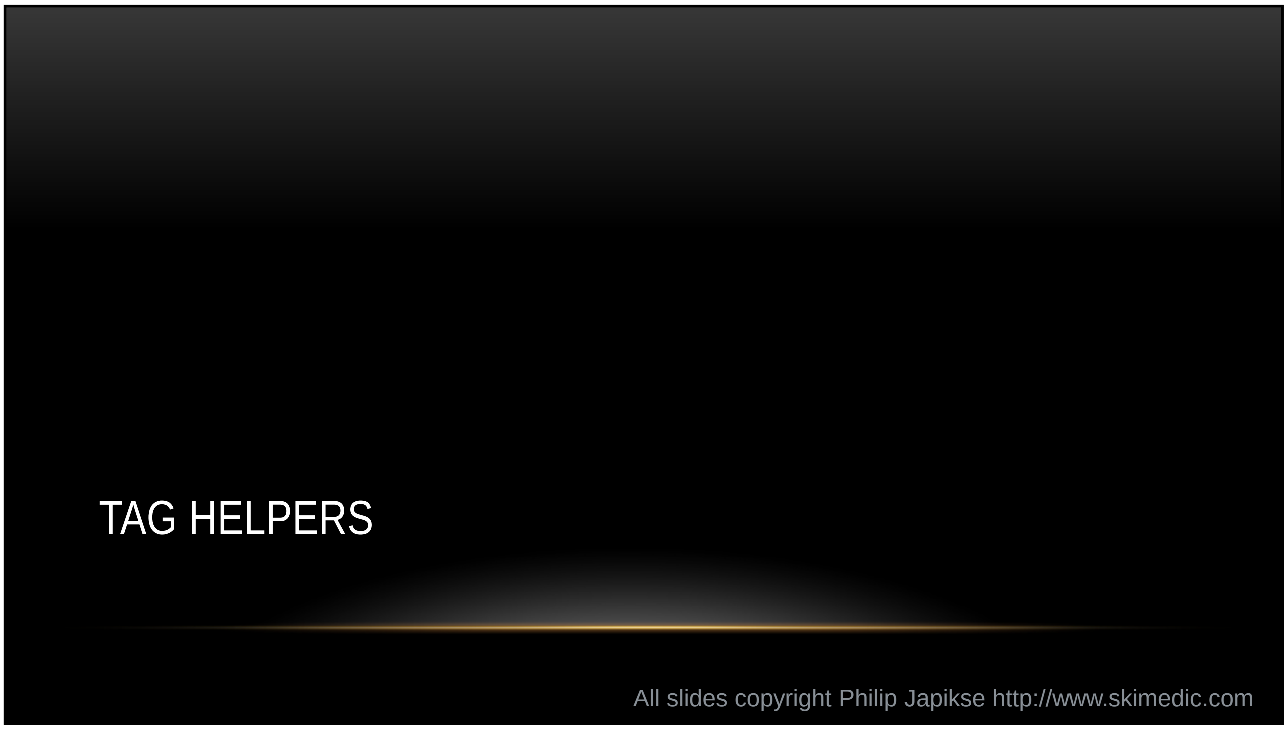


INVOKING VIEW COMPONENTS

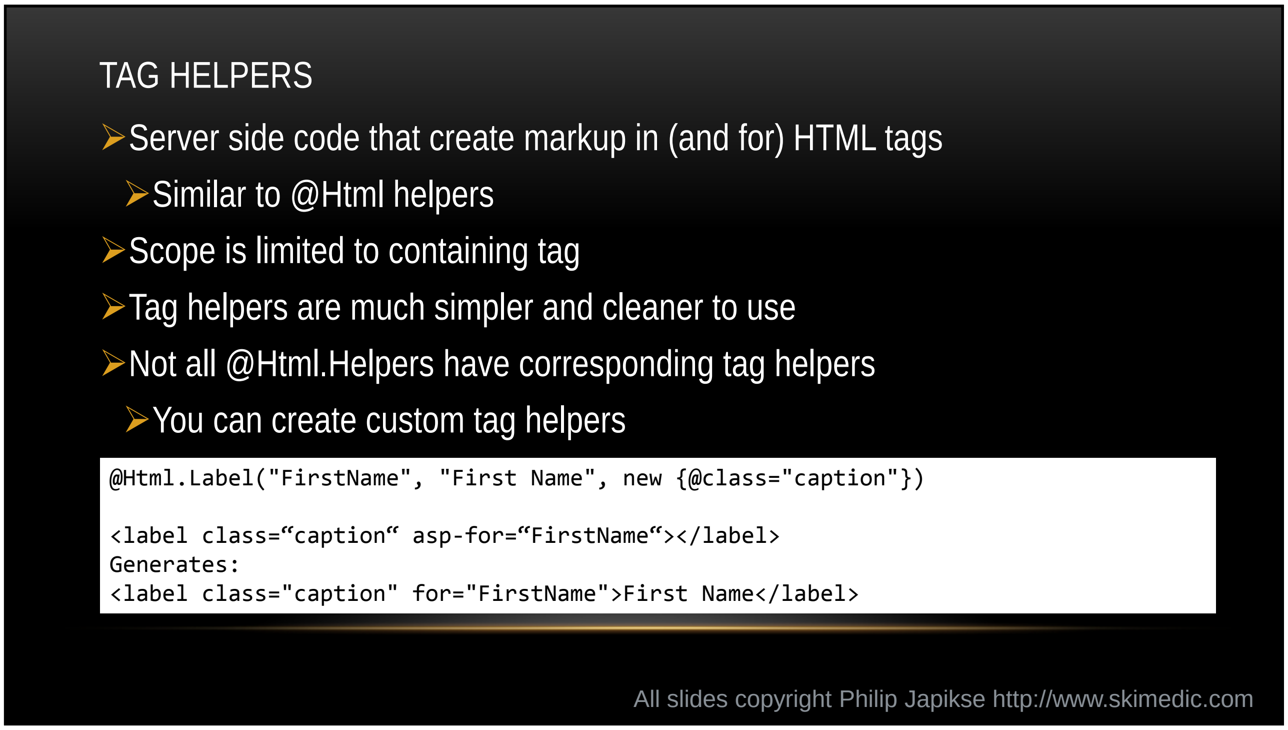
- Typically called from a view
  - Can be called from an action

```
@await Component.InvokeAsync(nameof(Menu) [, new {parameters}])
or
@Component.Invoke(nameof(Menu) [, new {parameters}])
```

- Views *must* be located in:
  - Views/<controller_name>/Components/<view_component_name>/<view_name>
  - Views/Shared/Components/<view_component_name>/<view_name>
    - View/Shared/Components/Menu/MenuView
  - Are named “Default.cshtml”
    - Can be changed



