ASP.NET CORE MVC MODULE 01 INTRODUCTION

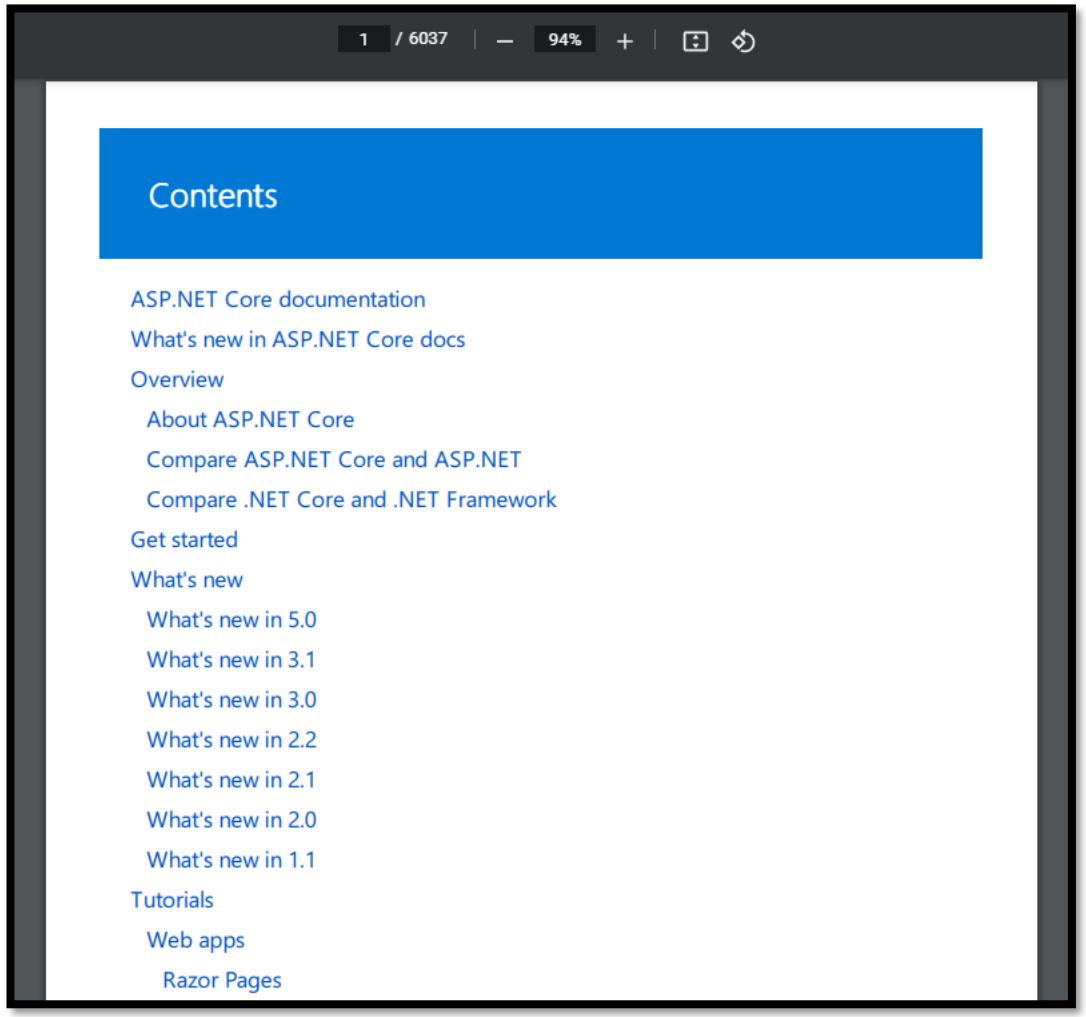
Summer 2021 – Web Development using ASP .Net Core MVC

MAIN SOURCES FOR THESE SLIDED

- Unless otherwise specified, the main sources for these slides are:
  - https://github.com/MicrosoftLearning/20486D-DevelopingASPNETMVCWebApplications ←for homework
  - https://docs.microsoft.com/en-us/aspnet/core/mvc/overview?view=aspnetcore-5.0 ←for "textbook"

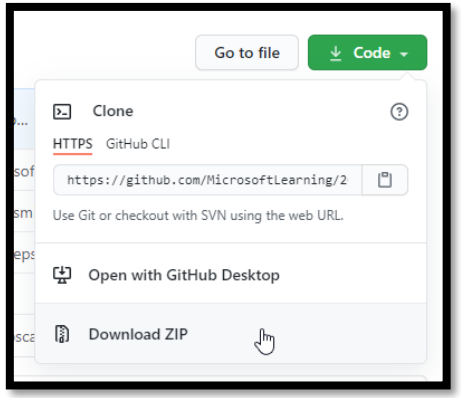
OFFICIAL ASP.NET CORE DOCUMENTATION

- Here is a link to the Official ASP.NET Core documentation from Microsoft
  - https://docs.microsoft.com/en-us/aspnet/core/opbuildpdf/0b7f664b/toc.pdf?branch=live&view=aspnetcore-5.0

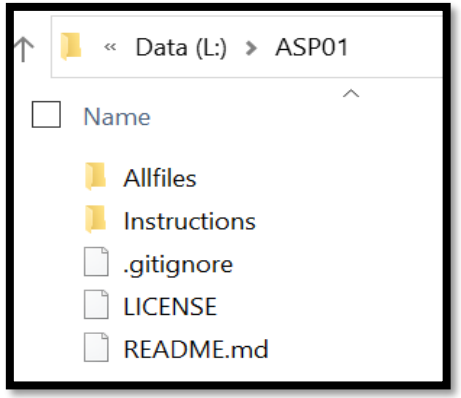


PREPARING THE DEV ENVIRONMENT & HOMEWORK ENVIRONMENT (FOR ASP)

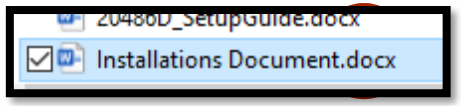
- **Step 1**: download the instructions and code from:
  - https://github.com/MicrosoftLearning/20486D-DevelopingASPNETMVCWebApplications
  - Note: this is a rather large file!



