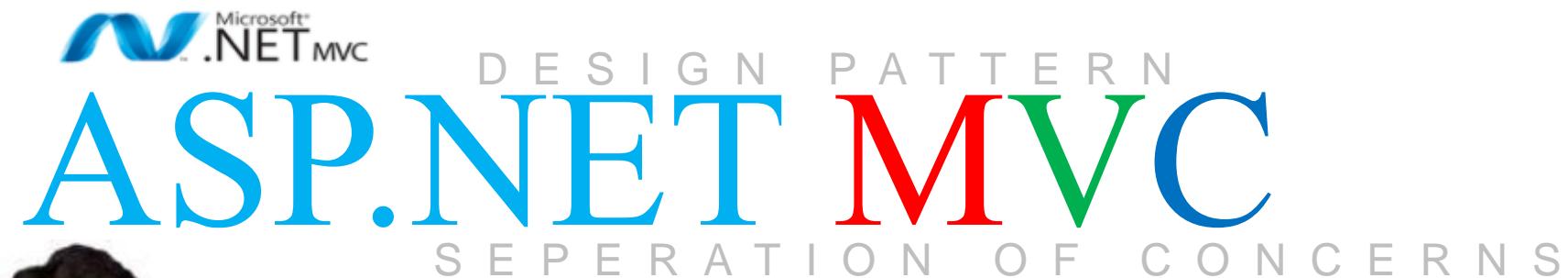




ASP.NET Web API 2

GROUND UP SERIES



Syed Awase Khirni

RESEARCHER | ENTREPRENEUR | TECHNOLOGY COACH

@sak008 | sak@sycliq.com/sak@territorialprescience.com | +91. 9035433124

Syed Awase earned his PhD from University of Zurich in GIS, supported by EU V Framework Scholarship from SPIRIT Project (www.geo-spirit.org). He currently provides consulting services through his startup www.territorialprescience.com and www.sycliq.com. He empowers the ecosystem by sharing his technical skills worldwide, since 2008.

Terms of Use

- You shall not circulate these slides without written permission from Territorial Prescience Research I Pvt ltd.
- If you use any material, graphics or code or notes from these slides, you shall seek written permission from TPRI and acknowledge the author Dr. Syed Awase Khirni
- If you have not received this material, post-training session, you shall destroy it immediately and not use it for unauthorized usage of the material. If any of the material, that has been shared is further used for any unauthorized training by the recipient, he shall be liable to be prosecuted for the damages. Any supporting material that has been provided by the author, shall not be used directly or indirectly without permission.
- If this material, has been shared to any organization prior to the training and the organization does not award the contract to TPRI, it should not use the training material internally. If by any chance, the organization is using this training material without written permission, the organization is liable to pay for the damages to TPRI and is subjected to legal action, jurisdiction being Bangalore.
- Without unauthorized usage, TPRI has right to claim damages ranging from USD 50000 to USD 10,0000 dollars as damages.
- Any organization, which does not intend to go ahead with training or does not agree with the terms and conditions, should destroy the material from its network immediately. The burden of proof lies on the client, with whom this material has been shared.
- Recovery of the damages and legal fees will be born by the client organization/candidate/party, which has violated the terms and conditions.
- Only candidates who have attended the training session in person from Dr. Syed Awase Khirni, TPRI are entitled to hold this training material. They cannot further circulate it, or use it or morph it, or change it to provide trainings.
- TPRI reserves all the rights to this material and code plays and right to modify them as and when it deems fit.
- If you agree with the terms and conditions, please go ahead with using the training material. Else please close and destroy the slide and inform TPRI immediately

Slide Version Updates

Please read terms and conditions of use

Last Updated	Version	Release Date	Updated by	Code Plays Done @

Original Series

Program Agenda

DAY 1

DAY 2

DAY 3

DAY 4

DAY 5

DAY 6

DAY 7

DAY 8

Please read terms and conditions of use

Original Series

SECTION -O

INSTALLATION AND DEV ENVIRONMENT CHECK

Installing ASP.NET MVC

- We can have multiple versions of ASP.NET MVC applications running side by side in your server/computer.
- ASP.NET MVC 4
 - <http://www.asp.net/mvc/mvc3> for version 3
 - Does not come with bootstrap view scaffolds
 - Uses modernizr views
 - <http://www.asp.net/mvc/mvc4> for version 4
 - Comes with bootstrap view scaffolds
 - Uses razorview engine for rendering
 - Requires .Net Framework 4 and above.

- ASP .NET MVC 5
 - Comes preinstalled in visual studio 2015.
 - Visual studio 2015 support for c#6.0
 - Requires .Net Framework 4.5 and above

- ASP .NET MVC 6
 - Comes preinstalled in visual studio 2017
 - Visual studio 2017 support for c# 7.0

Please read terms and conditions of use

Original Series

Alternative View Styles

Please read terms and conditions of use

Original Series

Databases

- MSSQL Server
- MongoDB

Please read terms and conditions of use

Original Series

SECTION -I

ARCHITECTURES

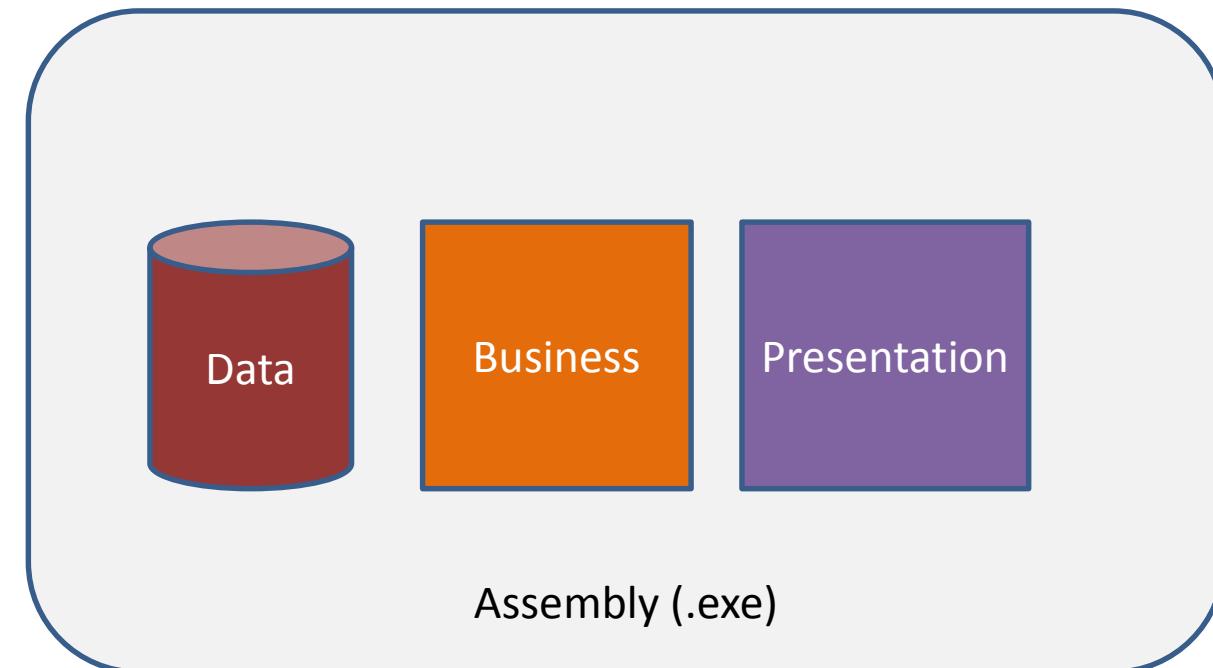
Architectures

- Three major concerns
 - Data / Storage (Files, Databases, External Storage)
 - Business Logic (Logic revolving adding/updating/deleting business objects)
 - Presentation
- One Tier
- Two Tier
- Three Tier
- n-Tier

One Tier Architecture

Please read terms and conditions of use

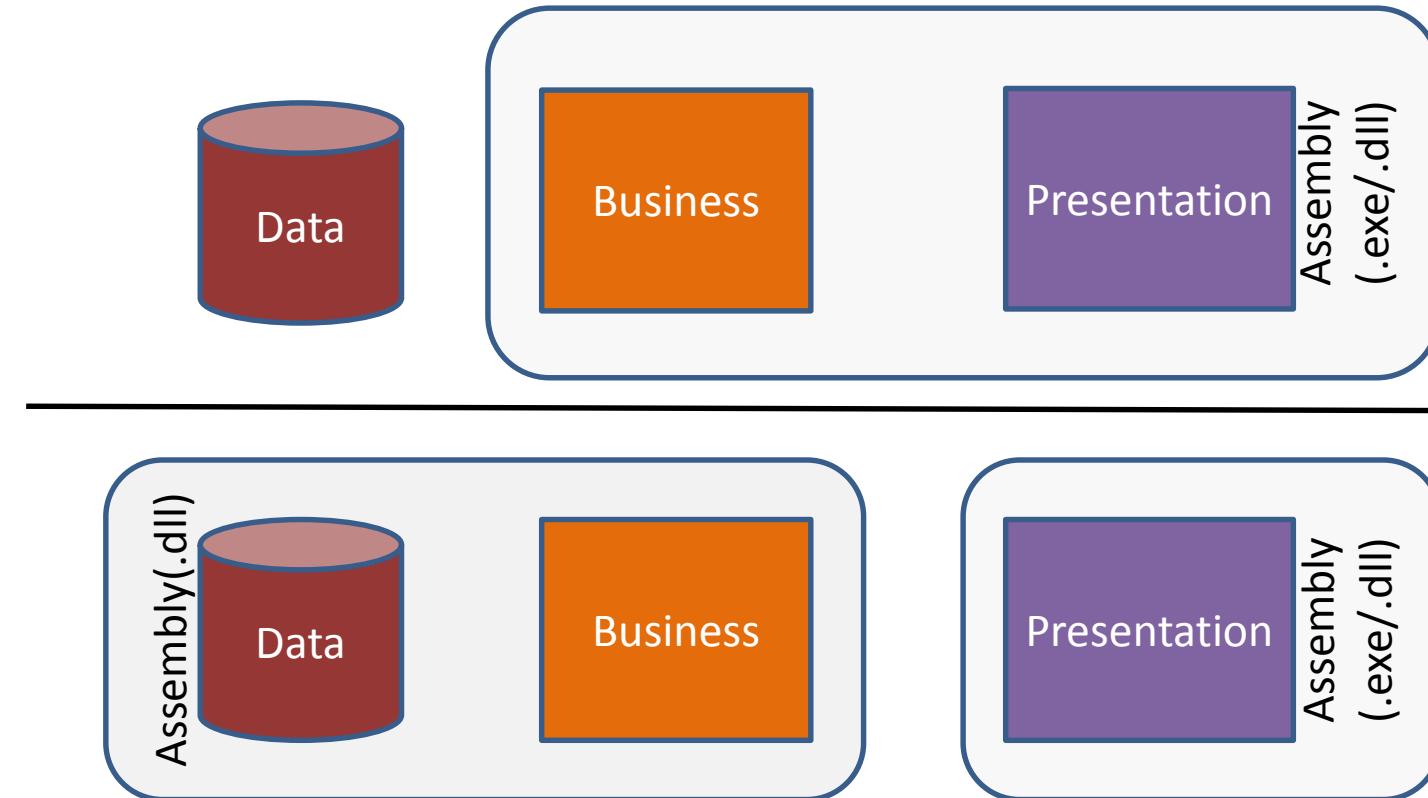
Original Series



Two Tier Architecture

Please read terms and conditions of use

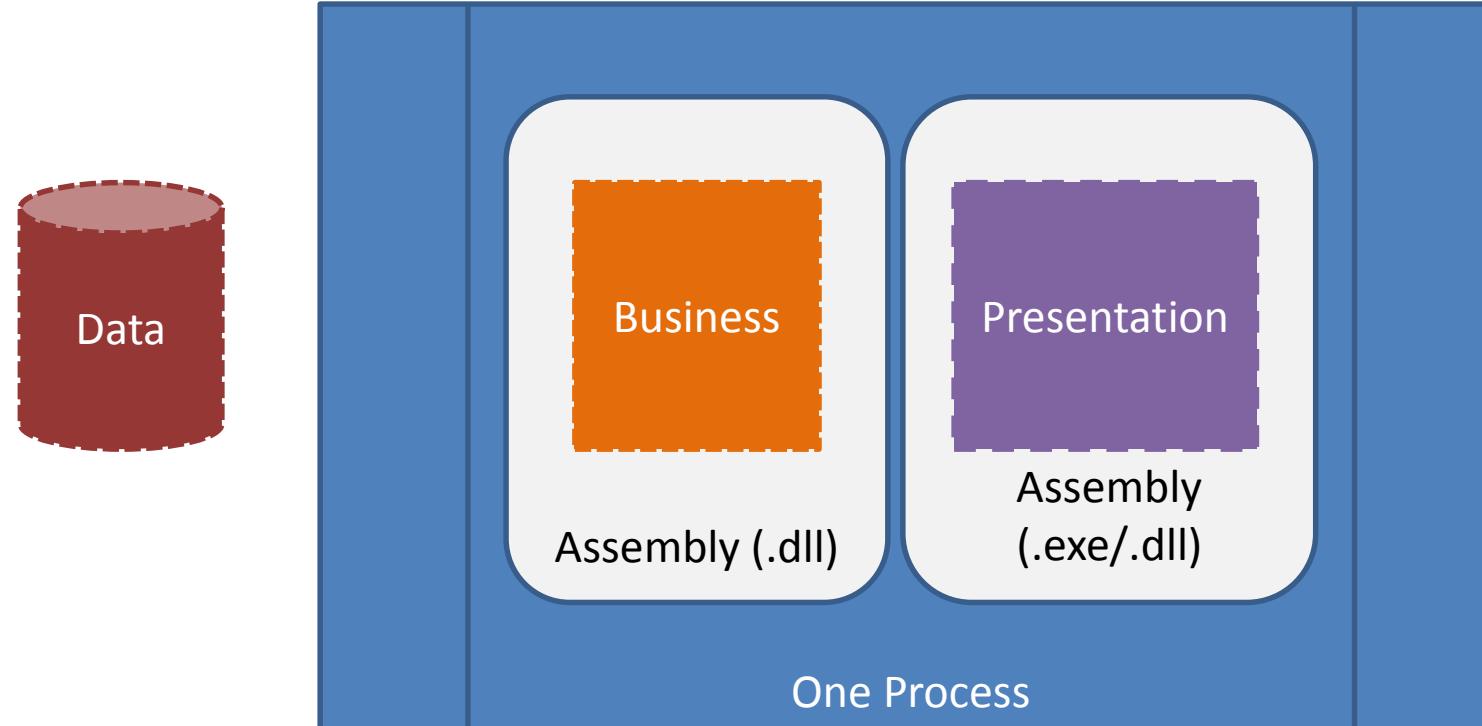
Original Series



Three Tier Architecture (Logical)

Please read terms and conditions of use

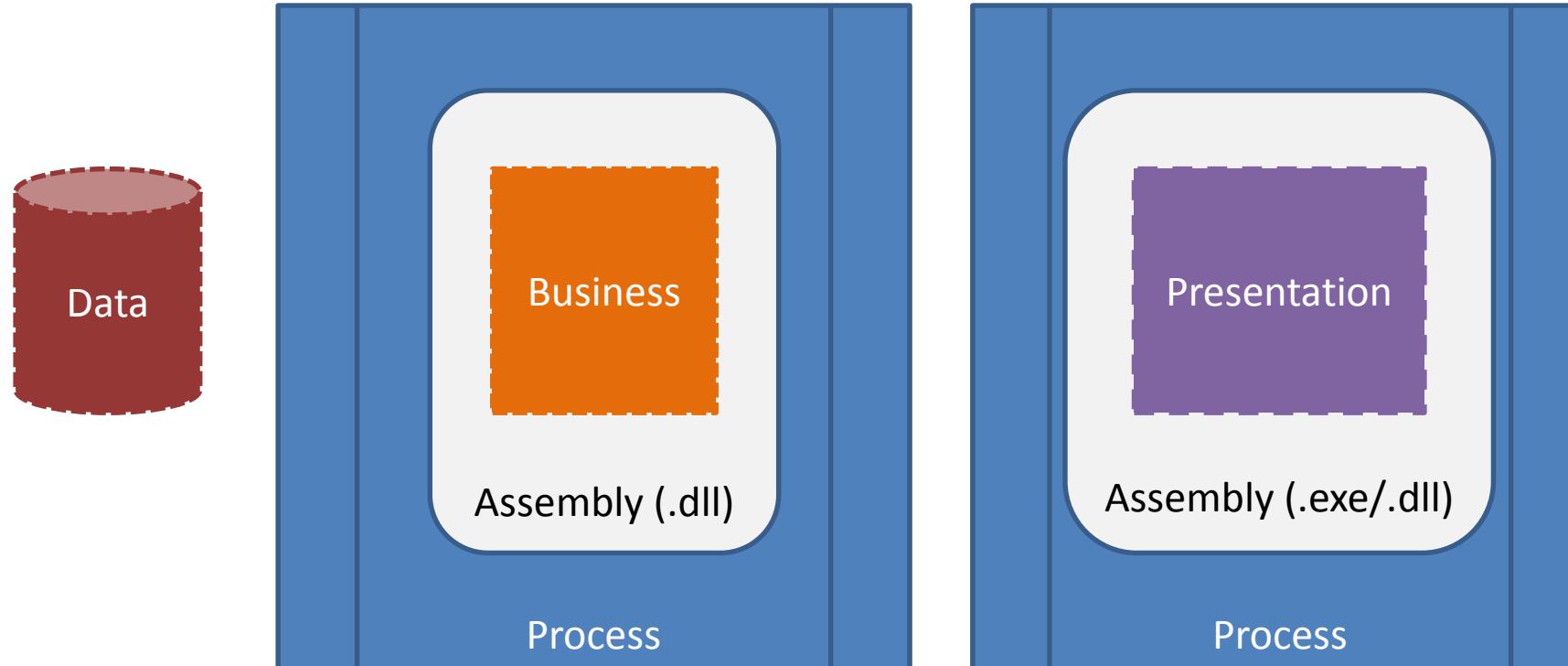
Original Series



Three Tier Architecture (Physical)