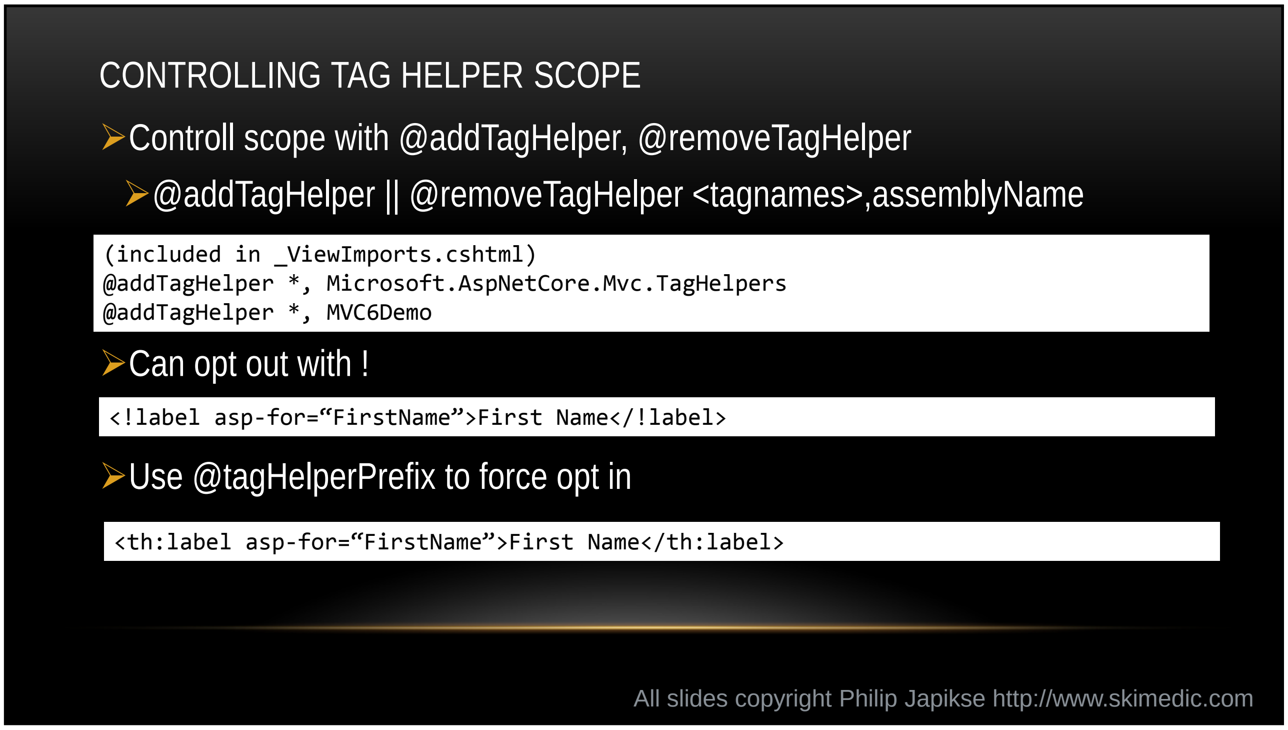
TAG HELPERS



TAG HELPERS

- Server side code that create markup in (and for) HTML tags
  - Similar to @Html helpers
- Scope is limited to containing tag
- Tag helpers are much simpler and cleaner to use
- Not all @Html.Helpers have corresponding tag helpers
  - You can create custom tag helpers

```
@Html.Label("FirstName", "First Name", new {@class="caption"})

<label class=“caption“ asp-for=“FirstName“></label>
Generates:
<label class="caption" for="FirstName">First Name</label>
```



CONTROLLING TAG HELPER SCOPE

- Controll scope with @addTagHelper, @removeTagHelper
  - @addTagHelper || @removeTagHelper <tagnames>,assemblyName

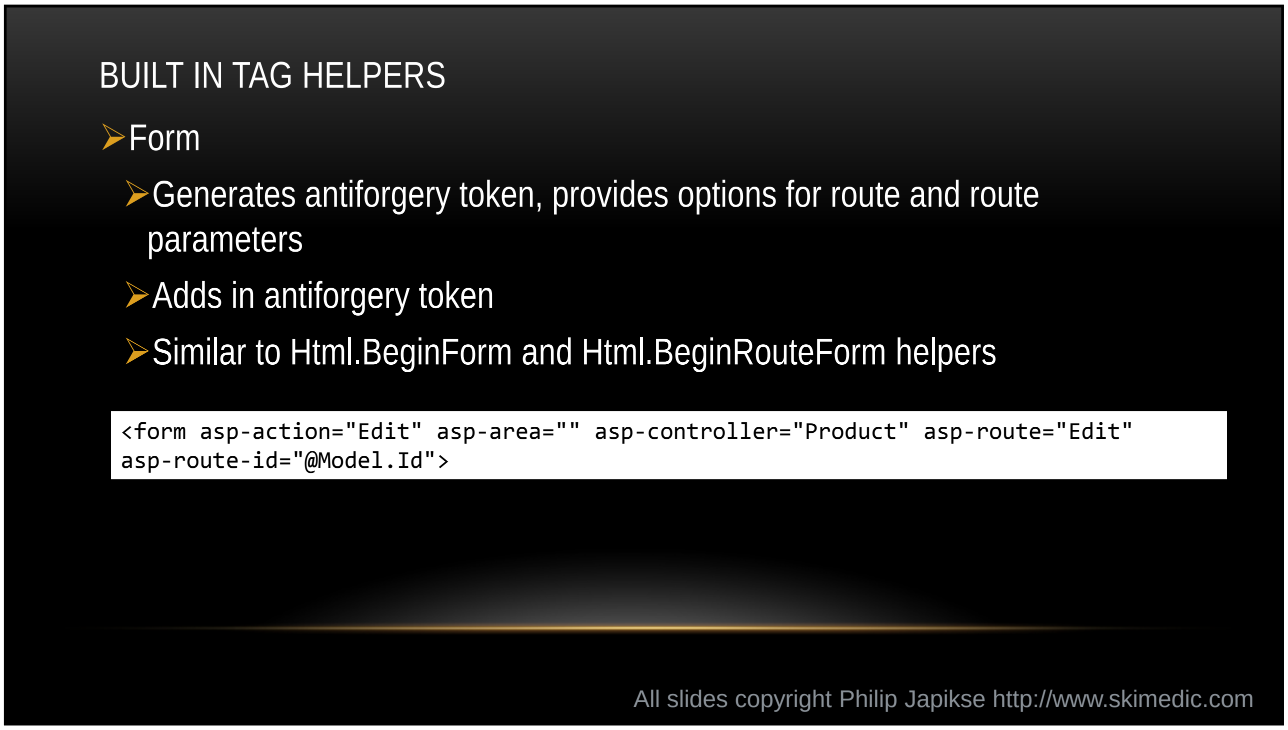
```
(included in _ViewImports.cshtml)
@addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers
@addTagHelper *, MVC6Demo
```

