

CSC496/CSC584 Winter 2018-2019

Course Outline

We will use ASP.NET Core 2.2 for all our projects and lectures. You may use either a Windows machine or a Mac OS X machine for all projects. For students using Windows machines, you can use the localDB as the database management system. For students using Mac OS X machines, you can use the SQLite database management system.

1. **12/17 .NET, .NET Core, ASP.NET, ASP.NET Core, ASP.NET Core Web App, Web API**
[\(https://docs.microsoft.com/en-us/dotnet/standard/\)](https://docs.microsoft.com/en-us/dotnet/standard/)

What is the State of the Art of the Emerging Web Technology?

As you will find out that since Microsoft tried to support ALL OS environments, using multiple IDEs (Visual Studio, Visual Studio on MAC, Visual Studio Core), using multiple but limited Database Providers (LocalDB on Windows, and SQLite on Mac), the learning curve and the reference material are lacking. The current statue quote can be considered as a turmoil state. When you follow the tutorial for Visual Studio for MAC until a step, the direction will point you to Visual Studio 2017 for Windows. We can conclude that the development of the emerging web technology generally known as .NET Core is still in progress! What we will use is the online material published on Microsoft official website step by step. Sometimes, we will follow the link posted on the left side of the webpages, while in some other occasions, we will have to work by clicking a “Next” button at the end of each tutorial. Even the style of the writing is a mess! The purpose of this Outline is to tell you what we expect you to do. If you miss any lecture, please fall back to this Outline to find out what we anticipated.

We now start with .NET Core – a new development of Microsoft unveiled in 2017 and has been a moving target since it was released. Books written and published in 2017 were using .Net Core 1.1 and has become outdated already. We could only use the online tutorial as our reference. I’ve tried one book and was able to convert the book examples to .NET Core 2.0. But it does not work on Mac OS X.

What is .NET and .NET Core?

A web server development environment/system, originally known as ASP, was used to handle server-side processing by moving away from the concept of creating a process for each HTTP request, and use multithreading instead! Later ASP turned into ASP.NET to develop an object-oriented platform and development environment. ASP supports interpreted scripts and ASP.NET supports compiled code. Later, ASP.NET used the concept of “Web Forms” and GUI-based IDE for server-side scripting! Hence, the software system is called ASP.NET Web Form. It later involved into ASP.NET MVC to

embrace the concept of MVC –(M)odel, (V)iew, and (C)ontroller. Now, the newly developed ASP.NET Core was replacing ASP.NET with the support of multi-platforms. It remains to be seen whether or not this move toward a new direction will be a success! Good news is that you don't have to know ASP to learn ASP.NET Core MVC!

You need to know that: (1) .NET is not .NET Core, (2) ASP.NET is not ASP.NET Core, (3) ASP.NET Web Form is not ASP.NET Core MVC, (4) ASP.NET Core Web Application is not ASP.NET Core Web API. Our targets are: (1) .NET Core and C#, (2) ASP.NET Core, and (3) Data Accessing. By ASP.NET Core, we meant ASP.NET Core Web Application with MVC components, and ASP.NET Core Web API with RESTful Services. Either of these two platforms can be used with or without using databases. When there are so many kinds of database management system providers, the development environment we use will follow what the tutorial supports. It will be an adventure if you attempt to wonder away from what's stated, I guarantee!

Luckily, the concept of MVC somewhat clears all these dusts! The letter 'M' stands for Model and is generally related to Database Accessing; the letter 'V' stands for View and is generally related to User-Side Scripting for presentation; and the letter 'C' stands for Controller and is generally related to the Server-Side Scripting. However, even MVC can be used for all possible components in AngularJS that is used for developing user-side scripts since all three components are needed in AngularJS. In this course, we will not cover the concepts or tools for user-side scripting because it is much easier to handle. We will leave it to future courses.

What is .NET Core?

I haven't answer this question, have I? The .NET Core is a platform for web development that used the traditional .NET architecture with additional multi-platform support. The recent development of Node.JS also influence the concept of Multi-Threading for developing a server. In Node.JS, a server works with a single thread to support Asynchronous Programming. The idea of using **async** and **await** keywords to annotate methods when introducing the concept of synchronization into .NET Core C#. In our lecture, you will run into the concept of asynchronous programming. But we will not discuss it in depth due to the lack of teaching material. Generally speaking, this concept is good for serving a **Single-Page Application (SPA)** or data-bound applications. Hence, when many web page requests can be processed with a single thread using non-blocking asynchronous programming! We may run into this emerging concept more often from this time on! That is really the "emerging" part of the web technology.

In this course, we begin with some terms and then use some on-line tutorial to work on lab exercises. As you follow the tutorial to work on examples, the results will be assigned as Projects and you can submit a Word Document showing some screen shots to illustrate your work for submission to the D2L Assignment folder. At the end of this

class, we will have one exam to focus on concepts and terms with the usage of the tool Visual Studio 2017.

Prepare for Actions

1. For Windows-based PC: (a) Download Visual Studio 2017 Community Edition, (b) Download .NET 2.0. If you download and install Visual Studio 2017 **Enterprise Edition** on a Windows-based PC, you need to download **PowerShell 3.0** that is enclosed in Windows Framework Management 3.0 or higher. If you are running Windows 7, you need to make sure that the Windows Service Pack 1 is installed.
2. For Mac OS X machines, you need the following: (a) Download Visual Studio 2017 Community Edition, (b) Download .NET 2.0.

Online Tutorial Labs about .NET Core

Since it is easier to find the counter part of the tutorial online, I am describing the lab exercises for Mac OS X. We will follow the following links to complete several lab exercises.

- a. Concept we will use: <https://docs.microsoft.com/en-us/dotnet/core/tutorials/using-on-mac-vs> (for MAC OS X). You can find similar tutorials for Windows (<https://docs.microsoft.com/en-us/dotnet/core/tutorials/using-on-windows-full-solution>).
- b. Building a complete .Net Core Solution on Mac OS Using Visual Studio on Mac. (<https://docs.microsoft.com/en-us/dotnet/core/tutorials/using-on-mac-vs-full-solution/> or <https://docs.microsoft.com/en-us/dotnet/core/tutorials/using-on-windows-full-solution> on windows.)
- c. Get Started with .NET Core CLI Commands (<https://docs.microsoft.com/en-us/dotnet/core/tutorials/using-with-xplat-cli>)
- d. About .NET Core: <https://docs.microsoft.com/en-us/dotnet/core/about>
- e. Choose between .NET Framework and .NET Core: <https://docs.microsoft.com/en-us/dotnet/standard/choosing-core-framework-server>

Now C#!

.NET C# Language

- f. Inside a C# Program: <https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/inside-a-program/>

- g. C# and .NET: <https://docs.microsoft.com/en-us/dotnet/csharp/getting-started/introduction-to-the-csharp-language-and-the-net-framework>
- h. Tour of C# - <https://docs.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/>
- i. **C# Concepts:** <https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/>
- j. **C# Programming Guide:** <https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/>
 - i. Basic Types, Arrays
 - ii. Classes & Structs
 - iii. Interfaces
 - iv. Enumerated Types
 - v. Methods
 - vi. Properties

Exercises: Reference –

<https://docs.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/classes-and-objects>

- a. Develop a base interface GeometryObject with x-axis and y-axis and MoveTo() method.
- b. Develop a Point with x-axis and y-axis using inheritance.
- c. Implement a MoveTo() method.
- d. Develop a class Line characterizing with two Point objects.
- e. Add a method to check if two lines of type Line are in parallel.

Note: C# supports Object.ToString() but not Object.toString().

- 2. **12/18 C# Programming Guide (Continued)** These examples are taken from the following two series of C# programming references:

- a. <https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/>

- b. <https://docs.microsoft.com/en-us/dotnet/csharp/tutorials/>

- i. Interfaces
- ii. Delegates
- iii. Properties
- iv. Indexers
- v. Events
- vi. Generics
- vii. Iterators
- viii. LINQ Query Expressions
- ix. Lambda Expressions
- x. Namespaces
- xi. Events
- xii. Indexers

Let's try some hands-on labs:

We can modify the HelloWorld project to try the concept of using a delegate to invoke methods. You can proceed now.

a. **Delegates with Named vs. Anonymous Methods**

<https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/delegates/delegates-with-named-vs-anonymous-methods>

b. **Using Delegates:**

<https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/delegates/using-delegates>

c. **Combine Delegates**

<https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/delegates/how-to-combine-delegates-multicast-delegates>

d. **Generics**

<https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/generics/>

e. **Inheritance:** Besides any types that may inherit from through single inheritance, all types in the .NET type system implicitly inherit from **Object** or a type derived from it. This is also known as “implicit inheritance”.