- **Step2**: Unzip the file into a directory with a short path



- **Step3**: The **Instructions** subdirectory contains installation steps
  - Follow the steps show in **Installations Document.docx**
  - Useful links:
    - https://www.visualstudio.com/downloads/ Visual Studio Community 2017/2019
    - https://go.microsoft.com/fwlink/?linkid=867670 SQL server management
    - https://nodejs.org/en Node.js



REGARDING LABS

- Please do the labs thoroughly
  - It may take you 2^+ hours per lab! So please plan accordingly! If time, we'll start the lab in class.
  - Some topics will be visited multiple times (in multiple lectures).
    - This means that some new "steps"/topics learnt in the first modules will appear again in future labs/modules

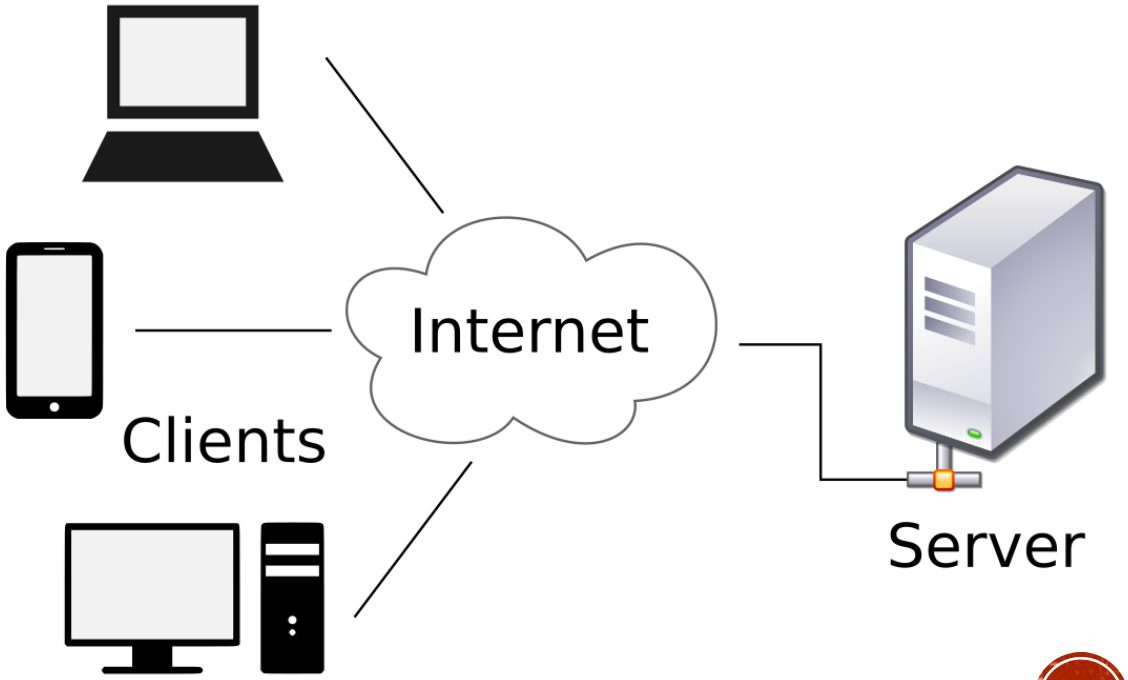
- You may feel lost from time to time but please do not lose focus.
  - Some of the code you'll see, will be covered in greater detail later.
    - Make sure to select a .cs file, if you cannot find a Run Without Debugging is grayed out!
  - Think about what you're trying to accomplish at each step. And try to see how it fits into the project.
  - Some people call this "productively lost". It is not wasted time!

- You're learning by example. You're not learning from a short, isolated code, but rather from a working project
  - As a new software developer, at least from my experience, you'll likely learn the same way at the company that hires you.

- Sometimes, the labs you'll try to submit are too large.
  - Email me in advance to get help remove unnecessary files so you can upload your work
  - Your grade on the homework will be effort grade since I won't be able to verify each file.
    I must see work/attempt, even if you don't complete these assignments. If you have challenges please let me know.

WHAT IS CLIENT-SERVER ARCHITECTURE?

- In-class discussion.
- Can you give examples of web-applications you use every day?
- See also:
  - https://www.geeksforgeeks.org/client-server-model/



WHAT IS ASP .NET?

