Building a RESTful API with ASP.NET Core *Web API*

Gill Cleeren
Mobile Architect
Snowball

IT/Dev CONNECTIONS

Gill Cleeren
Microsoft MVP & RD
Solution Architect

@gillcleeren
www.snowball.be
gill@snowball.be

IT/Dev CONNECTIONS

PLURALSIGHT

Xamarin, HTML5, WPF, social and Windows development

http://gicl.me/mypscourses

COVERED TOPICS

- ASP.NET Core & ASP.NET Core API
- Creating an API from scratch
- Dependency injection
- Entity Framework Core
- Logging
- Content Negotiation
- Tooling
- And much more!

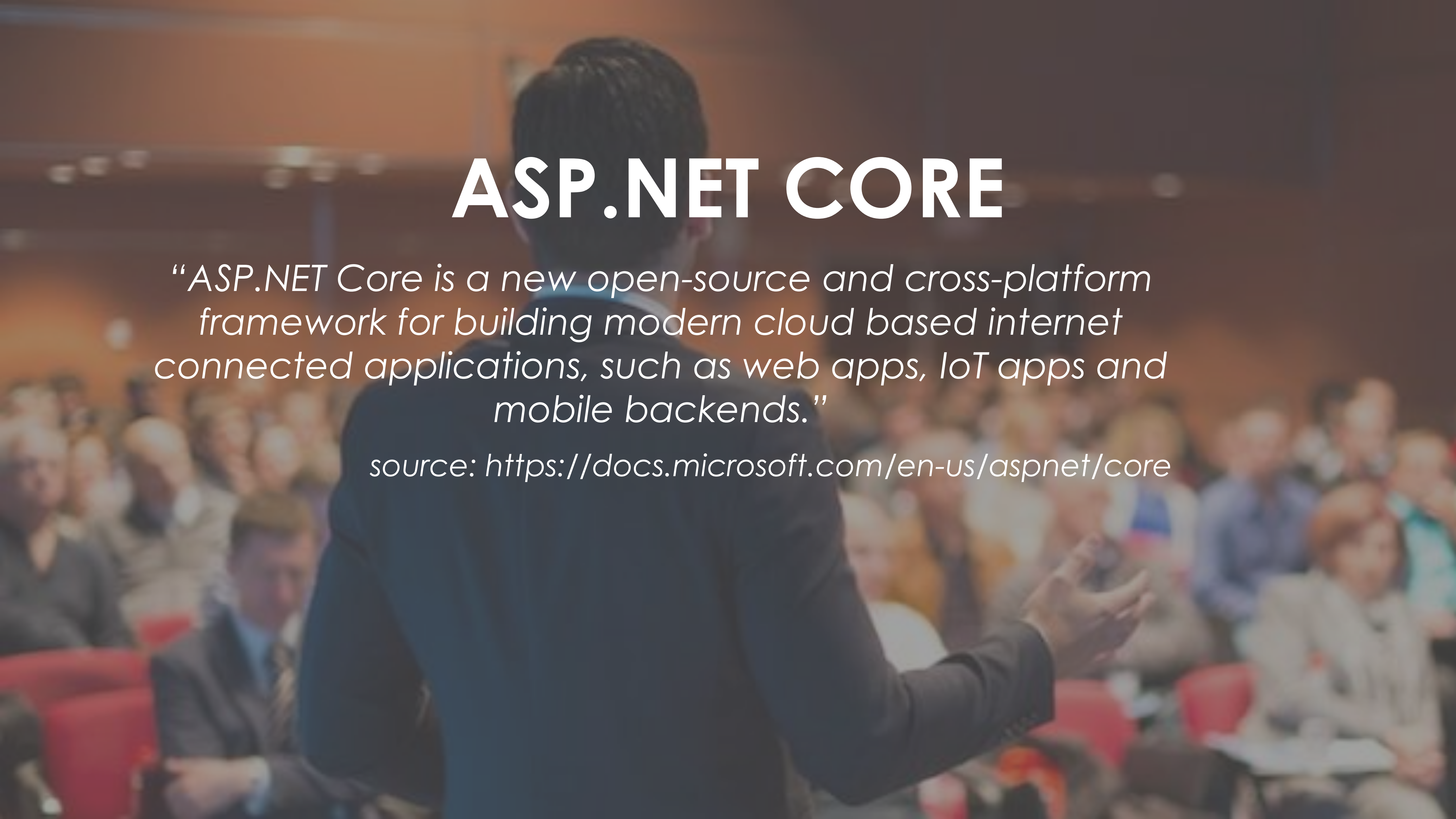
WHY DO WE NEED API'S IN THE FIRST PLACE?

Typical web apps are synchronous

SLOW… WHY?

SORRY, WE'RE

HELLO ASP.NET CORE



ASP.NET CORE

"ASP.NET Core is a new open-source and cross-platform framework for building modern cloud based internet connected applications, such as web apps, IoT apps and mobile backends."

source: https://docs.microsoft.com/en-us/aspnet/core

ASP.NET CORE

- Built on top of .NET Core
- Cross-platform
  - Windows, Mac & Linux
- Not tied to original .NET framework

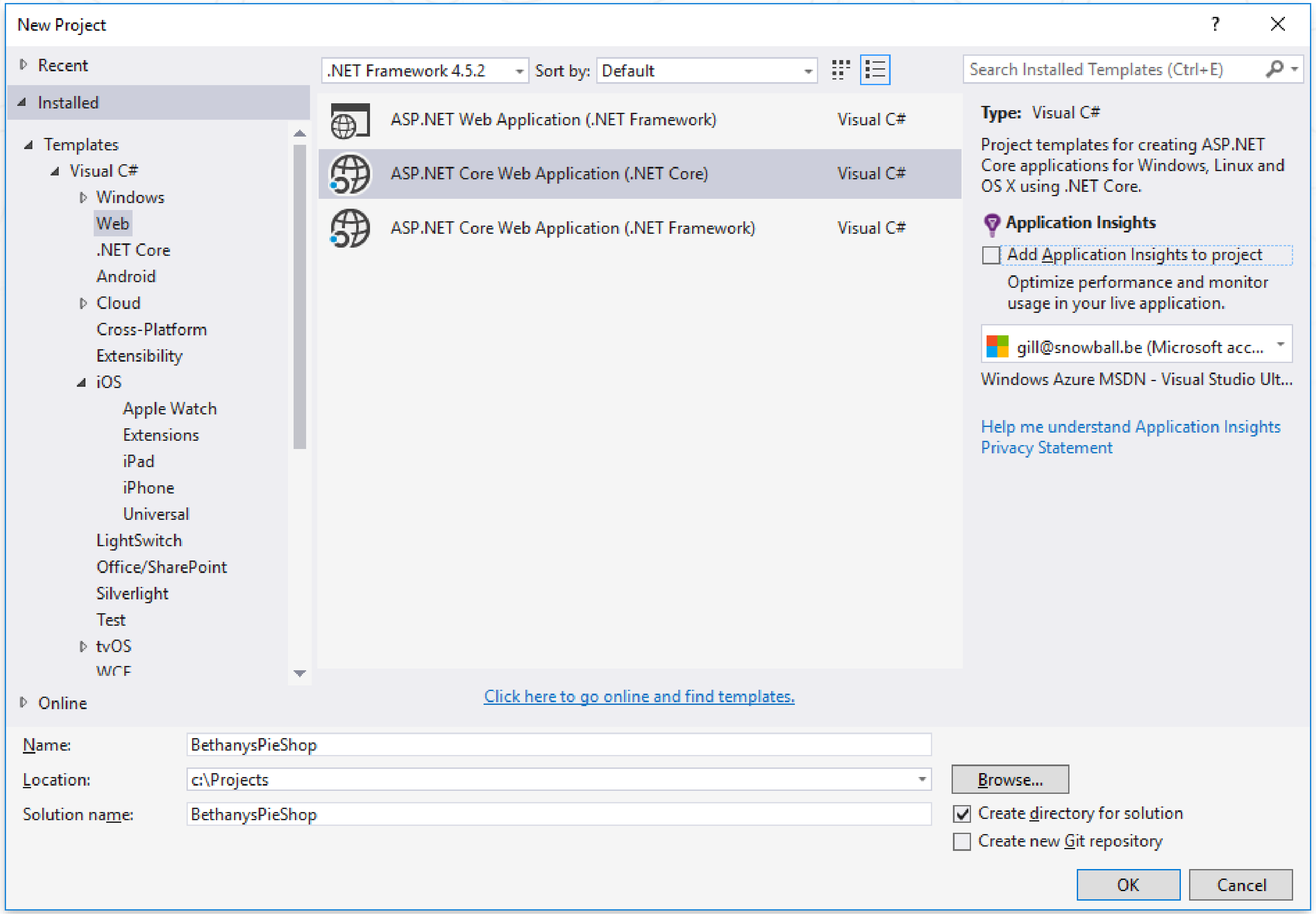
WHAT DOES ASP.NET CORE BRING US?

- Unification between MVC and Web API
- Dependency Injection
- Modular HTTP request pipeline
- Based on NuGet
- Cloud-ready
- IIS or self-host
- Open source
- Cross-platform
- New tooling
- Better integration of client-side frameworks
- Command-line support

MAIN NEW FEATURES IN ASP.NET CORE

- Startup class
- Dependency Injection
- Middleware
- New type of configuration
- Hosting
- Identity

FILE → NEW PROJECT



ASP.NET Web Application (.NET Framework)	Visual C#
ASP.NET Core Web Application (.NET Core)	Visual C#
ASP.NET Core Web Application (.NET Framework)	Visual C#

Type: Visual C#

Project templates for creating ASP.NET Core applications for Windows, Linux and OS X using .NET Core.

Application Insights

☐ Add Application Insights to project

Optimize performance and monitor usage in your live application.

gill@snowball.be (Microsoft acc...

Windows Azure MSDN - Visual Studio Ult...

Help me understand Application Insights

Privacy Statement

Click here to go online and find templates.