<https://docs.microsoft.com/en-us/dotnet/csharp/tutorials/inheritance>

f. **Console Application**

<https://docs.microsoft.com/en-us/dotnet/csharp/tutorials/console-teleprompter>

g. **String Interpolation**

<https://docs.microsoft.com/en-us/dotnet/csharp/tutorials/string-interpolation>

h. **Lambda Expression**

<https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/statements-expressions-operators/lambda-expressions>

i. **Overloaded Operators**

C# allows user-defined types to overload operators by defining static member functions using the **operator** keyword. Not all operators can be overloaded, however, and others have restrictions. Check the table:

<https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/statements-expressions-operators/overloadable-operators>

j. **yield and iterator**

Use the example program “Power of 2” to learn the concept of “yield”.

<https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/keywords/yield>

Q: How do we use “yield”?

A: You use a `yield return` statement to return each element one at a time. You consume an iterator method by using a `foreach` statement or LINQ query.

Q: What is the purpose of using 'yield' in a method returning an `IEnumerable<T>` data type?

A: Try the examples and think!!

Exercises: Go over the examples in the tutorial. In particular, you need to look into these examples in C# programming Guide:

<https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/>

3. 12/19 ASP.NET Core MVC with Razor Pages:

<https://docs.microsoft.com/en-us/aspnet/core/mvc/overview>

Summary: Now we have learned C# and the concept about .NET Core briefly. We are going to use this summary to wrap up what we have learned and prepare for the next topic: ASP.NET Core.

<https://docs.microsoft.com/en-us/visualstudio/get-started/csharp/tutorial-aspnet-core?view=vs-2017>

Q: What does MVC stand for?

A:

Once completed, you should be expected to know the following:

(1) Become familiar with the Visual Studio 2017 new project templates:

- a. Web Applications > Web API:
 - i. w/o Angular-related packages
 - ii. w/ controllers only (no models, no views)
 - iii. Cannot be browsed, can be tested with Postman
- b. Web Application > Web Application:
 - i. w/o any MVC folders
 - ii. w/ bower.json
- c. Web Application > Web Application(MVC)
 - i. w/ all three folders
 - ii. Inside view, there are two folders: Home, Shared (containing `_Layout.cshtml`)
 - iii. can be browsed

(2) Learn the concept of Razor Pages.

(3) Understand the features provided by Visual Studio 2017.

Razor Page Lab Exercises

We now begin the expedition of Razor Pages. We will click the side links one at a time. Do not use the links at the end of the first web page.

Windows:

<https://docs.microsoft.com/en-us/aspnet/core/tutorials/razor-pages/razor-pages-start>

- a. Create a Razor Pages web app on Windows
- b. Getting Started
- c. Adding a model
- d. Add scaffolding
- e. Update the pages
- f. (CSC584) Adding a search

Mac OSX: Follow the follow link to complete a tutorial project.

<https://docs.microsoft.com/en-us/aspnet/core/mvc/razor-pages/?tabs=visual-studio>

- a. Razor Page Project: The following link brings up an “introduction to Razor Pages” instead of a tutorial. It does not tell you detailed steps to complete the project. You may get confused easily. You try this only when you complete the tutorial. **Do not follow the recommendation** described at the beginning of the tutorial to try this link.

<https://docs.microsoft.com/en-us/aspnet/core/mvc/razor-pages/?tabs=visual-studio>

- b. Razor Syntax: <https://docs.microsoft.com/en-us/aspnet/core/mvc/views/razor>
- c. Adding a model
- d. Test it with SQLite – I will tell you what to do!!
- e. Add scaffolding
- f. Work with a DB
- g. Update the pages
- h. (CSC584) Add a search

4. 12/20 Work on Project #1(No Lecture)

- a. Razor Page Project - Windows: (<https://docs.microsoft.com/en-us/aspnet/core/mvc/razor-pages/?tabs=visual-studio>)
- b. Razor Syntax (<https://docs.microsoft.com/en-us/aspnet/core/mvc/views/razor>)
- c. Adding a model

- d. Adding Scaffolding
- e. Adding a search
- f. Adding a new field

5. **12/21 Work on Project #1(No Lecture)**

Winter Break until 1/1/2019. See you on 1/2/2019!

Merry Christmas and a Happy New Year!

6. 1/2 – 1/4 & 1/7 ASP.NET Core MVC Web App

1/2

Part II-Review of Razor Pages

We will take the approach that you worked on a project following the Online Tutorial first, then I will explain some important concepts wrapping up the major ideas in each topic. Today, we will explain the concepts about Razor Pages today using the Introduction to Razor Pages posted at the following link:

<https://docs.microsoft.com/en-us/aspnet/core/mvc/razor-pages/?tabs=visual-studio>

In this introduction, it consists of several projects. **First**, the project name is any name in which the concept about Startup.cs and a basic Razor Pages file is introduced. We can learn how to display the current Data and Time. **Second**, the project name is RazorPagesIntro in which the concept about a Razor Pages file. The author of the tutorial uses the term “Razor view file” and “Razor Pages” file without explaining what they meant. A Razor view file is an HTML file with Razor syntax embedded inside the HTML document without <html> header and <body> elements. A Razor Pages file adds the “@Page” or related directive at the beginning of this type of “Razor” file. In this introduction, the concept of matching a Razor Pages file with the “Code-Behind” file. (I still like to borrow the ASP.NET MVC term “Code-Behind” file here.) If the Razor Pages file is called Pages/Index2.cshtml, the Code-Behind file is called Pages/Index2.cshtml.cs. We will skip the first basic project and try the second project below.

a. Create a Razor Pages project RazorPagesIntro –

Step 1:

We will take a look at the file Startup.cs. This is a very important concept in the implementation of ASP.NET Core projects and I quote: Razor Pages is enabled in Startup.cs:

In particular, the method ConfigureServices() includes one statement: services.AddMvc() that includes the support for Razor Pages.

Step 2:

Next, we need to check a Razor Pages file.

Q: What is the first “directive” in a Razor Page?

A:

Q: What is the purpose of this directive?

A: This directive “makes the file into an MVC action” – which means that it handles requests directly, without going through a controller.

Step 3:

The next step involves two introductory files: one is a Razor Page, and another is a Code-Behind File similar to the Code: index2.cshtml and index2.cshtml.cs. You can right click the “Pages” folder and choose the “add new item” option. Then you need to add a View Page(C#) for adding a .cshtml file, and an “Empty Class” (C#) for adding a .cshtml.cs file.

- b. Writing a Basic Form - Create a Razor Pages project RazorPagesContacts –
For this part of the “Introduction”, you need to create a Web Application solution with the name “RazorPagesContacts”.
Step 1. Create a web app (not MVC) with the name RazorPagesContacts.
Step 2. Create a folder Data and add a data model file Customer.cs, and a db context file ApplicationDbContext.cs.
Step 3. Create one Razor Pages file Create.cshtml in the Pages folder.
Step 4. Add one Code-Behind file Create.cshtml.cs in the Pages folder.
The Pages/Create.cshtml.cs code-behind file defines the PageModel (for storing the data model CreateModel class) and the OnPostAsync() method. This code-behind file replaces the need to define a “control” action inside the Controllers folder of an MVC project.

The purpose of this Code-Behind file separates the logic of a page from its presentation.

- c. Using Layouts, partials, templates, and Tag Helpers with Razor Pages
I will leave this part as Exercises. You may proceed following the directions described in the “Introduction page”.

Q: What is the purpose of Anchor Tag Helper “asp-route-id”?

A: The [Anchor Tag Helper](#) used the `asp-route-{value}` attribute to generate a link to the Edit page. The link contains route data with the contact ID. For example, `http://localhost:5000/Edit/1`.

- d. TempData – Skip this part.
- e. Compare Razor Pages with ASP.NET MVC (Core): What are the main purposes of using Razor Pages instead of MVC?
<https://stackify.com/asp-net-razor-pages-vs-mvc/>
We may need to postpone this comparison to the end of this week after we get a handle on how to develop an ASP.NET Core MVC project. But you may begin to read this article.

At least, you should be able to find out the differences between the file structures of an MVC project and that of a Razor Pages project. In MVC, you will find three folders: Models, Views, and Controllers. In a Razor Pages project, there are only Models (or Data) and Pages folders. Inside the Pages folder, each view page (.cshtml) comes with a Code-Behind file (.cshtml.cs). There are no Views or Controllers folders.

Exercises:

Follow the direction to the Introduction to Razor Pages and study the code. Stop at “XSRF/CSRF” and skip the rest. XSRF or CSRF stands for Cross-Site Request Forgery. Please refer to the webpage below if you are interested in knowing more.

[https://www.owasp.org/index.php/Cross-Site_Request_Forgery_\(CSRF\)](https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF))

When you follow the direction described previously to create a Web Form and bring up a browser for testing it, you will notice two “create” buttons on the index.cshtml view. The second “create” is a hidden anti-forgery button automatically included in Razor Pages. You can check the source of the Razor Pages webpage to check for the source code of the hidden field.

1/3-1/4

Part II-b ASP.NET Core MVC – Now you can move on to the second part of the course, i.e., ASP.NET Core MVC.

a. Windows - <https://docs.microsoft.com/en-us/aspnet/core/tutorials/first-mvc-app/>

1. [Getting started](#)
2. [Adding a controller](#)
3. [Adding a view](#)
4. [Adding a model -](#)
5. [Working with SQL Server LocalDB](#)

This is the place where the Windows and Mac OS X developers must perform different actions.