- Can opt out with !

```
<!label asp-for=“FirstName”>First Name</!label>
```

- Use @tagHelperPrefix to force opt in

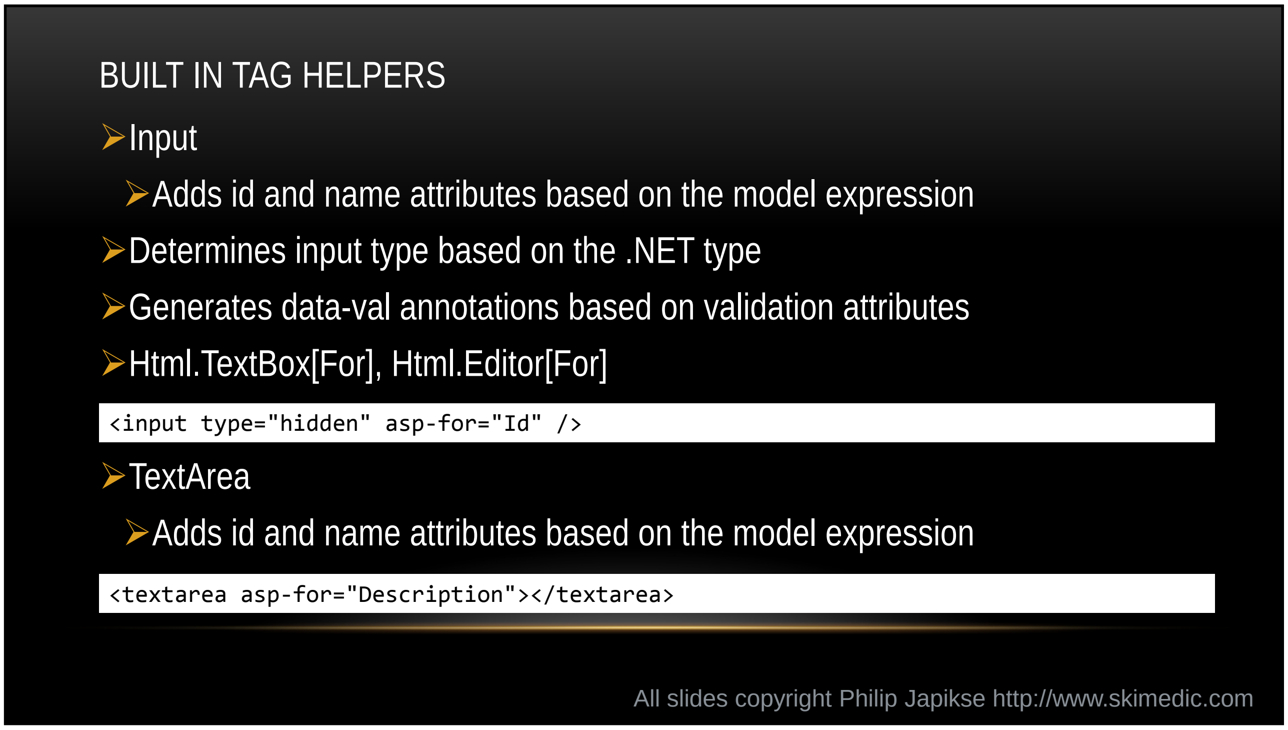
```
<th:label asp-for=“FirstName”>First Name</th:label>
```



BUILT IN TAG HELPERS

- Form
  - Generates antiforgery token, provides options for route and route parameters
  - Adds in antiforgery token
  - Similar to Html.BeginForm and Html.BeginRouteForm helpers

```
<form asp-action="Edit" asp-area="" asp-controller="Product" asp-route="Edit"
asp-route-id="@Model.Id">
```



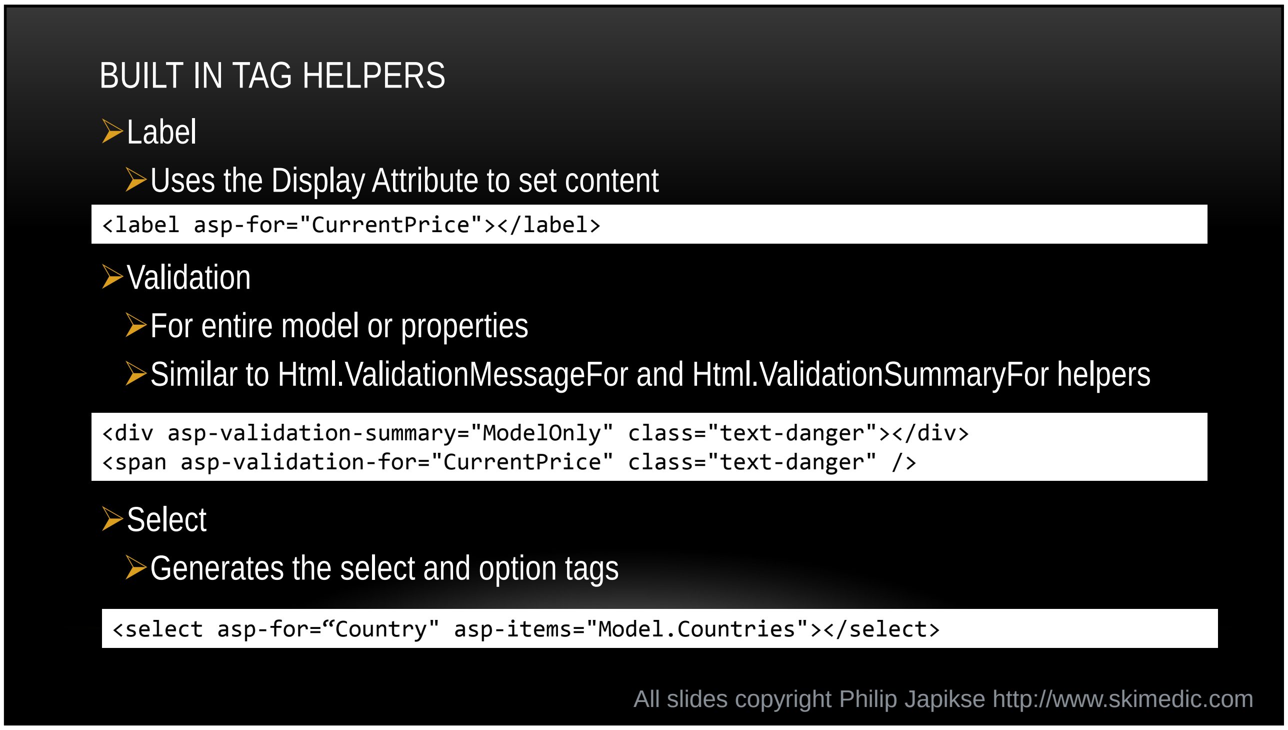
BUILT IN TAG HELPERS

- Input
  - Adds id and name attributes based on the model expression
- Determines input type based on the .NET type
- Generates data-val annotations based on validation attributes
- Html.TextBox[For], Html.Editor[For]

```
<input type="hidden" asp-for="Id" />
```