Please read terms and conditions of use

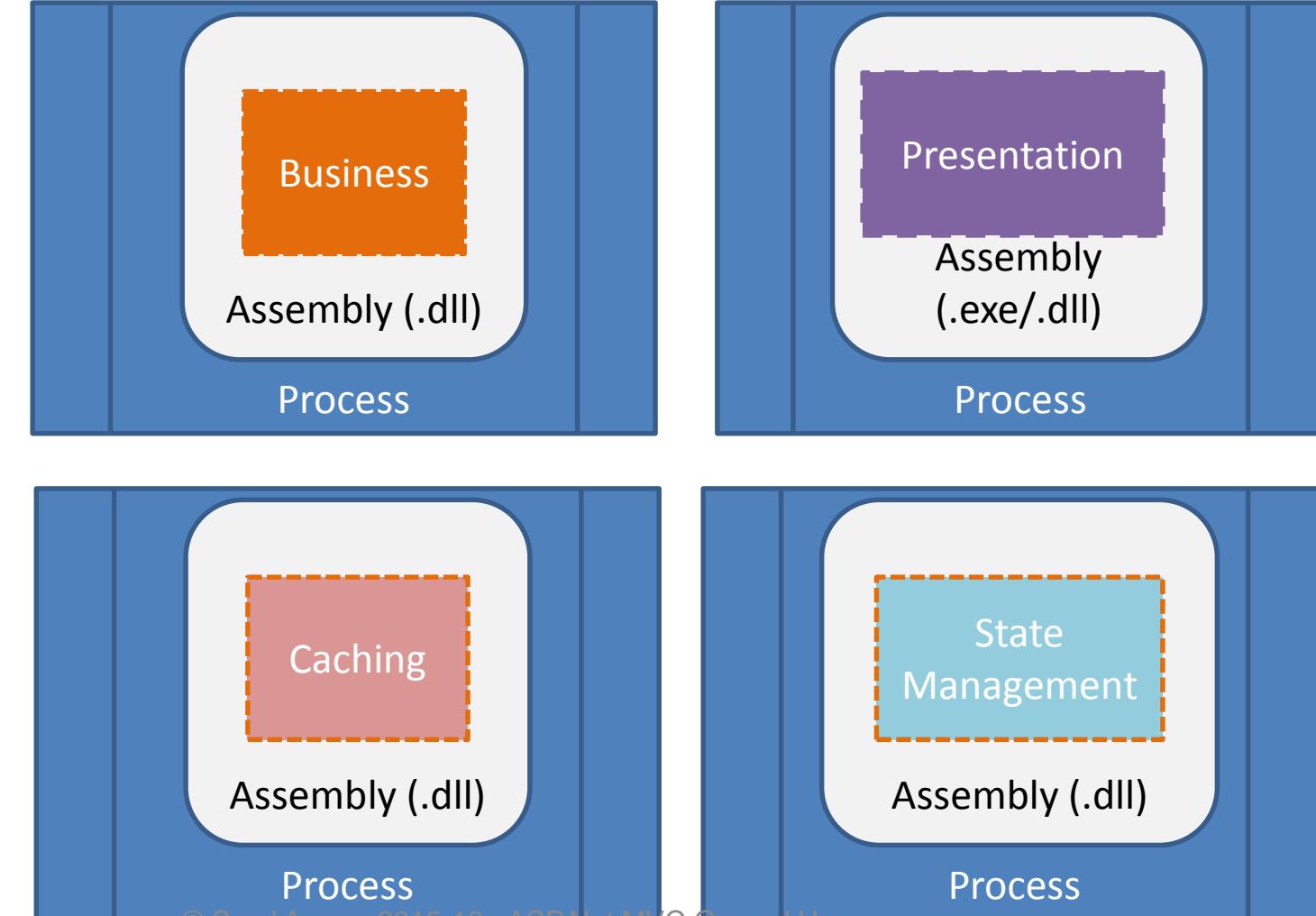
Original Series



n-Tier Architecture (Physical)

Please read terms and conditions of use

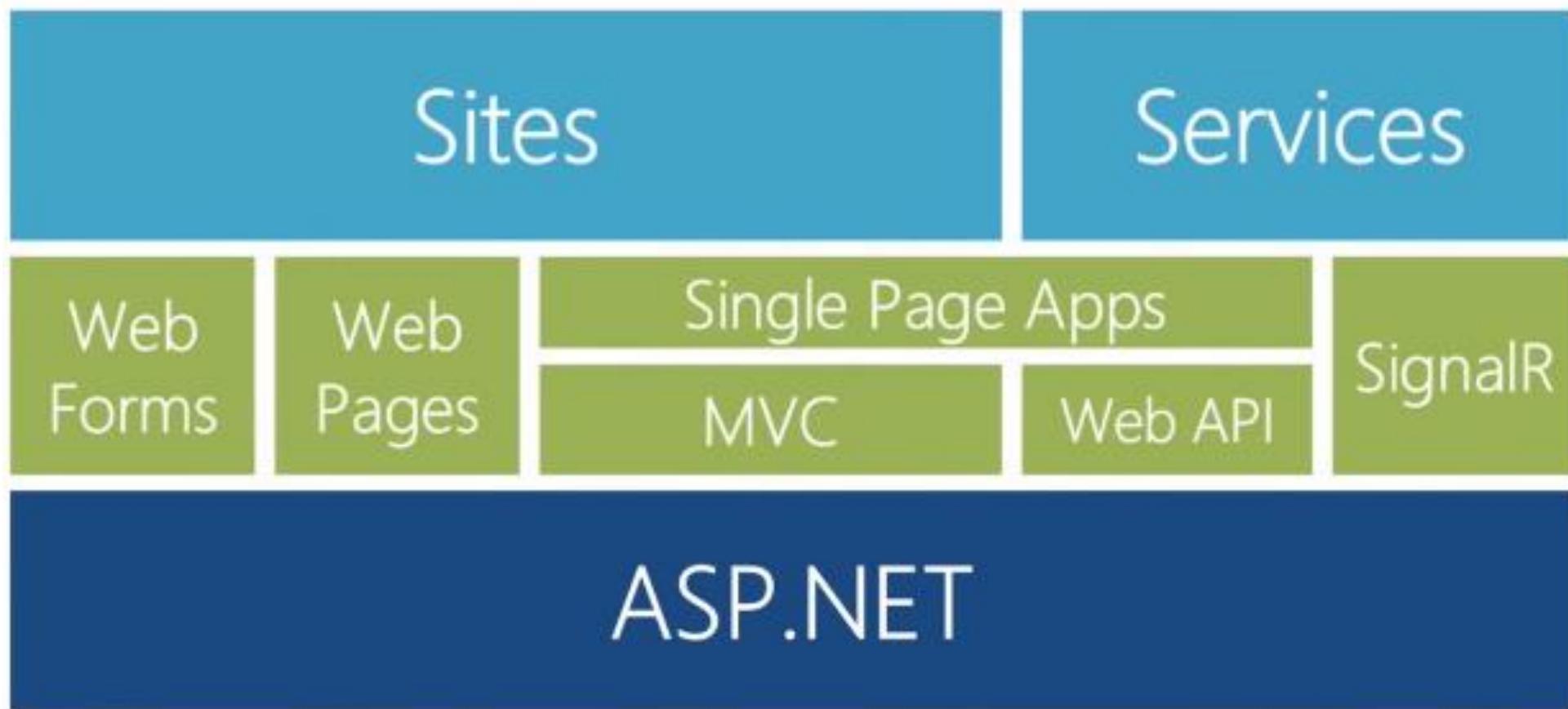
Original Series



ASP.NET OVERVIEW

Please read terms and conditions of use

Original Series

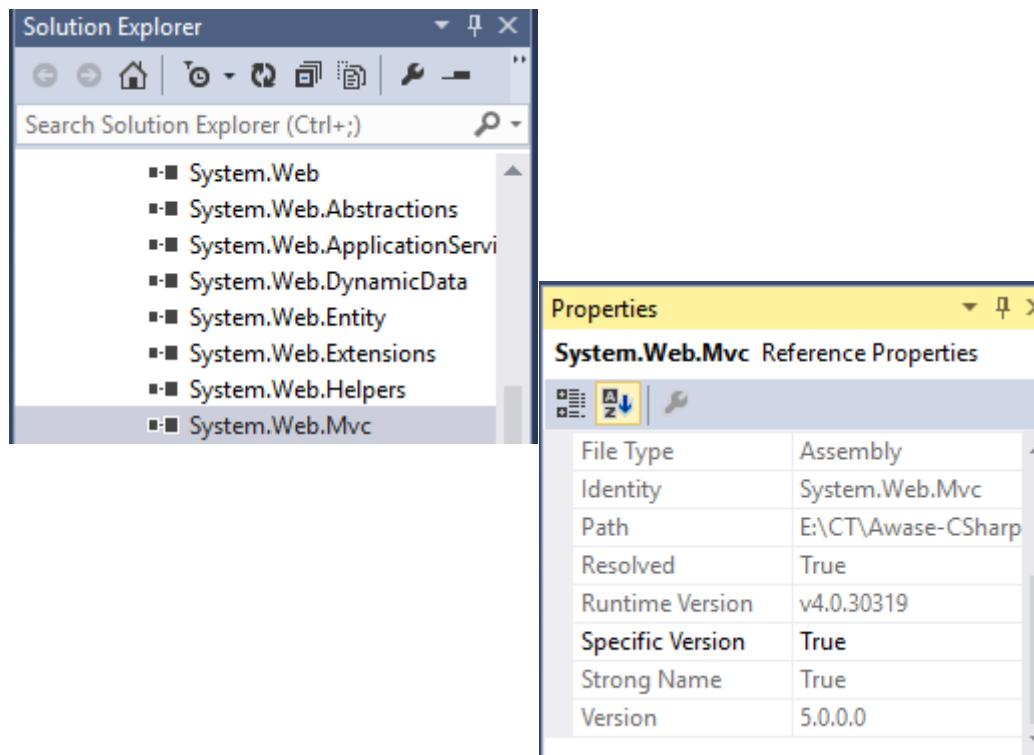


Installing ASP.NET MVC

Please read terms and conditions of use

Original Series

- ASP.NET MVC
 - <http://www.asp.net/mvc>



What MVC version

- 2 ways to identify
 - At design time – Go to solution explorer -> expand “References” folder. Right click on “**System.Web.Mvc**” assembly and select properties.
 - At run-time using the following code
 - `Typeof(Controller).Assembly.GetName().Version.ToString();`

Revisiting ASP.NET Web Forms

- First released in ASP .NET 1.0
- Replaced classic ASP (Active Server Pages)
 - Strongly typed code replace script
 - Abstract away the web
 - Click events replaced “POST” operations
- Original design from the late 90s
 - Web standards have strengthened
 - Client-side programming on the rise
- Web forms compete against other MVC frameworks
 - STURTS
 - RUBY ON RAILS (ROR)
 - DJANGO - PYTHON
 - ANGULARJS (VERY RECENTLY)
- Productive way to build web applications
- Control and event-based programming model
- Controls that abstract HTML, JS and CSS
- Rich UI Controls- datagrid, charts, AJAX
- Browser differences are handled for you

What's wrong with **ASP.NET WEB FORM**

- ViewState
- Page Life Cycle
- Limited Control over the rendered HTML
- Lack of **Separation of Concerns (SoC)**
- Untestable
- In a webform URL's are mapped to the **Physical Files**
 - `http://localhost/WebformsExampleOne/Default.aspx`

- Web Forms Supports rich server side controls and drag and drop option for designing View pages.
- Uses Event driven programming language.
- Easy to get started and less learning effort required.
- Logic is embedded in code behind files and page is heavy.
- Rapid Application Development and Support for ViewState.

DISADVANTAGES ASP.NET WEB FORMS

- No Application Structure guidance
- No standards on separation of concerns.
- Very complex page life cycle
- Unit testing and end to end testing becomes very difficult to manage.
- View states makes pages heavy and may reduce performance.
- Less support for parallel development.
- Tightly coupled with the view and code behind

When to USE ASP.NET MVC

- **ASP.NET MVC** is NOT a replacement of ASP.NET web forms based applications
- The approach of application development must be decided based on the application requirements and features provided by ASP.NET MVC to suite them
- Application development with ASP.NET MVC is more complex as compared to web forms based applications as they lack readily available rich controls and less knowledge of the pattern in ASP.NET Web Developers
- Application maintainability will be higher with separation of application tasks

MVC DESIGN PATTERN

- First proposed in 1970's
- Gives us a clean interaction model for web based development.
- View does not use Controller to update Model. Controller handles the events from View to manage user's interaction and data (via interaction with Model)
- Controller can be combined with View. Logical Separation of Model from the View
- The Controller does not contain the rendering logic.
- Controller uses view asking it to render new data.

SECTION -XIII

MVC 2.0

MVC 2.0

Please read terms and conditions of use

Original Series

- UI Helpers with Scaffolding templates
- Model validation on Client and Server Side
- Strongly typed HTML helpers
- Enhanced Visual Studio tooling.

MVC 2.0

Please read terms and conditions of use

Original Series

- Customer side validation
- Template helpers
- Regions
- Non-concurrent controllers
- `Html.ValidationSummary` Helper Method
- Restricting Binary data with Model Binders
- DataAnnotations Attributes
- Validator Providers
- New `RequireHttpsAttribute` Action Filter
- Templated Helpers
- Model-level Error diagnostics
- Displaying Model Level Error.

SECTION -XIII

MVC 3.0

MVC 3.0

Please read terms and conditions of use

Original Series

- Released in 2011.
- Inclusion of Razor View Engine
- .NET 4 Data Annotations
- Robust model binding and validations
- Inclusion of Global Action Filters
- Jquery Support and JavaScript Validations
- JSON Binding
- NuGet Integration to resolve the software dependency on the fly.

MVC 3.0

- Razor view engine introduced
- HTML5 format support
- Support for Multiple View Engines
- JavaScript and Ajax Support
- Validation Enhancements.
- Templates for HTML5 and CSS3
- Improved model Validation
- Improved Controller
- Improved Dependency Injection using `IDependencyResolver` interface
- Partial Page output caching.
- Introduced a new feature called Bundling of resources.
- `ViewBag` dynamic property introduced.
- `ActionResult` Types feature introduced.

SECTION -XIII

MVC 4.0

MVC 4.0

Please read terms and conditions of use

Original Series

- ASP.NET Web API
- Improved Project Templates, Added new ones
- Inclusion of Mobile Projects using Jquery Mobile
- Various Display Modes
- Asynchronous Controllers
- Bundling and Minification.

MVC 4.0

Please read terms and conditions of use

Original Series

- ASP.NET WEB API 1.x
- Improved support for rendering various venture formats.
- Structured Application development best practices.
- Modernizer, JavaScript and Ajax Support. Look and feel improvements.
- Empty Project Template.
- Mobile Project Template
- Support for adding Controller
- Task Support for Async Controller
- Bundling and Minification Support
- Support for Oauth and OpenID.
- Support for Windows Azure SDK1.6

SECTION -XIV

MVC 5.0

MVC 5.0

Please read terms and conditions of use

Original Series

- Improved Scaffolding
- ASP.NET Identity Management
- One ASP.NET
- Support for Bootstrap
- Attribute Routing
- Filter Overrides

MVC 5.0

Please read terms and conditions of use

Original Series

- ASP.NET WEB API 2.x
- ASP.NET IDENTITY MANAGEMENT SUPPORT AND SOCIAL LOGIN SUPPORT.
- BOOTSTRAP 3.x Support
- CONFIRMATION FILTERS
- CHANNEL SUPERSEDES

- IMPROVED ROUTING MECHANISM
 - Attribute based routing.
- Authentication Filters and Filter Overrides.
- Filter overrides
- Internet Application Template and Intranet Application Template
- ASP.NET WEB API Template
- Mobile Project Template

SECTION -XIV

MVC 6.0

MVC 6.0

Please read terms and conditions of use

Original Series

- Common framework for MVC, Web API and Web Pages
- Smooth Transiting from Web Pages to MVC
- Built DI First
- Runs on IIS or self host
- Based on the new Request Pipeline in ASP.NET vNext
- Runs cloud optimized
- No build dependency
- Enhanced developer experience
- Open source
- Cross-platform support.

MVC 6.0

Please read terms and conditions of use

Original Series

- ASP.NET MVC AND WEB API converged into one.
- Reliance infusion is inbuilt and part of MVC.
- Extended support for bundling and minification using NuGet, including the .NET runtime
- New JSON based venture structure
- New cloud computing optimization system of MVC, WebAPI, SignalR and Entity Framework
- IMPROVED ROUTING PERFORMANCE.
- IMPROVED COMPILATIONS AND HOT RELOADING
- NEW ROSYLN ONGOING COMPILER
- vNext is Opensource supported by .NET Foundation
- vNext and Roslyn support for Mono to run on Mac and Linux
- Removed the dependency of System.web.dll from MVC6.
- Added a Start-up class that replaces the global.asax file.

MVC

- A framework for building
 - Scalable
 - Flexible
 - Extensible
 - Standard based web application
- Complex Applications can be easily managed.
- ASP.NET MVC is light weight as they do not maintain view state in client page.

- MVC works on Conventions over Configurations.
- MVC uses the power of ASP.net and .NET framework
- MVC uses separation of Concerns
- MVC framework is defined in “System.Web.Mvc.Assembly” namespace in .NET/ASP.NET MVC
- Support for HTML5 enabled Razor view Engine.