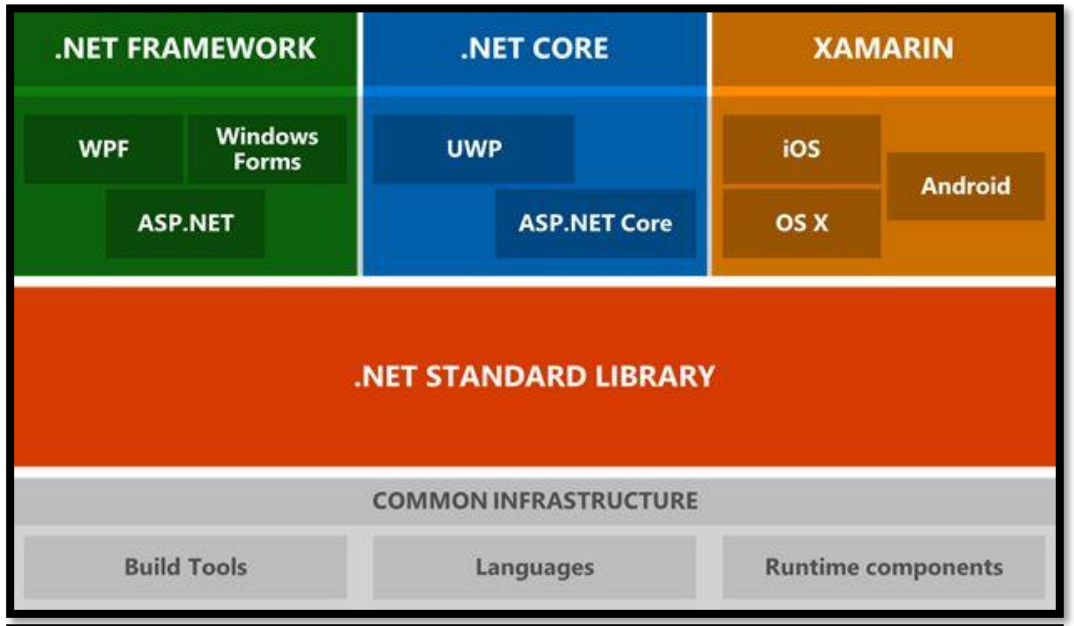
- Source:
  - https://dotnet.microsoft.com/learn/aspnet/what-is-aspnet

- "ASP.NET is an open source web framework, created by Microsoft, for building modern web apps and services with .NET."

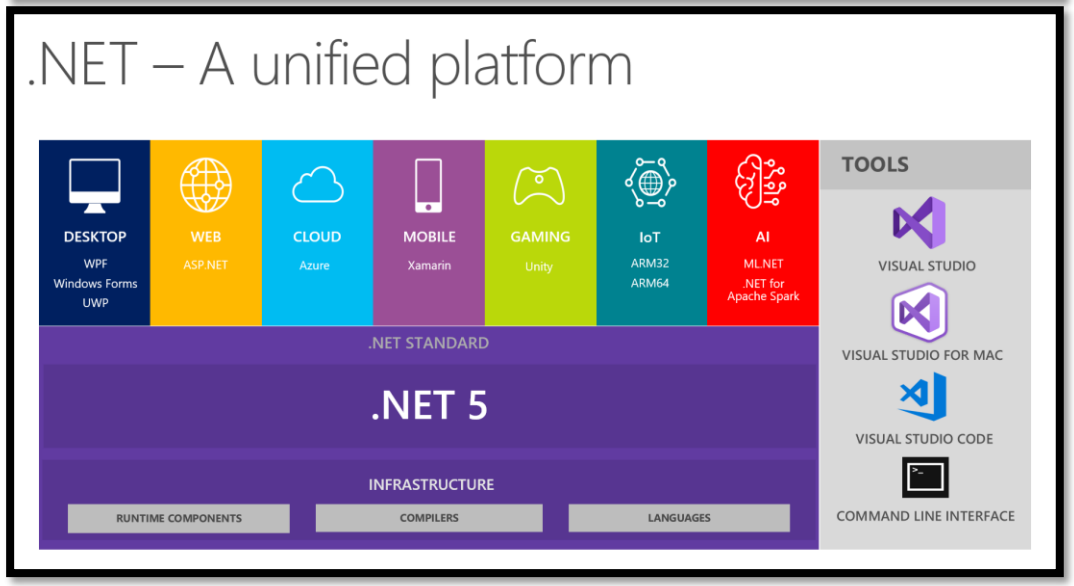
- ASP stands for **A**ctive **S**erver **P**ages

WHAT IS ASP .NET CORE

- There used to be .NET **Framework** (Windows only) and .NET **Core** (cross-platform)
  - In the past, ".NET" meant '.NET Framework"
  - Since this course was prepared a few years ago for the cross-platform version, it uses the name .NET **Core**
  - You are/this course is in here (a little old) ➔ ➔ ➔ ➔



- The latest/current version of .NET is .NET 5.0 which is a unified platform
- See also:
  - https://devblogs.microsoft.com/dotnet/introducing-net-5/
  - https://docs.microsoft.com/en-us/dotnet/core/dotnet-five



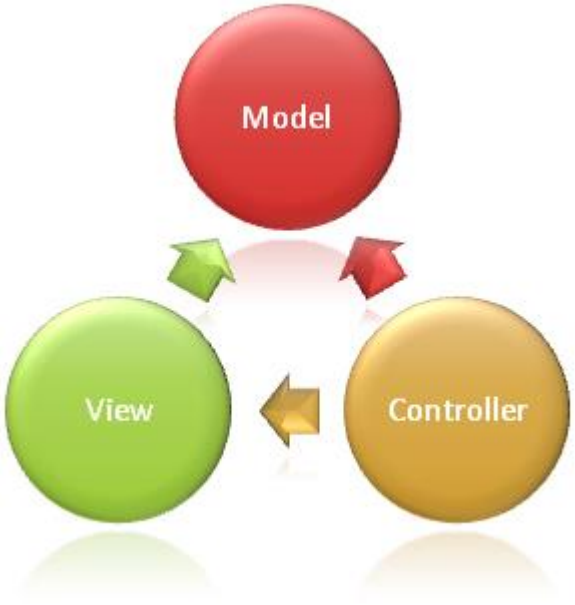
WHAT IS ASP .NET CORE MVC?