- TextArea
  - Adds id and name attributes based on the model expression

```
<textarea asp-for="Description"></textarea>
```



BUILT IN TAG HELPERS

- Label
  - Uses the Display Attribute to set content

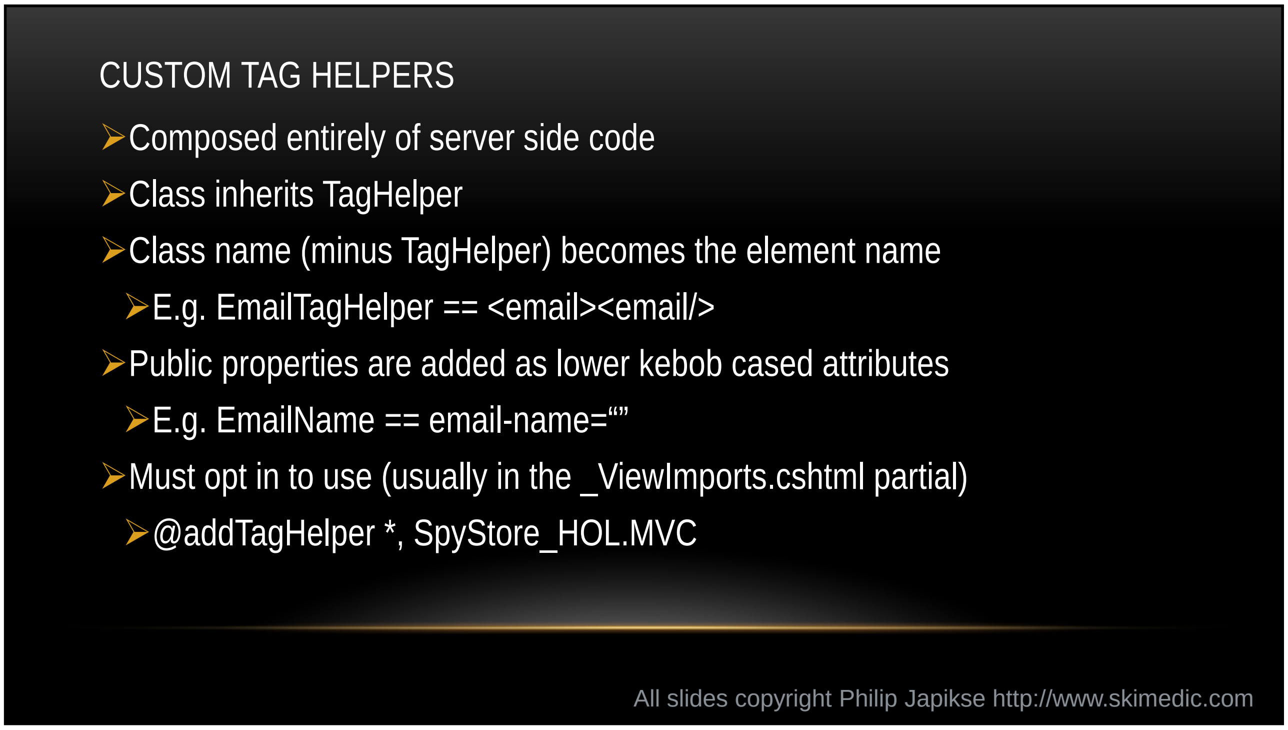
```
<label asp-for="CurrentPrice"></label>
```

- Validation
  - For entire model or properties
  - Similar to Html.ValidationMessageFor and Html.ValidationSummaryFor helpers

```
<div asp-validation-summary="ModelOnly" class="text-danger"></div>
<span asp-validation-for="CurrentPrice" class="text-danger" />
```

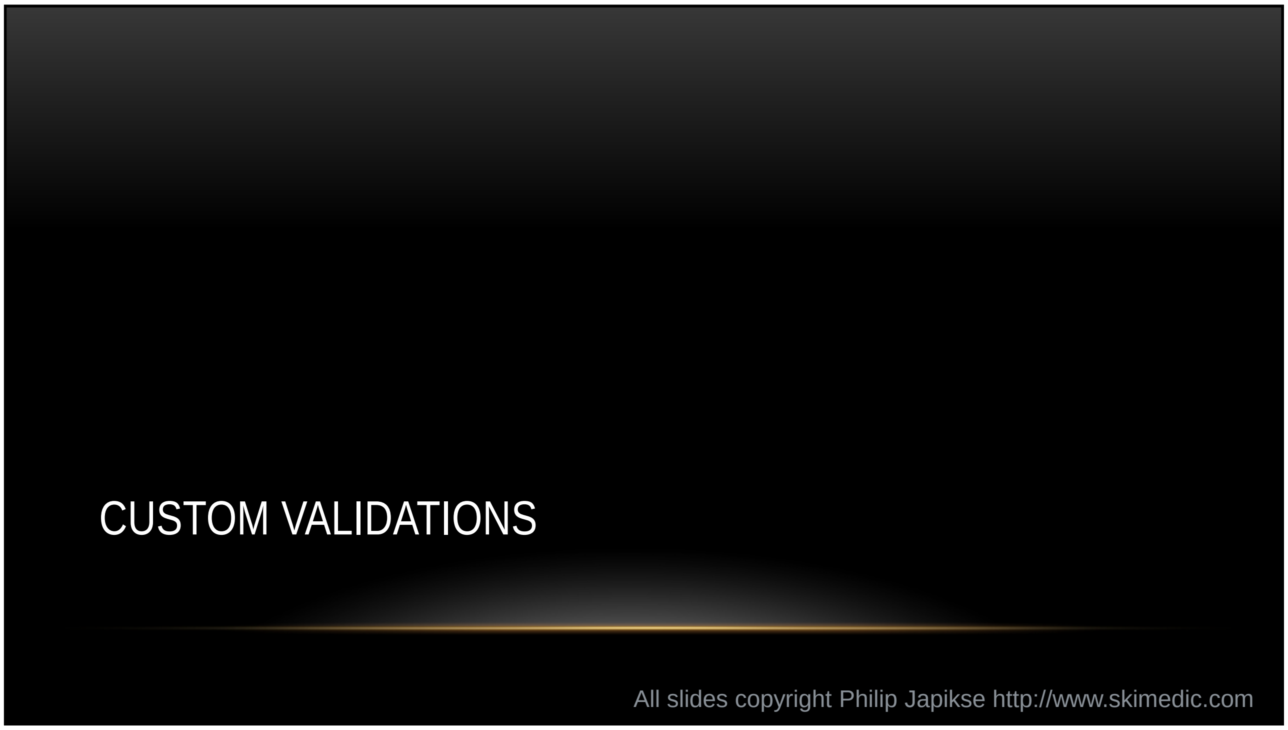
- Select
  - Generates the select and option tags

```
<select asp-for="Country" asp-items="Model.Countries"></select>
```