Name: BethanysPieShop

Location: c:\Projects

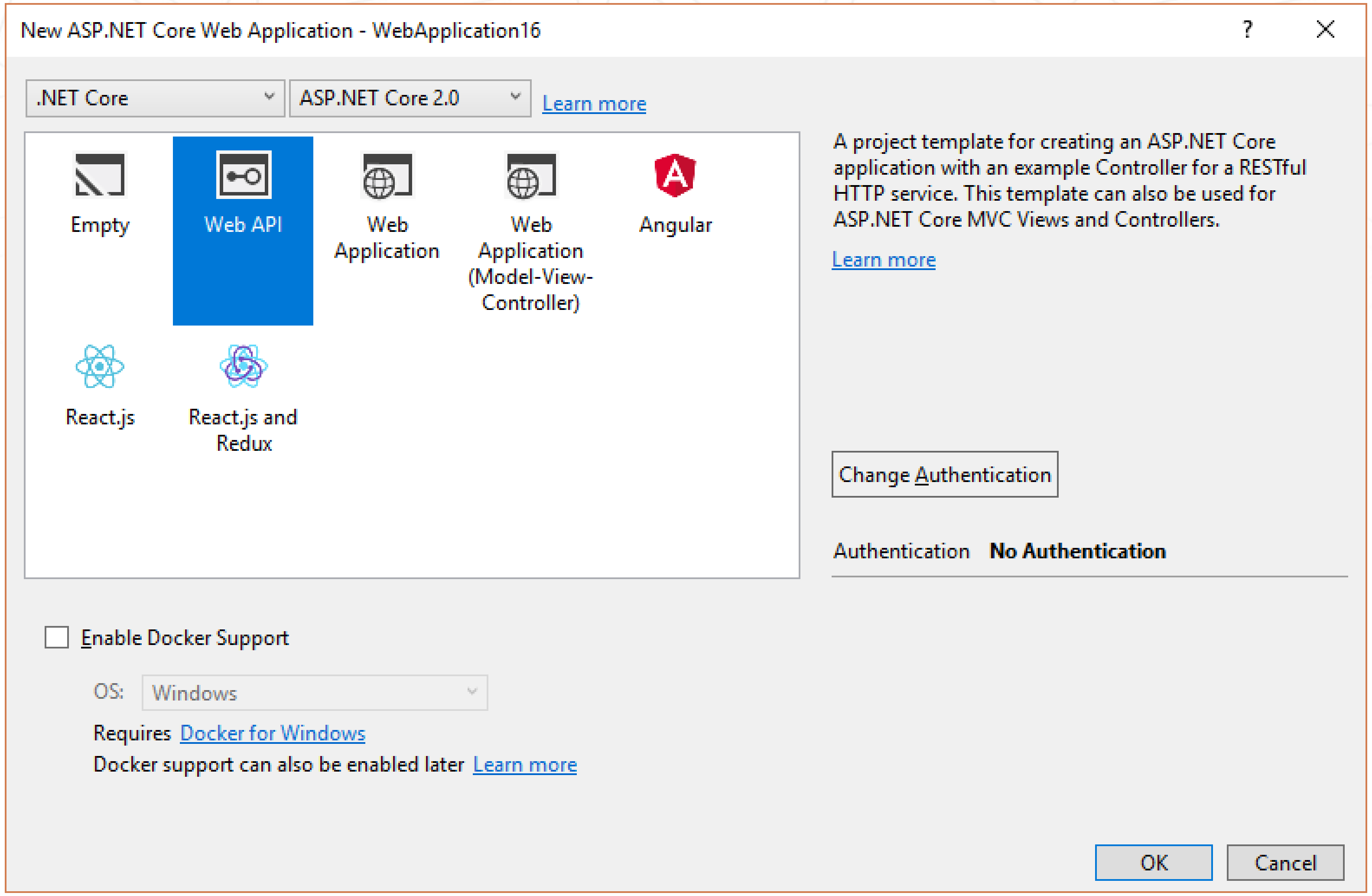
Solution name: BethanysPieShop

Browse...

☑ Create directory for solution

☐ Create new Git repository

OK Cancel



New ASP.NET Core Web Application - WebApplication16

.NET Core | ASP.NET Core 2.0 | Learn more

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.

Learn more

Change Authentication

Authentication **No Authentication**

☐ Enable Docker Support

OS: Windows

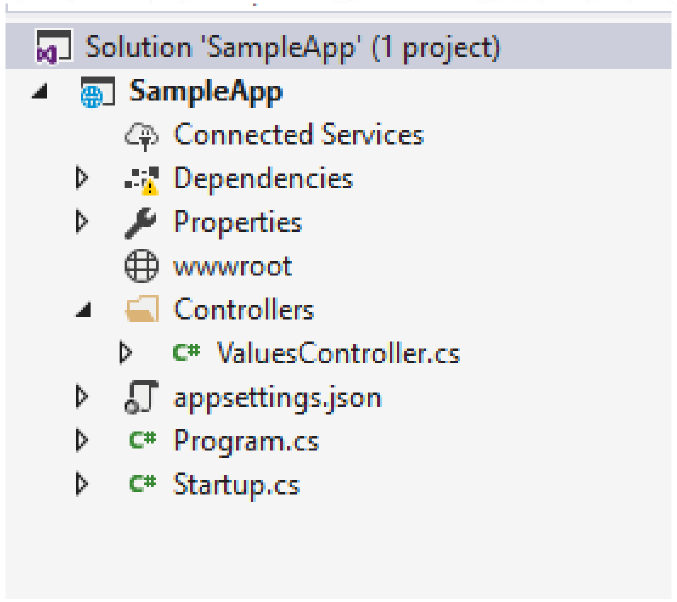
Requires Docker for Windows

Docker support can also be enabled later Learn more

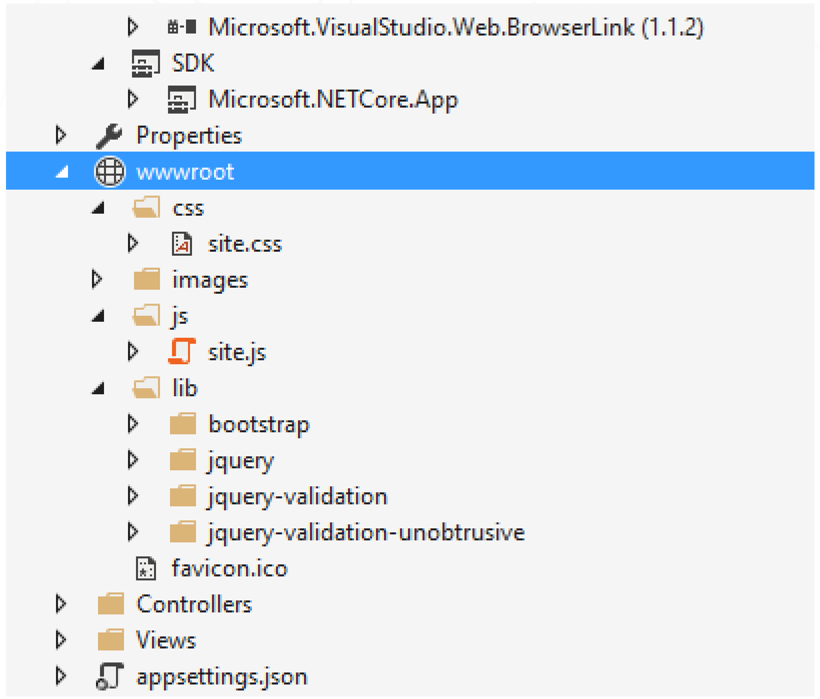
OK | Cancel

PROJECT STRUCTURE

- Important things to note
  - Dependencies
  - wwwroot
  - json files
    - bower.json (MVC only)
    - appsettings.json
  - Program.cs
  - Startup.cs
  - Regular folders
    - Controllers
    - Views (MVC only)

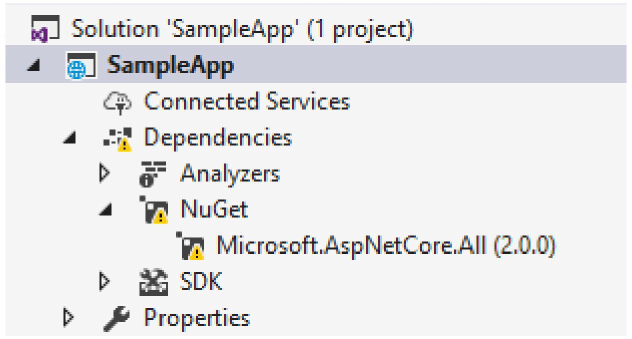


WWWROOT (MVC)



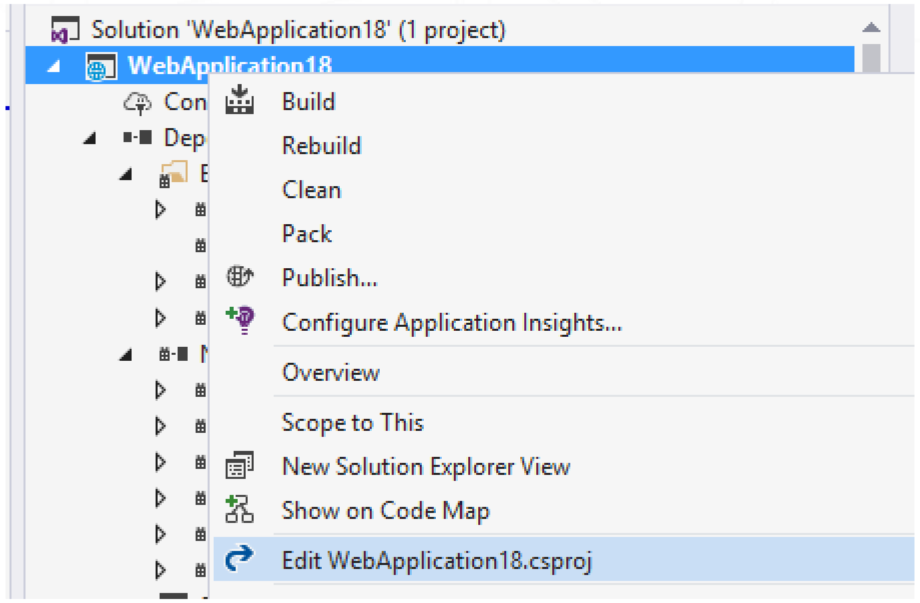
```
    ▷ Microsoft.VisualStudio.Web.BrowserLink (1.1.2)
   ◢ SDK
    ▷ Microsoft.NETCore.App
 ▷ Properties
 ◢ wwwroot
   ◢ css
    ▷ site.css
   ▷ images
   ◢ js
    ▷ site.js
   ◢ lib
    ▷ bootstrap
    ▷ jquery
    ▷ jquery-validation
    ▷ jquery-validation-unobtrusive
     favicon.ico
 ▷ Controllers
 ▷ Views
 ▷ appsettings.json
```