6. [Controller methods and views](#)
7. [Adding Search](#)
8. [Adding a New Field](#)
9. [Adding Validation](#)
10. [Examining the Details and Delete methods](#)

b. Mac OSX - <https://docs.microsoft.com/en-us/aspnet/core/tutorials/first-mvc-app-mac/>

11. [Getting started](#)
12. [Adding a controller](#)
13. [Adding a view](#)
14. [Adding a model:](#)

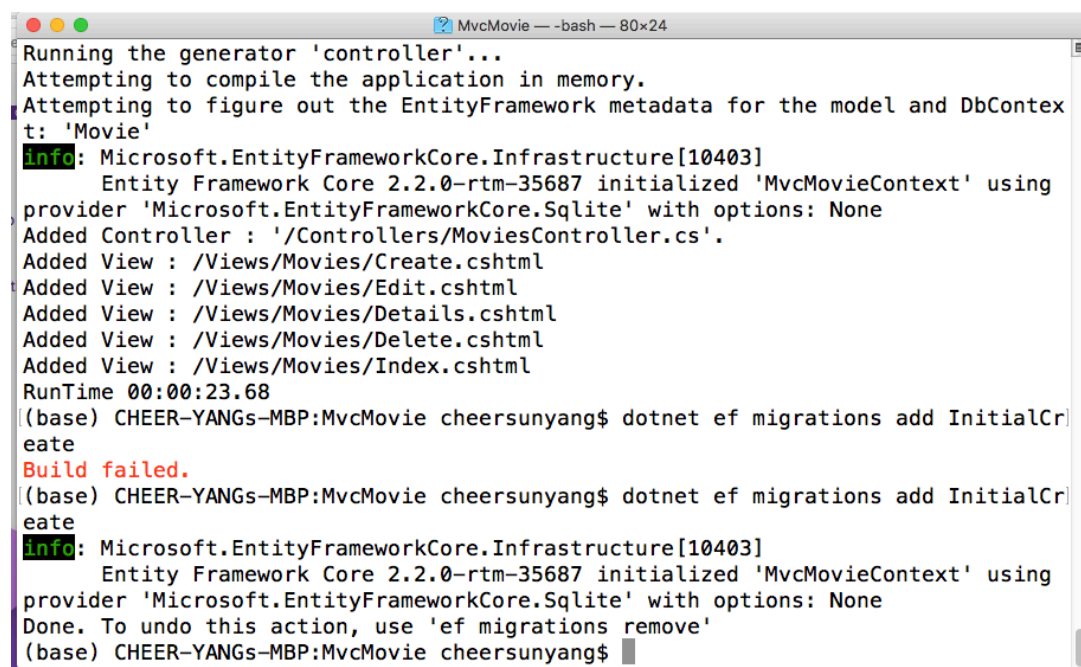
When you enter the dotnet command, you must change to the directory where the <name>.csproj is located first.

Before you enter the scaffolding tool, you need to get out of the visual studio for Mac and close the solution.

You may not need to install the aspnet-codegenerator tool since you might have done that when you worked on Project#1.

If you run into any error, please calm down and read the error message first.

Again, if you enter the “dotnet” command to conduct the migration without closing the project on Visual Studio first, you may receive an error message “Build failed.” Without giving any hint about the error. (This is bad.) I got the following screen:



```

Running the generator 'controller'...
Attempting to compile the application in memory.
Attempting to figure out the EntityFramework metadata for the model and DbContext: 'Movie'
info: Microsoft.EntityFrameworkCore.Infrastructure[10403]
      Entity Framework Core 2.2.0-rtm-35687 initialized 'MvcMovieContext' using
      provider 'Microsoft.EntityFrameworkCore.Sqlite' with options: None
Added Controller : '/Controllers/MoviesController.cs'.
Added View : /Views/Movies/Create.cshtml
Added View : /Views/Movies/Edit.cshtml
Added View : /Views/Movies/Details.cshtml
Added View : /Views/Movies/Delete.cshtml
Added View : /Views/Movies/Index.cshtml
RunTime 00:00:23.68
(base) CHEER-YANGs-MBP:MvcMovie cheersunyang$ dotnet ef migrations add InitialCreate
Build failed.
(base) CHEER-YANGs-MBP:MvcMovie cheersunyang$ dotnet ef migrations add InitialCreate
info: Microsoft.EntityFrameworkCore.Infrastructure[10403]
      Entity Framework Core 2.2.0-rtm-35687 initialized 'MvcMovieContext' using
      provider 'Microsoft.EntityFrameworkCore.Sqlite' with options: None
Done. To undo this action, use 'ef migrations remove'
(base) CHEER-YANGs-MBP:MvcMovie cheersunyang$

```

After closing the Visual Studio on Mac, I tried again and got it worked.

15. [SQLite](#)
16. [Controller methods and views](#)
17. [Adding Search](#)
18. [Adding a New Field](#)
19. [Adding Validation](#)

20. [Examining the Details and Delete methods](#)

1/7

Project 2:

1/8/2019 (1) Review of ASP.NET Core MVC (2) Web API

Review of ASP.NET Core MVC:

Before we move on to Web API, we need to review the concepts used in the ASP.NET Core MVC.

Q: What is the MVC Pattern?

A: <https://docs.microsoft.com/en-us/aspnet/core/mvc/overview>

Q: What is ASP.NET Core MVC?

A: ASP.NET Core MVC provides many features. Among them, there are some critical ones. We will explain some of them: Routing, Model Binding, Dependency Injection, Inversion of Control, etc.

Before we talk about all these features, we need to become aware of some “hidden rules” about the association of Controllers files and View files. First, we create a testMvc project, then we can explain several important concepts behind the scene. We will do the following:

1. Check the folders and files: Models, Views, Controllers, launchSettings.json (Where?), appsettings.json, Program.cs, Startup.cs.
2. Check each one of them and understand what the contents do.
3. Check Views. Know what folders are included, and what files are included in Home.
4. Find the Index.cshtml and the HomeController.cs files.
5. Investigate how they are related: You may comment out the Contact method in the HomeController.cs and rebuild the project.
6. Now uncomment it and put it back.
7. Investigate the effect of renaming the name of Contact.cshtml to Contact1.cshtml.
8. Now you should understand a rule about how a controller method is related with a view file. Is this a good thing comparing with the Razor Pages? For example, we need to store a Java class “public class Test” in a test.java file. Is this a good rule? Pros and cons? A razor Pages file uses <name>.cshtml and <name>.cshtml.cs to store the View and the Controller files, respectively. Is this better?
9. We need to know these details before we talk about Dependency Injection.
10. We will turn our attention to Program.cs and Startup.cs. What we ask you to do is to comment out some “using” directives and see what happens. By doing so, we see some wavy lines indicating some “undefined data types”.
11. Now we need to turn our attentions to some of the features.

Features of ASP.NET Core MVC

<https://docs.microsoft.com/en-us/aspnet/core/mvc/overview>

1. Routing: What is it?
2. Model Binding <https://docs.microsoft.com/en-us/aspnet/core/mvc/models/model-binding>

Model Binding Model binding in ASP.NET Core MVC maps data from HTTP requests to action method parameters.

3. Model Validation: How does MVC supports validation?
4. Dependency Injection
5. (We will skip the rest items on this list. We will talk about others later.)

Now we can turn to the concept of Dependency Injection.

Q: What is Dependency Injection into controllers?

A: <https://docs.microsoft.com/en-us/aspnet/core/mvc/controllers/dependency-injection> and Chapter 13 in the book mentioned below.

Q: What is Inverse of Control?

A: Chapter 13 in <http://www.cs.unsyiah.ac.id/~frdaus/PenelusuranInformasi/File-Pdf/Professional%20ASP.NET%20MVC%205.pdf>

Instead of hard-code a call to `Dao.GetData(request)`, the constructor assigns a parameter and uses a callback to the method passed in as an argument. By doing so, we allow the dependency on the implementation of `Dao.GetData()` out of the constructor. The control is “inversed” to the method being used, i.e., `Dao.GetData()`. You can also use a Service Locator to achieve the IoC.

Instead of this:

```
public class ServerFacade {
    public <K, V> V respondToRequest(K request) {
        if (businessLayer.validateRequest(request)) {
            DAO.getData(request);
            return Aspect.convertData(request);
        }
        return null;
    }
}
```

Use this and the control of the implementation for the `ServerFacade()` is no longer dependent of the implementation of `Dao.GetData()` because the implementation of DAO will be passed in as a parameter and is no longer hard-coded.

```
public class ServerFacade {
    public <K, V> V respondToRequest(K request, DAO dao) {
        return dao.getData(request);
    }
}
```

<Take a ten-minute break>

Now we can turn our attention to the association of a controller file with action methods, and the view file.

Q: How does Controller files (.cshtml.cs) become associated with View files (.cshtml)?

