

# Chapter 12: Finding your way around ASP.NET Core

Gill Cleeren  
@gillcleeren

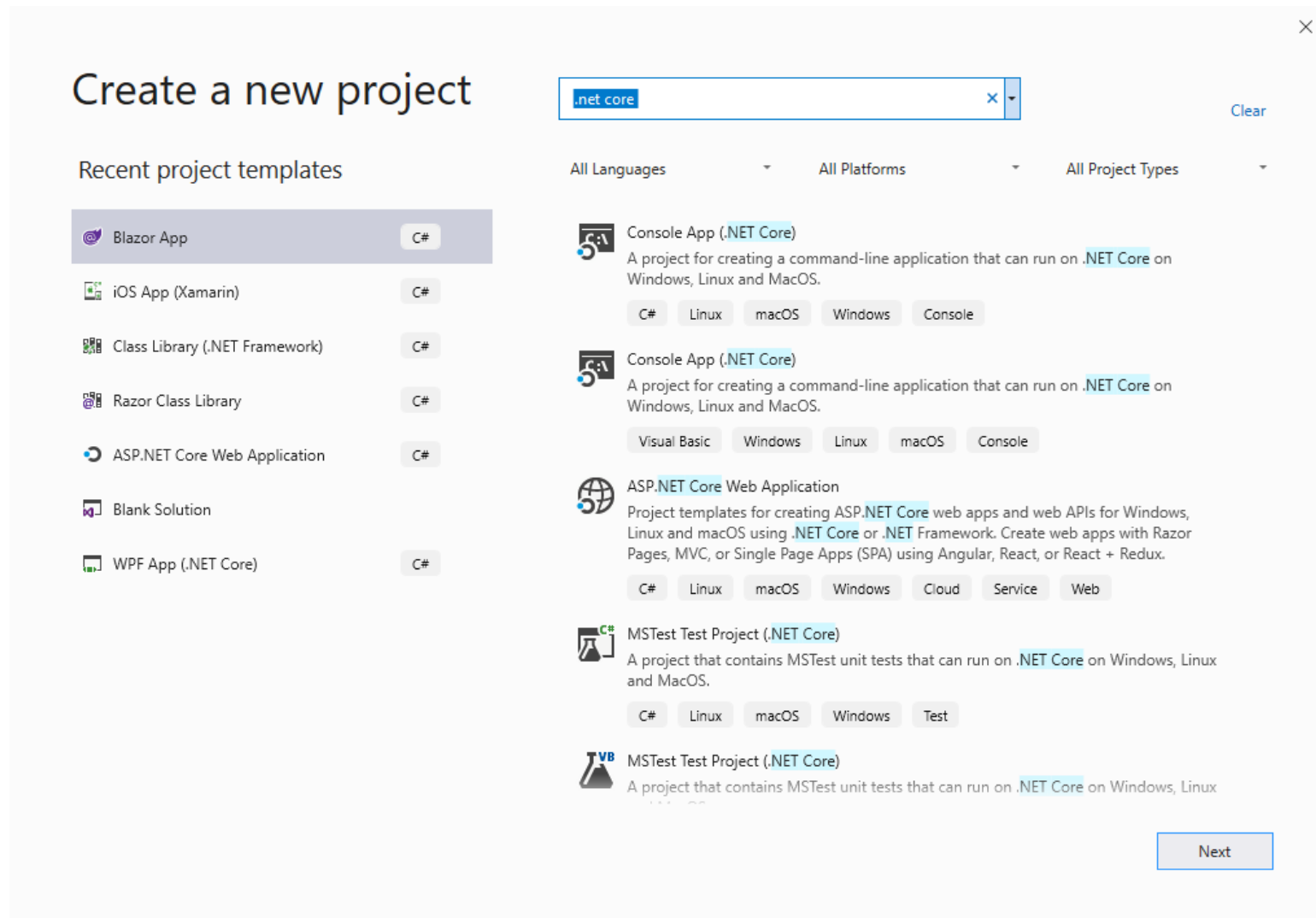
# Agenda

- File → New Project
- Project Structure
- Application configuration



File → New Project

# File → New Project



# Create a new ASP.NET Core web application

.NET Core

ASP.NET Core 3.0



## Empty

An empty project template for creating an ASP.NET Core application. This template does not have any content in it.



## API

A project template for creating an ASP.NET Core application with an example Controller for a RESTful HTTP service. This template can also be used for ASP.NET Core MVC Views and Controllers.



## Web Application

A project template for creating an ASP.NET Core application with example ASP.NET Razor Pages content.



## Web Application (Model-View-Controller)

A project template for creating an ASP.NET Core application with example ASP.NET Core MVC Views and Controllers. This template can also be used for RESTful HTTP services.