MVC

Please read terms and conditions of use

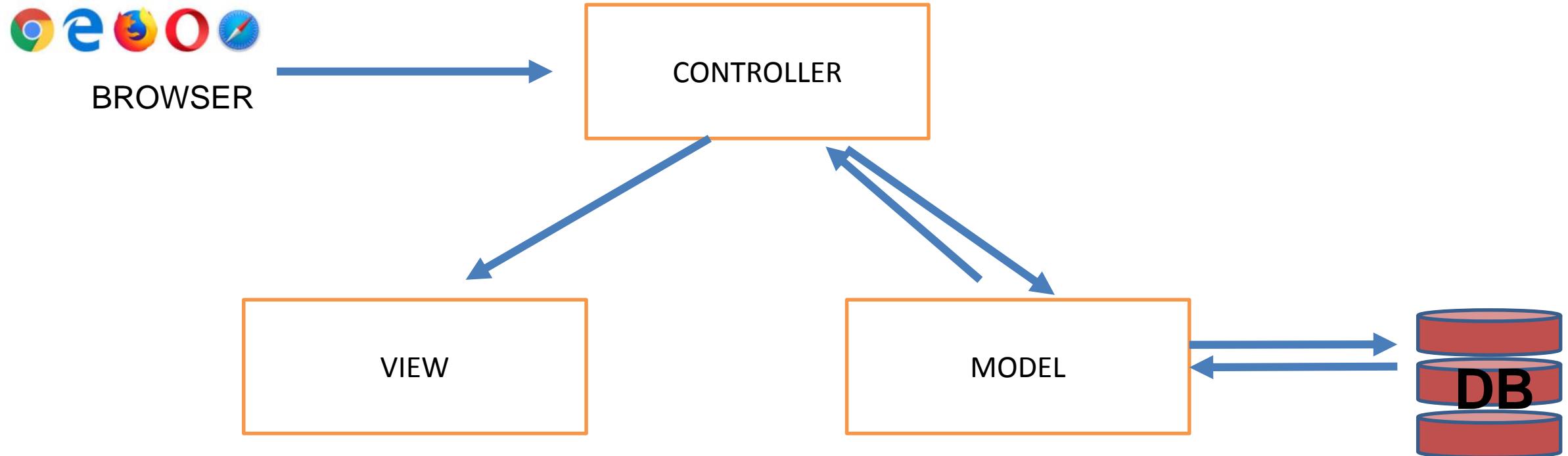
Original Series

- Search Engine Optimization (SEO) – Clean URL's and no extension methods used for locating the files.
- Rich Javascript support with unobtrusive javascript validation, Jquery validation and JSON binding.
- Requirements should be identified much in depth at design phase.

MVC flow

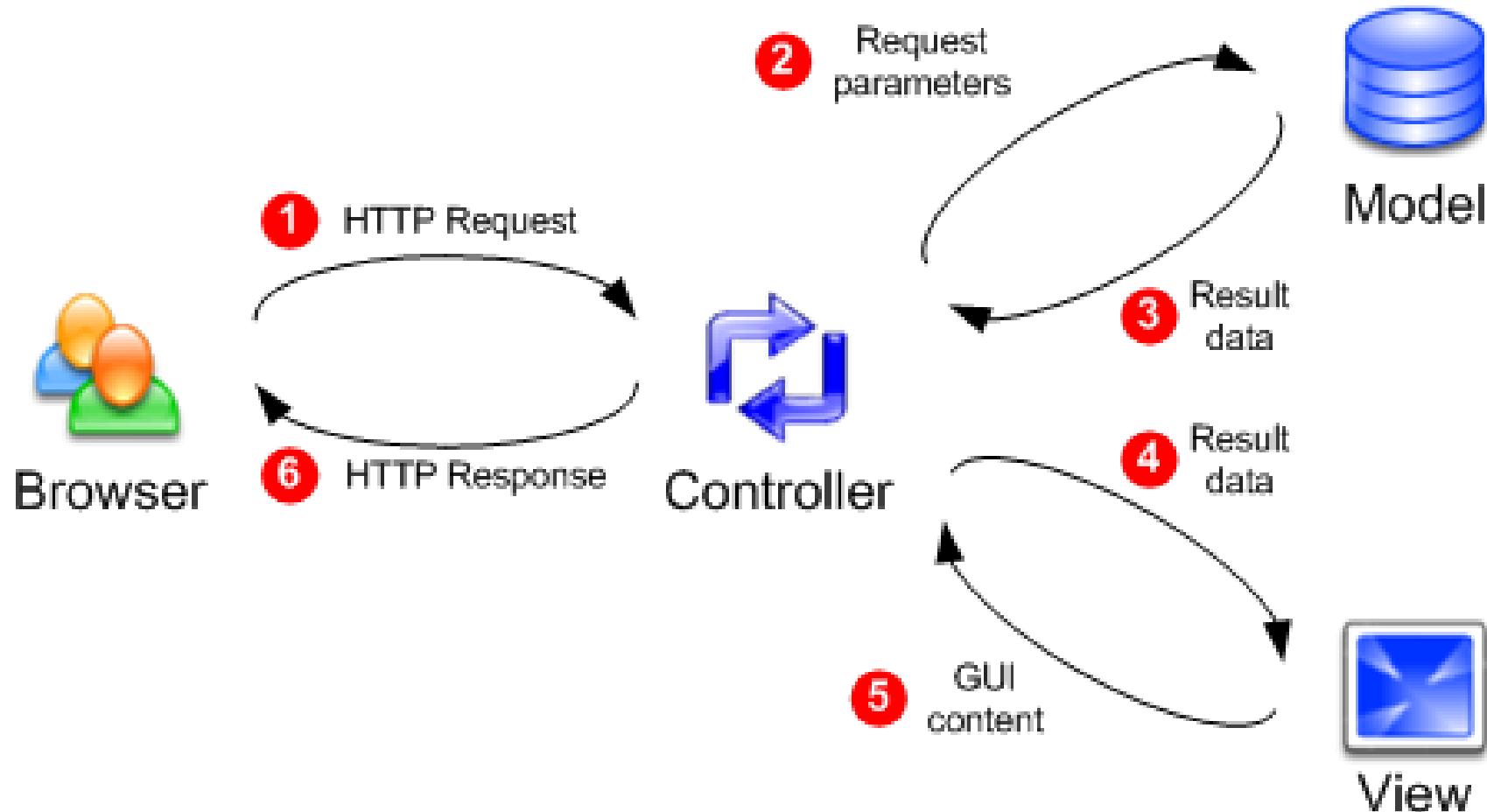
Please read terms and conditions of use

Original Series



MVC IN ACTION

Please read terms and conditions of use

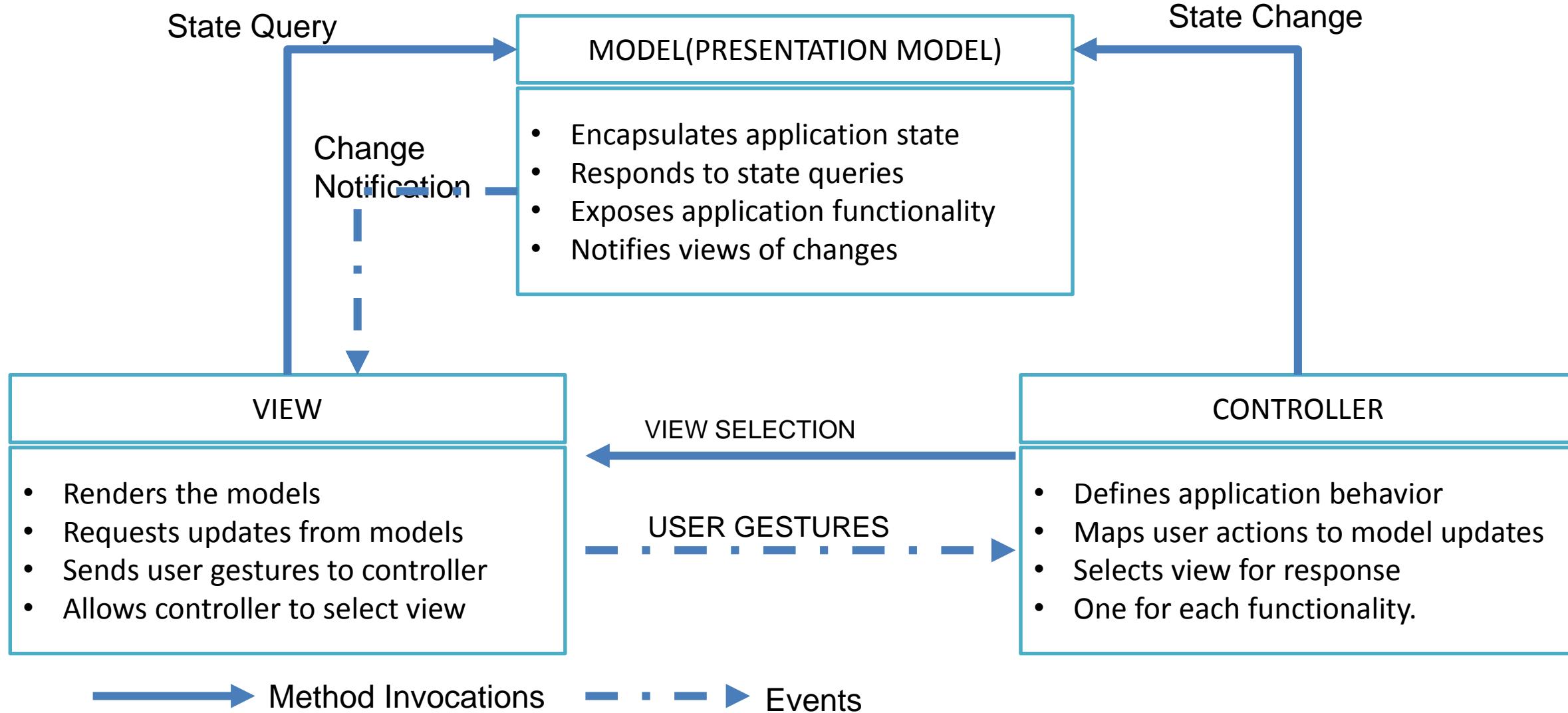


Original Series

MODEL CONTROLLER VIEW

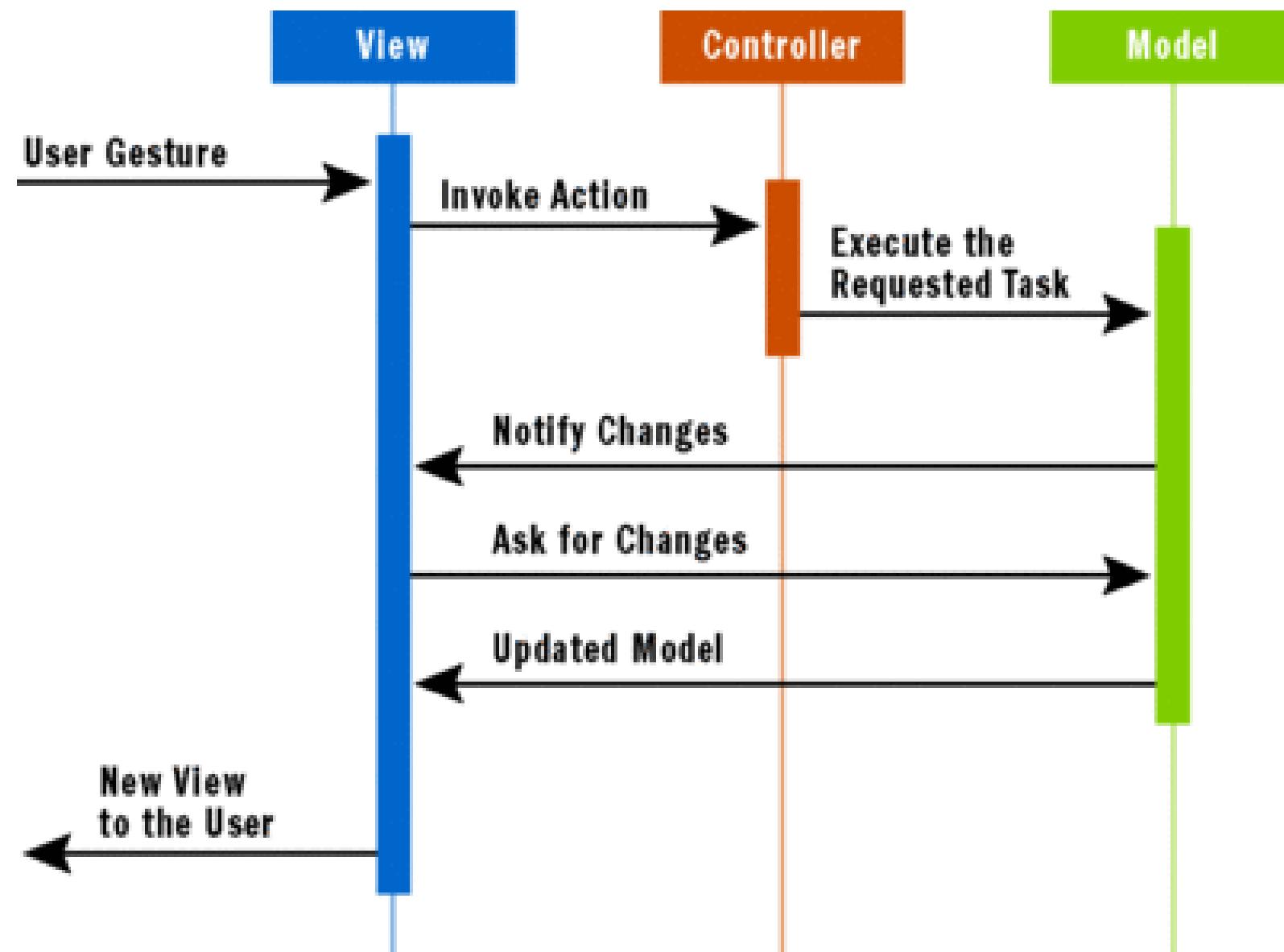
Please read terms and conditions of use

Original Series



Please read terms and conditions of use

Original Series

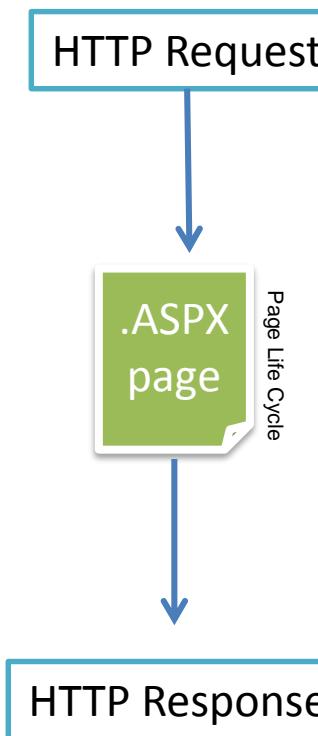


WEB FORMS vs MVC

Please read terms and conditions of use

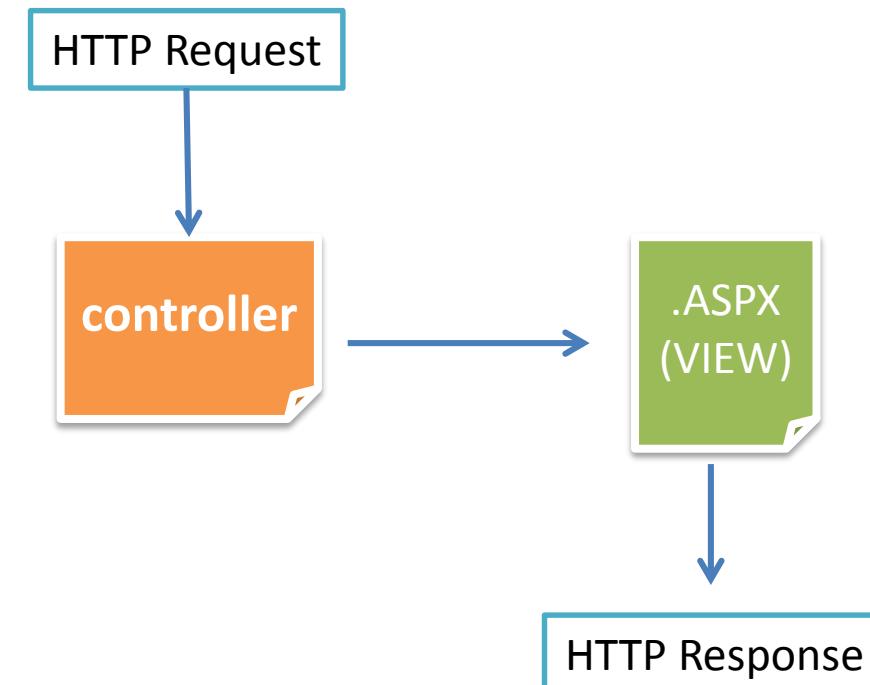
Original Series

ASP.NET WEB FORMS



url's are mapped to Physical files

ASP.NET MVC



url's are mapped to controller
action methods

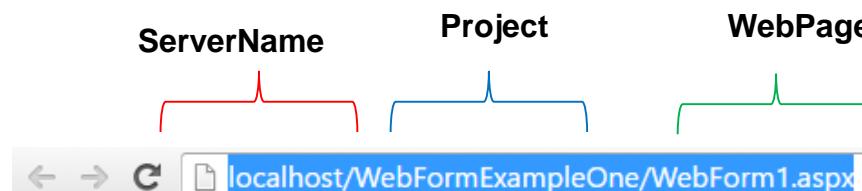
WEB FORMS vs MVC

Please read terms and conditions of use

Original Series

Web Forms

- In a **webForms** URL's are mapped to **Physical Files**

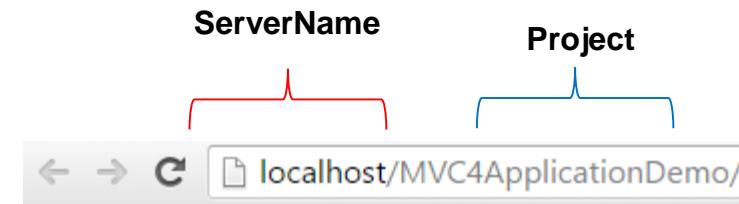


Hello from Webforms application

```
protected void Page_Load(object sender, EventArgs e)
{
    Response.Write("Hello from Webforms application");
}
```