A: <https://docs.microsoft.com/en-us/aspnet/core/mvc/views/overview>

This web page introduces several important concepts used in ASP.NET Core MVC. We point out several of them from the excerpts of the article.

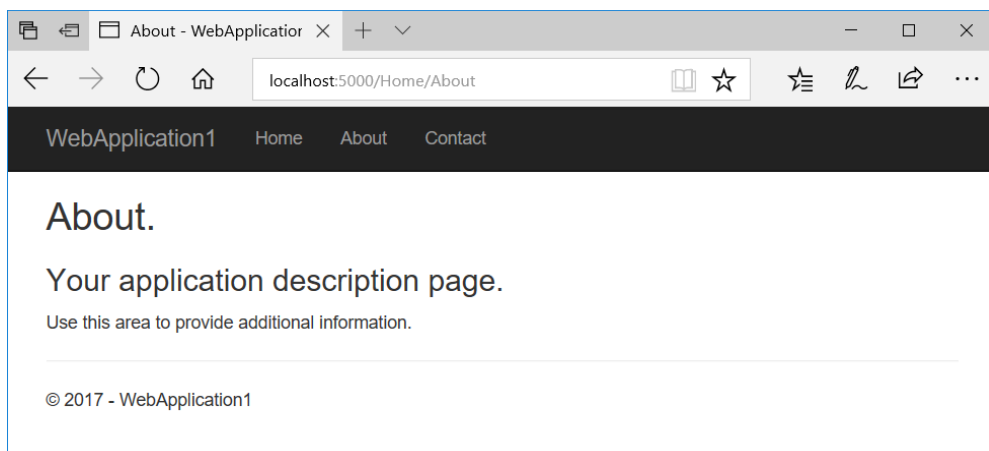
- a) Views (.cshtml) are associated with a controller file (.cshtml.cs) using the name convention described below.

Views are typically returned from actions as a `ViewResult`, which is a type of `ActionResult`. Your action method can create and return a `ViewResult` directly, but that isn't commonly done. Since most controllers inherit from `Controller`, you simply use the `View` helper method to return the `ViewResult`:

```
HomeController.cs
```

```
public IActionResult About()  
{  
    ViewData["Message"] = "Your application description page.";  
  
    return View();  
}
```

When this action returns, the `About.cshtml` view shown in the last section is rendered as the following webpage:



b) View discovery

When an action returns a view, a process called view discovery takes place. This process determines which view file is used based on the view name.

The default behavior of the View method (return View();) is to return a view with the same name as the action method from which it's called. For example, the About ActionResult method name of the controller is used to search for a view file named About.cshtml. First, the runtime looks in the Views/[ControllerName] folder for the view. If it doesn't find a matching view there, it searches the Shared folder for the view.

It doesn't matter if you implicitly return the ViewResult with return View(); or explicitly pass the view name to the View method with return View("<ViewName>");. In both cases, view discovery searches for a matching view file in this order:

- 1 Views/[ControllerName][ViewName].cshtml
- 2 Views/Shared/[ViewName].cshtml

c) Passing data to views

You can pass data to views using several approaches. The most robust approach is to specify a model type in the view. This model is commonly referred to as a viewmodel. You pass an instance of the viewmodel type to the view from the action.

Using a viewmodel to pass data to a view allows the view to take advantage of strong type checking. Strong typing (or strongly-typed) means that every variable and constant has an explicitly defined type (for example, string, int, or DateTime). The validity of types used in a view is checked at compile time.

Visual Studio and Visual Studio Code list strongly-typed class members using a feature called **IntelliSense**. When you want to see the properties of a viewmodel, type the variable name for the viewmodel followed by a period (.). This helps you write code faster with fewer errors.

Specify a model using the @model directive. Use the model with @Model:

```
@model WebApplication1.ViewModels.Address

<h2>Contact</h2>
<address>
  @Model.Street<br>
  @Model.City, @Model.State @Model.PostalCode<br>
  <abbr title="Phone">P:</abbr> 425.555.0100
</address>
```

To provide the model to the view, the controller passes it as a parameter:

```
public IActionResult Contact()
{
    ViewData["Message"] = "Your contact page.";

    var viewModel = new Address()
    {
        Name = "Microsoft",
        Street = "One Microsoft Way",
        City = "Redmond",
        State = "WA",
        PostalCode = "98052-6399"
    };

    return View(viewModel);
}
```

There are no restrictions on the model types that you can provide to a view. We recommend using Plain Old CLR Object (POCO) viewmodels with little or no behavior (methods) defined. Usually, viewmodel classes are either stored in the Models folder or a separate ViewModels folder at the root of the app. The Address viewmodel used in the example above is a POCO viewmodel stored in a file named Address.cs inside **Models**: (NOTE: change the sample code to stored in WebApplication1.Models.)

```
namespace WebApplication1.Models
{
    public class Address
    {
        public string Name { get; set; }
        public string Street { get; set; }
    }
}
```

```

    public string City { get; set; }
    public string State { get; set; }
    public string PostalCode { get; set; }
}
}

```

d) Weakly-typed data (ViewData and ViewBag)

Note: ViewBag is not available in the Razor Pages.

In addition to strongly-typed views, views have access to a weakly-typed (also called loosely-typed) collection of data. Unlike strong types, weak types (or loose types) means that you don't explicitly declare the type of data you're using. You can use the collection of weakly-typed data for passing small amounts of data in and out of controllers and views.

Passing data between a ...	Example
Controller and a view	Populating a dropdown list with data.
View and a layout view	Setting the <title> element content in the layout view from a view file.
Partial view and a view	A widget that displays data based on the webpage that the user requested.

e) ViewData

ViewData is a ViewDataDictionary object accessed through string keys. String data can be stored and used directly without the need for a cast, but you must cast other ViewData object values to specific types when you extract them. You can use ViewData to pass data from controllers to views and within views, including partial views and layouts.

The following is an example that sets values for a greeting and an address using ViewData in an action:

```

public IActionResult SomeAction()
{
    ViewData["Greeting"] = "Hello";
    ViewData["Address"] = new Address()
    {
        Name = "Steve",
        Street = "123 Main St",
        City = "Hudson",
        State = "OH",
        PostalCode = "44236"
    };

    return View();
}

```

```
}
```

Work with the data in a view CSHTML file:

```
@{
    // Since Address isn't a string, it requires a cast.
    var address = ViewData["Address"] as Address;
}
@ViewData["Greeting"] World!

<address>
    @address.Name<br>
    @address.Street<br>
    @address.City, @address.State @address.PostalCode
</address>
```

Exercise:

1. If you haven't done the exercise we had mentioned at the beginning of this lesson, i.e., comment out some directives and check what the purposes are for each directive, you may want to do that.
2. Using the testMvc project we created previously, you may test to see if you can add a string type data with the name "Person" and the value of your own name. You can change the Contact.cshtml file to display the name of the Person.
3. What folders are created by MVC? What folders are created in a Razor Pages project using the "ASP.NET Core Web App" option for creating a new project?

1/9 Prepare for ASP.NET Web API

<https://docs.microsoft.com/en-us/aspnet/core/web-api/?view=aspnetcore-2.1>

There are four out of the five tutorial experiments we can do following the tutorial of building WEB APIs. We will work on the following:

1. 1/9 - Building your first Web API with ASP.NET Core using Visual Studio
2. 1/10 - ASP.NET Core Web API Help Pages using Swagger
3. 1/11 – Project #3 Web API Tutorial and the Swagger Package (without the customization of UI- step 8 below).

1/9/2019 First Web API with ASPNET Core Using Visual Studio

<https://docs.microsoft.com/en-us/aspnet/core/tutorials/first-web-api>

We will use a Web API to implement a RESTful Web Service. Thus, we need to understand the concept of a Web Service first, then the RESTful.

A web service is the operations or actions a server provides to another server via the Internet. For example, if you purchase used books via Amazon.com, the server at Amazon.com will have to “talk” to other used-book sellers’ servers before Amazon.com can reply to a customer’s request. The interactions between the Amazon.com’s server and other used-book seller’s server is known as a Web Service. So, in the implementation of a web service, two parties are involved: a web service server and a web service client. However, in the previous example, both parties are actually “servers”. Therefore, instead of calling them servers or clients, they are all referred to as “end points”.

Traditionally, a web service was implemented with a specific protocol known as a Simple Object Access Protocol (SOAP). It is built on top of HTTP. SOAP works well when it supports for requests such as that used in Amazon.com to retrieve data from another end point, it also supports remote procedure calls. But, gradually, RPC-type of web services are not getting enough users and the other means of getting remote data was simplified. Another protocol known as RESTful Web Service was proposed by Dr. Roy Fielding in his dissertation. The term REST stands for REpresentational State Transfer (REST). The request for remote data is built-in in the header of HTTP requests, and the replies are represented in the format known as JavaScript Object Notation (JSON). This approach eliminates the additional layer of accessing software on top of HTTP. As a result, the performance is improved. It has several advantages over SOAP for transferring remote data. What a service provider needs to implement include similar operations as CRUD (Create, Read, Update, and Delete): Post, Get, Put, and Delete. The testing of a web service at the server side cannot be using a regular browser directly since the reply is not formatted in an HTML form. We will use a tool known as Postman to test the web service we create.