- Source: https://docs.microsoft.com/en-us/aspnet/core/mvc/overview?view=aspnetcore-5.0

- **Model-View-Controller** (MVC) is an architectural pattern used to design a web application.

- MVC separates a web application into three components: Models, Views, and Controllers.
  - Such pattern helps achieve **separation of concerns** which helps code, debug, and test the code.

- We'll see them in more details but here is what each component does:
  - A **model** defines a set of classes that represent the object types that the web application manages.
    - For example: a **product** (having properties such as: unique identifier, price, description, picture ID, seller), or a **user**, etc.
  - A **controller**: is a class that handles user interaction
    (it receives user requests, it may create/modify **model** classes, and it returns the appropriate **view**)
    - For example, when a user can send a request to the server (say you're searching for laptops). Based on the request, a **controller** will create a list of instances of the **product** model and passes it to a view, which displays the list to the user.
  - A **view** is a component that builds the webpages that make up the **user interface**.
    - Typically, a **controller** will pass an instance of a model class (or a list) to a view. The **view** will display the properties of a **model**. E.g., if the controller passes a list of **product** objects, the **view** might display them in a table name.



- Example: let's see this model on a size, such as Amazon.com or Facebook.com

QUICK OVERVIEW OF THE MVC

- **Models**
  - These are the typical **C# classes** - so we'll use .cs files.
  - A **model** contains business logic, validation, and if needed, database access logic.

- **Views**
  - These are used to generate the user interface.
  - Views are markup pages that will contain both HTML and C# - so we'll use **.cshtml** files.

- **Controllers**
  - A user interacts with a website by clicking on links and buttons. Controllers respond to user actions, they may create an instance of a model, and pass it to a view, so that it can be displayed in a webpage.
  - Controllers are .NET classes that inherit from the **Microsoft.AspNetCore.Mvc.Controller** class – so we'll use **.cs** files.

SOME FEATURES WE MAY SEE

- Source:
  - https://docs.microsoft.com/en-us/aspnet/core/mvc/overview?view=aspnetcore-5.0

- ASP.NET Core MVC includes the following:
  - Routing
  - Model binding
  - Model validation
  - Dependency injection
  - Filters
  - Areas
  - Web APIs
  - Testability
  - Razor view engine
  - Strongly typed views
  - Tag Helpers
  - View Components

IMPORTANT NOTE

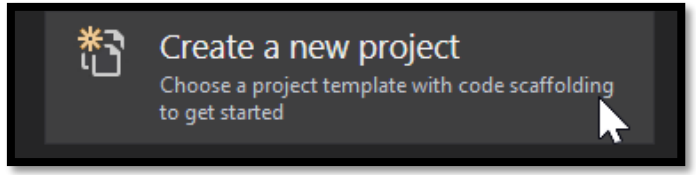
- For small/simple web applications, you probably wouldn't choose the **MVC** pattern.
  - If you have time check **Razor Pages** which are easier.

- The **MVC** pattern is more suitable for more complex web applications, where we need:
  - separation of concerns (separate the business logic (models), input logic (controllers), user interface (views))
  - scaling (as your application gets larger and larger, how will you be able to maintain the code efficiently?)
  - take control of the URLs use (we'll talk about routing)

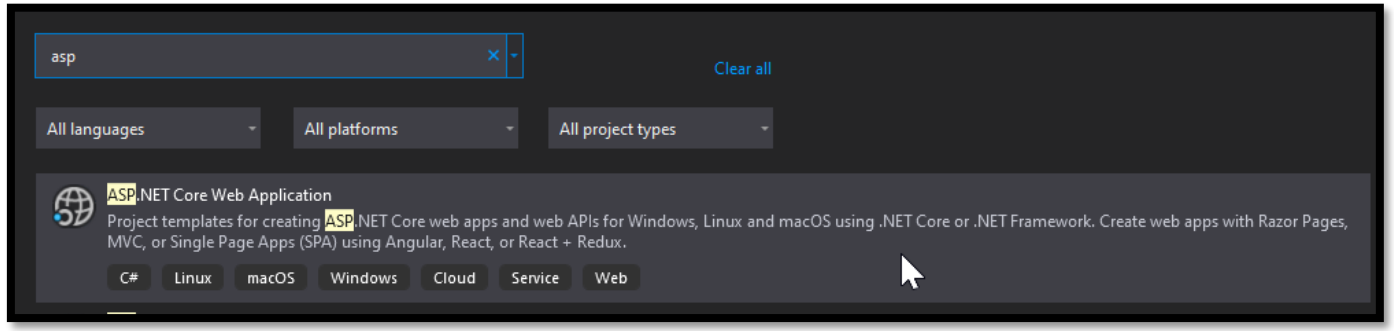
GET STARTED WITH ASP.NET CORE MVC

- Source:
  - https://github.com/dotnet/AspNetCore.Docs/blob/master/aspnetcore/tutorials/first-mvc-app/start-mvc.md
  - See also page 1621 from the official documentation pdf…