DEPENDENCIES

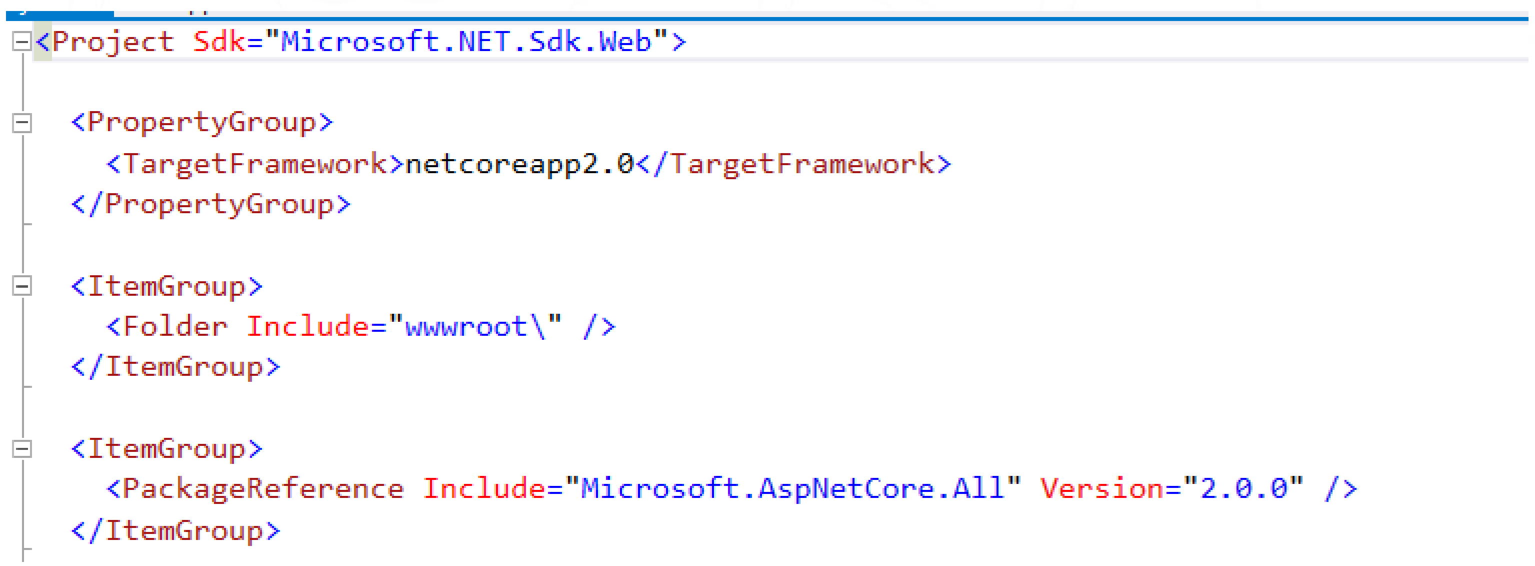


```
Solution 'SampleApp' (1 project)
◢ SampleApp
      Connected Services
   ◢  Dependencies
      ▷  Analyzers
      ◢  NuGet
            Microsoft.AspNetCore.All (2.0.0)
      ▷  SDK
   ▷  Properties
```

MANAGING DEPENDENCIES: CSPROJ FILE


Solution 'WebApplication18' (1 project)
WebApplication18
Con
Dep
Build
Rebuild
Clean
Pack
Publish...
Configure Application Insights...
Overview
Scope to This
New Solution Explorer View
Show on Code Map
Edit WebApplication18.csproj

MANAGING DEPENDENCIES: CSPROJ FILE



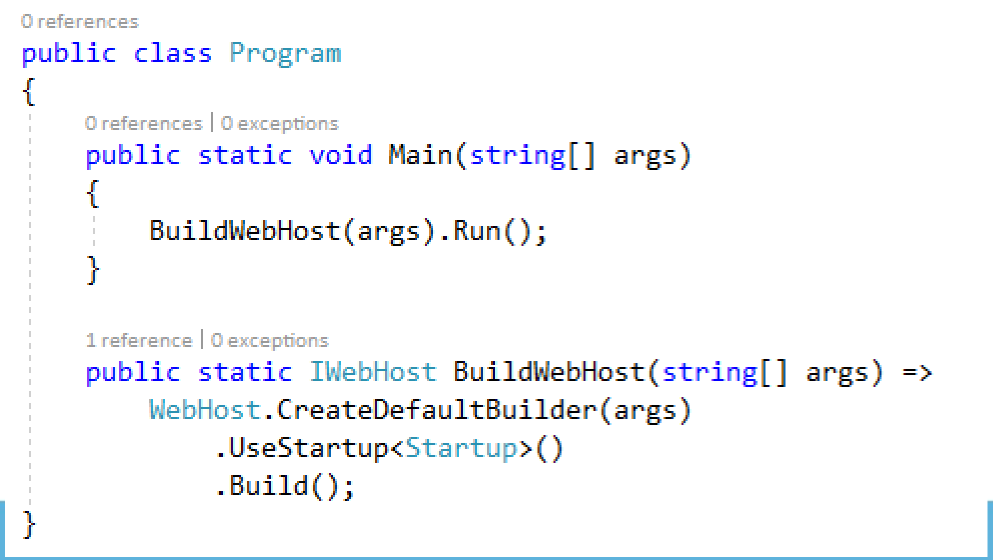
```
<Project Sdk="Microsoft.NET.Sdk.Web">

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

  <ItemGroup>
    <Folder Include="wwwroot\" />
  </ItemGroup>

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

DEMO

File → New Project

BASE CONCEPTS IN ASP.NET CORE

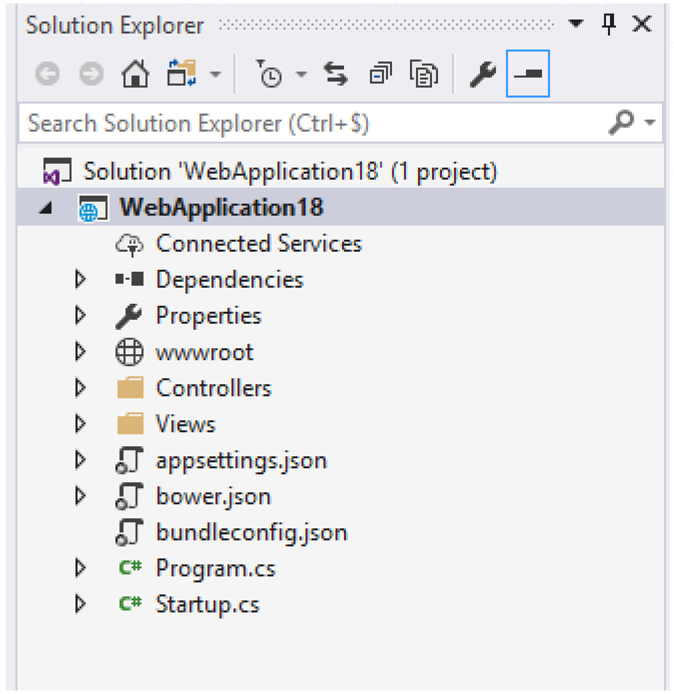
- An ASP.NET Core app is a console app that creates a web server in its Main method
  - MVC will get added soon!



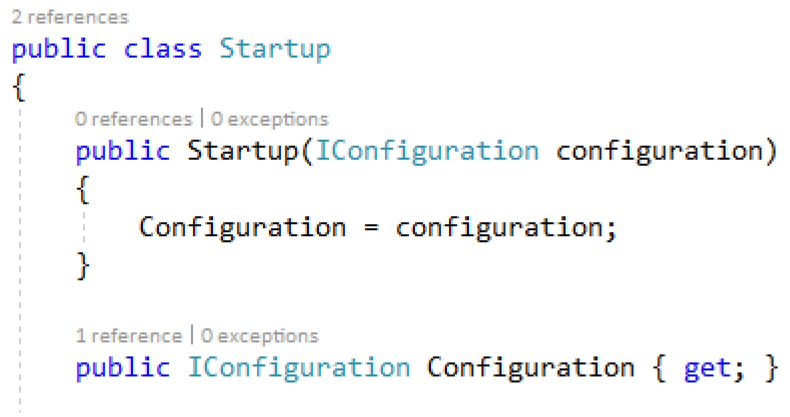
```
0 references
public class Program
{
    0 references | 0 exceptions
    public static void Main(string[] args)
    {
        BuildWebHost(args).Run();
    }

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

STARTUP CLASS

- The UseStartup method on WebHostBuilder specifies the Startup class for your app
  - Use to define request handing 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 CONSTRUCTOR IN STARTUP CLASS



```
2 references
public class Startup
{
    0 references | 0 exceptions
    public Startup(IConfiguration configuration)
    {
        Configuration = configuration;
    }

    1 reference | 0 exceptions
    public IConfiguration Configuration { get; }
```

STARTUP CLASS

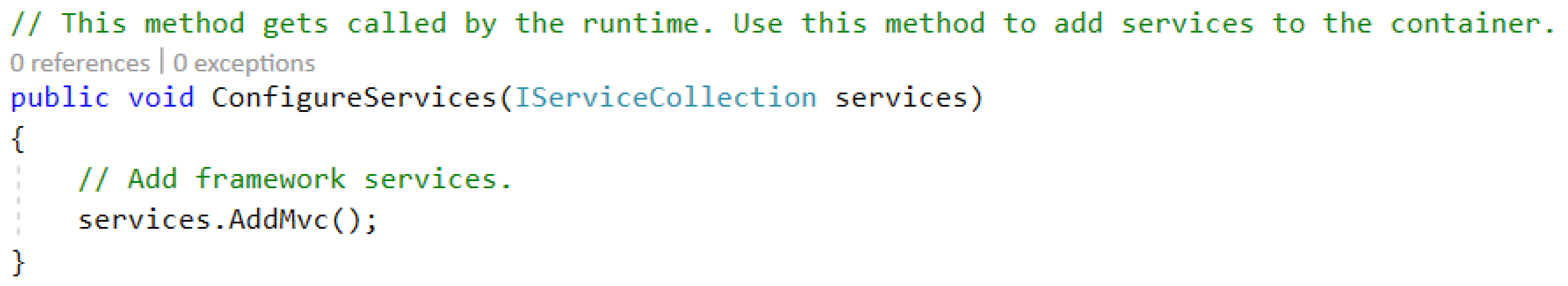
- Note: this is dependency injection at work

```
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