In the following several days from M-R, you will follow the online tutorial to become familiar with the development of a Web Service also known as a Web API. Then you will complete Project III.

Reference: RESTful Web Services: The Basics (IBM)

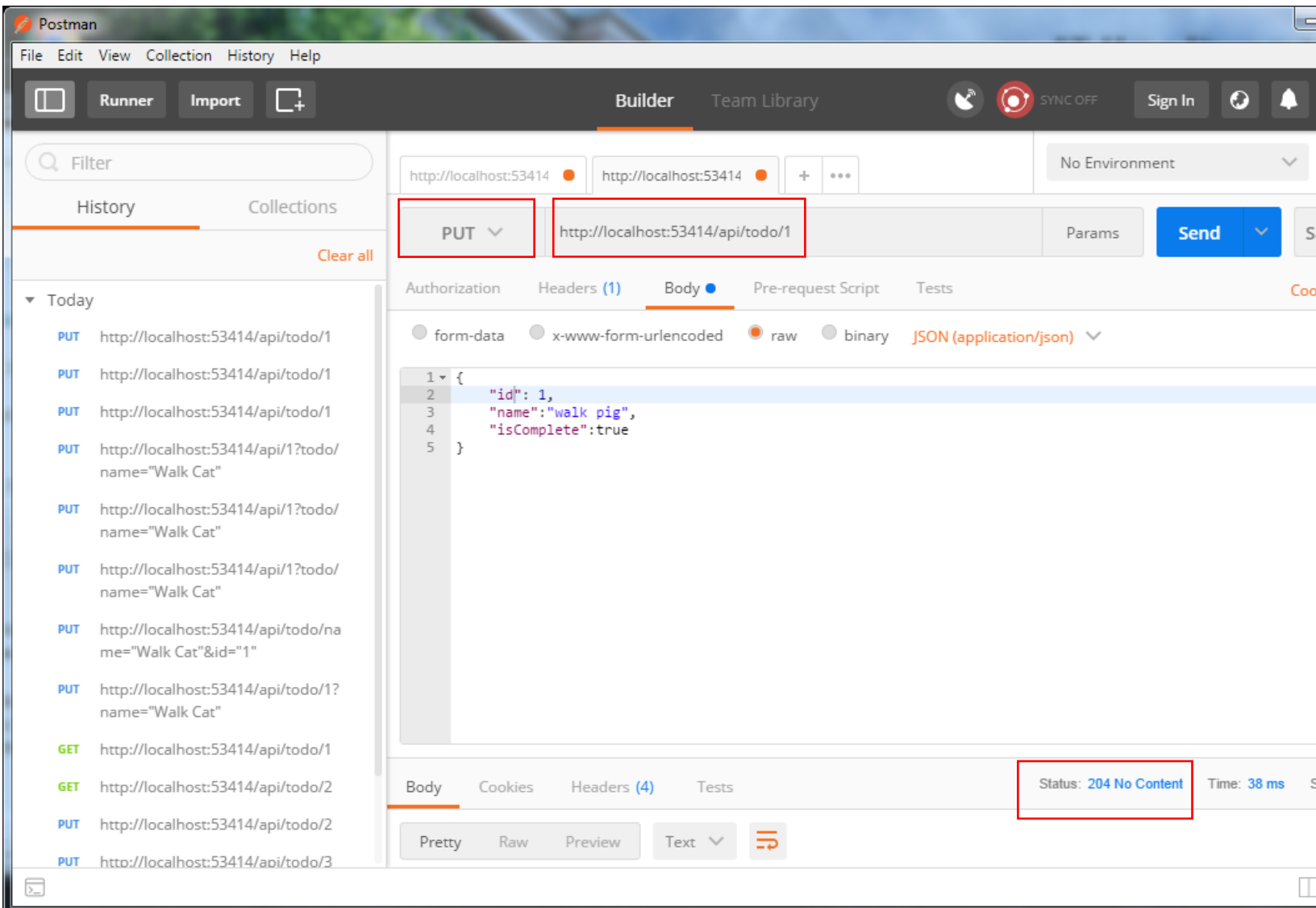
<https://www.ibm.com/developerworks/library/ws-restful/index.html>

1/9/2019

- a. Windows or Mac OS X: <https://docs.microsoft.com/en-us/aspnet/core/tutorials/first-web-api>
- b. After finishing the tutorial, you can find more references:
 - i. Routing: <https://docs.microsoft.com/en-us/aspnet/core/mvc/controllers/routing>
 - ii. File Uploads:
 - iii. Dependency Injection:
 - iv. Testing:
 - v. Advanced:

Do the following Steps:

- a. Follow the online tutorial to create a Web API project on Windows/Mac OS X.
- b. Test run the project and make sure it works.
- c. Click “browse with Google Chrome” for the test run
- d. Download Postman and install it.
- e. Execute the Postman.
- f. Change the controller file TodoController.cs to add Create code and route.
- g. For testing with Postman, you need to provide a JSON file with two properties: “name” and “isComplete”. Check the output status, body to make sure it is correct.
- h. When you test your code, if you forget to rebuild your code, you may receive a “**405 Method Not Allowed**” error. It is because the action specified in the URL is not correct. For example, if you add the code to do “POST”, but you **forget to rebuild your code**. The POST operation is considered a method that is not allowed. Also, if you enter a POST request and specify the URL as <http://localhost:5000/api/todo/3>, you will get a “405 Method not Allowed” error because **the ID of a record cannot specified by a user** via the URL. The correct POST request is <http://localhost:5000/api/todo/>.
- i. For Update, you need to add the PutTodoItem() code in the TodoController.cs.
- j. For testing, you need to use “PUT” with THREE properties: “id”, “name”, and “isComplete”. The return status is “204 no content”. But if you use “Get” to browse the record, it should be updated to the new values.



1/10 -

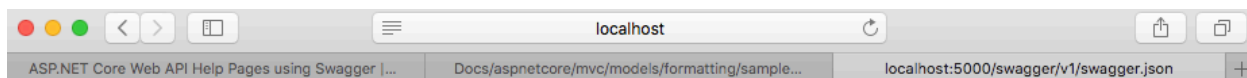
ASP.NET Core Web API Help Pages Using Swagger

<https://docs.microsoft.com/en-us/aspnet/core/tutorials/web-api-help-pages-using-swagger?view=aspnetcore-2.1>

Please follow the tutorial to complete the development of help pages for the Web API. You must complete the first Web API project first before you can try this tutorial in the same project.

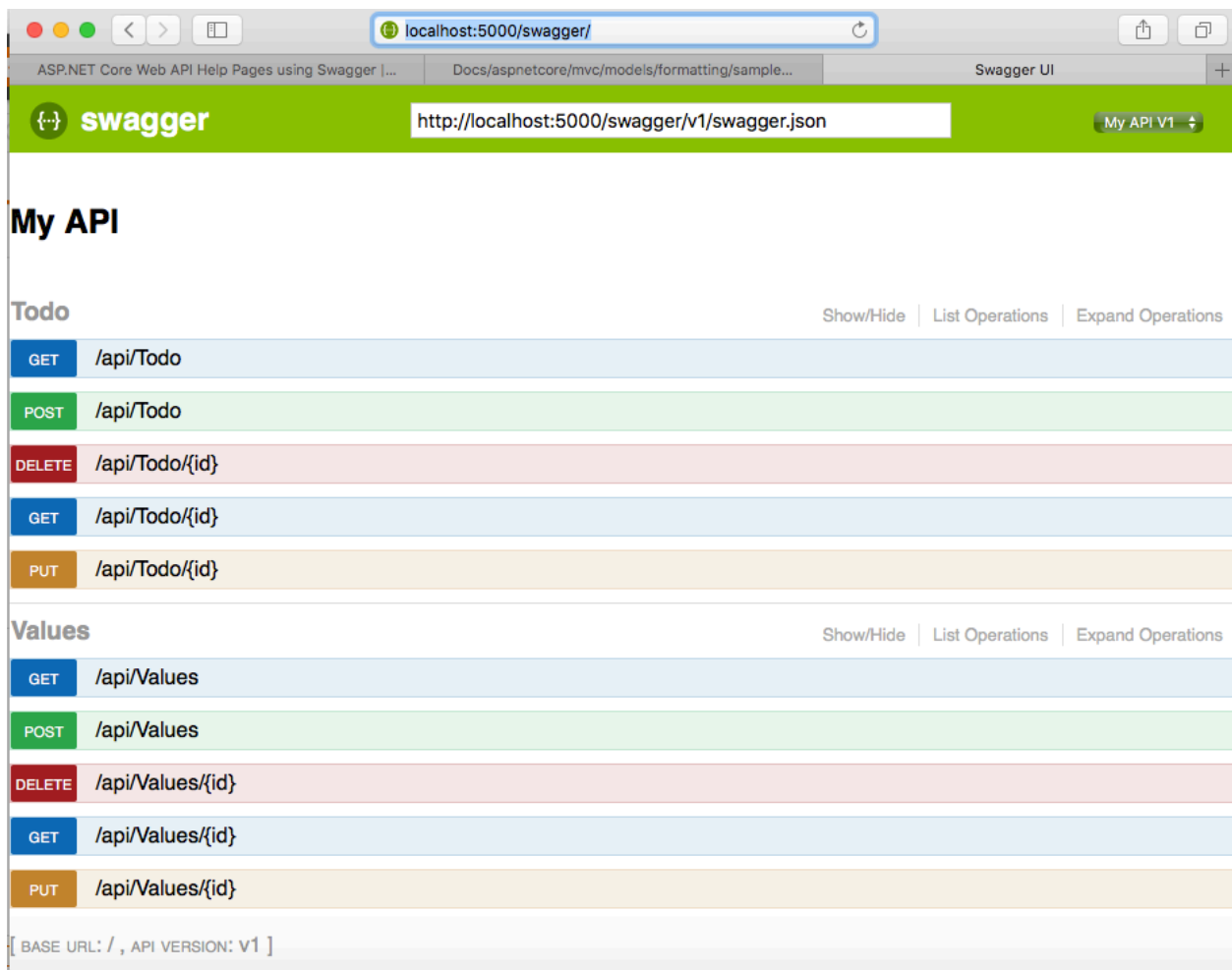
We will follow the following steps to complete the tutorial experiments.