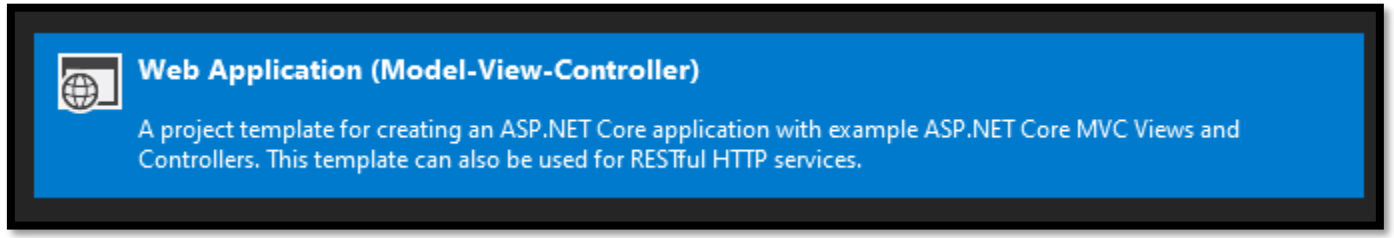
- Let's create our first ASP .NET Core MVC web application (for **VS Code**, see instructions …)
  - Start Visual Studio, and create a new project:



  - Search for **ASP.NET Core Web Application**



  - Select a project name then choose **Web App (Model-View-Controller).**

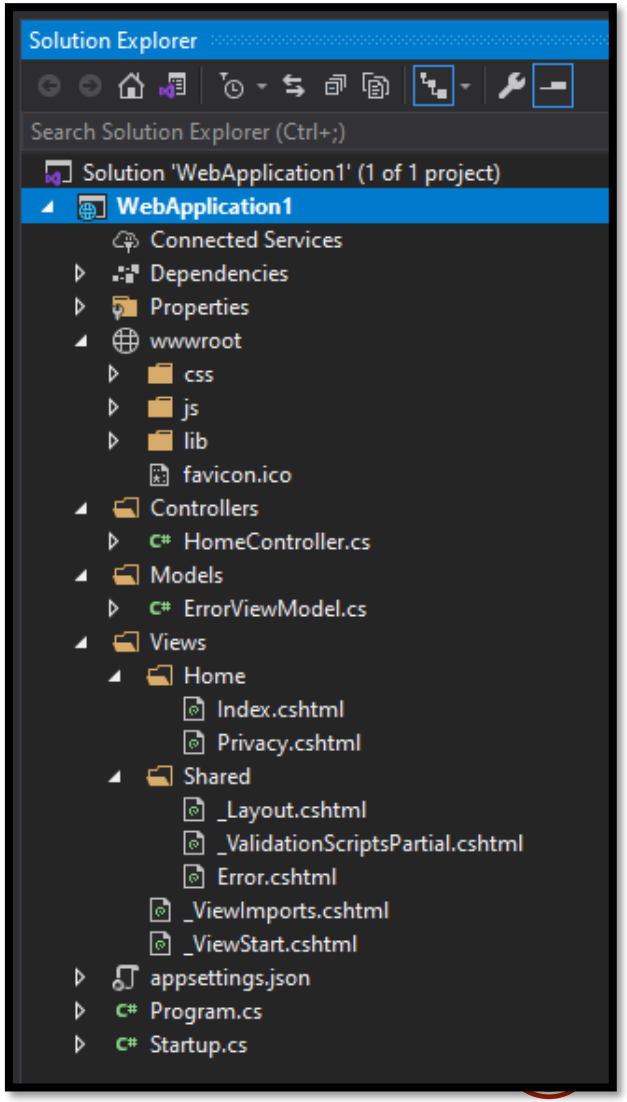


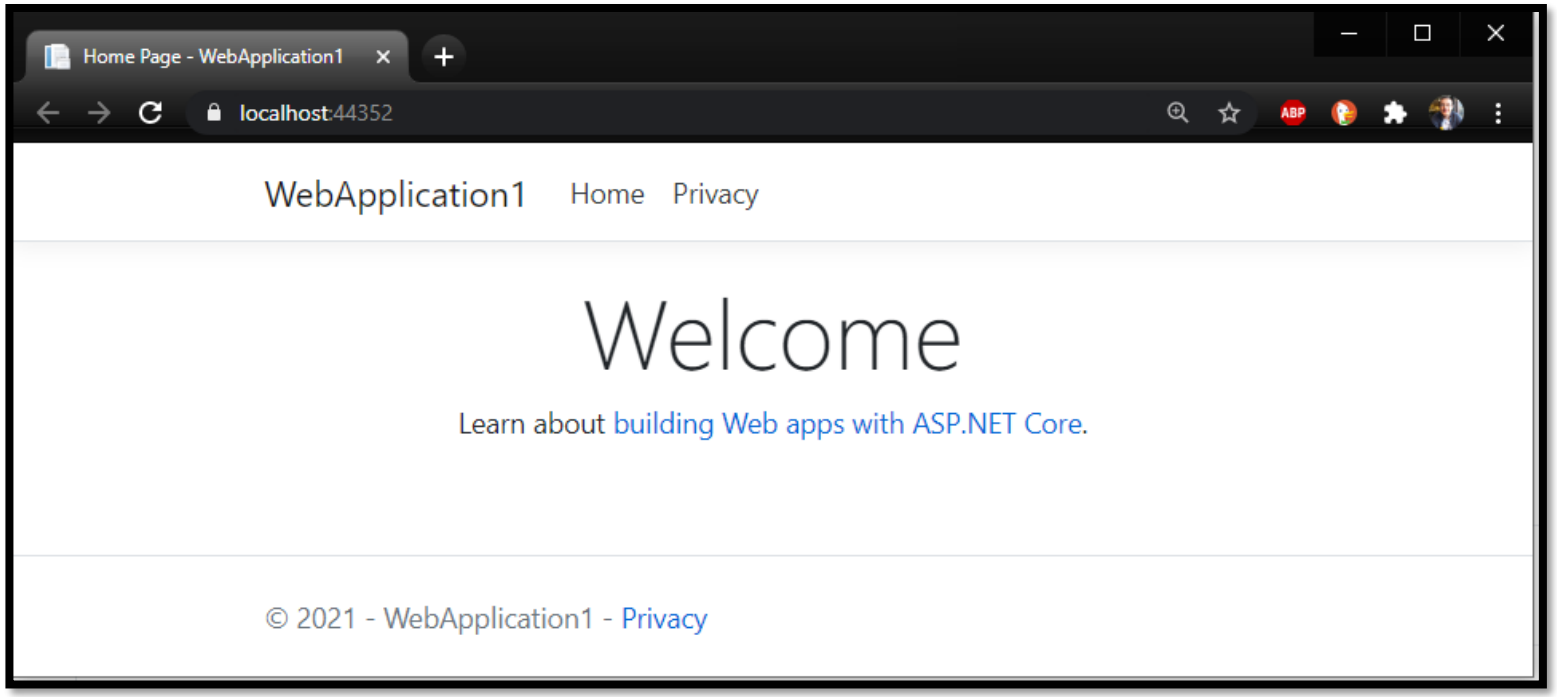
GET STARTED WITH ASP.NET CORE MVC

- Source:
  - https://github.com/dotnet/AspNetCore.Docs/blob/master/aspnetcore/tutorials/first-mvc-app/start-mvc.md
  - See also page 1621 from the official documentation pdf…

- Briefly go over the structure of the project …

- Then run the application (Ctrl+F5)





IN-CLASS DEMO

Demonstration: How to Explore an ASP.NET Core MVC Application

- Source/Steps
- https://github.com/MicrosoftLearning/20486D-DevelopingASPNETMVCWebApplications/blob/master/Instructions/20486D_MOD01_DEMO.md

Don't panic if what I cover doesn't completely make sense. We'll learn them in more details in the upcoming weeks. This is just an exploration!

DEMO

HomeController.cs

PhotoSharingSample | PhotoSharingSample.Controllers.HomeController | Index()

```
22
23      public IActionResult Index()
24      {
25          return View(_dbContext.Photos.ToList());
26      }
27
28      public IActionResult GetImage(int PhotoId)
29      {
30          Photo requestedPhoto = _dbContext.Photos.FirstOrDef
31          if (requestedPhoto != null)
32          {
33              string webRootpath = _environment.WebRootPath;
34              string folderPath = "\\images\\";