## Angular

A project template for creating an ASP.NET Core application with Angular



## React.js

A project template for creating an ASP.NET Core application with React.js

[Get additional project templates](#)

## Authentication

No Authentication

[Change](#)

## Advanced

☒ Configure for HTTPS

☐ Enable Docker Support  
(Requires [Docker Desktop](#))

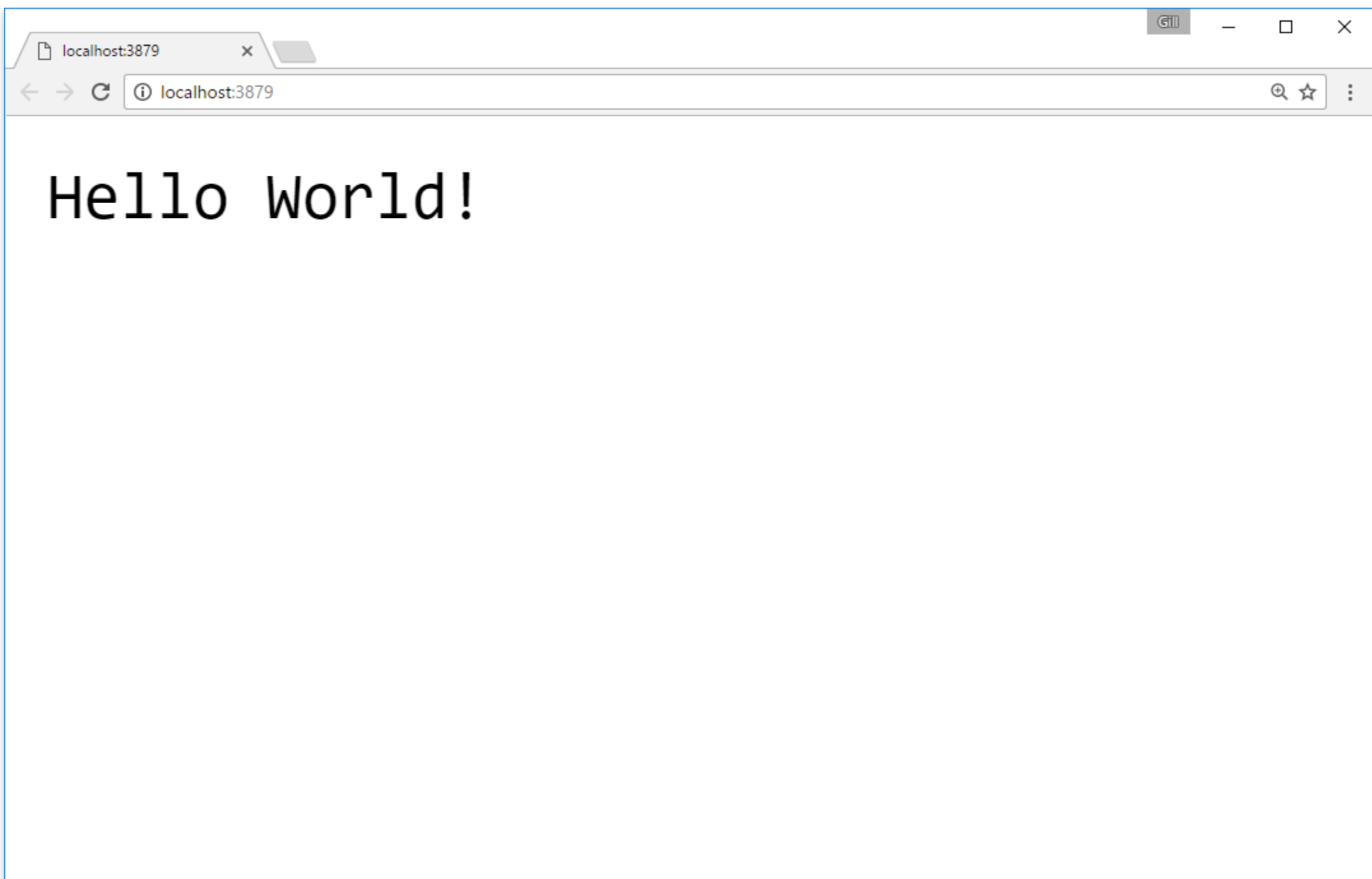
Linux

Author: Microsoft

Source: .NET Core 3.0.0

Back

Create



# Welcome

Learn about [building Web apps with ASP.NET Core](#).