1. Add a NuGet Package Swashbuckle.AspNetCore to your project.
2. Change your Startup.cs to add the Swagger generator to the services collection in the ConfigureServices method of Startup.cs.
3. You will need to add a using directive Swashbuckle.AspNetCore.Swagger.
4. Change the Configure method as indicated in the tutorial.
5. Test your project by entering the URL as:
<http://localhost:5001/swagger/v1/swagger.json>
6. You may see the following:



```
{
  "swagger": "2.0",
  "info": {
    "version": "v1",
    "title": "My API",
    "basePath": "/",
    "paths": {
      "/api/ToDo": {
        "get": {
          "tags": [
            "ToDo"
          ],
          "operationId": "ApiToDoGet",
          "consumes": [],
          "produces": [
            "text/plain",
            "application/json",
            "text/json"
          ],
          "responses": {
            "200": {
              "description": "Success",
              "schema": {
                "type": "array",
                "items": {
                  "$ref": "#/definitions/ToDoItem"
                }
              }
            }
          },
          "post": {
            "tags": [
            "ToDo"
          ],
          "operationId": "ApiToDoPost",
          "consumes": [
            "application/json-patch+json",
            "application/json",
            "text/json",
            "application/*+json"
          ],
          "produces": [],
          "parameters": [
            {
              "name": "item",
              "in": "body",
              "required": false,
              "schema": {
                "$ref": "#/definitions/ToDoItem"
              }
            }
          ],
          "responses": {
            "200": {
              "description": "Success"
            }
          },
          "/api/ToDo/{id}": {
            "get": {
              "tags": [
                "ToDo"
              ],
              "operationId": "ApiToDoByIdGet",
              "consumes": [],
              "produces": [],
              "parameters": [
                {
                  "name": "id",
                  "in": "path",
                  "required": true,
                  "type": "integer",
                  "format": "int64"
                }
              ],
              "responses": {
                "200": {
                  "description": "Success"
                }
              },
              "put": {
                "tags": [
                  "ToDo"
                ],
                "operationId": "ApiToDoByIdPut",
                "consumes": [
                  "application/json-patch+json",
                  "application/json",
                  "text/json",
                  "application/*+json"
                ],
                "produces": [],
                "parameters": [
                  {
                    "name": "id",
                    "in": "path",
                    "required": true,
                    "type": "integer",
                    "format": "int64"
                  },
                  {
                    "name": "item",
                    "in": "body",
                    "required": false,
                    "schema": {
                      "$ref": "#/definitions/ToDoItem"
                    }
                  }
                ],
                "responses": {
                  "200": {
                    "description": "Success"
                  },
                  "delete": {
                    "tags": [
                      "ToDo"
                    ],
                    "operationId": "ApiToDoByIdDelete",
                    "consumes": [],
                    "produces": [],
                    "parameters": [
                      {
                        "name": "id",
                        "in": "path",
                        "required": true,
                        "type": "integer",
                        "format": "int64"
                      }
                    ],
                    "responses": {
                      "200": {
                        "description": "Success"
                      }
                    },
                    "/api/Values": {
                      "get": {
                        "tags": [
                          "Values"
                        ],
                        "operationId": "ApiValuesGet",
                        "consumes": [],
                        "produces": [
                          "text/plain",
                          "application/json",
                          "text/json"
                        ],
                        "responses": {
                          "200": {
                            "description": "Success",
                            "schema": {
                              "type": "array",
                              "items": {
                                "type": "string"
                              }
                            }
                          }
                        },
                        "post": {
                          "tags": [
                            "Values"
                          ],
                          "operationId": "ApiValuesPost",
                          "consumes": [
                            "application/json-patch+json",
                            "application/json",
                            "text/json",
                            "application/*+json"
                          ],
                          "produces": [],
                          "parameters": [
                            {
                              "name": "value",
                              "in": "body",
                              "required": false,
                              "schema": {
                                "type": "string"
                              }
                            }
                          ],
                          "responses": {
                            "200": {
                              "description": "Success"
                            }
                          },
                          "/api/Values/{id}": {
                            "get": {
                              "tags": [
                                "Values"
                              ],
                              "operationId": "ApiValuesByIdGet",
                              "consumes": [],
                              "produces": [
                                "text/plain",
                                "application/json",
                                "text/json"
                              ],
                              "parameters": [
                                {
                                  "name": "id",
                                  "in": "path",
                                  "required": true,
                                  "type": "integer",
                                  "format": "int32"
                                }
                              ],
                              "responses": {
                                "200": {
                                  "description": "Success",
                                  "schema": {
                                    "type": "string"
                                  }
                                }
                              },
                              "put": {
                                "tags": [
                                  "Values"
                                ],
                                "operationId": "ApiValuesByIdPut",
                                "consumes": [
                                  "application/json-patch+json",
                                  "application/json",
                                  "text/json",
                                  "application/*+json"
                                ],
                                "produces": [],
                                "parameters": [
                                  {
                                    "name": "id",
                                    "in": "path",
                                    "required": true,
                                    "type": "integer",
                                    "format": "int32"
                                  },
                                  {
                                    "name": "value",
                                    "in": "body",
                                    "required": false,
                                    "schema": {
                                      "type": "string"
                                    }
                                  }
                                ],
                                "responses": {
                                  "200": {
                                    "description": "Success"
                                  },
                                  "delete": {
                                    "tags": [
                                      "Values"
                                    ],
                                    "operationId": "ApiValuesByIdDelete",
                                    "consumes": [],
                                    "produces": [],
                                    "parameters": [
                                      {
                                        "name": "id",
                                        "in": "path",
                                        "required": true,
                                        "type": "integer",
                                        "format": "int32"
                                      }
                                    ],
                                    "responses": {
                                      "200": {
                                        "description": "Success"
                                      }
                                    }
                                  }
                                }
                              },
                              "definitions": {
                                "ToDoItem": {
                                  "type": "object",
                                  "properties": {
                                    "id": {
                                      "format": "int64",
                                      "type": "integer"
                                    },
                                    "type": {
                                      "type": "string"
                                    },
                                    "isComplete": {
                                      "type": "boolean"
                                    }
                                  }
                                }
                              },
                              "securityDefinitions": {}
                            }
                          }
                        }
                      }
                    }
                  }
                }
              }
            }
          }
        }
      }
    }
  }
}
```

7. If you enter <http://localhost:5000/swagger/> you will be getting the screen:



8. (Optional) You can customize the UI. Change the ConfigureServices() method in the Startup.cs to the following Swagger generator:

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddDbContext<TodoContext>(opt =>
        opt.UseInMemoryDatabase("TodoList"));

    services.AddMvc();
    services.AddSwaggerGen(c =>
    {
        c.SwaggerDoc("v1", new Info
        {
            Title = "My API",
            Version = "v1",
            Description = "A simple example ASP.NET Core
                        Web API",