```
// This method gets called by the runtime. Use this method to add services to the container.
0 references | 0 exceptions
public void ConfigureServices(IServiceCollection services)
{
    // Add framework services.
    services.AddMvc();
}
```

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

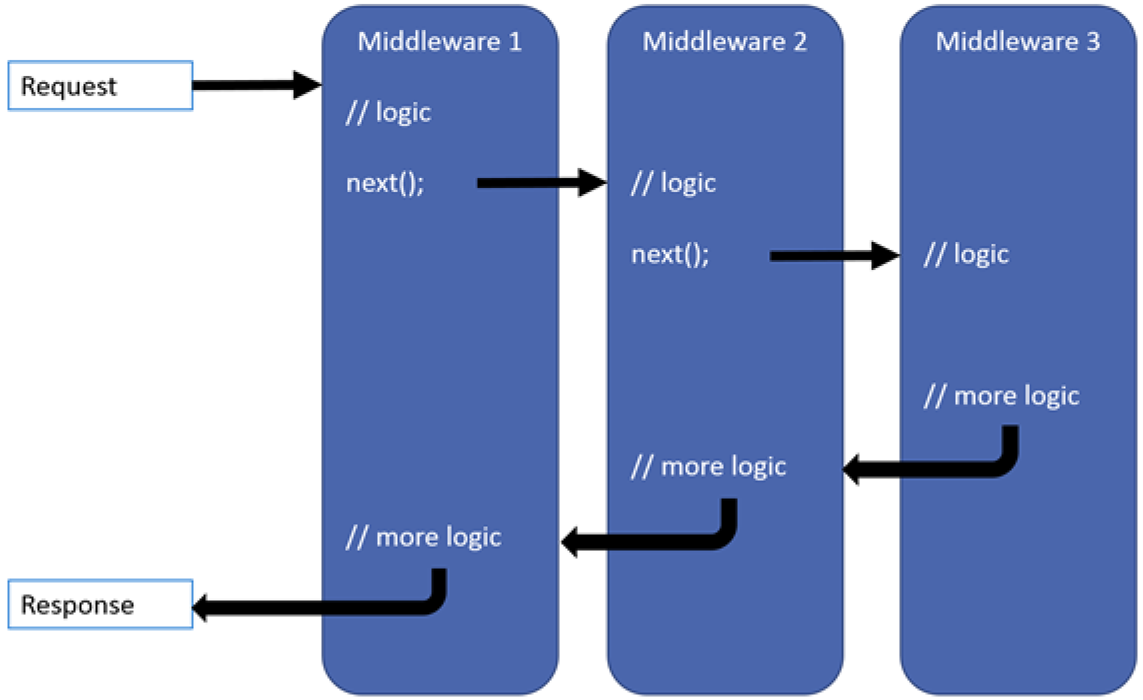
- Request pipeline gets composed using middleware
- ASP.NET Core middleware performs asynchronous logic on an HttpContext and then either invokes the next middleware in the sequence or terminates the request directly
- Typically used by adding a reference to a NuGet package
- Followed by calling UseXXX()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

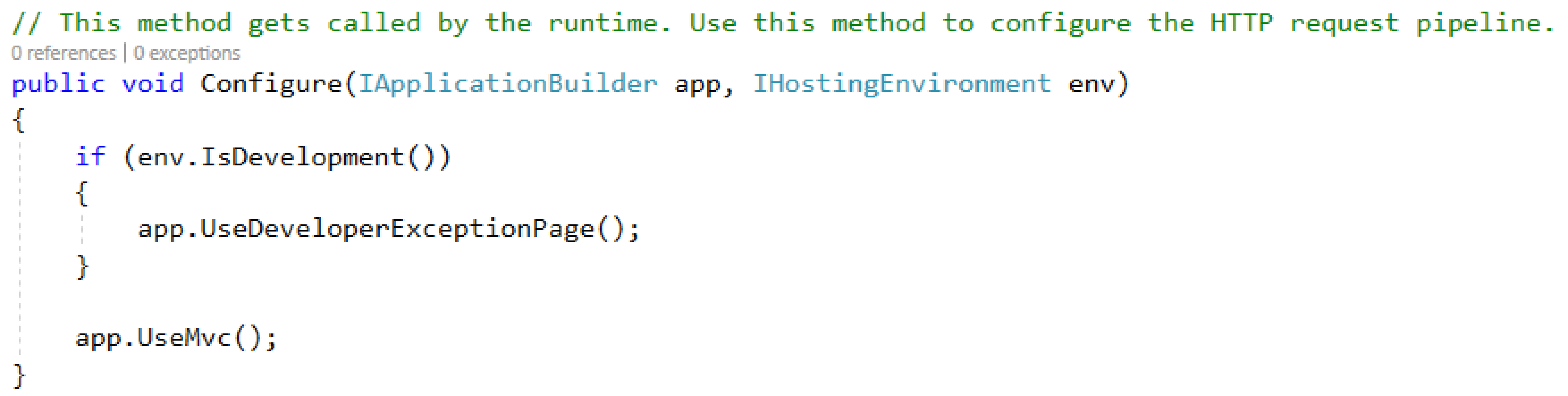
REQUEST PIPELINE

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



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.
0 references | 0 exceptions
public void Configure(IApplicationBuilder app, IHostingEnvironment env)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }

    app.UseMvc();
}
```