# DEMO

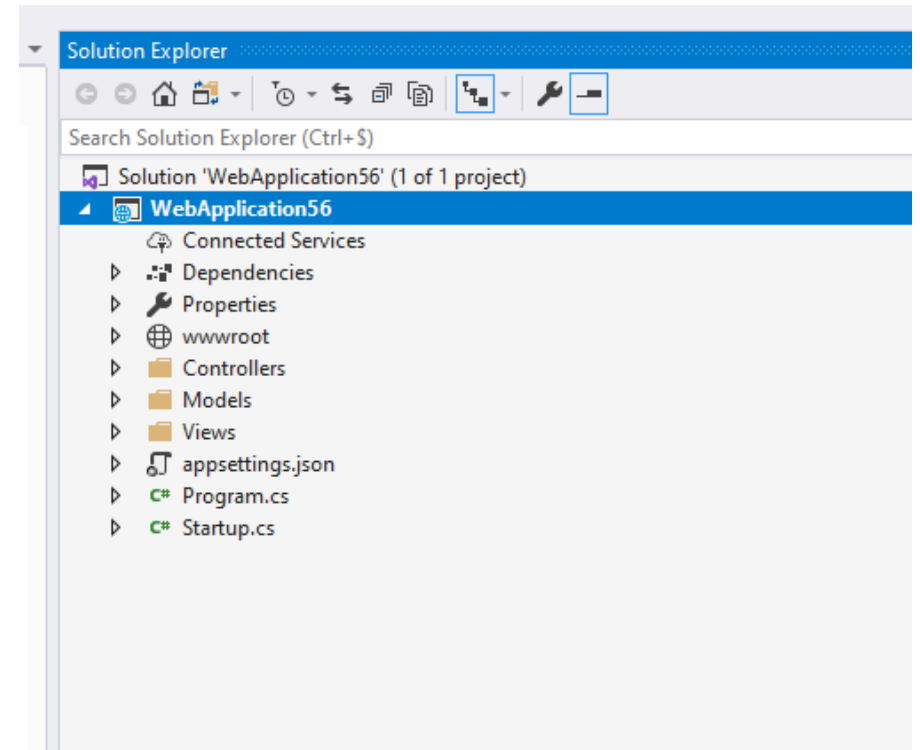
Project Templates



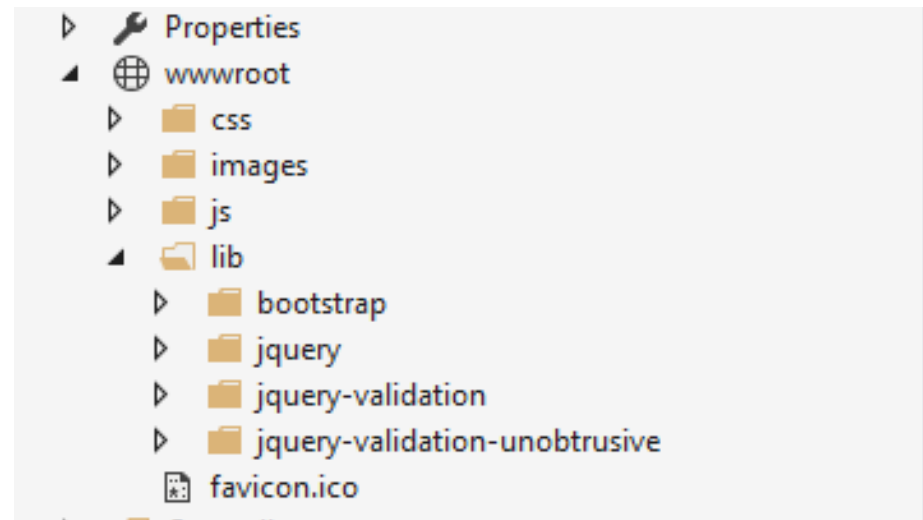
# Project Structure

# Generated Project Structure

- Important things to note
  - wwwroot
  - json files
    - bower.json (only in “older” versions... see later)
    - appsettings.json
  - No web.config...
  - Program.cs
  - Startup.cs
  - Regular folders
    - Controllers
    - Views
    - Models