MVC

- MVC URL's are mapped to **controller Action Methods**
- Functions in a controller are generally called as **Controller Action Methods**



Hello from MVC4 Application using razor Engine

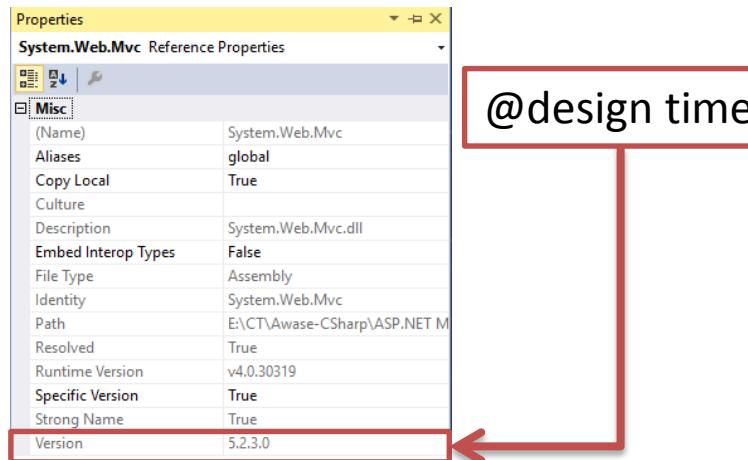
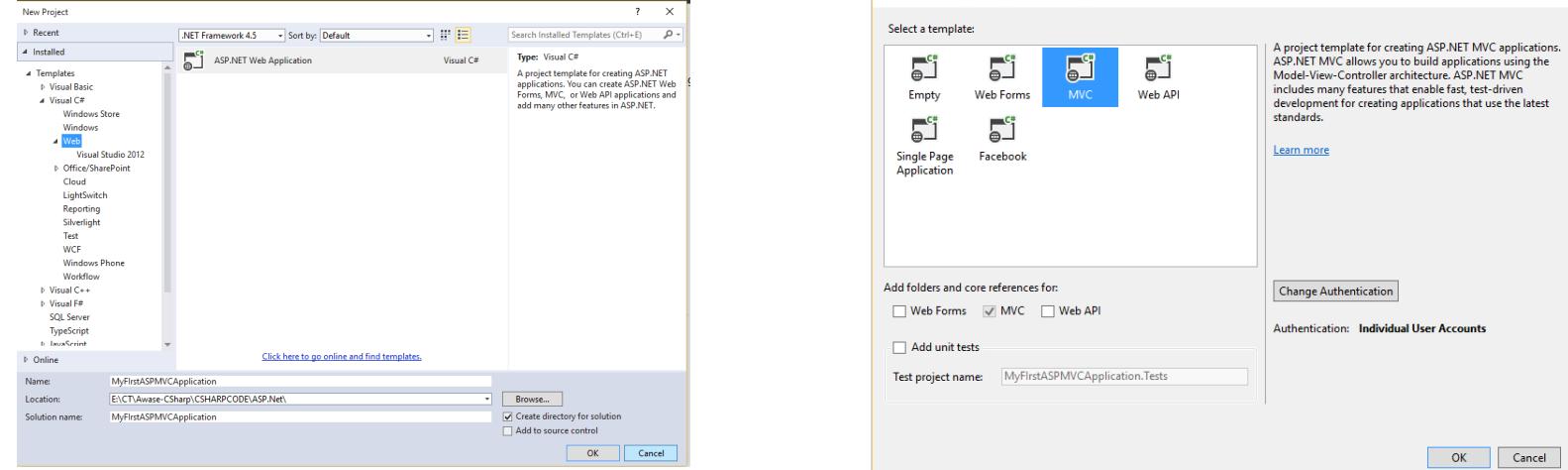
```
0 references
public String Index()
{
    //ViewBag.Message = "Modify this template to jump-start your ASP.NET MVC
    //application.";

    //return View();
    return "Hello from MVC4 Application using razor Engine";
}
```

MVC Application with VS

Please read terms and conditions of use

Original Series



Identifying MVC Version @ runtime using reflection

```
public ActionResult Index()
{
    //1. Getting the version of the MVC libraries at runtime using reflection.
    //2. Using ViewBag.Dictionary
    //3. Basic Controllers defined in the scaffolding template provided by Visual
    //   Studio
    ViewBag.MVCVersionQuery = typeof(Controller).Assembly.GetName
        ().Version.ToString();
    //4. Conventional view called by the Controller action method - Index()
    //   (without any parameters passed).
    //5. Using ViewData to pass on MVC Version Property
    ViewData["MVCversion"] = typeof(Controller).Assembly.GetName
        ().Version.ToString();
    //6. using model to pass on MVC Version Property
}
```

ASP.NET MVC DESIGN GOALS

- Does not replace web forms
 - An alternative project type
- Still runs on ASP.NET
 - Caching
 - Modules
 - Master pages
 - Providers
 - Handlers
 - Session state
- Embrace the web
 - No illusion of state – no page life cycle
 - Clean URLs and clean HTML
- Extensible
 - Pluggable view engines
 - Controller factories
- Testable
 - Maintains a strict separation of concerns

Features of MVC

- Separation of application tasks viz.business logic, UI logic and input logic
- Supports Test Driven Development
- Highly Testable framework
- Powerful URL-mapping component for comprehensible and searchable URLs
- Support existing ASP.net features viz.authentication, authorization, memberships and roles, caching, state management, configuration, health monitoring etc..

Model

- It represents the application domain.
- Applications business logic can also be contained in model.
- Model contains pieces of C# classes with set of entity properties and data annotation validations defined on top of it and data access logic.
- Model can be entities or business objects.

Controller

- Controller contains the control flow logic
- Controller handles the user interaction with the web application.
- Controller contains a set of action methods.
- User requests comes through controller to model and manipulate the records from it and then render the required data using view to UI.

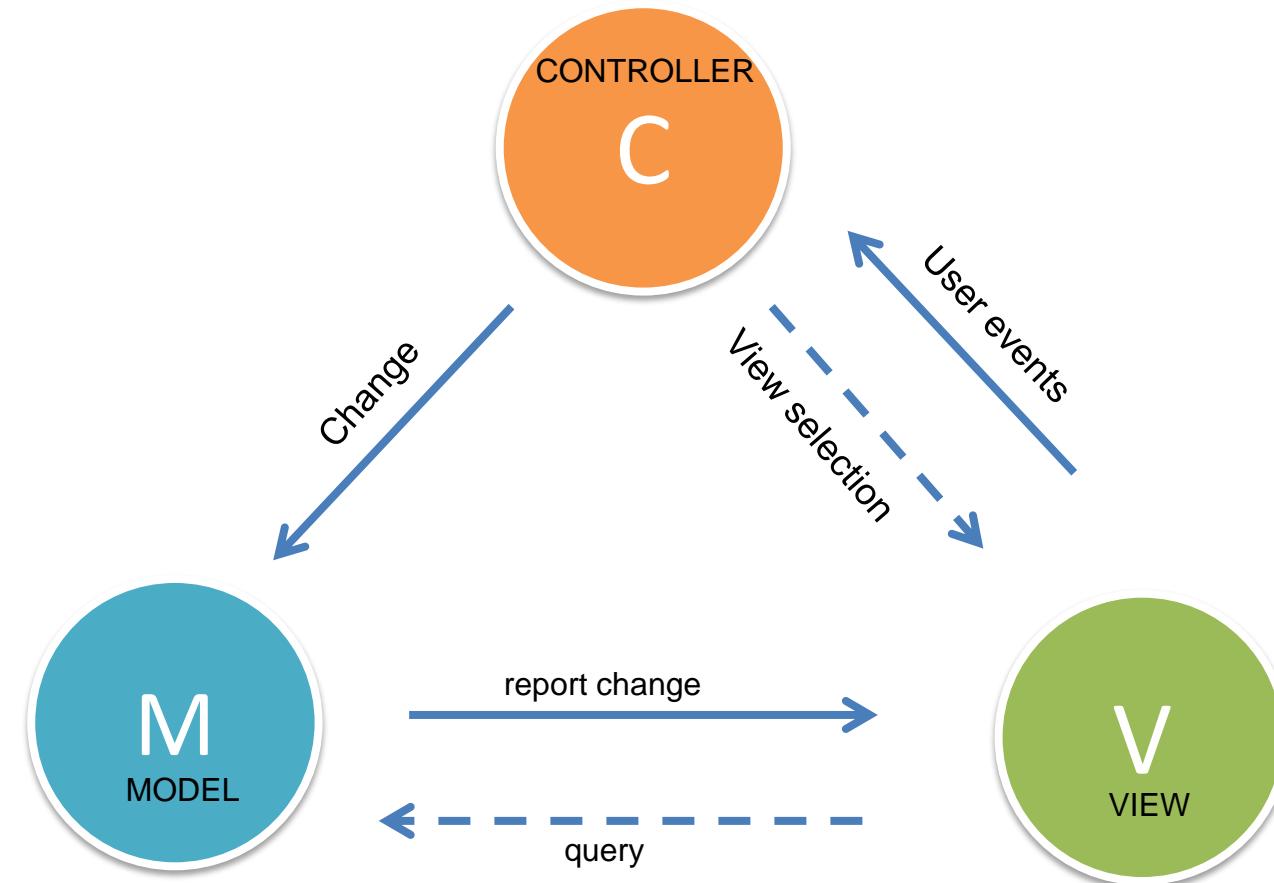
Views

- Views represent the presentation layer of the web application
- User interface (UI) logic will be contained in the view.
- Views should be dumb(shouldn't contain any application logic, only display logic).
- No code behind and No business logic.
- Action methods by default call view with the same name (Conventions over Configurations)
- Views are not tied to specific controller or action method.

MVC DESIGN PATTERN

Please read terms and conditions of use

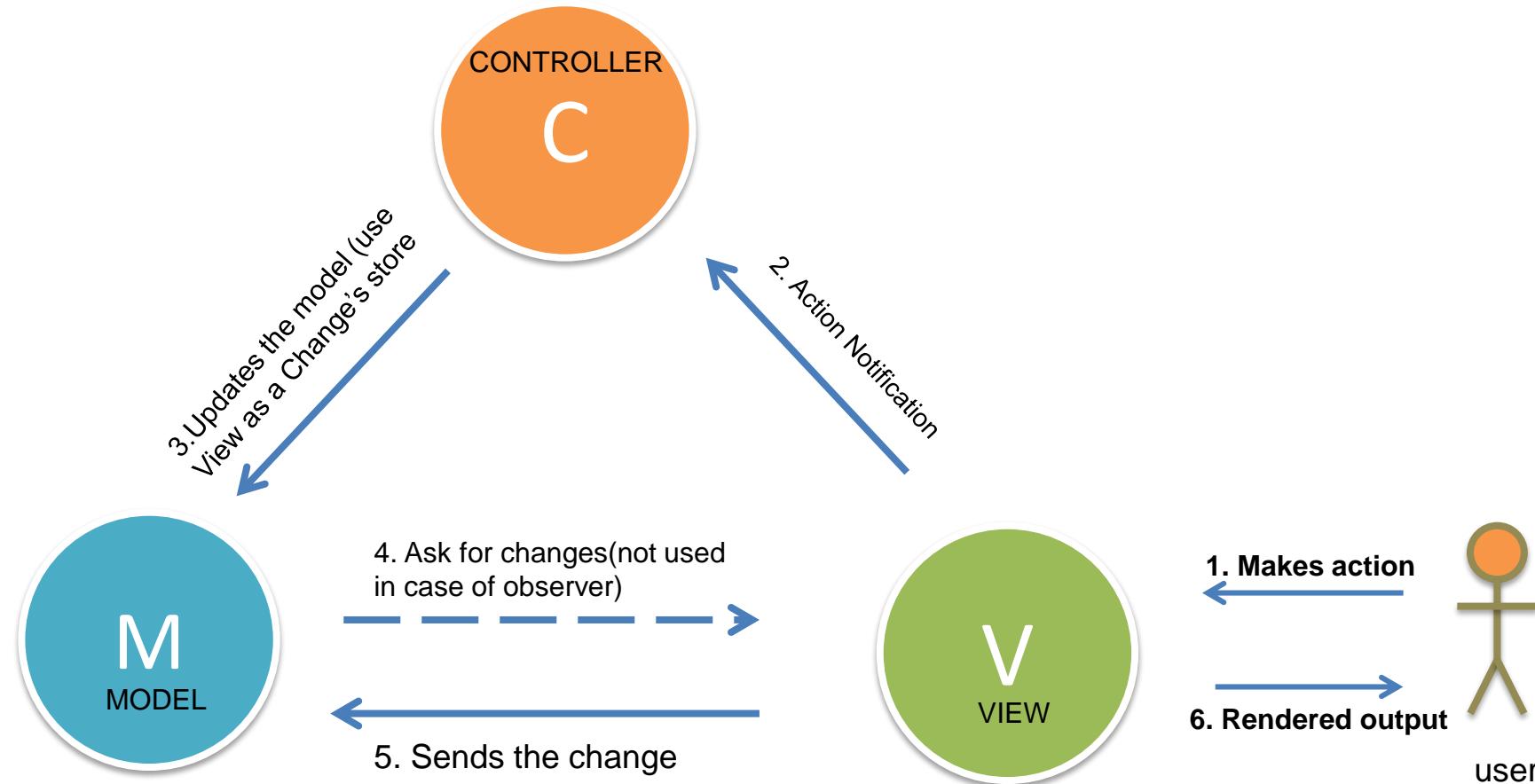
Original Series



MVC DESIGN PATTERN

Please read terms and conditions of use

Original Series

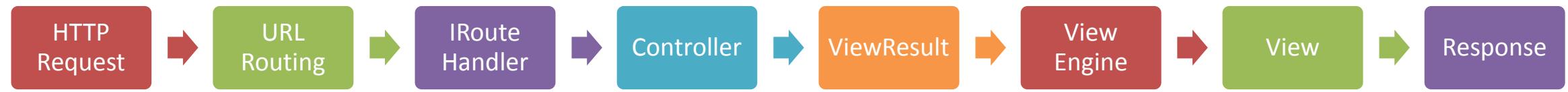


MVC DESIGN PATTERN

- Controller initializes the events of View interface to interact with model and controller.
- The user interacts with the View (UI)
- Controller handles user's events (can be the “observer” pattern) and asks Model to update.
- Model raises events, informing subscribers (View) about changes
- View (UI) (subscribes to model events) handles Model's events and shows new Model's data.
- The UI user interface waits for further user actions