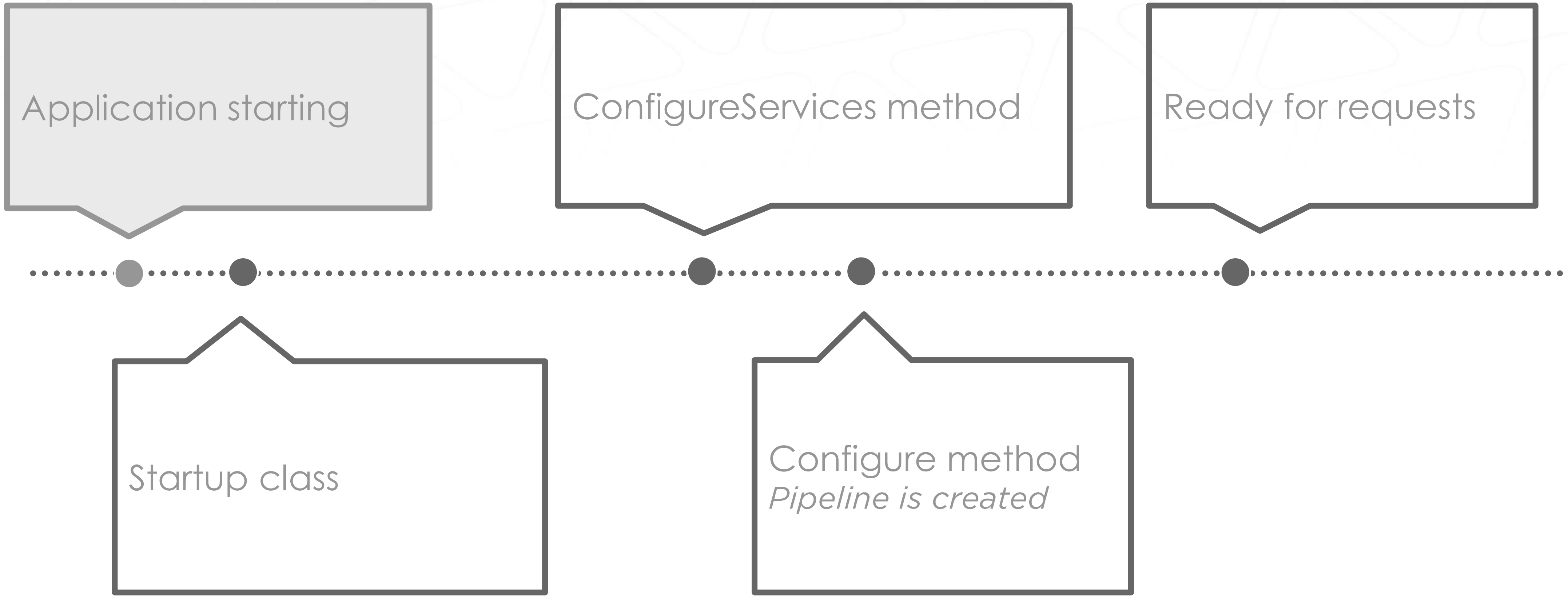
IT/Dev CONNECTIONS

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.
}
```

THE STARTUP OF THE APPLICATION

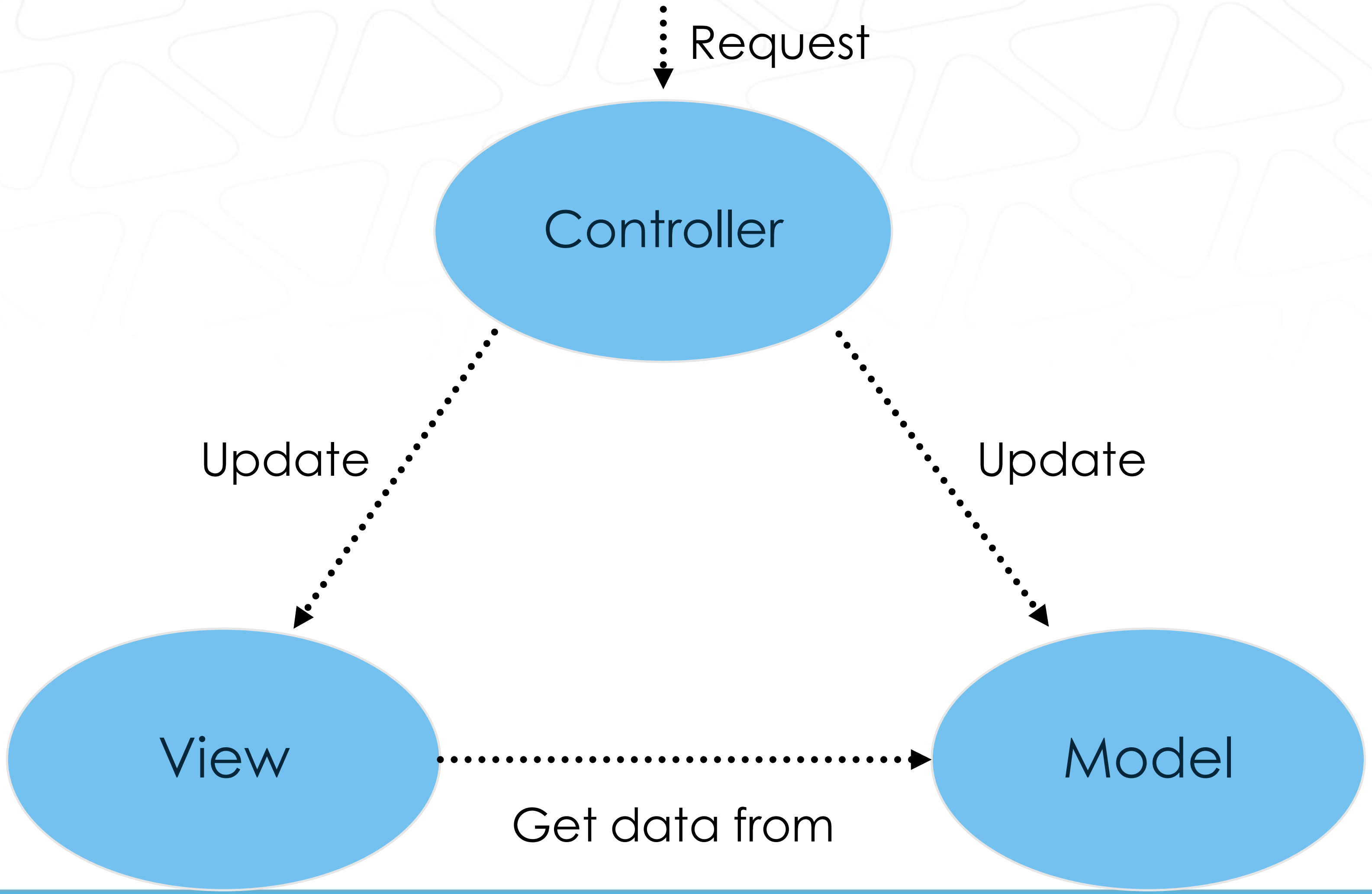


DEMO

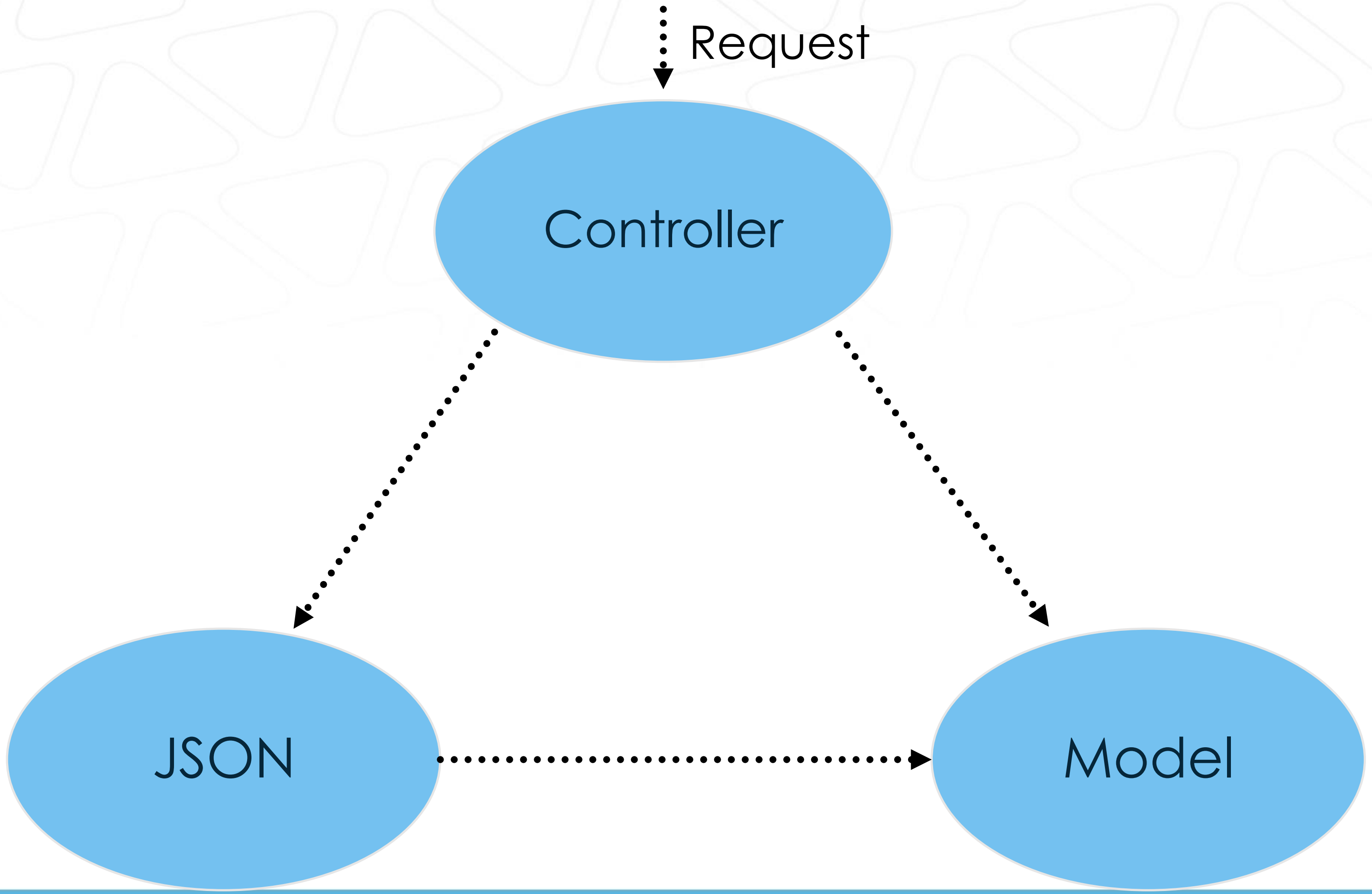
Application configuration

CREATING AN API WITH ASP.NET CORE *WEB API*

THE MVC PATTERN



WHEN USING AN API…



CREATING AN API CONTROLLER

- API controller == MVC controller that gives access to data without HTML
- Data delivery is most often done using REST
  - Representational State Transfer
- REST is based on HTTP verbs and URLs/resources
  - HTTP verb specifies what we want to do
  - Resource represent what we want to use, which objects
  - Format is JSON (preferred) or XML

A VERY SIMPLE CONTROLLER

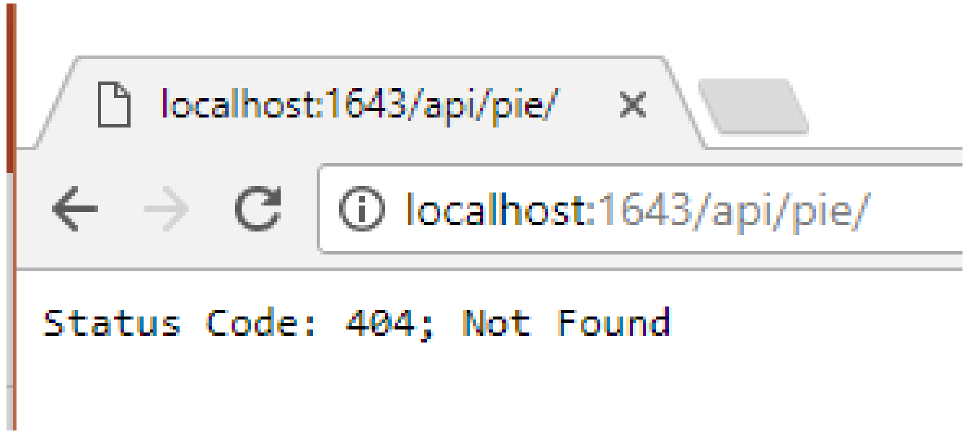
- Similar to controllers in plain ASP.NET Core MVC
- Base ApiController is no more
  - All is now in one class: Controller
  - You can (and should) now have mixed classes

```
public class PieController : Controller
{
 ...
}
```

A VERY SIMPLE ACTION METHOD

```
public class PieController : Controller
{
  public JsonResult Get()
  {
    var pies = new List<Pie>()
    {
      ...
    };
    return new JsonResult(pies);
  }
}
```

AND THE RESULT IS…


localhost:1643/api/pie/

```
Status Code: 404; Not Found
```

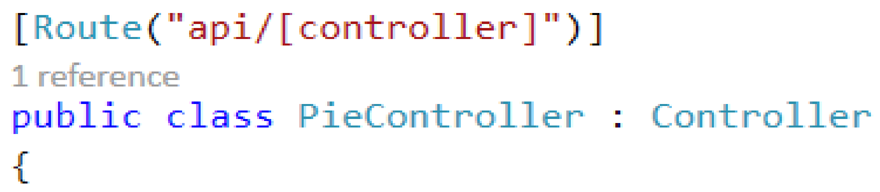
WE'LL NEED SOME ROUTING…

ROUTING

- Routing will allow us to match the request to the action method on the Controller
- 2 options exist
  - Convention-based routing
    - Useful for MVC applications
    - Similar to “old” model
  - Attribute-based routing
    - More natural fit for APIs

WORKING WITH THE ROUTE

- Convention is starting the route with *api*, followed by name of the controller
  - Can be reached using /api/pie



```
[Route("api/[controller]")]
1 reference
public class PieController : Controller
{
```

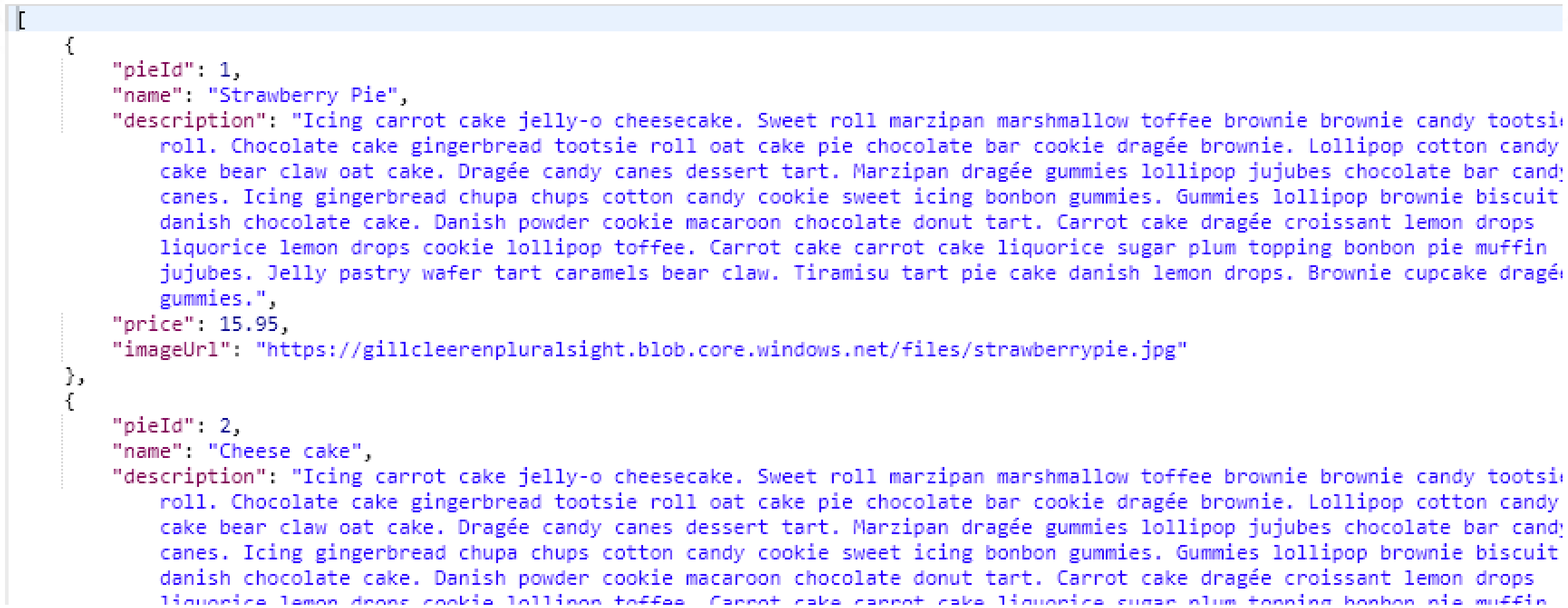
HTTP METHODS

- Routes are combined with HTTP methods
  - Used to indicate which action we want to do
  - Each will have a result, a payload
  - Response code is used to report on the outcome of the operation

HTTP METHODS

Verb	Method	Result
GET	/api/pie	Gets collection of all pies
GET	/api/pie/1	Gets specific pie
POST	/api/pie	Contains data to create a new pie instance, will return saved data
PUT	/api/pie	Contains changed data to update an existing pie instance
DELETE	/api/pie/1	Deletes certain pie

THE RESULT: JSON