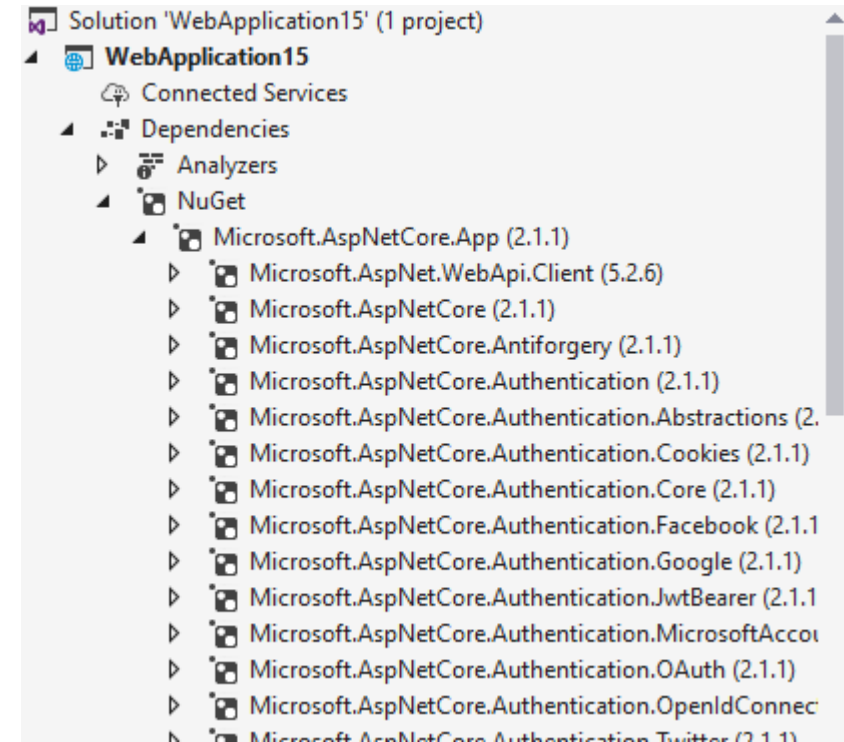
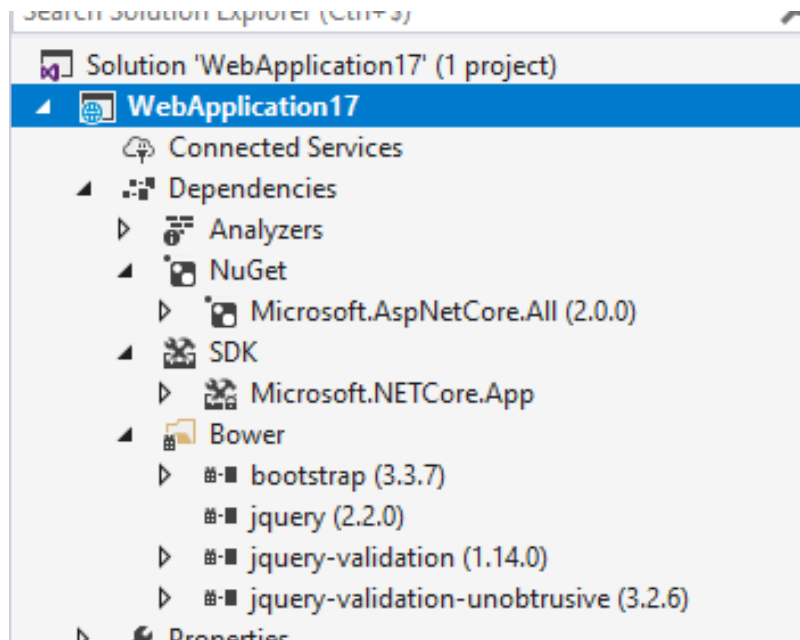


# wwwroot



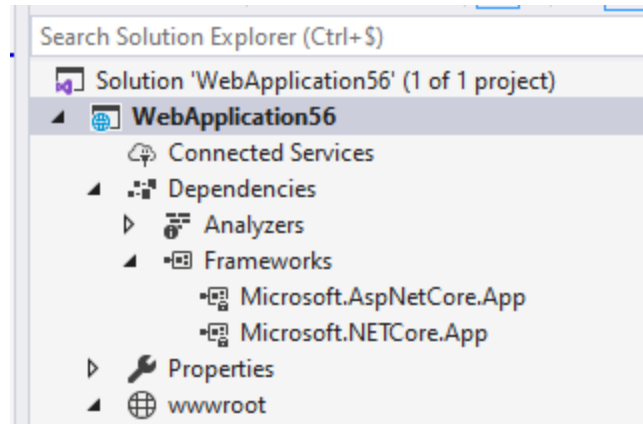
# Dependencies

- 2.0 and 2.1 are slightly different in terms of package names

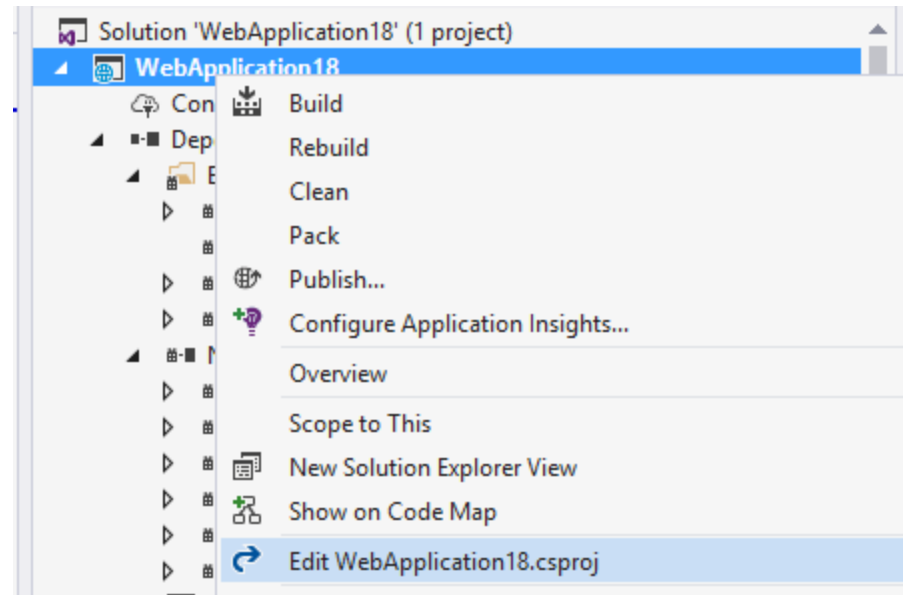


# Dependencies

- ASP.NET Core 3.0 doesn't use NuGet packages anymore...