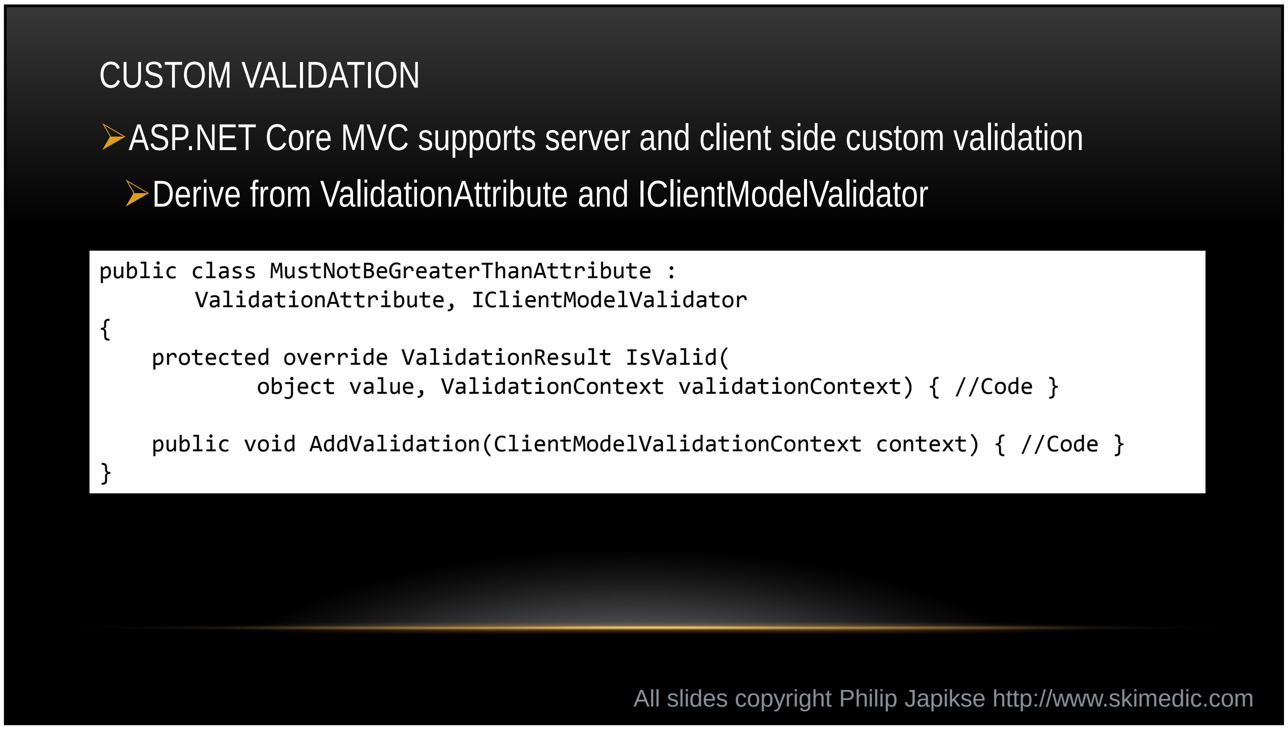


CUSTOM TAG HELPERS

- Composed entirely of server side code
- Class inherits TagHelper
- Class name (minus TagHelper) becomes the element name
  - E.g. EmailTagHelper == <email><email/>
- Public properties are added as lower kebob cased attributes
  - E.g. EmailName == email-name="“”"
- Must opt in to use (usually in the _ViewImports.cshtml partial)
  - @addTagHelper *, SpyStore_HOL.MVC