```
[
    {
        "pieId": 1,
        "name": "Strawberry Pie",
        "description": "Icing carrot cake jelly-o cheesecake. Sweet roll marzipan marshmallow toffee brownie brownie candy tootsi
            roll. Chocolate cake gingerbread tootsie roll oat cake pie chocolate bar cookie dragée brownie. Lollipop cotton candy
            cake bear claw oat cake. Dragée candy canes dessert tart. Marzipan dragée gummies lollipop jujubes chocolate bar cand
            canes. Icing gingerbread chupa chups cotton candy cookie sweet icing bonbon gummies. Gummies lollipop brownie biscuit
            danish chocolate cake. Danish powder cookie macaroon chocolate donut tart. Carrot cake dragée croissant lemon drops
            liquorice lemon drops cookie lollipop toffee. Carrot cake carrot cake liquorice sugar plum topping bonbon pie muffin
            jujubes. Jelly pastry wafer tart caramels bear claw. Tiramisu tart pie cake danish lemon drops. Brownie cupcake dragé
            gummies.",
        "price": 15.95,
        "imageUrl": "https://gillcleerenpluralsight.blob.core.windows.net/files/strawberrypie.jpg"
    },
    {
        "pieId": 2,
        "name": "Cheese cake",
        "description": "Icing carrot cake jelly-o cheesecake. Sweet roll marzipan marshmallow toffee brownie brownie candy tootsi
            roll. Chocolate cake gingerbread tootsie roll oat cake pie chocolate bar cookie dragée brownie. Lollipop cotton candy
            cake bear claw oat cake. Dragée candy canes dessert tart. Marzipan dragée gummies lollipop jujubes chocolate bar cand
            canes. Icing gingerbread chupa chups cotton candy cookie sweet icing bonbon gummies. Gummies lollipop brownie biscuit
            danish chocolate cake. Danish powder cookie macaroon chocolate donut tart. Carrot cake dragée croissant lemon drops
            liquorice lemon drops cookie lollipop toffee. Carrot cake carrot cake liquorice sugar plum topping bonbon pie muffin
```

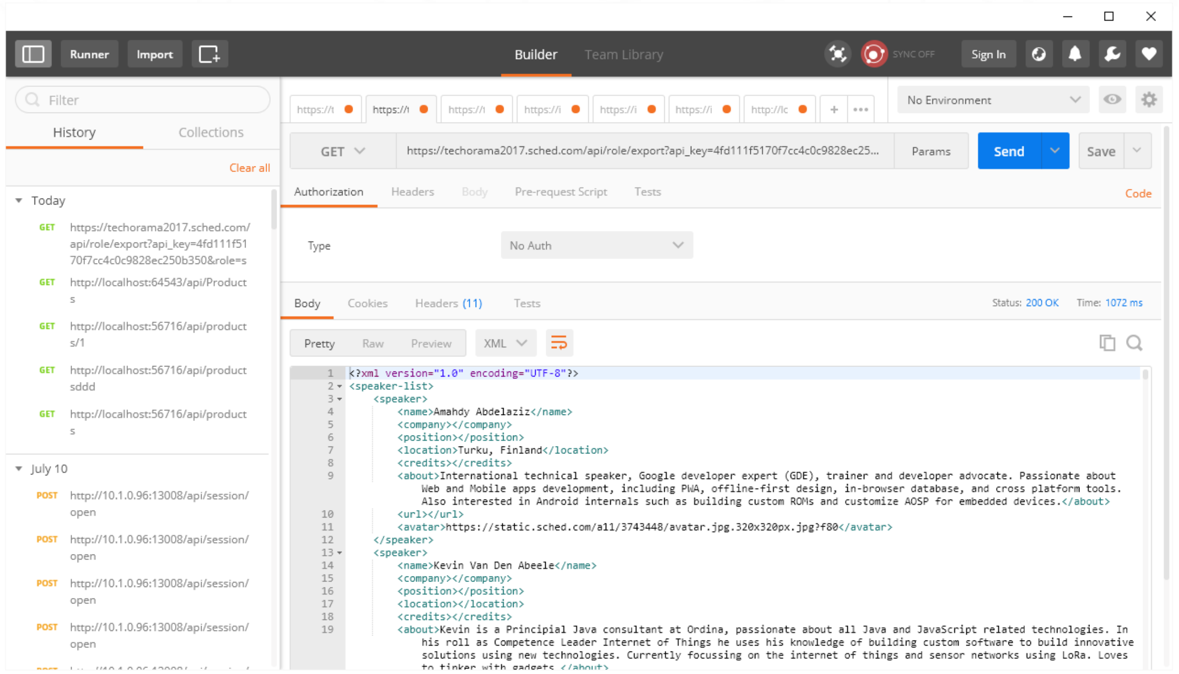
DEMO

Creating a simple API

TESTING AND USING YOUR CONTROLLERS

TESTING YOUR CONTROLLERS

- Different options
  - Fiddler
  - PostMan
  - Swashbuckle
    - NuGet package that will add API description page and allow for testing



Today

GET https://techorama2017.sched.com/api/role/export?api_key=4fd111f5170f7cc4c0c9828ec250b350&role=s

GET http://localhost:64543/api/Products

GET http://localhost:56716/api/products/1

GET http://localhost:56716/api/productsddd

GET http://localhost:56716/api/products

July 10

POST http://10.1.0.96:13008/api/session/open

POST http://10.1.0.96:13008/api/session/open

POST http://10.1.0.96:13008/api/session/open

POST http://10.1.0.96:13008/api/session/open

GET https://techorama2017.sched.com/api/role/export?api_key=4fd111f5170f7cc4c0c9828ec25...

Type: No Auth

Status: 200 OK Time: 1072 ms

```xml
<?xml version="1.0" encoding="UTF-8"?>
<speaker-list>
    <speaker>
        <name>Amahdy Abdelaziz</name>
        <company></company>
        <position></position>
        <location>Turku, Finland</location>
        <credits></credits>
        <about>International technical speaker, Google developer expert (GDE), trainer and developer advocate. Passionate about
            Web and Mobile apps development, including PWA, offline-first design, in-browser database, and cross platform tools.
            Also interested in Android internals such as building custom ROMs and customize AOSP for embedded devices.</about>
        <url></url>
        <avatar>https://static.sched.com/a11/3743448/avatar.jpg.320x320px.jpg?f80</avatar>
    </speaker>
    <speaker>
        <name>Kevin Van Den Abeele</name>
        <company></company>
        <position></position>
        <location></location>
        <credits></credits>
        <about>Kevin is a Principial Java consultant at Ordina, passionate about all Java and JavaScript related technologies. In
            his roll as Competence Leader Internet of Things he uses his knowledge of building custom software to build innovative
            solutions using new technologies. Currently focussing on the internet of things and sensor networks using LoRa. Loves
            to tinker with gadgets.</about>
```

DEMO

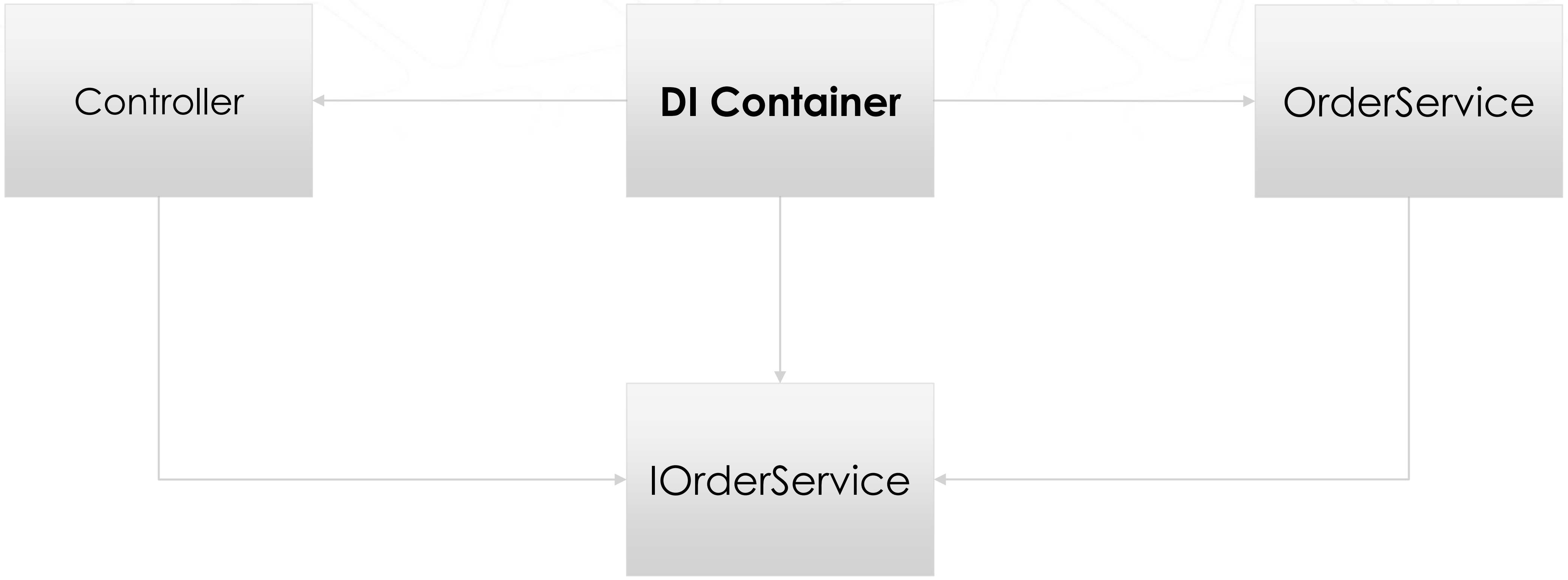
Testing the API using Postman

USING DEPENDENCY INJECTION AND REPOSITORIES

DEPENDENCY INJECTION

- Type of inversion of control (IoC)
- Another class is responsible for obtaining the required dependency
- Results in more loose coupling
  - Container handles instantiation as well as lifetime of objects
- Built-in into ASP.NET Core
  - Can be replaced with other container if needed

DEPENDENCY INJECTION



REGISTERING DEPENDENCIES

```
public void ConfigureServices(IServiceCollection services)
{
  services.AddSingleton<ISomethingService, SomethingService>();

  // Add framework services.
  services.AddMvc();
}
```

REGISTERING DEPENDENCIES

- AddSingleton
- AddTransient
- AddScoped

DEPENDENCY INJECTION IN CONTROLLER

- Constructor injection: adds dependencies into constructor parameters

```
public class PieController : Controller
{
  private readonly ISomethingService _something;

  public PieController(ISomethingService something)
  {
    _something = something;
  }
}
```

“ USE A **REPOSITORY** TO SEPARATE THE LOGIC THAT RETRIEVES THE DATA AND MAPS IT TO THE ENTITY MODEL FROM THE BUSINESS LOGIC THAT ACTS ON THE MODEL. THE BUSINESS LOGIC SHOULD BE AGNOSTIC TO THE TYPE OF DATA THAT COMPRISES THE DATA SOURCE LAYER ”

THE REPOSITORY

```
public class PieRepository : IPieRepository
{
  private List<Pie> _pies;

  public PieRepository()
  {
    _pies = new List<Pie>()
      {
        ...