# Managing dependencies: csproj file



# Managing dependencies: csproj file (.NET Core 1.1)

```
<Project Sdk="Microsoft.NET.Sdk.Web">

  <PropertyGroup>
    <TargetFramework>netcoreapp1.1</TargetFramework>
  </PropertyGroup>

  <PropertyGroup>
    <PackageTargetFallback>$(PackageTargetFallback);portable-net45+win8+wp8+wpa81;</PackageTargetFallback>
  </PropertyGroup>

  <ItemGroup>
    <PackageReference Include="Microsoft.ApplicationInsights.AspNetCore" Version="2.0.0" />
    <PackageReference Include="Microsoft.AspNetCore" Version="1.1.2" />
    <PackageReference Include="Microsoft.AspNetCore.Mvc" Version="1.1.3" />
    <PackageReference Include="Microsoft.AspNetCore.StaticFiles" Version="1.1.2" />
    <PackageReference Include="Microsoft.Extensions.Logging.Debug" Version="1.1.2" />
    <PackageReference Include="Microsoft.VisualStudio.Web.BrowserLink" Version="1.1.2" />
  </ItemGroup>

  <ItemGroup>
    <DotNetCliToolReference Include="Microsoft.VisualStudio.Web.CodeGeneration.Tools" Version="1.0.1" />
  </ItemGroup>

</Project>
```

# Managing dependencies: csproj file (.NET Core 2.0)

```
<Project Sdk="Microsoft.NET.Sdk.Web">

  <PropertyGroup>
    <TargetFramework>netcoreapp2.0</TargetFramework>
  </PropertyGroup>

  <ItemGroup>
    <PackageReference Include="Microsoft.AspNetCore.All" Version="2.0.0" />
  </ItemGroup>

  <ItemGroup>
    <DotNetCliToolReference Include="Microsoft.VisualStudio.Web.CodeGeneration.Tools" Version="2.0.0" />
  </ItemGroup>

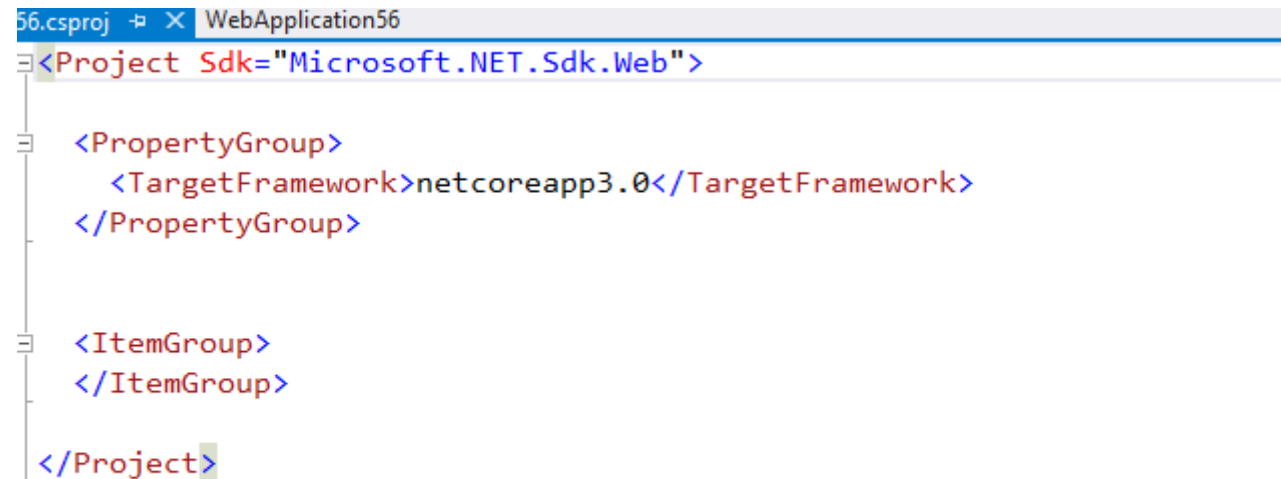
</Project>
```



# Managing dependencies: csproj file (.NET Core 2.1)

```
1 <Project Sdk="Microsoft.NET.Sdk.Web">
2
3   <PropertyGroup>
4     <TargetFramework>netcoreapp2.1</TargetFramework>
5   </PropertyGroup>
6
7   <ItemGroup>
8     <PackageReference Include="Microsoft.AspNetCore.App" />
9     <PackageReference Include="Microsoft.AspNetCore.Razor.Design" Version="2.1.2" PrivateAssets="All" />
10  </ItemGroup>
11
12 </Project>
```

# Managing dependencies: csproj file (.NET Core 3.0)



```
56.csproj  WebApplication56
<Project Sdk="Microsoft.NET.Sdk.Web">
  <PropertyGroup>
    <TargetFramework>netcoreapp3.0</TargetFramework>
  </PropertyGroup>
  <ItemGroup>
  </ItemGroup>
</Project>
```