MVC Life Cycle

Please read terms and conditions of use

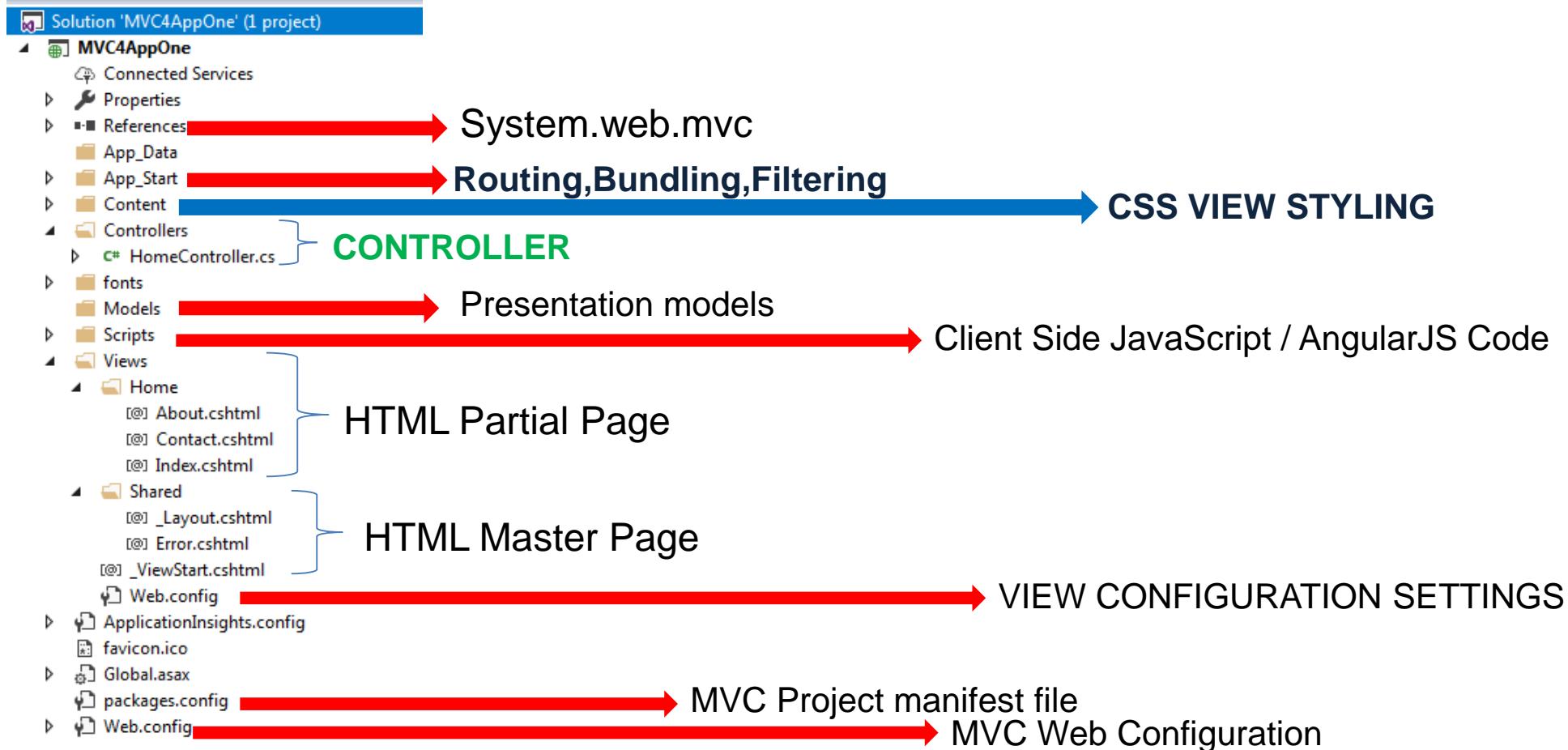


Original Series

MVC Application Structure

Please read terms and conditions of use

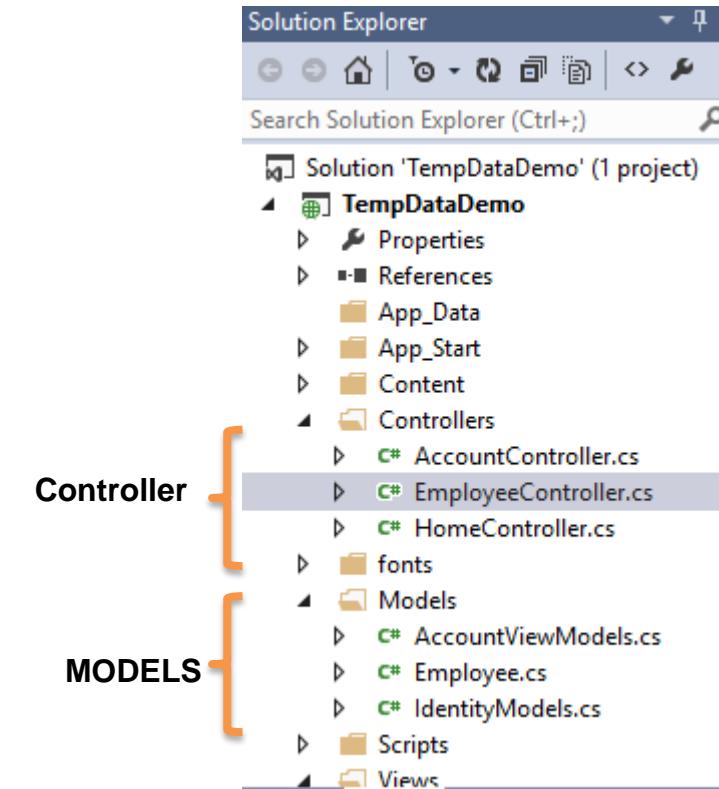
Original Series



MODELS in ASP.NET MVC

```
2 references
public class Employee
{
    1 reference
    public int EmployeeId { get; set; }
    1 reference
    public string Name { get; set; }
    1 reference
    public string Gender { get; set; }
    1 reference
    public string city { get; set; }
}

0 references
public class EmployeeController : Controller
{
    //
    // GET: /Employee/
    0 references
    public ActionResult Index()
    {
        Employee employee = new Employee
        {
            EmployeeId = 101,
            Name = "John",
            Gender = "male",
            city = "London"
        };
        return View(employee);
    }
}
```



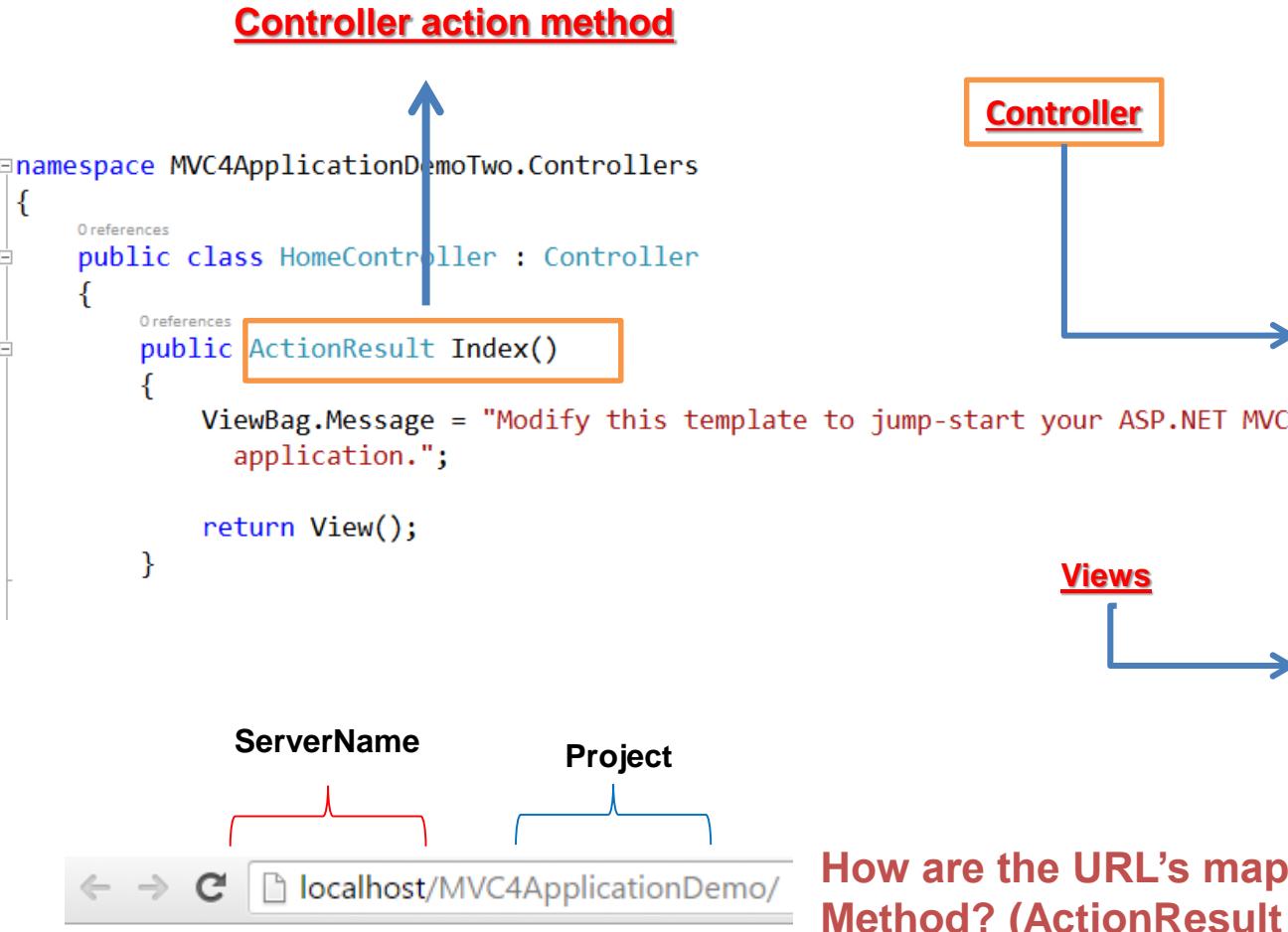
Controller

MODELS

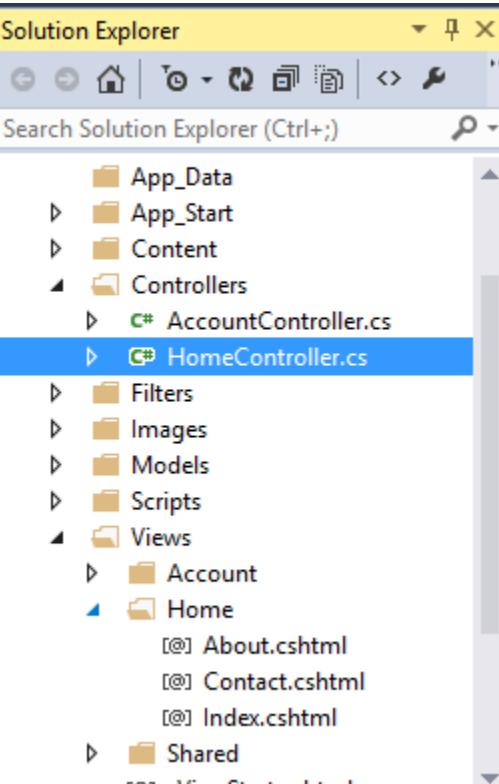
Controllers in an MVC Application

Please read terms and conditions of use

Original Series



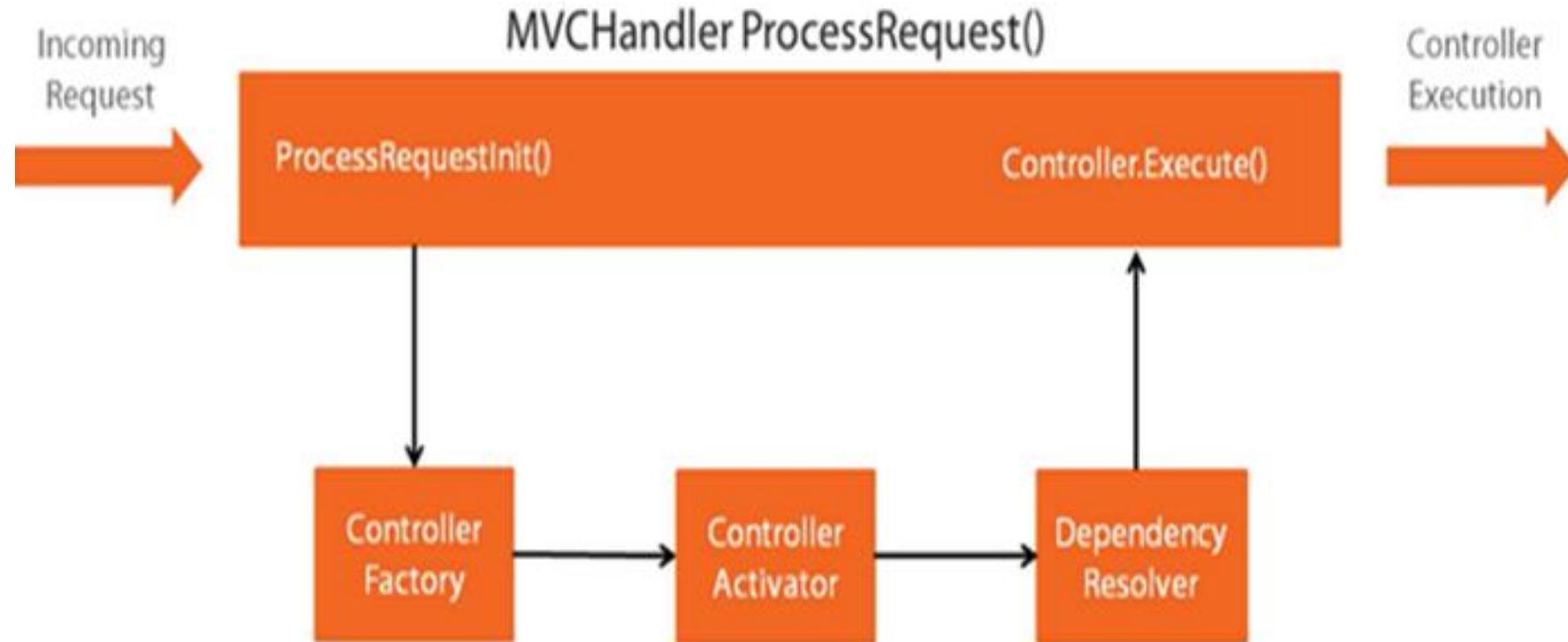
How are the URL's mapped to Controller action Method? (ActionResult Index())?



Controller Initialization

Please read terms and conditions of use

Original Series



MVC Controllers

- Public methods are “actions”
 - Method invoked by ASP.NET once it determines the proper route
 - Controller can build the model and place in ViewData
 - Return value of ActionResult tells the framework where to go next

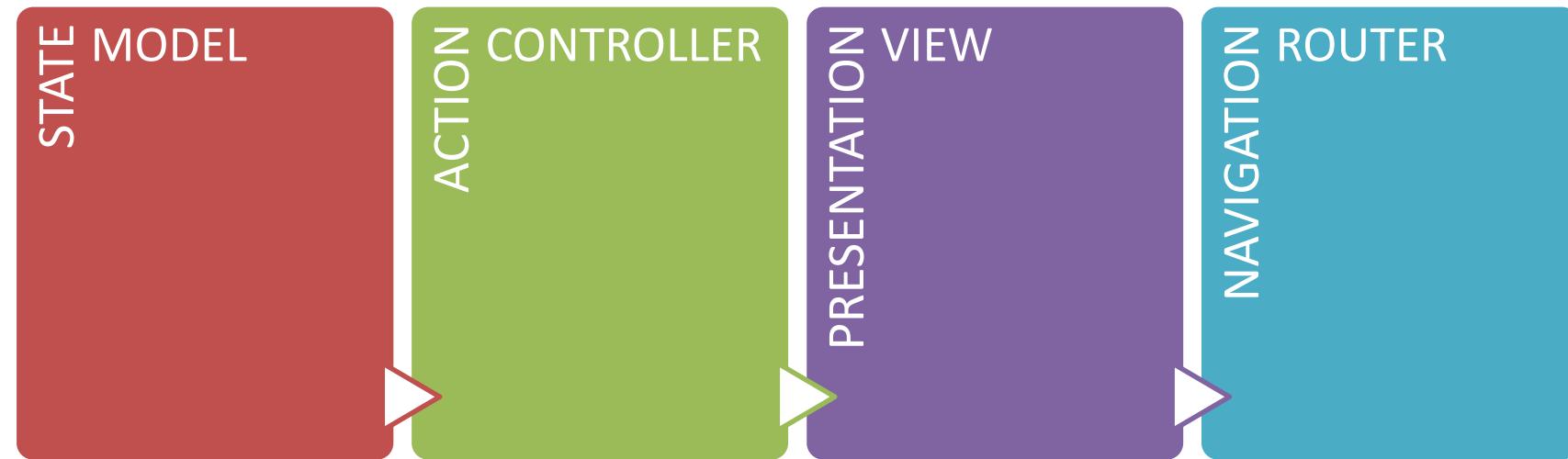
```
public class HomeController : Controller
{
    public ActionResult Index()
    {
        return View();
    }

    public ActionResult About()
    {
        ViewBag.Message = "Your application description page.";

        return View();
    }

    public ActionResult Contact()
    {
        ViewBag.Message = "Your contact page.";

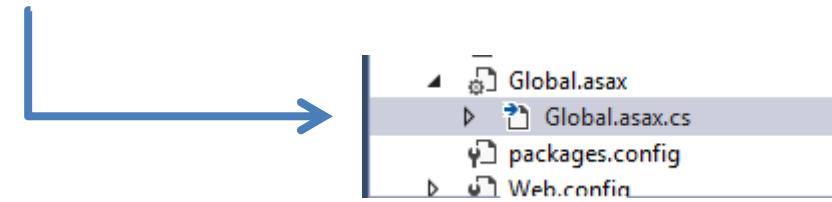
        return View();
    }
}
```



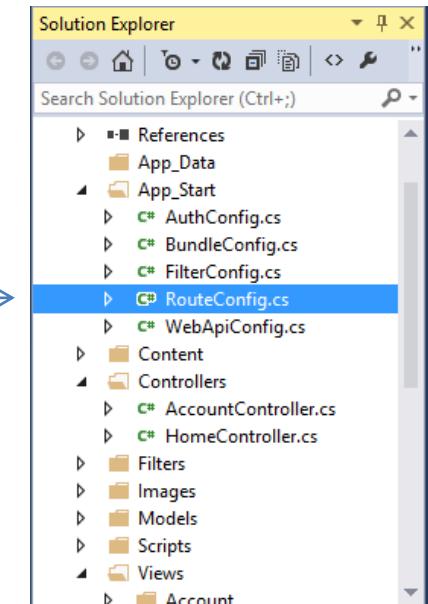
MVC Routing

- System.Web.Routing
 - Part of ASP.NET and released with .NET 3.5 SP1
- Directs incoming request to and MVC controller
 - Defines routes during application startup
 - Map URLs to controller action with parameters

```
routes.MapRoute(  
    "Default", // Route name  
    "{controller}/{action}/{id}", //URL with parameters  
    new {  
        controller = "Home",  
        action = "Index",  
        id = UrlParameter.Optional  
    } //Parameter defaults
```



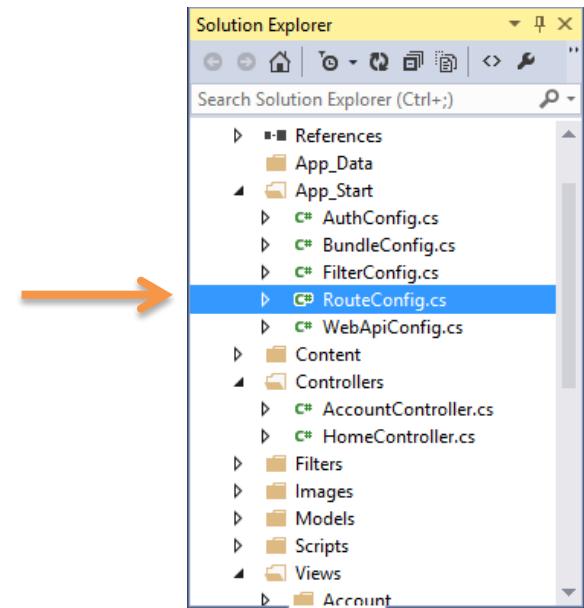
```
protected void Application_Start()  
{  
    AreaRegistration.RegisterAllAreas();  
  
    WebApiConfig.Register(GlobalConfiguration.Configuration);  
    FilterConfig.RegisterGlobalFilters(GlobalFilters.Filters);  
    RouteConfig.RegisterRoutes(RouteTable.Routes);  
    BundleConfig.RegisterBundles(BundleTable.Bundles);  
    AuthConfig.RegisterAuth();  
}
```