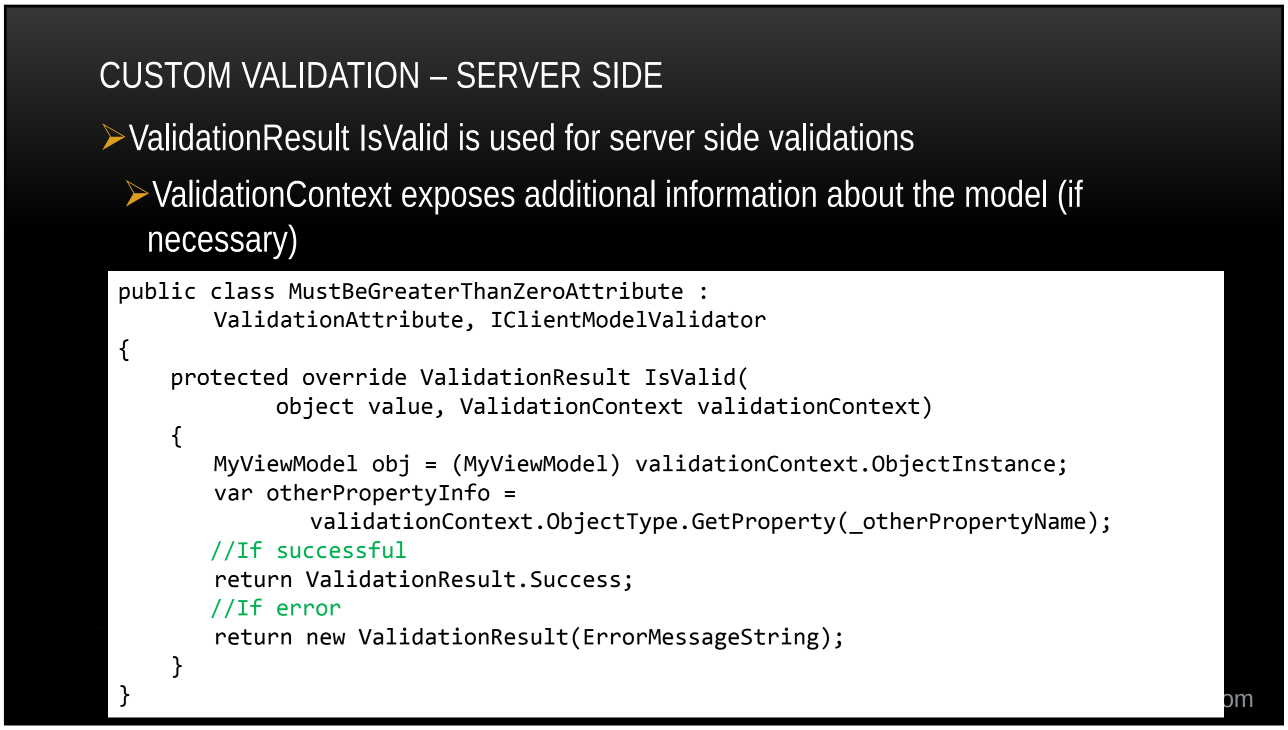
CUSTOM VALIDATIONS



CUSTOM VALIDATION

- ASP.NET Core MVC supports server and client side custom validation
  - Derive from ValidationAttribute and IClientModelValidator

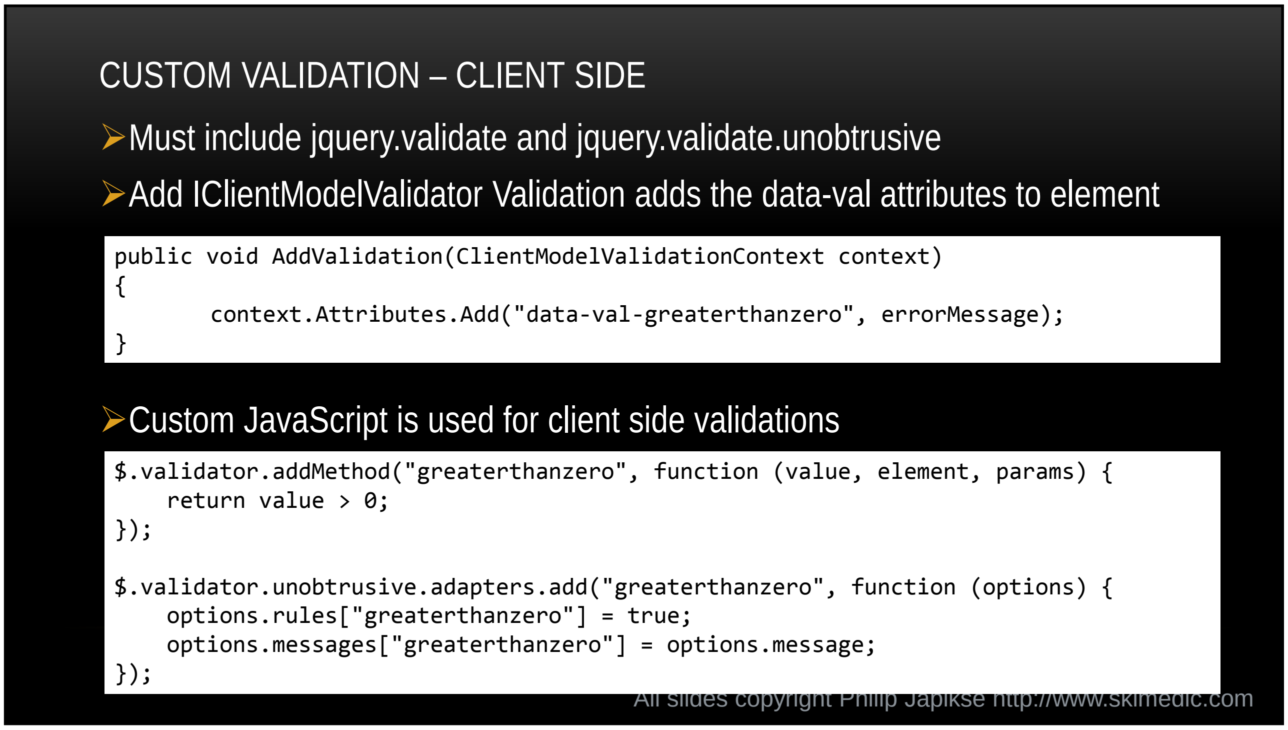
```
public class MustNotBeGreaterThanAttribute :
       ValidationAttribute, IClientModelValidator
{
    protected override ValidationResult IsValid(
            object value, ValidationContext validationContext) { //Code }

    public void AddValidation(ClientModelValidationContext context) { //Code }
}
```



CUSTOM VALIDATION – SERVER SIDE

- ValidationResult IsValid is used for server side validations
  - ValidationContext exposes additional information about the model (if necessary)

```
public class MustBeGreaterThanZeroAttribute :
       ValidationAttribute, IClientModelValidator
{
    protected override ValidationResult IsValid(
            object value, ValidationContext validationContext)
    {
       MyViewModel obj = (MyViewModel) validationContext.ObjectInstance;
       var otherPropertyInfo =
              validationContext.ObjectType.GetProperty(_otherPropertyName);
       //If successful
       return ValidationResult.Success;
       //If error
       return new ValidationResult(ErrorMessageString);
    }
}
```