# DEMO

Exploring the new project

Exploring the added packages

Application configuration

# Base concepts in ASP.NET Core 1.1

- An ASP.NET Core app is a console app that creates a web server in its Main method
  - MVC will get added soon!

```
0 references
public class Program
{
    0 references | 0 exceptions
    public static void Main(string[] args)
    {
        var host = new WebHostBuilder()
            .UseKestrel()
            .UseContentRoot(Directory.GetCurrentDirectory())
            .UseIISIntegration()
            .UseStartup<Startup>()
            .UseApplicationInsights()
            .Build();

        host.Run();
    }
}
```

# Base concepts in ASP.NET Core 2.X

```
0 references
public class Program
{
    0 references | 0 exceptions
    public static void Main(string[] args)
    {
        BuildWebHost(args).Run();
    }

    1 reference | 0 exceptions
    public static IWebHost BuildWebHost(string[] args) =>
        WebHost.CreateDefaultBuilder(args)
            .UseStartup<Startup>()
            .Build();
}
```

# Base concepts in ASP.NET Core 3.X

- HostBuilder instead of WebHostBuilder

```
public class Program
{
    public static void Main(string[] args)
    {
        CreateHostBuilder(args).Build().Run();
    }

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

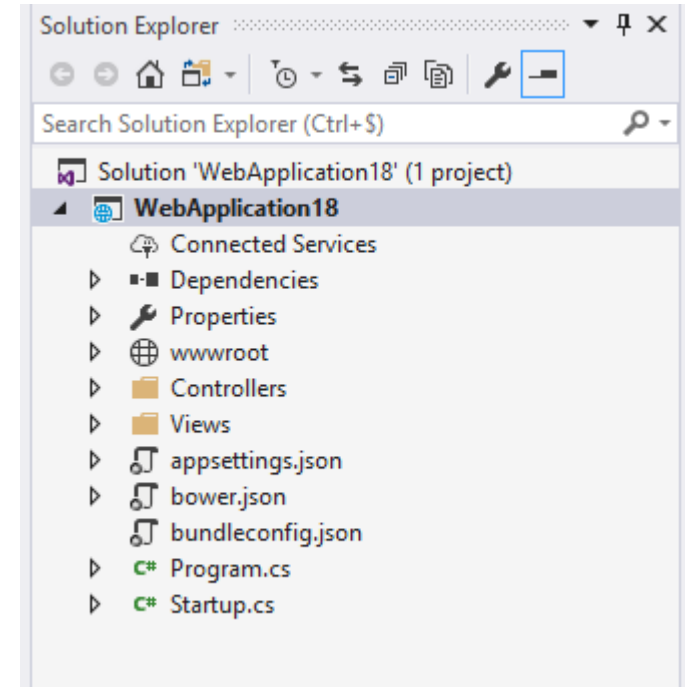
# Running your site

- Different options
  - Kestrel
  - IIS
  - Other options are available



# Startup class

- The UseStartup method on WebHostBuilder specifies the Startup class for your app
  - Use to define request handling pipeline and services for the app are configured
- Contains 2 important methods
  - Configure: used to configure the middleware in the request pipeline
  - ConfigureServices: defines services that we'll be using in the app
    - MVC, EF, Logging...



# Startup method in Startup class

- Changed slightly in ASP.NET Core 2/3 as well
  - See the demo for this

```
0 references | 0 exceptions
public Startup(IHostingEnvironment env)
{
    var builder = new ConfigurationBuilder()
        .SetBasePath(env.ContentRootPath)
        .AddJsonFile("appsettings.json", optional: false, reloadOnChange: true)
        .AddJsonFile($"appsettings.{env.EnvironmentName}.json", optional: true)
        .AddEnvironmentVariables();
    Configuration = builder.Build();
}
```

# Startup class

- Note: this is dependency injection at work

```
public class Startup
{
    public void ConfigureServices(IServiceCollection services)
    {
    }

    public void Configure(IApplicationBuilder app)
    {
    }
}
```

# Services in ASP.NET Core MVC

- Service is a component for usage in an application
- Are injected and made available through DI
  - ASP.NET Core comes with built-in DI container
  - Can be replaced if needed
- Supports more loosely-coupled architecture
- Components are available throughout the application using this container
- Typically, things like logging are injected this way

# ConfigureServices

```
public void ConfigureServices(IServiceCollection services)
{
    // Add framework services.
    services.AddDbContext<ApplicationDbContext>(options =>
        options.UseSqlServer(Configuration.GetConnectionString("DefaultConnection")));

    services.AddIdentity<ApplicationUser, IdentityRole>()
        .AddEntityFrameworkStores<ApplicationDbContext>()
        .AddDefaultTokenProviders();

    services.AddMvc();