35              string fullPath = webRootpath + folderPath + re
36
37              FileStream fileOnDisk = new FileStream(fullPath
38              byte[] fileBytes;
39              using (BinaryReader br = new BinaryReader(fileO
```

Solution Explorer

Search Solution Explorer (Ctrl+;)

- Solution 'PhotoSharingSample' (1 project)
  - PhotoSharingSample
    - Connected Services
    - Dependencies
    - Properties
    - wwwroot
    - Controllers
      - HomeController.cs
    - Models
      - Photo.cs
      - PhotoSharingDB.cs
    - Views
      - Home
        - Index.cshtml
    - appsettings.json
    - Program.cs
    - Startup.cs

Solution Explorer | Team Explorer | Class View

Note: This code block sends a list of photos to the view.

Index.cshtml

```
1   @model IEnumerable<PhotoSharingSample.Models.Photo>
2
3   @{
4       Layout = null;
5   }
6
7   <!DOCTYPE html>
8
9   <html>
10  <head>
11      <meta name="viewport" content="width=device-width" />
12      <title>Home page</title>
13
14      <link type="text/css" rel="stylesheet" href="~/css/photo-sharing-styles.css" />
15  </head>
16  <body>
17      <h1 class="main-title">Photos Unlimited</h1>
18      <p class="into">Join, upload, share, and comment instantly. Bring your photographic
19      <p class="into">Our latest Photos</p>
20      <br />
21      @foreach (var item in Model)
22      {
23          <div class="photo-index-card">
24              <h3 class="display-picture-title">
25                  "@Html.DisplayFor(modelItem => item.Title)"
26              </h3>
27              @if (item.PhotoFileName != null)
```

Solution Explorer

Search Solution Explorer (Ctrl+;)