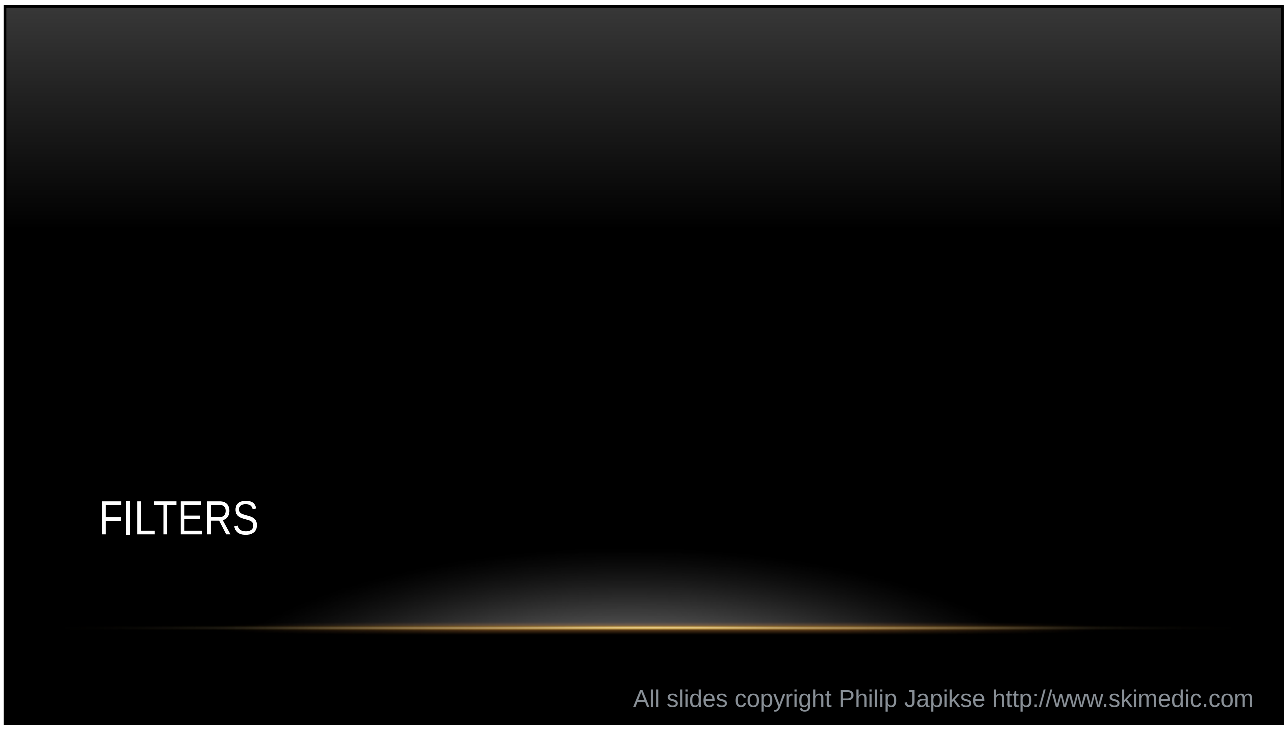
CUSTOM VALIDATION – CLIENT SIDE

- Must include jquery.validate and jquery.validate.unobtrusive
- Add IClientModelValidator Validation adds the data-val attributes to element

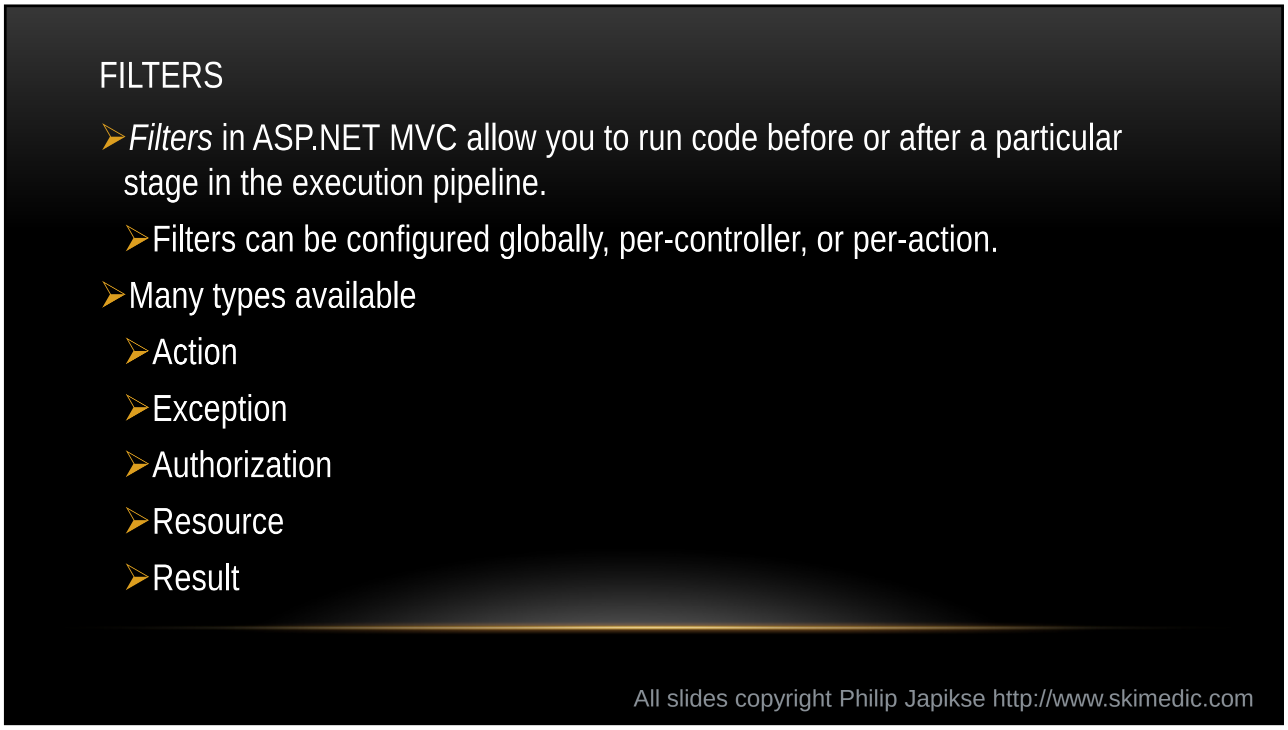
```
public void AddValidation(ClientModelValidationContext context)
{
       context.Attributes.Add("data-val-greaterthanzero", errorMessage);
}
```

- Custom JavaScript is used for client side validations

```
$.validator.addMethod("greaterthanzero", function (value, element, params) {
    return value > 0;
});

$.validator.unobtrusive.adapters.add("greaterthanzero", function (options) {
    options.rules["greaterthanzero"] = true;
    options.messages["greaterthanzero"] = options.message;
});
```