    // Add application services.
    services.AddTransient<IEmailSender, AuthMessageSender>();
    services.AddTransient<ISmsSender, AuthMessageSender>();
}
```

# Middleware

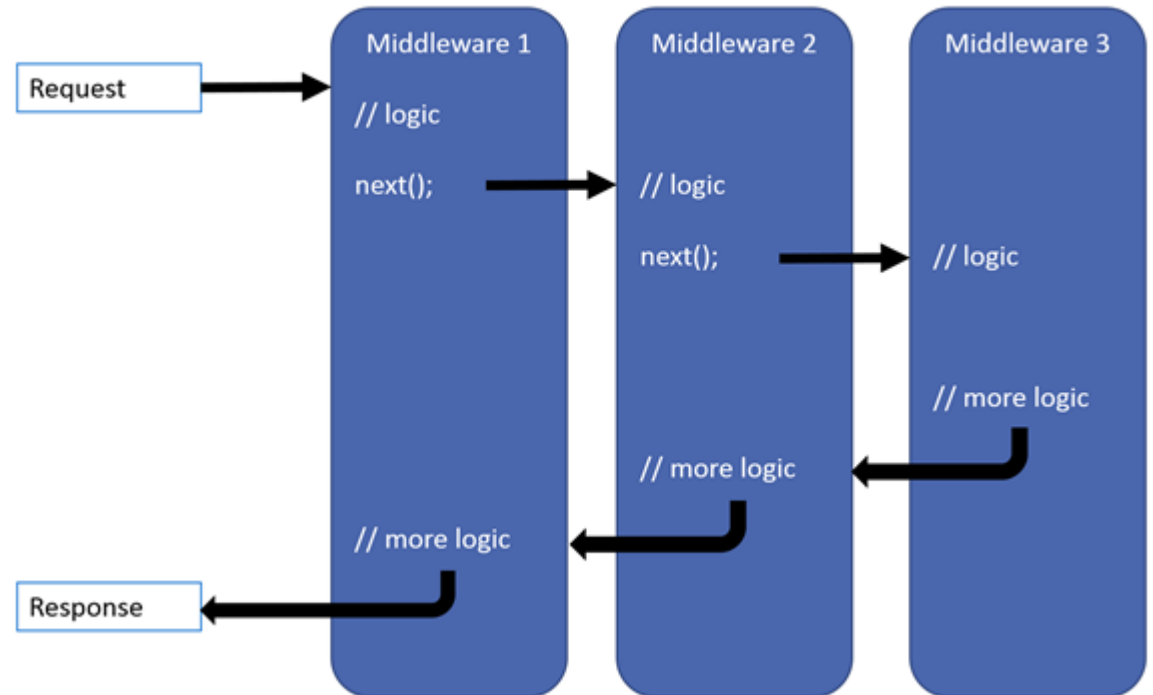
- Request pipeline gets composed using middleware
- ASP.NET Core middleware performs asynchronous logic on an HttpContext and then either invokes the next middleware in the sequence or terminates the request directly
- Typically used by adding a reference to a NuGet package
- Followed by calling UseXXX() on IApplicationBuilder in the Configure method

# Middleware

- ASP.NET Core comes with lots of built-in middleware
  - Static file support
  - Routing
  - Authentication
  - MVC...
- You can also write your own middleware
  - Outside of scope of this course though

# Request pipeline

- Requests are passed from delegate to delegate
- Can be short-circuited
  - Static files
  - Keeps load on server lower





# Most simple pipeline

```
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.AspNetCore.Http;

public class Startup
{
    public void Configure(IApplicationBuilder app)
    {
        app.Run(async context =>
        {
            await context.Response.WriteAsync("Hello, World!");
        });
    }
}
```

# Configure

- Ex: UseMvc
  - Adds routing
  - Configures MVC as default handler
- Parameters are injected using DI

```
// This method gets called by the runtime. Use this method to configure the HTTP request pipeline.  
References  
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)  
{  
    if (env.IsDevelopment())  
    {  
        app.UseDeveloperExceptionPage();  
    }  
    else  
    {  
        app.UseExceptionHandler("/Home/Error");  
        // The default HSTS value is 30 days. You may want to change this for production scenarios, see https://aka.ms/aspnetcore-hsts.  
        app.UseHsts();  
    }  
    app.UseHttpsRedirection();  
    app.UseStaticFiles();  
  
    app.UseRouting();  
  
    app.UseAuthorization();  
  
    app.UseEndpoints(endpoints =>  
    {  
        endpoints.MapControllerRoute(  
            name: "default",  
            pattern: "{controller=Home}/{action=Index}/{id?}");  
    });  
}
```

# Ordering middleware components

- Order in which they are added will decide how they are called!
  - Critical for security, performance and even functionality
- Default order
  - Exception/error handling
  - Static file server
  - Authentication
  - MVC

```
public void Configure(IApplicationBuilder app)
{
    app.UseExceptionHandler("/Home/Error");
    // Call first to catch exceptions thrown in the
    // following middleware.

    app.UseStaticFiles();
    // Return static files and end pipeline.