      };
  }

  public IEnumerable<Pie> Pies => _pies;
}
```

REGISTERING THE REPOSITORY

```
public void ConfigureServices(IServiceCollection services)
{
  services.AddSingleton<IPieRepository, PieRepository>();
}
```

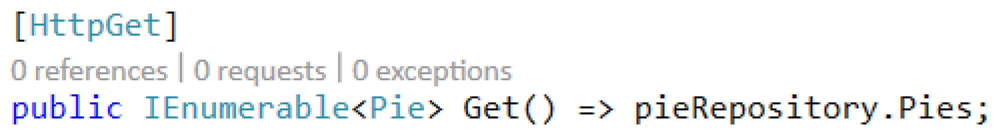
DEMO

Adding a Repository and Dependency Injection

ADDING MORE FUNCTIONALITY ON THE CONTROLLER

ACTION METHODS

- Action methods need to be declared with attribute that specifies the HTTP method

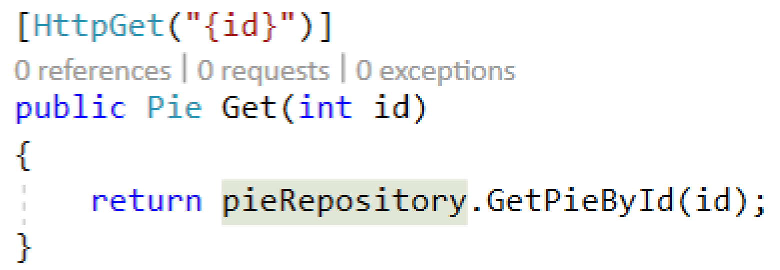


```
[HttpGet]
0 references | 0 requests | 0 exceptions
public IEnumerable<Pie> Get() => pieRepository.Pies;
```

- Available attributes
  - HttpGet
  - HttpPost
  - HttpPut
  - HttpDelete
  - HttpPatch
  - HttpHead
  - AcceptVerbs: for multiple verbs on one method

ACTION METHODS

- We can pass values (routing fragment)

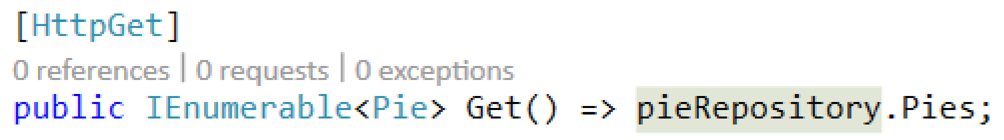


```
[HttpGet("{id}")]
0 references | 0 requests | 0 exceptions
public Pie Get(int id)
{
    return pieRepository.GetPieById(id);
}
```

  - Can be reached by combining the route with the parameter
    - /api/pie/{id}
- Routes in general don't contain an action segment
  - Action name isn't part of the URL to reach an action method
  - HTTP method is used to differentiate between them

ACTION RESULTS

- API action methods don't rely on ViewResult
  - Instead, return data



```
[HttpGet]
0 references | 0 requests | 0 exceptions
public IEnumerable<Pie> Get() => pieRepository.Pies;
```

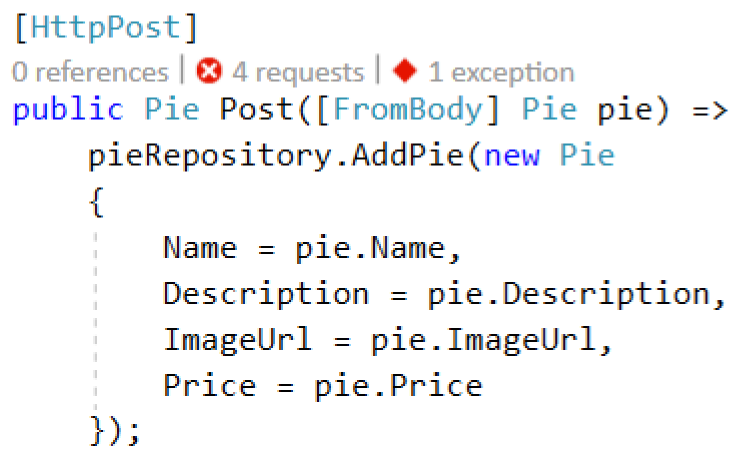
  - MVC will make sure they get returned in a correct format
    - By default JSON
    - It's possible to change this, even from the client
      - Media Type formatting is applied here

STATUS CODES

- Status code are part of the response from the API
- Used for information:
  - Whether or not the request succeeded
  - Why did it fail

STATUS CODES

- MVC is good at knowing what to return
  - If we return null, it'll return 204 – No Content
- We can override this by returning a different IActionResult
  - NotFound leads to 404
  - Ok just sends the object to the client and returns 200



```
[HttpPost]
0 references | 4 requests | 1 exception
public Pie Post([FromBody] Pie pie) =>
    pieRepository.AddPie(new Pie
    {
        Name = pie.Name,
        Description = pie.Description,
        ImageUrl = pie.ImageUrl,
        Price = pie.Price
    });
```

MOST IMPORTANT STATUS CODES

- 20X: Success
  - 200: OK
  - 201: Created
  - 204: No content
- 40X: Client error
  - 400: Bad request
  - 403: Forbidden
  - 404: Not found
- 50X: Server error
  - 500: Internal server error

DEMO

Completing the Controller and repository

ADDING ENTITY FRAMEWORK CORE

“

ENTITY FRAMEWORK (EF) CORE IS A LIGHTWEIGHT AND EXTENSIBLE VERSION OF THE POPULAR ENTITY FRAMEWORK DATA ACCESS TECHNOLOGY

”

ENTITY FRAMEWORK CORE

- ORM
- Lightweight
- Cross-platform
- Open source
- Works with several databases
- Code-first

DEMO

Adding Entity Framework Core

LOGGING

LOGGING IN ASP.NET CORE

- Built-in into ASP.NET Core
- Available as middleware
  - ILogger
  - Provider-based model
  - External providers can be added if needed

BUILT-IN PROVIDERS

- Console
- Debug
- EventSource
- EventLog
- Azure
- …

DEMO

Logging

CONTENT FORMATTING

CONTENT FORMATTING

- MVC has to work out which data format it should return
  - Basically, which encoding to use
- Can be influenced by the client request
- Content format depends on
  - Format(s) accepted by client
  - Format(s) that can be produced
  - Content policy specified by the action
  - Type returned by the action
- Can be hard to figure out
  - In most cases, the default will do

DEFAULT CONTENT POLICY

- Easiest situation: client and action method define no restrictions on the format that can be used
  - If method returns string, string is returned to client, content-type header is set to text/plain
  - All other data types are returned as JSON, content-type is set to application/json
- Strings aren't returned as JSON because of possible issues
  - Double quoted strings: “”Hello world””
  - An int is simply returned as “2”

DEMO

Default content policy

CONTENT NEGOTIATION

- Clients will typically include an Accept header
  - Specifies possible accepted formats (MIME types)
- Browser will send
  - Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
  - Prefers HTML and XHTML, can accept XML and Webp
  - Q indicated preference (xml is less preferred)
  - */* indicated that all will work
    - But only for 0.8

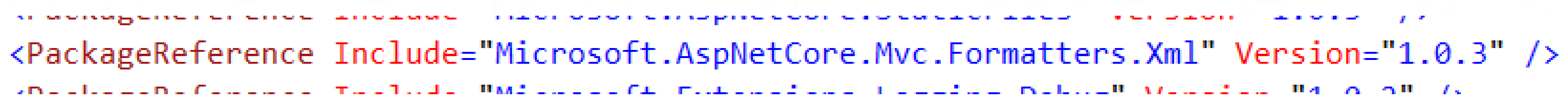
CHANGING THE ACCEPT HEADER

- If we ask the API for XML…
  - Headers Accept="application/xml"
- We'll get back json… (Thank you MVC)

- MVC doesn't know about XML at this point

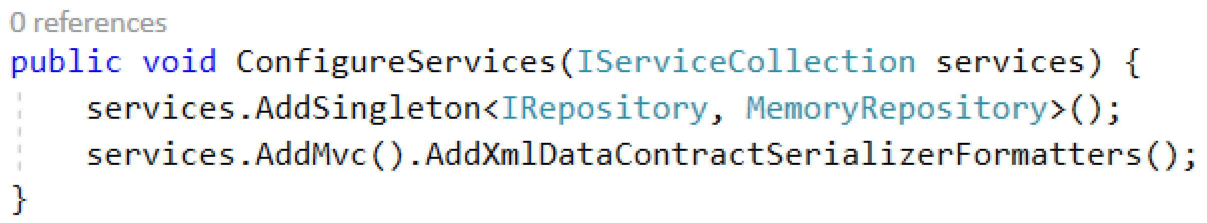
XML PLEASE

- Requires extra package to be included