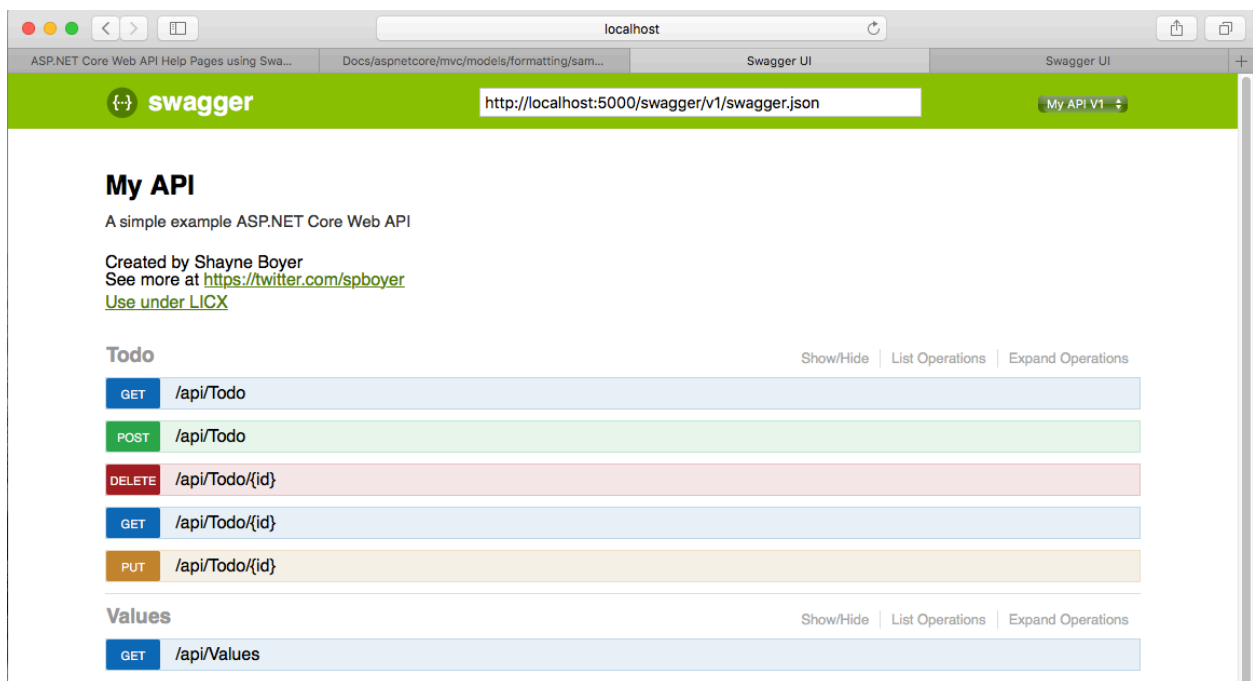
```

```

TermsOfService = "None",
Contact = new Contact { Name = "Shayne Boyer",
    Email = "",
    Url = "https://twitter.com/spboyer" },
License = new License { Name =
    "Use under LICX",
    Url = "https://example.com/license" },
//c.IncludeXmlComments(xmlPath)
});
});
}

```

9. I commented out the `c.IncludeXmlComments(xmlPth)` to build the project. Then I can test the project:



Exercise:

You may follow the direction to complete the XML Comments. Hint: Right click the project `ToDoApi` instead of the Solution to enable the XML comment.

1/11/2018

Project 3: Complete Project3 starting Friday, January 11, 2019 until January 14, 2019.

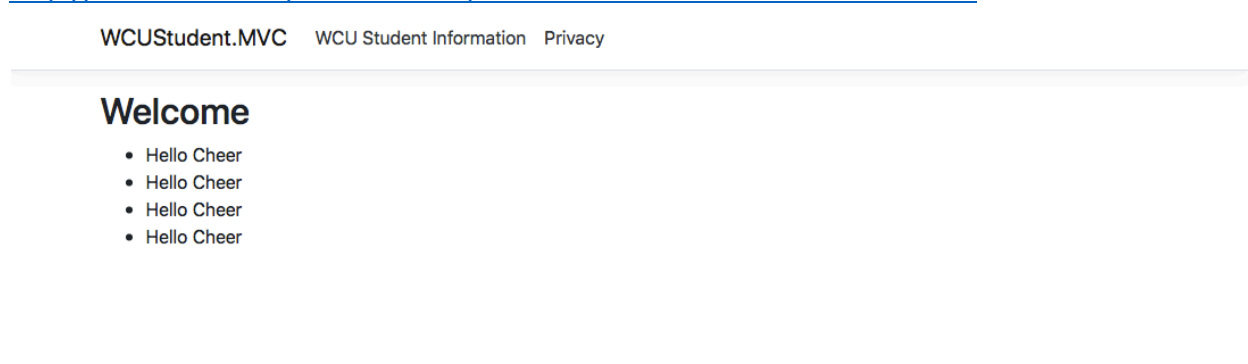
1/14 – 1/15 Putting it altogether!

We will begin to build the App together until the stage I can let you go free!

ASP.NET Core Web App – WCU Student Information

1. Create an ASP.NET Core Web App WCUSTudent.MVC (yes, with a dot MVC as part of the name of the solution).
2. Add a controller and modify the controller similarly to what the tutorial states. Test it with name and numTimes specified as

<http://localhost:5001/WCUSTudent/Welcome?name=Cheer&numTimes=4>



3. Add a model: Student.cs including two fields name and major. This class should inherit from the DbContext class. You don't need to worry about Data Annotation.

```
using System;
namespace WCUSTudent.MVC.Models
{
    public class Student
    {
        public int Id { get; set; }
        public string name { get; set; }
        public string major { get; set; }
    }
}
```

4. Add a StudentContext.cs class in the models folder.
5. `using System;`
`using Microsoft.EntityFrameworkCore;`

```
namespace WCUSTudent.MVC.Models
{
    public class WCUSTudentContext : DbContext
    {
        public WCUSTudentContext(DbContextOptions<WCUSTudentContext> options)
            : base(options)
        {
        }

        public DbSet<WCUSTudent.MVC.Models.Student> students { get; set; }
    }
}
```

6. Add required NuGet packages: `Microsoft.EntityFrameworkCore.Sqlite` and `Microsoft.VisualStudio.Web.CodeGeneration.Design`.
7. Register the database in the Startup.cs. (1) Add the two using directives:

```
using WCUSStudent.MVC.Models;
using Microsoft.EntityFrameworkCore;
```

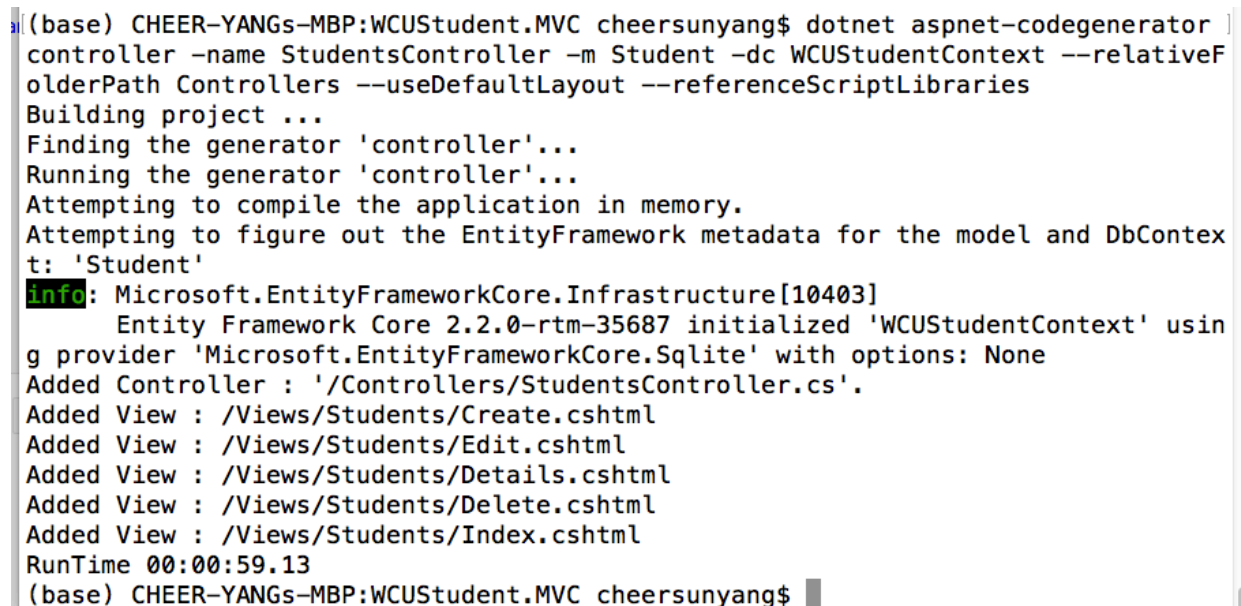
Register the database context with the [dependency injection](#) container in `Startup.ConfigureServices` before

```
services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version_2_2);
```