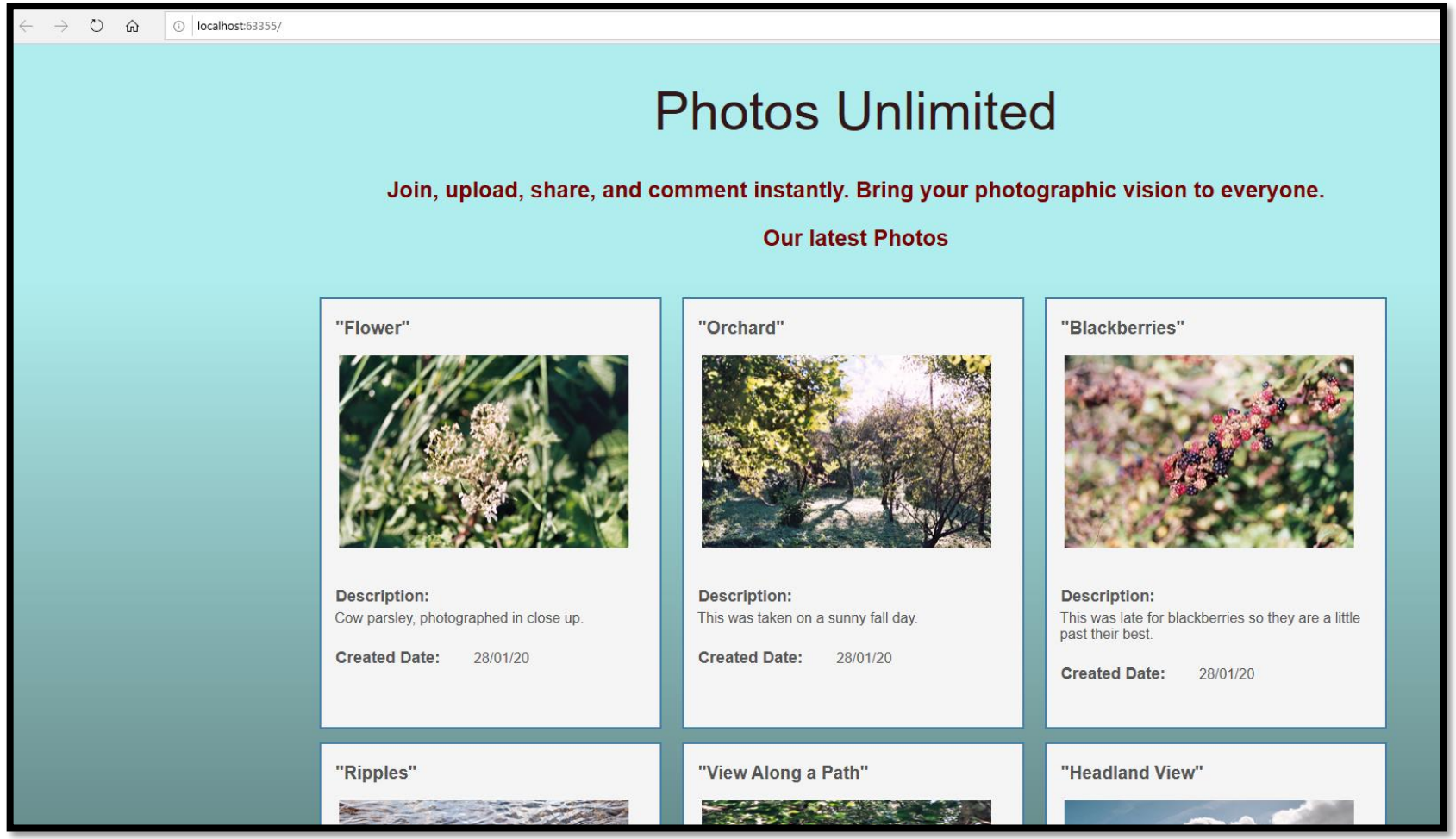
- Solution 'PhotoSharingSample' (1 project)
  - PhotoSharingSample
    - Connected Services
    - Dependencies
    - Properties
    - wwwroot
    - Controllers
      - HomeController.cs
    - Models
      - Photo.cs
      - PhotoSharingDB.cs
    - Views
      - Home
        - Index.cshtml
    - appsettings.json
    - Program.cs
    - Startup.cs

Solution Explorer | Team Explorer | Class View

Properties

Note: This code block represents how the view accepts the list of photos from the **Index** action.

DEMO



Note: The browser displays the default home page.
You have reached the **Index** action of the **Home** controller.

DEMO

Startup.cs

PhotoSharingSample | PhotoSharingSample.Startup | Configure(IApplicationBuilder app, PhotoSharingDB photoSharingD

```
27
28          services.AddDbContext<PhotoSharingDB>(options =>
29                  options.UseSqlServer(_configuration.GetConnectionString("PhotoSharing
30      }
31
32      public void Configure(IApplicationBuilder app, PhotoSharingDB photoSharingDB)
33      {
34          photoSharingDB.Database.EnsureDeleted();
35          photoSharingDB.Database.EnsureCreated();
36
37          app.UseStaticFiles();
38
39          app.UseMvcWithDefaultRoute();
40      }
41  }
42 }
43
```

Solution Explorer

Search Solution Explorer (Ctrl+;)

- Solution 'PhotoSharingSample' (1 project)
  - PhotoSharingSample
    - Connected Services
    - Dependencies
    - Properties
    - wwwroot
    - Controllers
      - HomeController.cs
    - Models
      - Photo.cs
      - PhotoSharingDB.cs
    - Views
      - Home
        - Index.cshtml
    - appsettings.json
    - Program.cs
    - Startup.cs

Note: This code block adds MVC to the request execution pipeline,

with a default route which contains the following template:

{controller=Home}/{action=Index}/{id?}.

DEMO

Photo.cs

PhotoSharingSample | PhotoSharingSample.Models.Photo | PhotoID

```
8
9   namespace PhotoSharingSample.Models
10  {
11      public class Photo
12      {
13          public int PhotoID { get; set; }
14
15          [Required]
16          public string Title { get; set; }
17
18          [DisplayName("Picture")]
19          [MaxLength]
20          public string PhotoFileName { get; set; }
21
22          [HiddenInput(DisplayValue = false)]
23          public string ImageMimeType { get; set; }
24
25          [DataType(DataType.MultilineText)]
26          public string Description { get; set; }
27
28          [DataType(DataType.DateTime)]
29          [DisplayName("Created Date")]
30          [DisplayFormat(DataFormatString = "{0:dd/MM/yy}", ApplyFormatInEditMode = true)]
31          public DateTime CreatedDate { get; set; }
32      }
33  }
```

Solution Explorer

Search Solution Explorer (Ctrl+;)

- Solution 'PhotoSharingSample' (1 project)
  - PhotoSharingSample
    - Connected Services
    - Dependencies
    - Properties
    - wwwroot
    - Controllers
      - HomeController.cs
    - Models
      - Photo.cs
      - PhotoSharingDB.cs
    - Views
      - Home
        - Index.cshtml
    - appsettings.json
    - Program.cs
    - Startup.cs

Solution Explorer | Team Explorer | Class View

Properties

Note: This code block represents the **Title** property for a photo stored in the application.