```
<PackageReference Include="Microsoft.AspNetCore.Mvc.Formatters.Xml" Version="1.0.3" />
```

- And as always, some configuration change in the Startup class



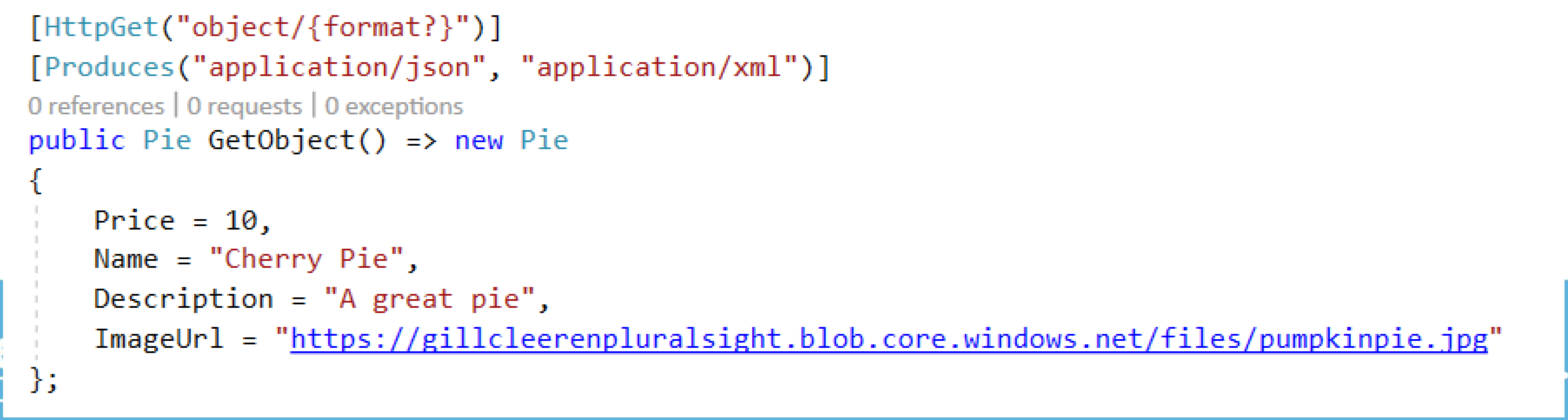
```
0 references
public void ConfigureServices(IServiceCollection services) {
    services.AddSingleton<IRepository, MemoryRepository>();
    services.AddMvc().AddXmlDataContractSerializerFormatters();
}
```

DEMO

Adding support for XML

OVERRIDING THE ACTION DATA FORMAT

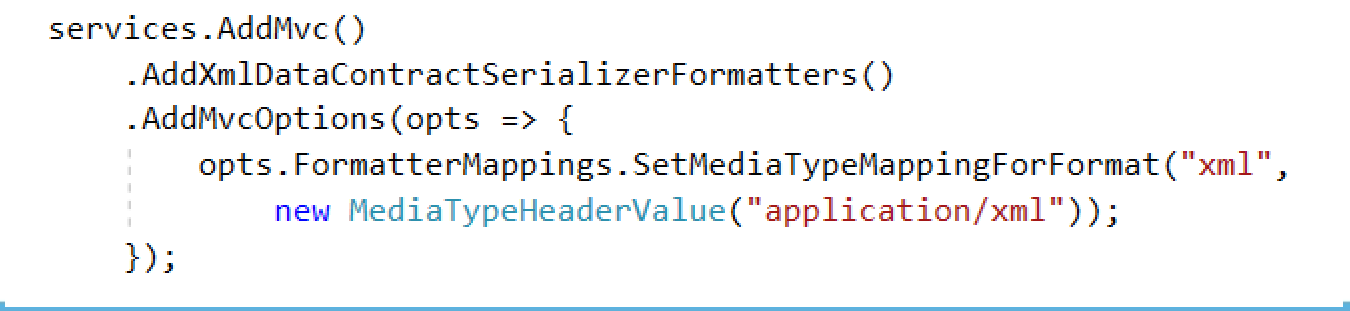
- We can specify the data format on the action method itself
  - Produces is filter
  - Changes the content type
  - Argument is the returned data format



```
[HttpGet("object/{format?}")]
[Produces("application/json", "application/xml")]
0 references | 0 requests | 0 exceptions
public Pie GetObject() => new Pie
{
    Price = 10,
    Name = "Cherry Pie",
    Description = "A great pie",
    ImageUrl = "https://gillcleerenpluralsight.blob.core.windows.net/files/pumpkinpie.jpg"
};
```

PASSING THE DATA FORMAT USING THE ROUTE OR THE QUERY STRING

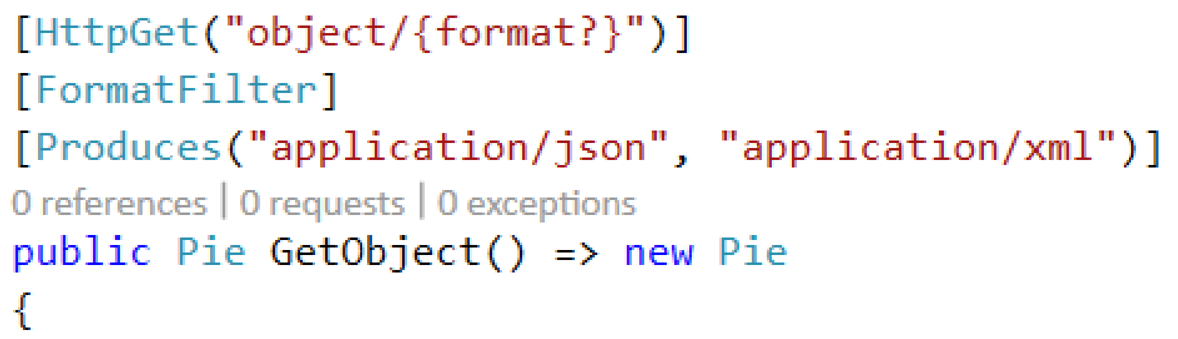
- ACCEPT header can't always be controlled from client
- Data format can therefore also be specified through route or query string to reach an action
- We can define a shorthand in the Startup class
  - xml can now be used to refer to application/xml



```
services.AddMvc()
    .AddXmlDataContractSerializerFormatters()
    .AddMvcOptions(opts => {
        opts.FormatterMappings.SetMediaTypeMappingForFormat("xml",
            new MediaTypeHeaderValue("application/xml"));
    });
```

PASSING THE DATA FORMAT USING THE ROUTE OR THE QUERY STRING

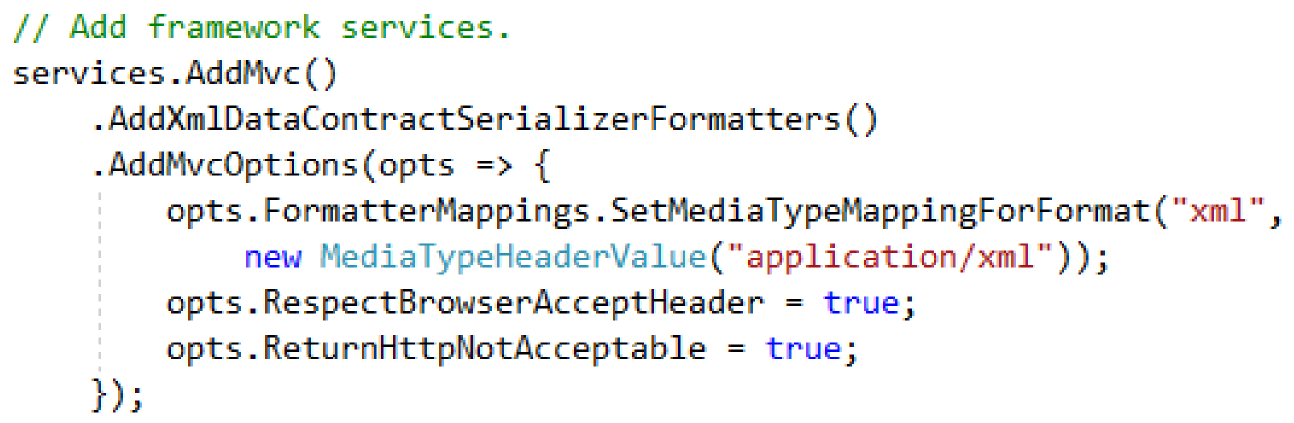
- On the action, we now need to add the FormatFilter
- Upon receiving the shorthand (xml), the mapped type from the Startup is retrieved
- Route can now also include the format (optional)
  - If combined with Produces and other type is requested, 404 is returned



```
[HttpGet("object/{format?}")]
[FormatFilter]
[Produces("application/json", "application/xml")]
0 references | 0 requests | 0 exceptions
public Pie GetObject() => new Pie
{
```

REAL CONTENT NEGOTIATION

- RespectBrowserAcceptHeader: if true, accept header will be fully respected
- ReturnHttpNotAcceptable: if no format can be found, 406 – Not Acceptable is returned



```
// Add framework services.
services.AddMvc()
    .AddXmlDataContractSerializerFormatters()
    .AddMvcOptions(opts => {
        opts.FormatterMappings.SetMediaTypeMappingForFormat("xml",
            new MediaTypeHeaderValue("application/xml"));
        opts.RespectBrowserAcceptHeader = true;
        opts.ReturnHttpNotAcceptable = true;
    });
```

DEMO

Content Negotiation

VERSIONING THE API

VERSIONING AN API

- Something that's often forgotten by developers…
- Versioning is required
  - Clients evolve, API needs to follow as well
  - Old versions often still required
- Many ways exist
  - URI versioning
  - Header versioning
  - Accept-headers versioning
  - …

VERSIONING AN API

- Open-source package
  - microsoft.aspnetcore.mvc.versioning
- Supports
  - ASP.NET Web API
  - ASP.NET Web API with oData
  - ASP.NET Core
- Added using
  - services.AddApiVersioning();

VERSIONING USING QUERYSTRING PARAMETER VERSIONING

```
namespace My.Services.V1
{
    [ApiVersion("1.0")]
    public class HelloWorldController : Controller
    {
        public string Get() => "Hello world v1.0!";
    }
}
namespace My.Services.V2
{
    [ApiVersion("2.0")]
    public class HelloWorldController : Controller
    {
        public string Get() => "Hello world v2.0!";
    }
}
```

VERSIONING USING URL PATH SEGMENT

```
[ApiVersion("1.0")]
[Route("api/v{version:apiVersion}/[controller]")]
public class HelloWorldController : Controller {
    public string Get() => "Hello world!";
}

[ApiVersion("2.0")]
[ApiVersion("3.0")]
[Route("api/v{version:apiVersion}/helloworld")]
public class HelloWorld2Controller : Controller {
    [HttpGet]
    public string Get() => "Hello world v2!";

    [HttpGet, MapToApiVersion( "3.0" )]
    public string GetV3() => "Hello world v3!";
}
```

VERSIONING USING HEADER

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddMvc();
    services.AddApiVersioning(o => o.ApiVersionReader = new HeaderApiVersionReader("api-version"));
}
```

DEMO

Versioning the API

DEPLOYING THE API

DEPLOYMENT OPTIONS

- Azure
- IIS
- Docker
- Linux

DEPLOYING TO AZURE

- Account via portal.azure.com
- Create an Azure App Service
  - SQL Server database if using a DB
- Visual Studio Publish from Solution Explorer

DEMO

Deploying the API to an Azure App Service

SUMMARY

QUESTIONS?

THANKS!

Building a RESTful API with ASP.NET Core Web API

Gill Cleeren
Mobile Architect
Snowball