MVC Routing

```
1 reference
public class RouteConfig
{
    1 reference
    public static void RegisterRoutes(RouteCollection routes)
    {
        routes.IgnoreRoute("{resource}.axd/{*pathInfo}");

        routes.MapRoute(
            name: "Default",
            url: "{controller}/{action}/{id}",
            defaults: new { controller = "Home", action = "Index", id =
                UrlParameter.Optional }
        );
    }
}
```

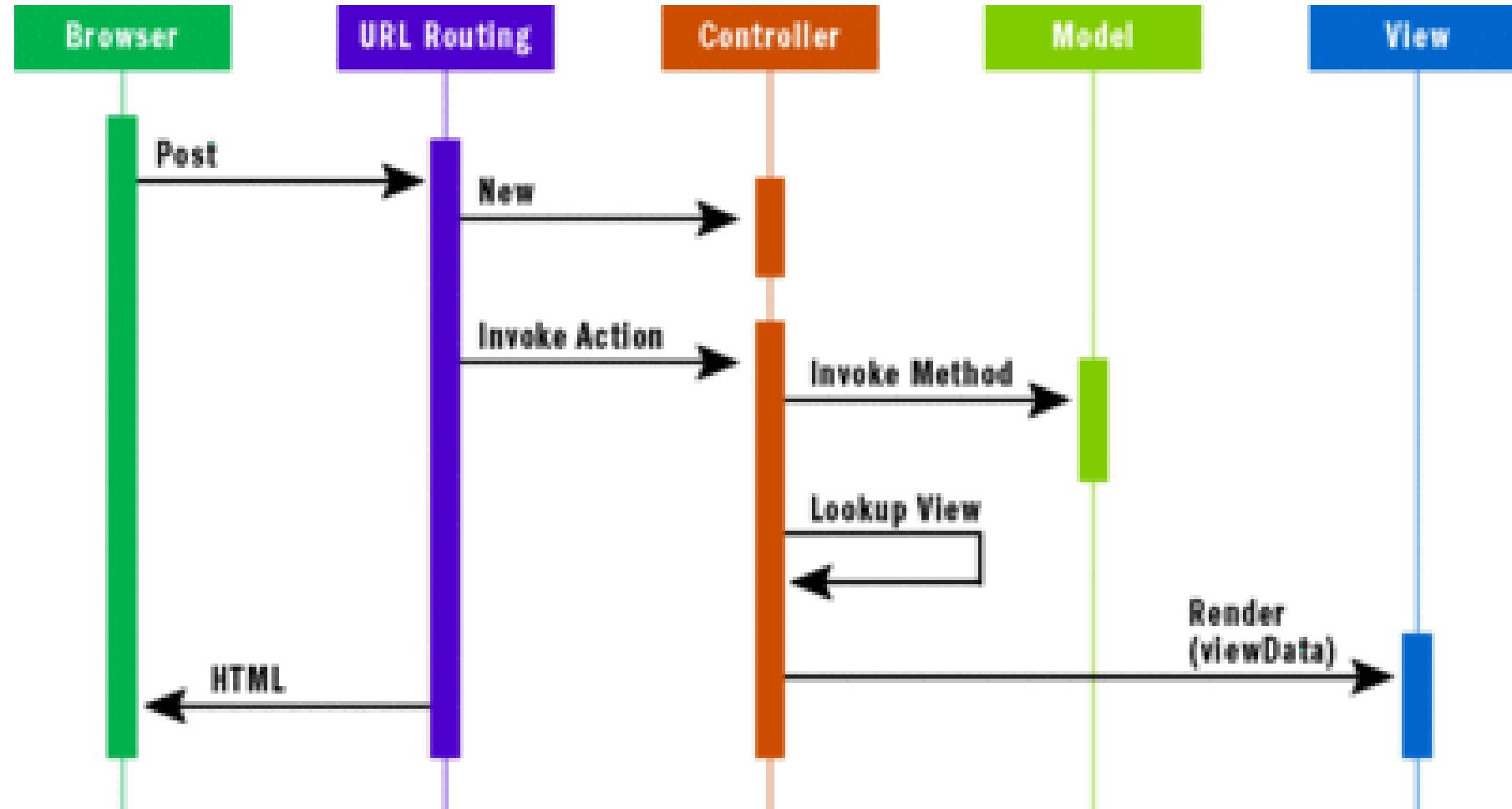


Enable trace in web.config

```
<system.web>
  <compilation debug="true" targetFramework="4.0" />
  <trace enabled="true" pageOutput="false"/>
```

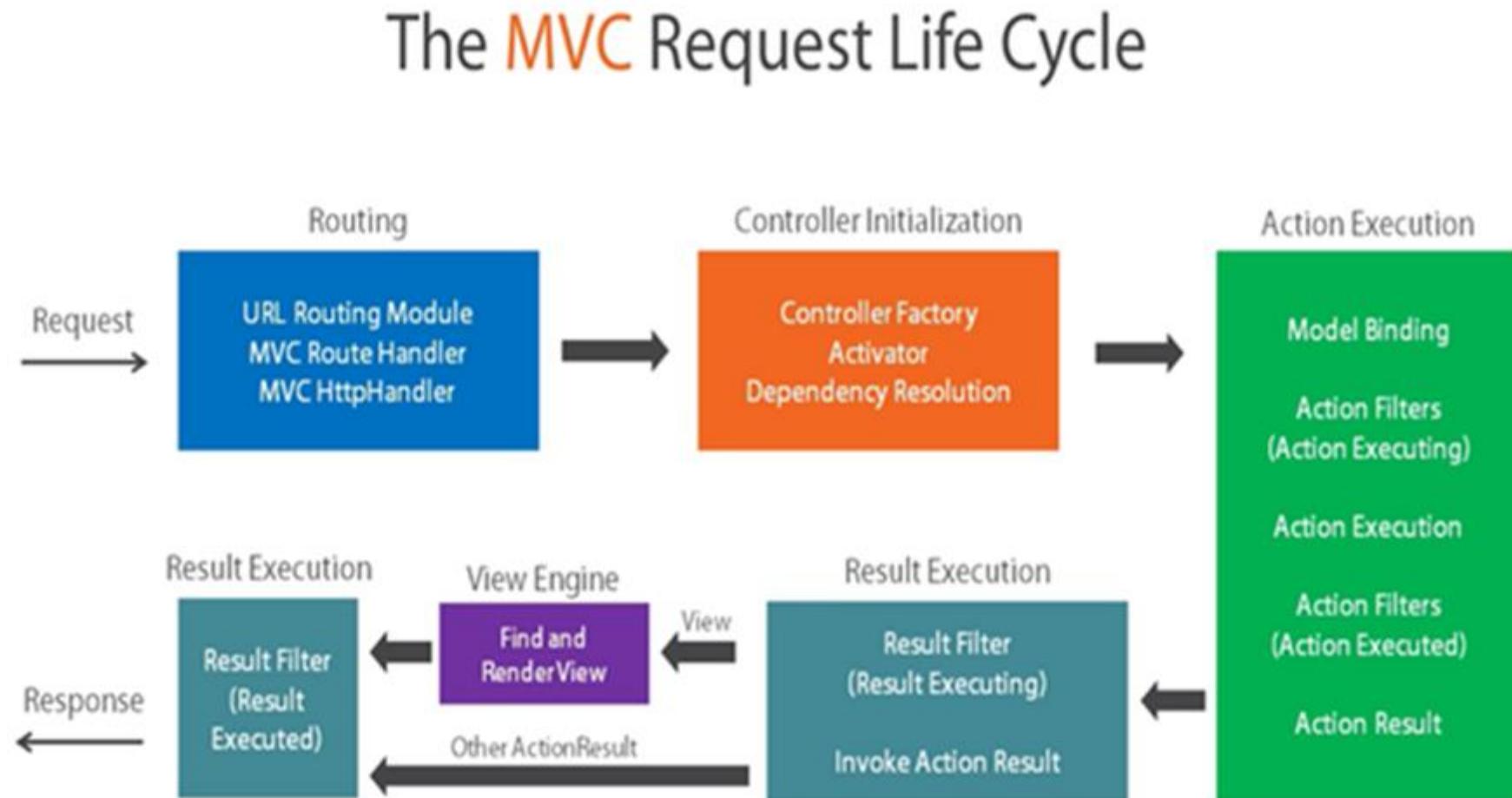
A screenshot of a browser window displaying the 'Application Trace' page. The URL is 'localhost:2525/trace.axd'. The page shows a table titled 'Requests to this Application' with one entry. The table has columns for No., Time of Request, File, Status Code, Verb, and View Details. The entry shows '1' in the No. column, '10/03/2016 11:43:14' in the Time of Request column, 'File' in the File column, '200' in the Status Code column, 'GET' in the Verb column, and a 'View Details' link. At the bottom of the page, it says 'Microsoft .NET Framework Version:4.0.30319; ASP.NET Version:4.6.114.0'.

<http://localhost:2525/trace.axd>



Stages of Request Execution

Stage	Description
httpRequest to the Application	Global.asax file, Route Objects are added to the Route Table Object
Perform routing	UrlRoutingModule uses the first matching Route Object in the RouteTable collection to create the RouteData Object, which it then uses to create a RequestContext Object
Create MVC request handler	MvcRouteHandler object creates an instance of the MvcHandler class and passes the RequestContext instance to the handler
Create Controller	MvcRouteHandler object uses the RequestContext instance to identify the IControllerFactory object (typically an instance of the DefaultControllerFactory class) to create the controller instance with.
Execute Controller	MvcHandler instance calls the controller's EXECUTE method
Invoke Action	For controllers that inherit from the ControllerBase class. The ControllerActionInvoker object that is associated with the controller determines which action method of the controller class to call and then calls that method
Execute Result	Action method receives user input, prepares appropriate response data and then executes the result by returning a result type. The built-in result type that can be executed include the following: ViewResult(which renders a view and is most often used result type), RedirectToRouteResult, RedirectResult, ContentResult, JsonResult, FileResult and EmptyResult.



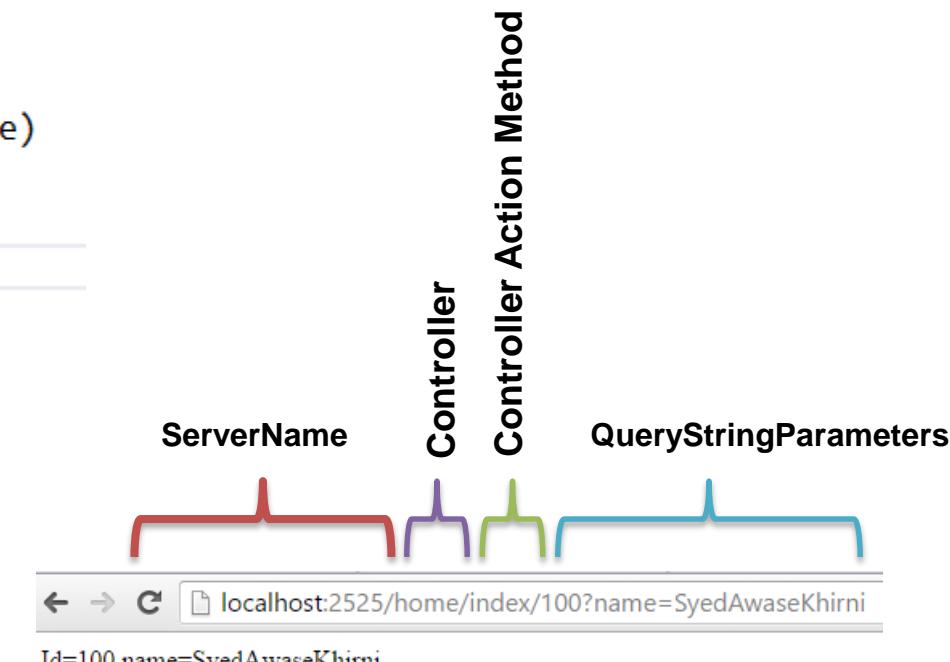
ASP.NET MVC FORM POST PARAMETERS

Please read terms and conditions of use

- ASP.NET MVC will automatically pass any query string or form post parameters named “name” to Index action method when it is invoked.

```
0 references
public class HomeController : Controller
{
    0 references
    public string Index(string id, string name)
    {
        return "Id=" + id + " name=" + name;
    }
}
```

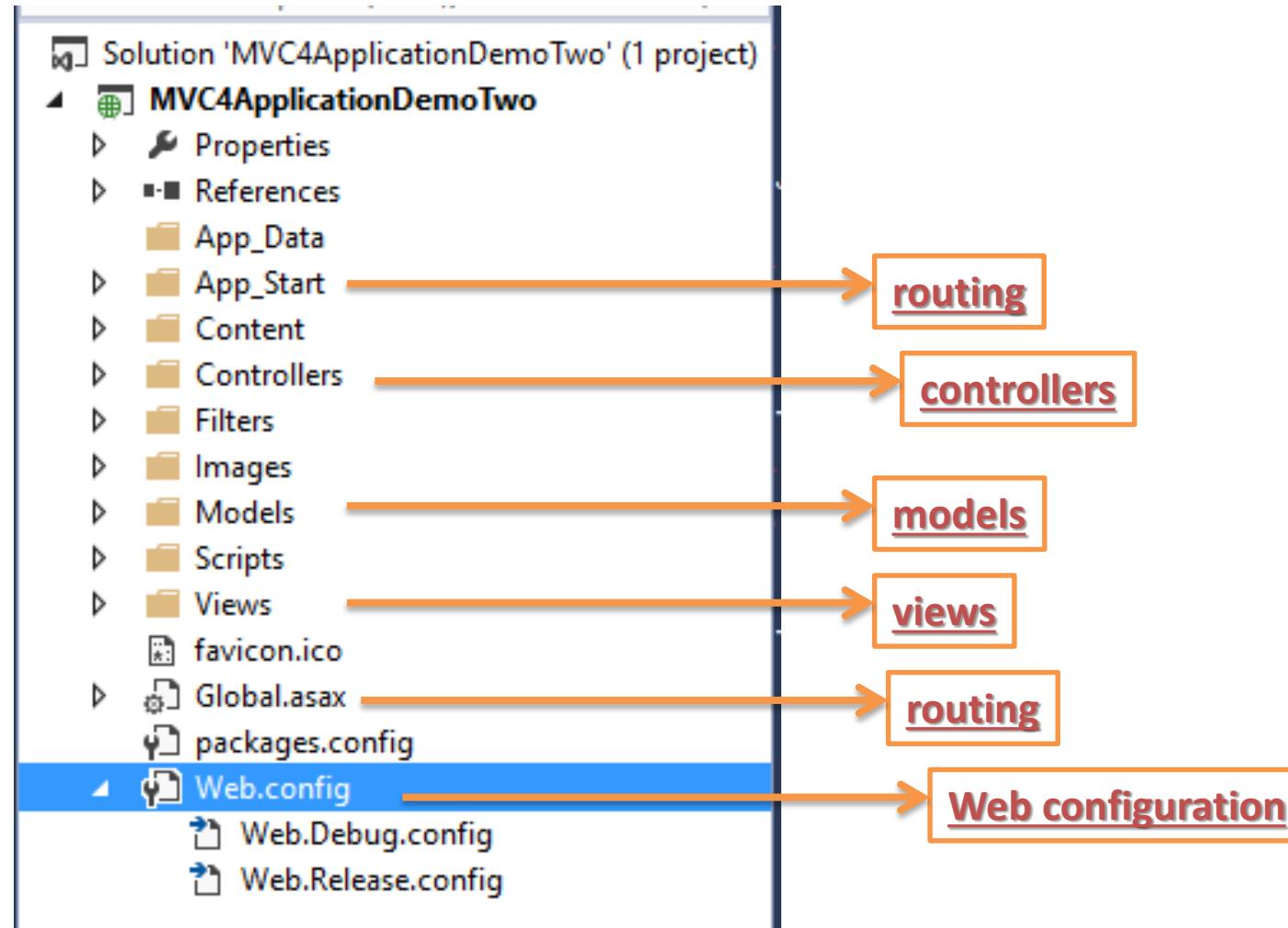
Original Series



Controller and View Conventions

Please read terms and conditions of use

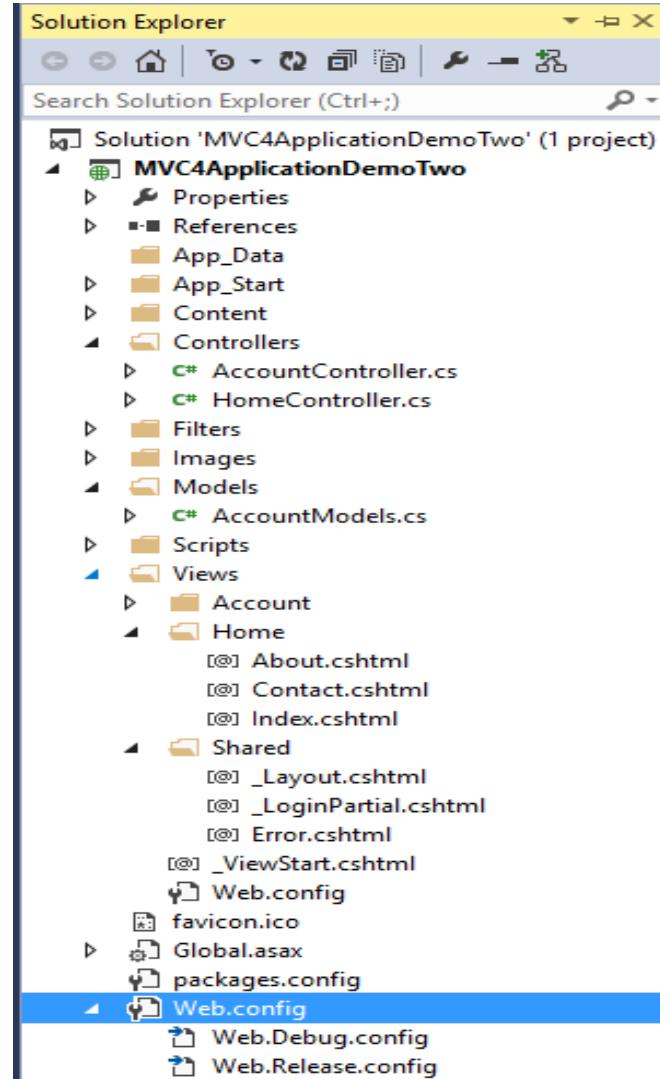
Original Series



Controller and View Conventions

Please read terms and conditions of use

Original Series



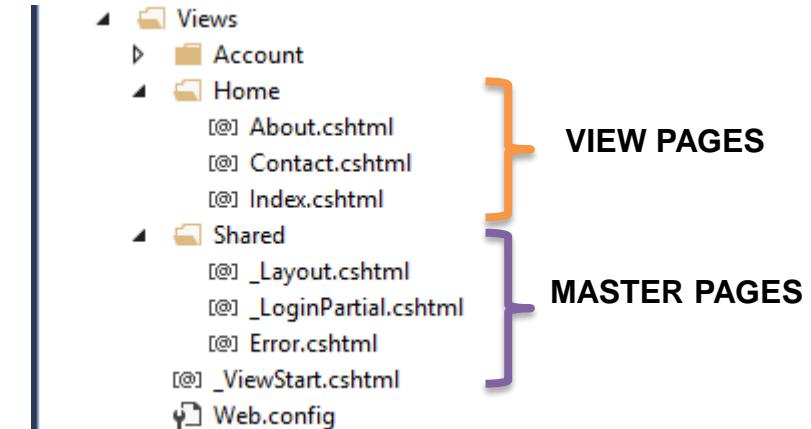
- **Controllers folder**
 - Recommended location for controllers
 - Controller type name must end with **Controller**(thus omitted from the route)
- **Views folder**
 - Recommended location for views
 - .aspx, .ascx, .master files
 - Subfolders for every controller
 - Shared folder contains views used by multiple controllers