DEMO

HomeController.cs

PhotoSharingSample | PhotoSharingSample.Controllers.HomeController | GetImage(int PhotoId)

```
24      {
25          return View(_dbContext.Photos.ToList());
26      }
27
28      public IActionResult GetImage(int PhotoId)
29      {
30          Photo requestedPhoto = _dbContext.Photos.FirstOrDefault(p => p.PhotoID == PhotoId);
31          if (requestedPhoto != null)
32          {
33              string webRootpath = _environment.WebRootPath;
34              string folderPath = "\\images\\";
35              string fullPath = webRootpath + folderPath + requestedPhoto.PhotoFileName;
36
37              FileStream fileOnDisk = new FileStream(fullPath, FileMode.Open);
38              byte[] fileBytes;
39              using (BinaryReader br = new BinaryReader(fileOnDisk))
40              {
41                  fileBytes = br.ReadBytes((int)fileOnDisk.Length);
42              }
43              return File(fileBytes, requestedPhoto.ImageMimeType);
44          }
45          else
46          {
47              return NotFound();
48          }
49      }
```

Solution Explorer

Search Solution Explorer (Ctrl+;)

- Solution 'PhotoSharingSample' (1 project)
  - PhotoSharingSample
    - Connected Services
    - Dependencies
    - Properties
    - wwwroot
    - Controllers
      - HomeController.cs
    - Models
      - Photo.cs
      - PhotoSharingDB.cs
    - Views
      - Home
        - Index.cshtml
    - appsettings.json
    - Program.cs
    - Startup.cs

Solution Explorer | Team Explorer | Class View

Properties

Note: This code block represents the **GetImage** action of the **HomeController** class.

DEMO

HomeController.cs | Index.cshtml

```
@model IEnumerable<PhotoSharingSample.Models.Photo>

@{
    Layout = null;
}

<!DOCTYPE html>

<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>Home page</title>

    <link type="text/css" rel="stylesheet" href="~/css/photo-sharing-styles.css" />
</head>
<body>
    <h1 class="main-title">Photos Unlimited</h1>
    <p class="into">Join, upload, share, and comment instantly. Bring your photographic vision to everyone.</p>
    <p class="into">Our latest Photos</p>
    <br />
    @foreach (var item in Model)
    {
        <div class="photo-index-card">
            <h3 class="display-picture-title">
                "@Html.DisplayFor(modelItem => item.Title)"
            </h3>
            @if (item.PhotoFileName != null)
            {
                <div class="photo-display">
                    <img class="photo-display-img" src="@Url.Action("GetImage", "Home", new { PhotoId = item.PhotoID })" />
                </div>
            }

            <div>
                <span class="display-label">
                    @Html.DisplayNameFor(model => model.Description):
                </span>
                <br />
                <span class="display-field">
                    @Html.DisplayFor(model => item.Description)
                </span>
            </div>
```

Solution Explorer

Search Solution Explorer (Ctrl+;)

- Solution 'PhotoSharingSample' (1 project)
  - PhotoSharingSample
    - Connected Services
    - Dependencies
    - Properties
    - wwwroot
    - Controllers
      - HomeController.cs
    - Models
      - Photo.cs
      - PhotoSharingDB.cs
    - Views
      - Home
        - Index.cshtml
    - appsettings.json
    - Program.cs
    - Startup.cs

Solution Explorer | Team Explorer | Class View

Properties

Note: The Razor view engine runs this code and renders the **Photo** image.

QUERY STRINGS

- A query string is the part of a URL after the question mark

https://www.amazon.com/Coupons/b/?_encoding=UTF8&node=2231352011&ref_=nav_cs_coupons

- We use **query strings** to send values to a server.
  - Note: these are sent in plain sight, so do not send usernames and passwords using query strings!
- Typically, we use the query strings when we want to preserve a small amount of data from one page request to another. For example when you search for "laptops"
- All browsers support query strings, but some impose a URL length limit of 2,083 characters.
  - A more secure way, and without a limit, is to send values using the request **body.**
- See also:
  - https://support.microsoft.com/en-us/topic/maximum-url-length-is-2-083-characters-in-internet-explorer-174e7c8a-6666-f4e0-6fd6-908b53c12246

REQUEST LIFE CYCLE – IF TIME

- Note: We'll see details in the upcoming weeks!

- Here is an example: a request for the details of a product with the ID "1":

1. The **user requests the web address**: http://www.adventure-works.com/product/display/1
2. The **MVC routing engine** examines the request and determines that it should forward the request to the **Product controller,** namely the **Display** action
3. Then, the **Display** action (that is part of the **Product** Controller) creates a new instance of the **Product** model class.
   - The **Product** model instance would **query the database** for the record containing the product with ID "1"
4. The **Display** action then creates an instance of a **View** and passes the **Product** model to it.
5. The **Razor view engine** runs the server-side code in **Product Display View** to render **HTML**.
   - For example, the server-side code could insert the values of Title, Description, Catalog Number, and Price into the HTML.
6. The rendered **HTML page is then returned to the browser** that made the request and is being displayed.

IF TIME

- You'll need to know:
- Classes
- Interfaces
- Inheritance