```
services.AddDbContext<WCUSStudentContext>(options =>
    options.UseSqlite(Configuration.GetConnectionString("StudentInfoContext")));
```

8. In this project, the Student model is scaffolded. You need to do a final check before you move on, mainly the two files in the Models folder: `Student.cs`, and the `WCUSStudentContext.cs`. Make sure that there are no wiggly lines inside the `StartUp.cs`. You can build the project and make sure that there are no errors. Check the `appsetting.json` file to see if you have the same name for the ConnectionStrings as the one you registered in the `StartUp.cs`. You also need to make sure you had matching curly braces inside the `appsetting.json`.
9. Close the project after you build it correctly.
10. Open a Terminal window and enter the `dotnet` command. Notice that the `-m` specifies the `Student.cs` in the Models folder, the `-dc` specifies the data context `WCUSStudentContext`. The name of the generated controller is `StudentsController`.

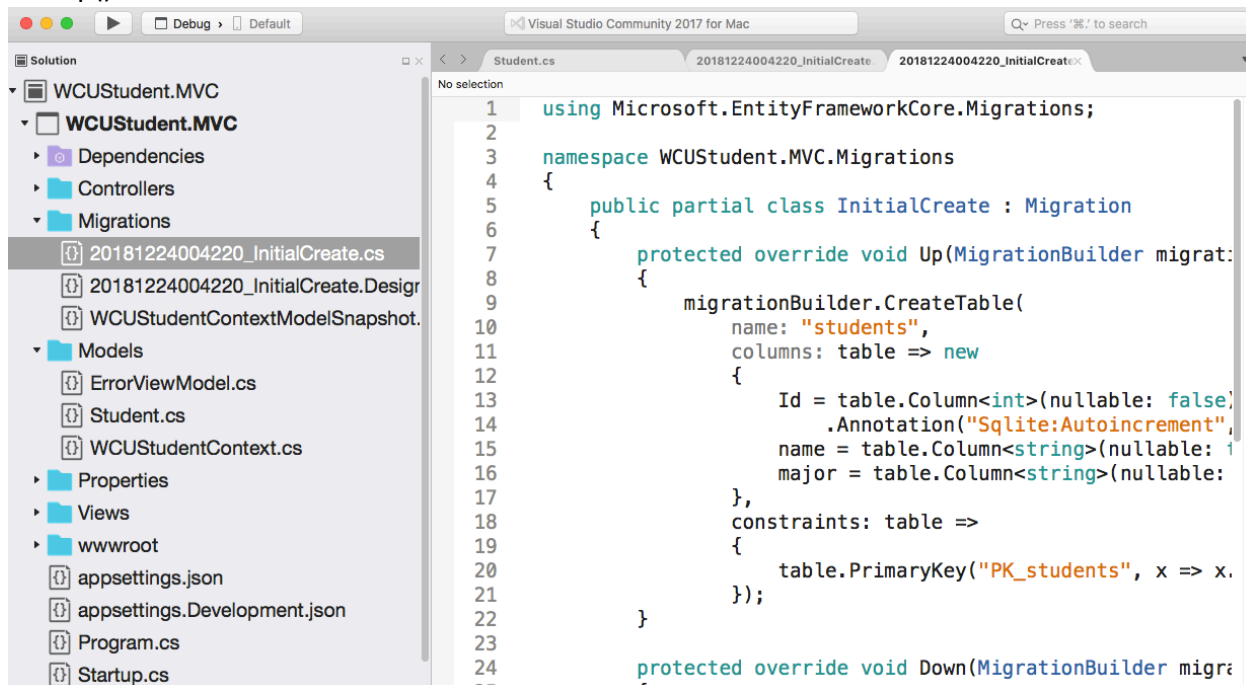
```
>dotnet aspnet-codegenerator controller -name StudentsController -m Student -dc
WCUSStudentContext --relativeFolderPath Controllers --useDefaultLayout --
referenceScriptLibraries
```



```
((base) CHEER-YANGs-MBP:WCUSStudent.MVC cheersunyang$ dotnet aspnet-codegenerator
controller -name StudentsController -m Student -dc WCUSStudentContext --relativeF
olderPath Controllers --useDefaultLayout --referenceScriptLibraries
Building project ...
Finding the generator 'controller'...
Running the generator 'controller'...
Attempting to compile the application in memory.
Attempting to figure out the EntityFramework metadata for the model and DbContex
t: 'Student'
Info: Microsoft.EntityFrameworkCore.Infrastructure[10403]
      Entity Framework Core 2.2.0-rtm-35687 initialized 'WCUSStudentContext' usin
g provider 'Microsoft.EntityFrameworkCore.Sqlite' with options: None
Added Controller : '/Controllers/StudentsController.cs'.
Added View : /Views/Students/Create.cshtml
Added View : /Views/Students/Edit.cshtml
Added View : /Views/Students/Details.cshtml
Added View : /Views/Students/Delete.cshtml
Added View : /Views/Students/Index.cshtml
RunTime 00:00:59.13
(base) CHEER-YANGs-MBP:WCUSStudent.MVC cheersunyang$
```

11. Once the controller is generated as “`StudentController`”, we can begin the initial migration process.

12. Generate the code to run the Up() method in the initial migration file. Notice that we haven't stored any data into the database, but generating code for generating the database, and CRUD code. This is known as "Code First" provided by the Entity-Framework Core. You can browse the Initial migration file with Visual Studio and check the Up() method. Check Line 7 of the screen shot below:



13. Make sure you find the Info below to indicate that the Database Table is generated with the structure specified in the Student.cs.

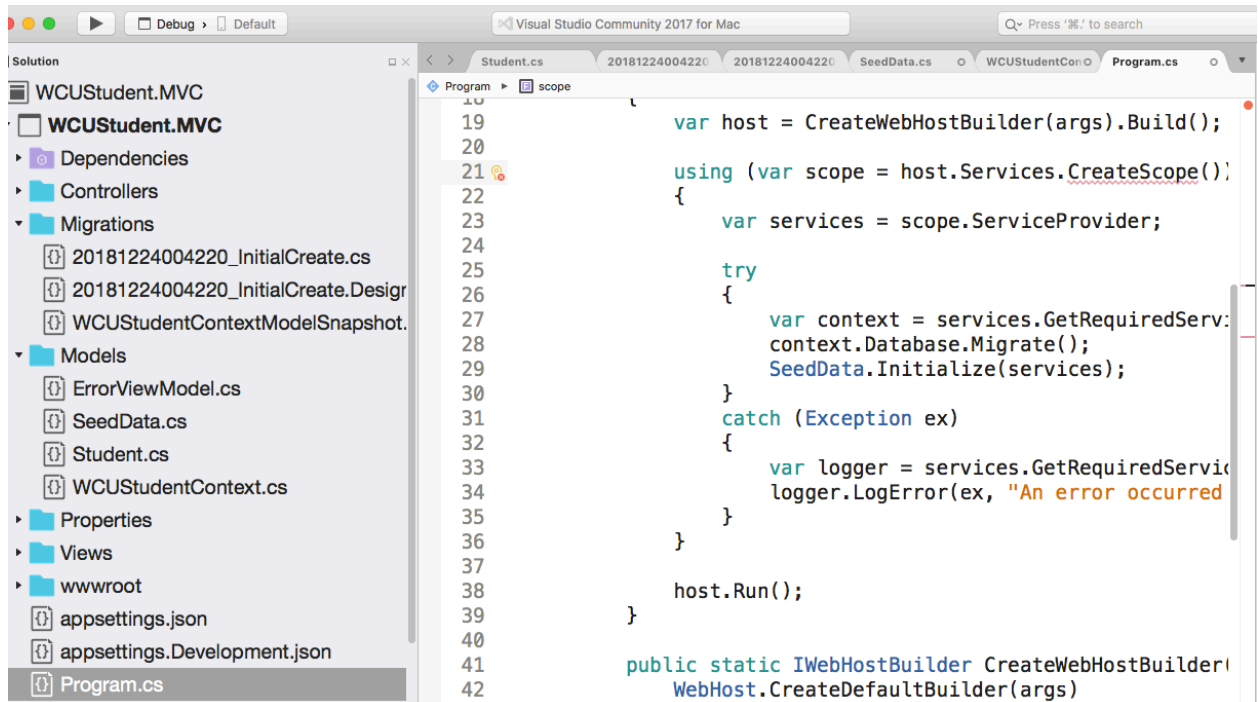
```

CREATE TABLE "students" (
    "Id" INTEGER NOT NULL CONSTRAINT "PK_students" PRIMARY KEY AUTOINCREMENT,
    "name" TEXT NULL,
    "major" TEXT NULL
);
  
```

14. Build the app and enter the URL: <https://localhost:5001/students>. If you see an exception message, you might not have done the migration correctly. Try the creation link.
15. I've created two student records for the purpose of testing. We needed to delete all of them before running SeedData.cs.
16. We need to write a SeedData.cs C# code in the Models folder. You may run into problems here. When you see wiggly lines under any word, you may right click on it and choose quick fix. You may need to add the two using directives:

```

using WCUSStudent.MVC.Models;
using Microsoft.Extensions.DependencyInjection;
  
```



17. Notice that I had a wiggly line at line 21 before I add the using directive for dependency injection. I was using “quick fix” to resolve this issue.
18. The second part is to add the WCUSStudent model to a Web API with the four operations CRUD. I am leaving that to you.

1/16 Exam starts at 1:00 pm (If you are late, I will wait until 2:00 pm)

1/17 Project (No class)

1/18 Project 4 Due (No Class)

Project 4 Due by 11:59 pm, January 18, 2019. Late submission is only accepted until Saturday night (January 19, 2019), with late penalty of -1/15 per day.