Views/View Engine

- WebForms views
 - Use .aspx, ascx and .master files
 - No server-side form required, no viewstate used
- Razor
 - Preferred view engine moving forward
 - Introduced in MVC 5.0

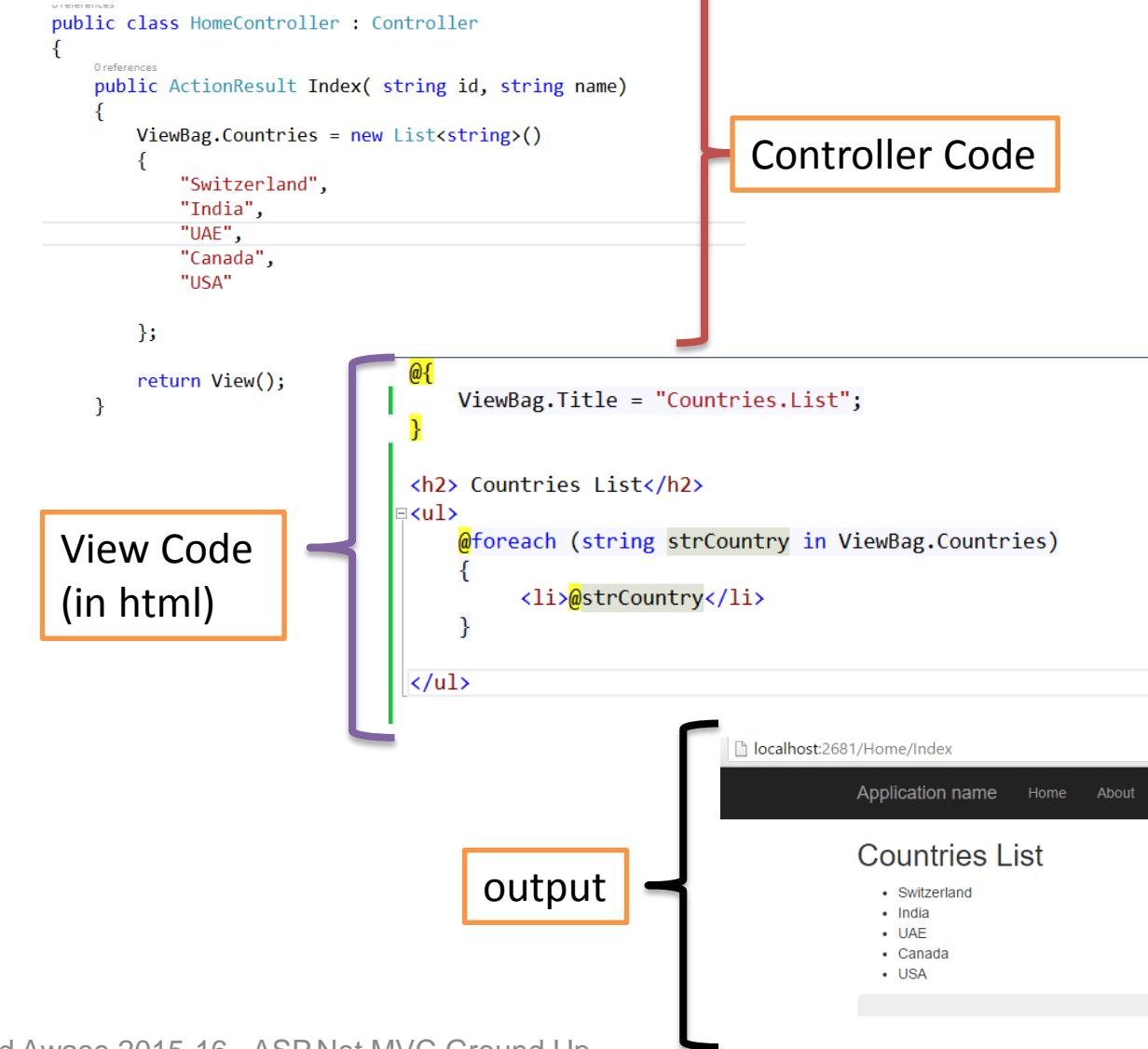
Views

- **Views are .aspx files**
 - Derive from ViewPage, which derives from Page
 - Have a ViewData dictionary property populated from the controller
 - Still use markup, can still contain server-side script
 - No server-side form required
 - No _VIEWSTATE
 - No control state
- **Strongly types views**
 - Derive from ViewPage<T> instead of ViewPage
 - Generic type Parameter represents the type of the model



Views

- ViewData, ViewBag and TempData mechanisms to pass on data from the Controller to the View.
- To pass data from Controller to a View. *It's always a good practice to use strongly typed view models*
- *"@" symbol is used to switch between html and C# code*



Strongly Typed View

- *It's always a good practice to use strongly typed view models*
- *ViewData and ViewBag are used to pass data from a controller to a view.*
- *Strongly types view models provide compile time error checking.*

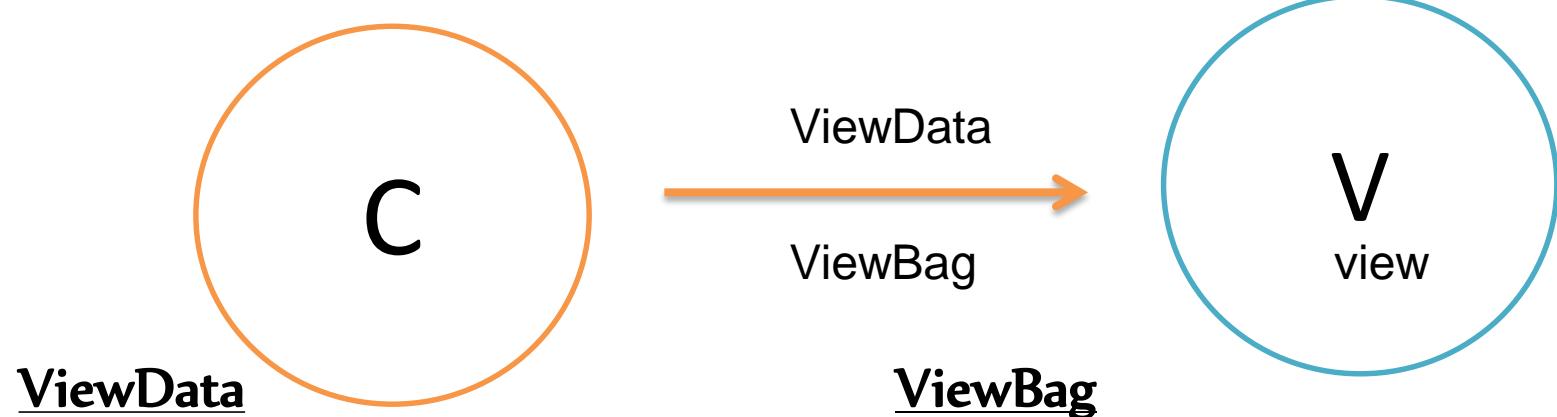
```
public class HomeController : Controller
{
    public ActionResult Index()
    {
        return View();
    }

    public ActionResult About()
    {
        ViewBag.Message = "Your application description page.";

        return View();
    }
}
```

Both ViewData and ViewBag do not Provide compile time error checking

ViewData vs ViewBag



- ViewData is a dictionary of objects that is derived from ViewDataDictionary class and is accessible using strings as keys.
- ViewData requires typecasting for complex data type and check for null values to avoid error.

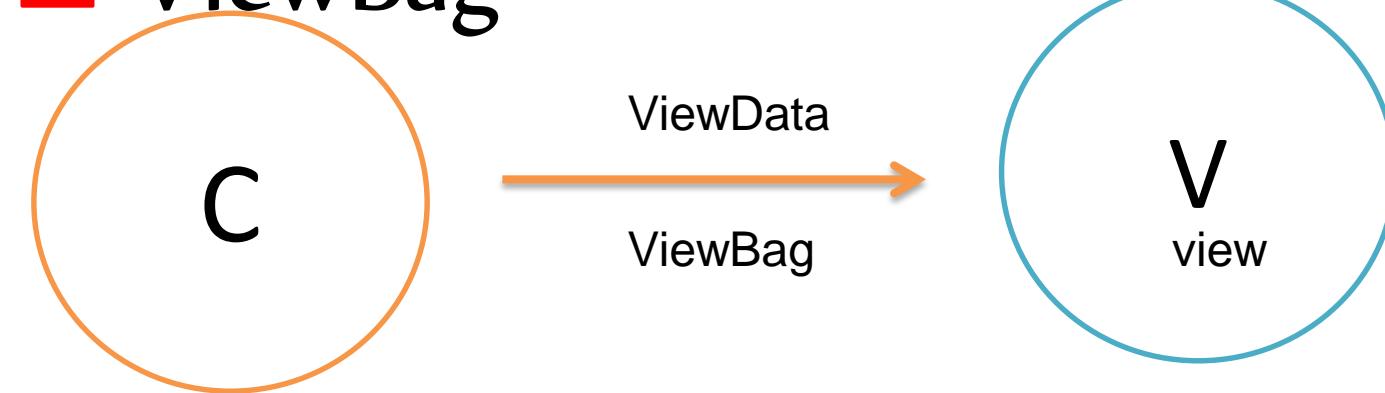
```
ViewData["YourKey"] = "YourData";
```

ViewBag

- A dynamic feature introduced in C# 4.0
- Allows an object to have properties dynamically added to it.
- ViewBag doesn't require typecasting for complex data type.

```
ViewBag.YourProperty="YourData";
```

ViewData ≈ ViewBag



- Similarities between ViewBag & ViewData:
 - Helps to maintain data when you move from controller to view.
 - Used to pass data from controller to corresponding view.
 - **Short life** means value becomes null when redirection occurs. This is because their *goal is to provide a way to communicate between controllers and views*. **It's a communication mechanism within the server call.**

TEMPDATA

- A dictionary derived from TempDataDictionary Class and stored in short lives Session and it is a string key and object value.
- The difference is the lifecycle of the object.
- **Works with 302/303 redirection because it's in the same HTTP Request.**
- **Used to move data from one controller to other controller or from one action to other action.**
- **On redirection, "Tempdata" helps to maintain data between those redirects.**
- **Internally uses session variables.**
- **Tempdata is used during the current and subsequent request only, means it is used when you are sure that next request will be redirecting to next view.**
- **Requires typecasting for complex data type and check for null values to avoid error.**
- **Used to store only one time messages like error messages, validation messages**

Scenarios when to use ViewBag, ViewData and TempData

- **ViewBag and View Data object work well in the following scenarios:**
 - Incorporating dropdown lists of lookup data into an entity
 - Components like a shopping cart
 - Widgets like a user profile widget
 - Small amounts of aggregate data
- **TempData object works well in one basic scenario:**
 - Passing data between the current and next HTTP requests

View Helpers

Please read terms and conditions of use

- **Helpers available via properties of a ViewPage**

Property	Class	Description
Ajax	AjaxHelper	Invoke controller actions asynchronously and update client content
Html	HtmlHelper	Create anchor tags, encode HTML
Url	UrlHelper	Create URLs to invoke controller actions

```
<div>
    @Html.ActionLink("About Us", "About", "Home");
</div>
```



`
About us`

Original Series

Preparing Models

- **Attributes**
 - Decorate properties
- **Available Attributes**
 - `DataType Attribute`
 - `Display Attribute`
 - `Validation`
 - `RequiredAttribute`
 - `StringLength Attribute`
 - `RegularExpressionAttribute`
 - `CompareAttribute`

ASP.NET WEBFORM vs MVC

Original Series
Please read terms and conditions of use

Features	Web Forms	ASP.NET MVC
Separation of concerns	NO	YES
Familiar Event Driven Model	YES	NO
ViewState Issues	YES	NO
Server Controls	Yes	No
Control over HTML	No	Yes
Test Driven Development	No	Yes

ASP.NET WEBFORM vs MVC

ASP.NET Web Forms	ASP.NET MVC
ASP.NET Web Forms uses Page controller pattern approach for rendering layout. In this approach, every page has it's own controller i.e. code-behind file that processes the request.	ASP.NET MVC uses Front Controller approach . That approach means ,a common controller for all pages, processes the requests.
No separation of concerns. As we discussed that every page (.aspx) has it's own controller (code behind i.e. aspx.cs/.vb file), so both are tightly coupled.	Very clean separation of concerns. View and Controller are neatly separate.
Because of this coupled behavior, automated testing is really difficult.	Testability is key feature in ASP.NET MVC. Test driven development is quite simple using this approach.
In order to achieve stateful behavior, viewstate is used. Purpose was to give developers, the same experience of a typical WinForms application.	ASP.NET MVC approach is stateless as that of the web. So here no concept of viewstate.
Statefulness has a lots of problem for web environment in case of excessively large viewstate. Large viewstate means increase in page size.	As controller and view are not dependent and also no viewstate concept in ASP.NET MVC, so output is very clean.

ASP.NET WEBFORM vs MVC

Please read terms and conditions of use

Original Series

ASP.NET WebForms model follows a Page Life cycle.	No Page Life cycle like WebForms. Request cycle is simple in ASP.NET MVC model.
Along with statefulness, microsoft tries to introduce server-side controls as in Windows applications. Purpose was to provide somehow an abstraction to the details of HTML. In ASP.NET Web Forms, minimal knowledge of HTML, JavaScript and CSS is required.	In MVC, detailed knowledge of HTML, JavaScript and CSS is required.
Above abstraction was good but provides limited control over HTML, JavaScript and CSS which is necessary in many cases.	Full control over HTML, JavaScript and CSS.
With a lots of control libraries availability and limited knowledge of other related technologies, ASP.NET WebForms is RAD(Rapid Application Development) approach.	It's a step back. For developers decrease in productivity.
It's good for small scale applications with limited team size.	It's better as well as recommended approach for large-scale applications where different teams are working together.

	WebForms is an abstraction of web application programming; envisaged to ease the transition for the Visual Basic programmers who were the primary target for .NET when it was launched. Uses VB's event-based model, such as viewstate and postbacks
	Concurrency between multiple clients goes out the window.
	It is stored as a massive chunk of data in a hidden field leading to increased page load sizes.
	It is wholly dependent on POST methods, so it breaks navigation in HTML.

Summary

- ASP.NET MVC is an alternative to WEB FORMS
 - Builds on ASP.NET, does not replace ASP.NET
- Strives for simplicity
 - Clean URLs
 - Clean HTML
- Separation of concerns
 - It's what Model, View, Controller is about.