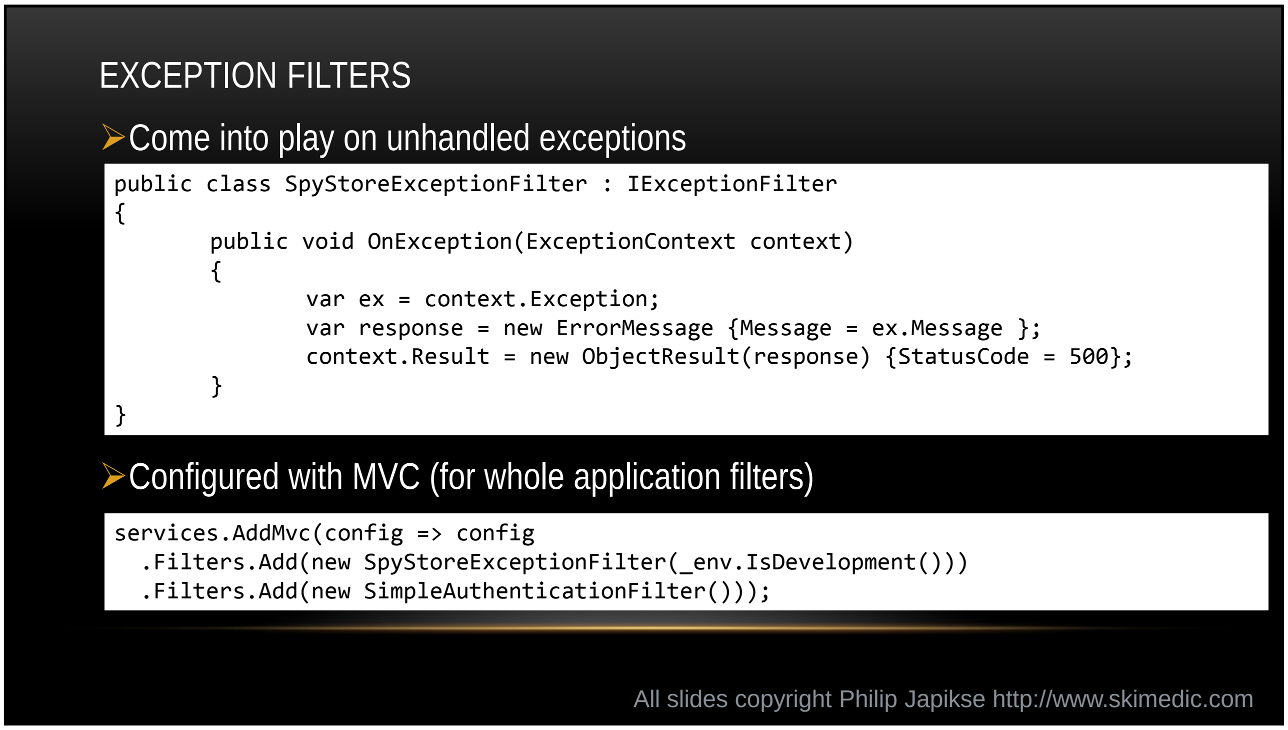


FILTERS



FILTERS

- *Filters* in ASP.NET MVC allow you to run code before or after a particular stage in the execution pipeline.
  - Filters can be configured globally, per-controller, or per-action.
- Many types available
  - Action
  - Exception
  - Authorization
  - Resource
  - Result



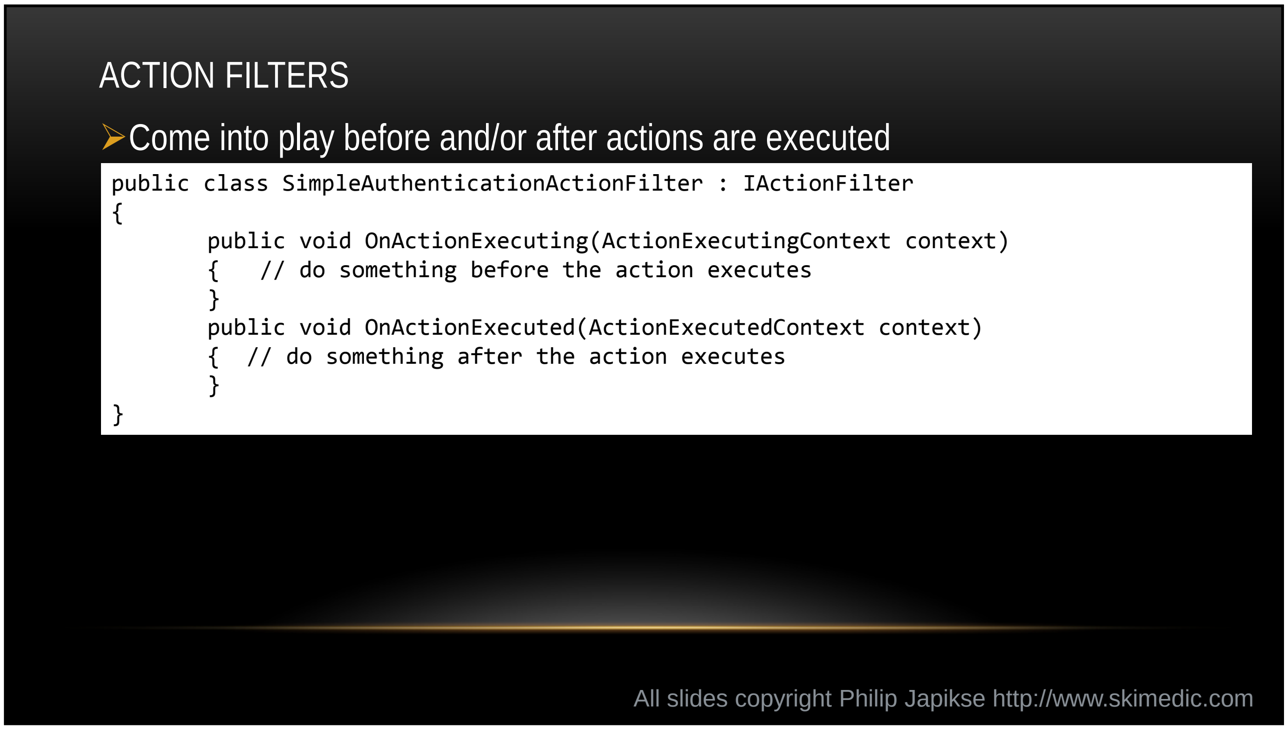
EXCEPTION FILTERS

- Come into play on unhandled exceptions

```
public class SpyStoreExceptionFilter : IExceptionFilter
{
        public void OnException(ExceptionContext context)
        {
                var ex = context.Exception;
                var response = new ErrorMessage {Message = ex.Message };
                context.Result = new ObjectResult(response) {StatusCode = 500};
        }
}
```

- Configured with MVC (for whole application filters)

```
services.AddMvc(config => config
  .Filters.Add(new SpyStoreExceptionFilter(_env.IsDevelopment()))
  .Filters.Add(new SimpleAuthenticationFilter()));
```



ACTION FILTERS

- Come into play before and/or after actions are executed

```
public class SimpleAuthenticationActionFilter : IActionFilter
{
        public void OnActionExecuting(ActionExecutingContext context)
        {   // do something before the action executes
        }
        public void OnActionExecuted(ActionExecutedContext context)
        {  // do something after the action executes
        }
}
```

Contact Me

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

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

www.hallwayconversations.com

Thank You!

Find the code at: https://github.com/skimedic/dotnetcore_hol/tree/master/2.0