    app.UseIdentity();
    // Authenticate before you access secure resources.

    app.UseMvcWithDefaultRoute();
    // Add MVC to the request pipeline.
}
```

# Ordering middleware components

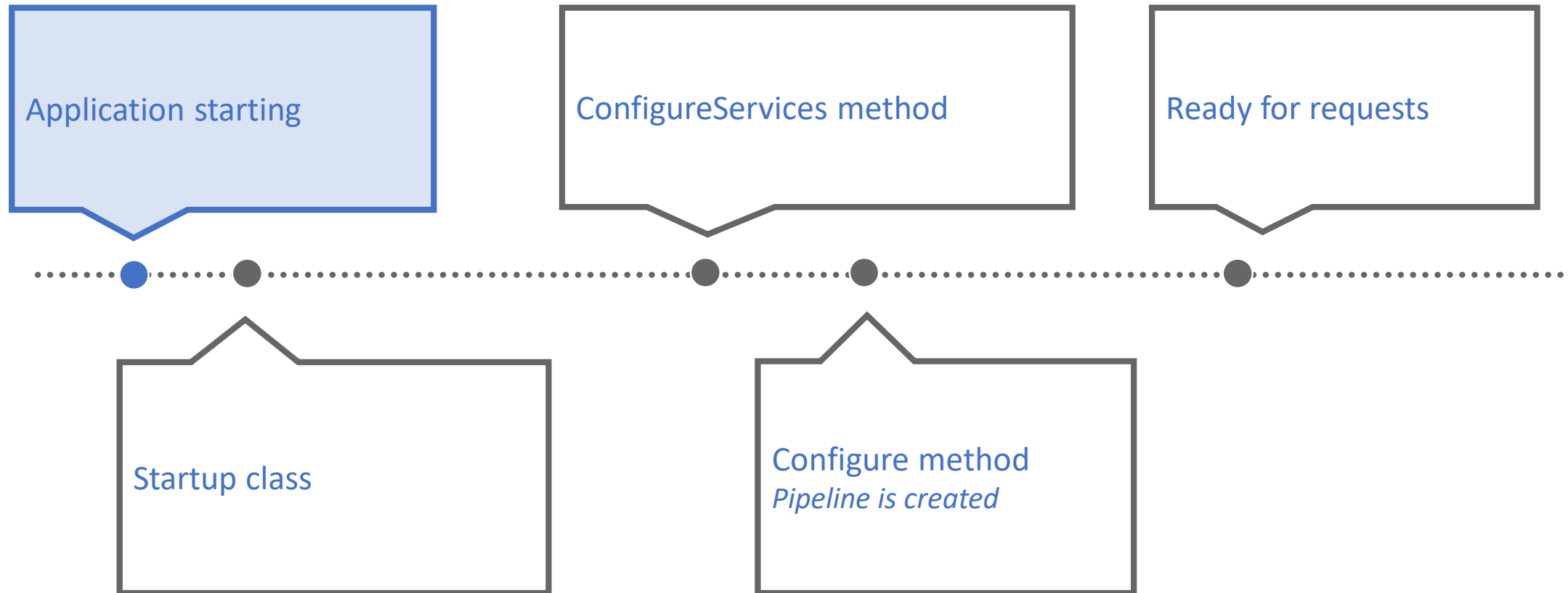
- No compression is done here on static files because of the order

```
public void Configure(IApplicationBuilder app)
{
    app.UseStaticFiles();           // Static files not compressed
                                   // by middleware.
    app.UseResponseCompression();
    app.UseMvcWithDefaultRoute();
}
```

# Built-in middleware

Middleware	Description
Authentication	Provides authentication support.
CORS	Configures Cross-Origin Resource Sharing.
Response Caching	Provides support for caching responses.
Response Compression	Provides support for compressing responses.
Routing	Defines and constrains request routes.
Session	Provides support for managing user sessions.
Static Files	Provides support for serving static files and directory browsing.
URL Rewriting Middleware	Provides support for rewriting URLs and redirecting requests.

# The Startup of the Application



# Content root

- An ASP.NET Core app has a content root directory
  - Views
  - Web Content
- Defaults to same folder as the executable hosting the application
  - Can be configured

# Web root

- Directory where static files are located
  - CSS
  - Images
  - JS
- Served by static files middleware if configured from wwwroot folder
  - Can also be configured if needed



# Application configuration

- Not using web.config anymore
  - One will get created if you publish the app on IIS though
- Configuration is pulled from set of configuration providers
- Built-in providers:
  - File formats (INI, JSON, and XML)
  - Command-line arguments
  - Environment variables
  - In-memory .NET objects
  - An encrypted user store
  - Azure Key Vault
  - Custom providers, which you install or create
- Also support different environments

# Environments

- ASP.NET Core now knows 3 different environments
  - Development
  - Staging
  - Production
- More can be configured



# DEMO

Application configuration

# Summary

- Default way of working of an ASP.NET Core app has changed
- Middleware is important
- DI is built-in