SECTION -II

CONTROLLERS

Overview

Please read terms and conditions of use

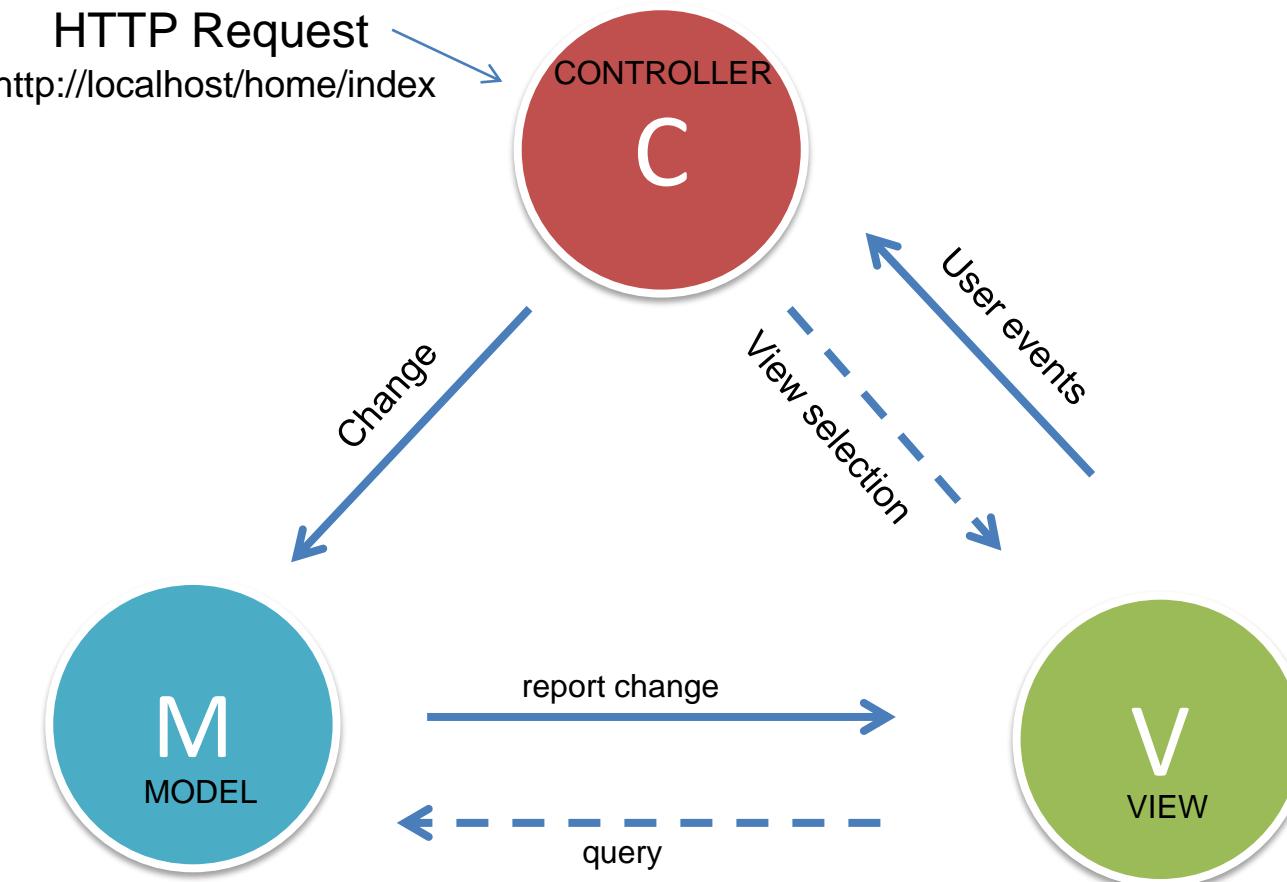
- MVC Controllers
- Routing Rules
- Controller Actions
- Action filters
- Action parameters

Original Series

MVC CONTROLLER

Please read terms and conditions of use

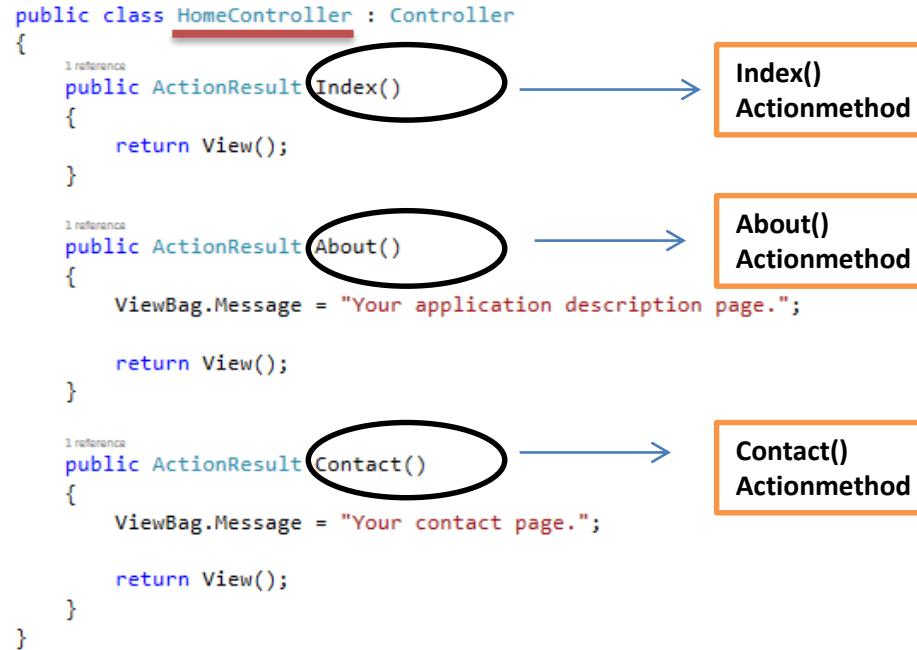
Original Series



MVC Controllers

- ASP.NET MVC Framework maps URLs to classes that are referred to as **Controllers**
- **Controllers process incoming requests, handle user inputs and interacts, execute appropriate application logic.**
- **It calls a separate view component to generate the HTML markup for the request.**
- The base class for all controllers is the [ControllerBase](#) class, which provides general MVC handling. The [Controller](#) class inherits from [ControllerBase](#) and is the default implement of a controller.
- The **Controller** class is responsible for the following processing stages:
 - Locating the appropriate action method to call and validating that it can be called
 - Getting the values to use as the action method's arguments
 - Handling all errors that might occur during the execution of the action method
 - Providing the default Redering Engine to render views.

Controller Action Methods



- User Interaction with ASP.NET MVC Applications is organized around controllers and action methods.
- The controller defines action methods.
- Controllers can include as many action methods as needed
- **Action methods typically have a one-to-one mapping with user interactions.**
- On http url request, MVC application uses routing rules that are defined in the GLOBAL.ASAX to parse the URL and determine the path of the Controller
- Controller determines appropriate action method to handle the request.

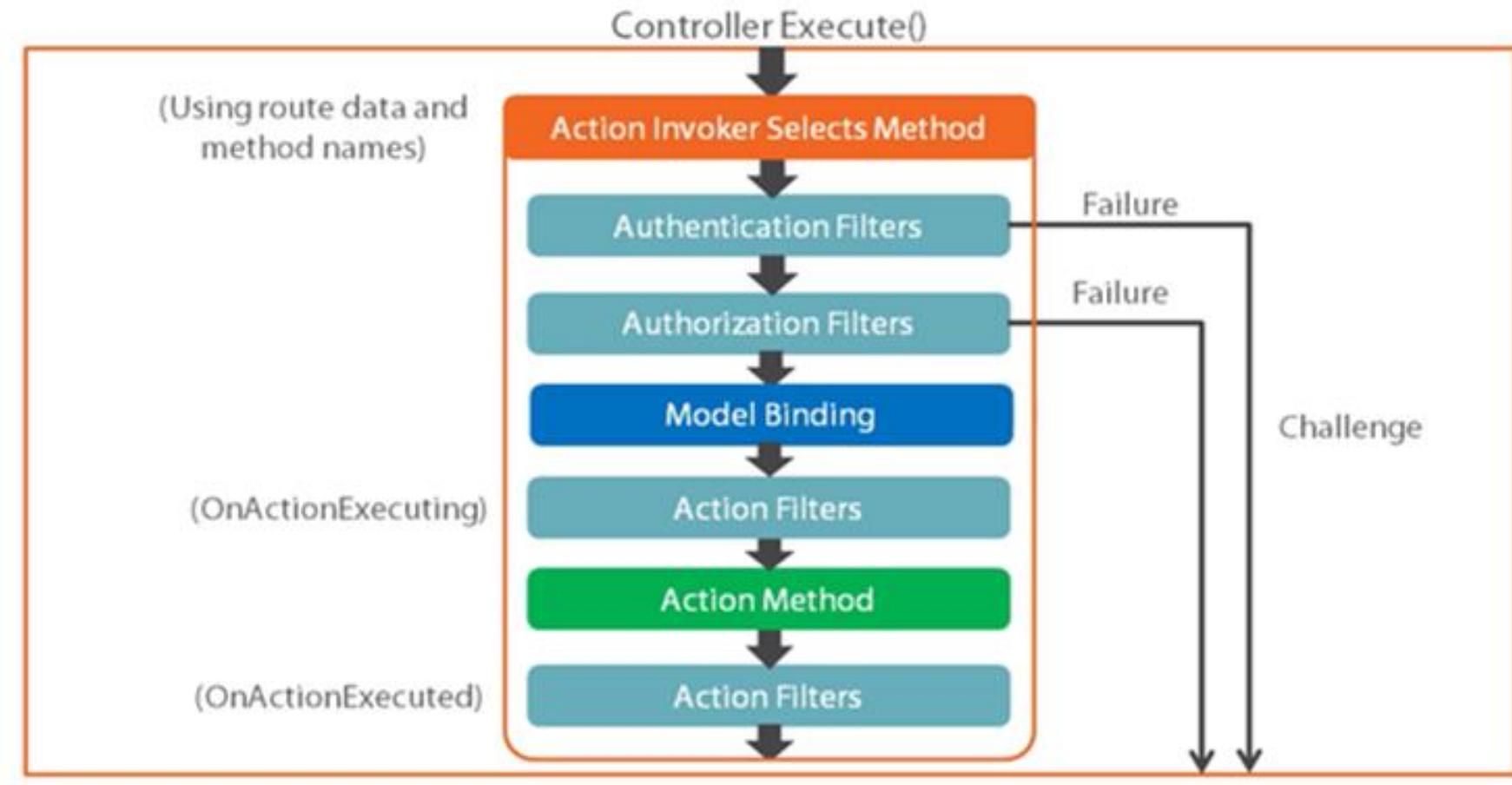
<http://localhost/home/Index> -> Index()

<http://localhost/home/About> -> About()

<http://localhost/home/Contact> -> Contact()

The Action Method Execution Process

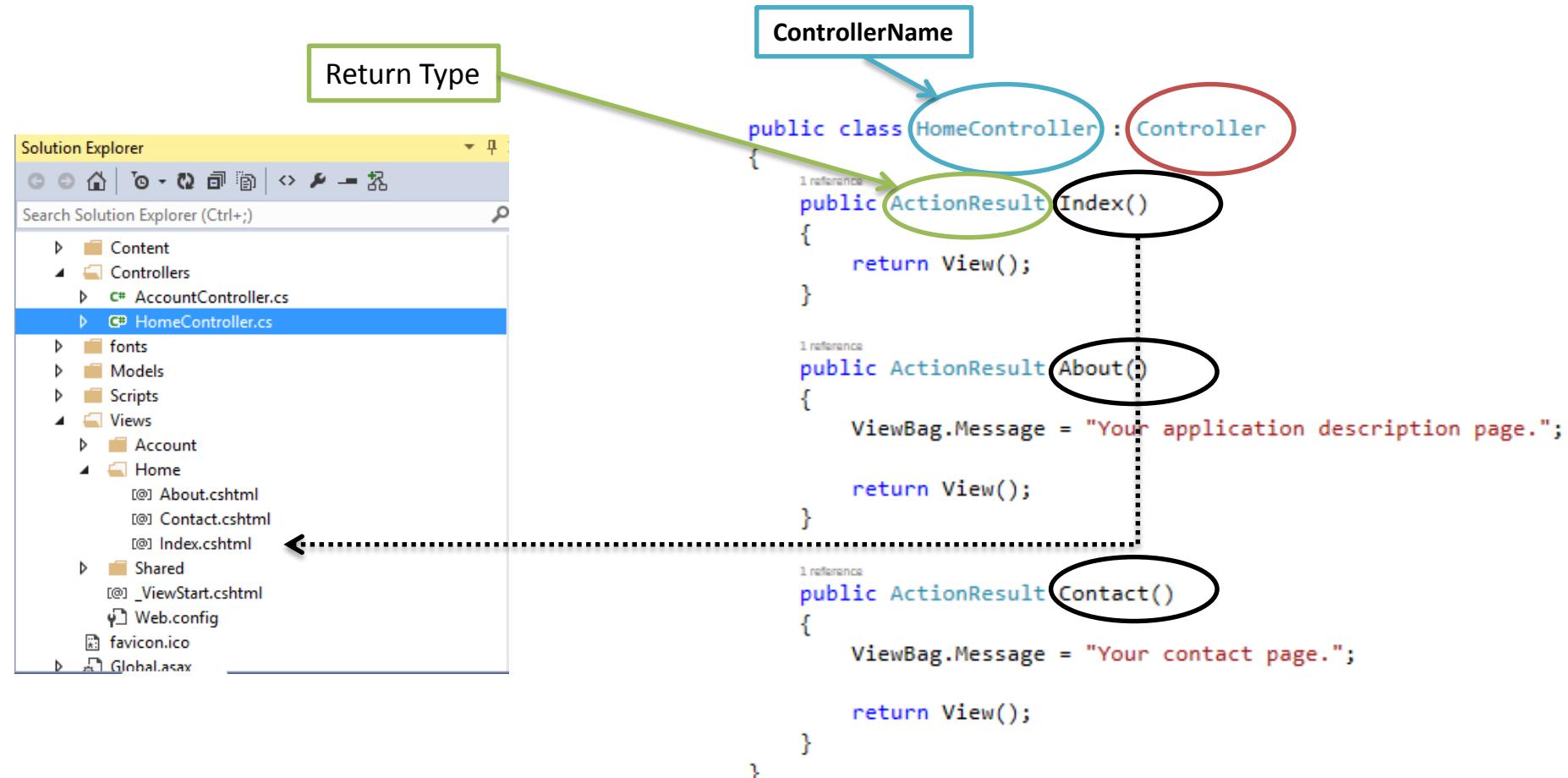
Original Series
Please read terms and conditions of use



Action Results

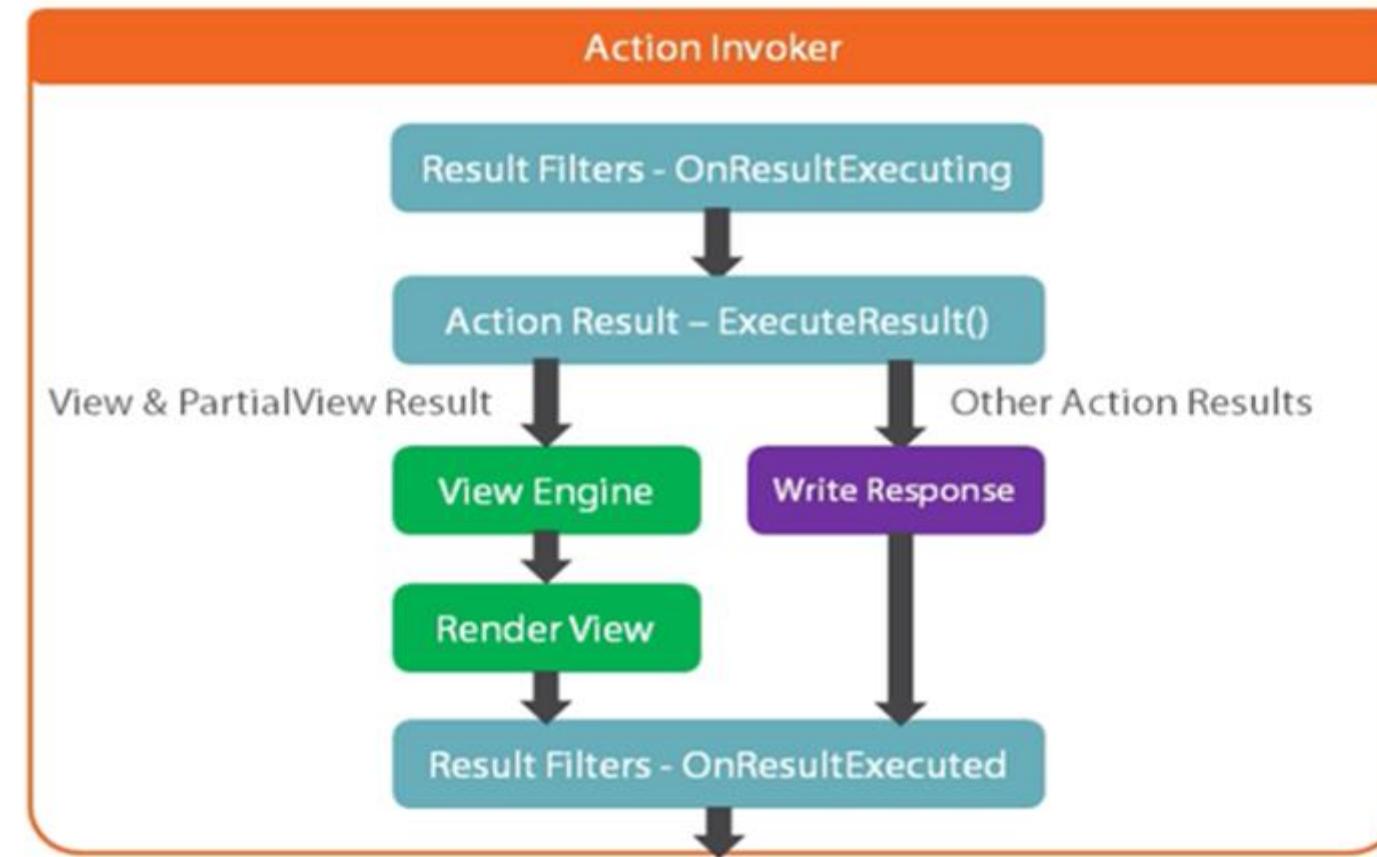
Please read terms and conditions of use

Original Series



- **ActionResult Return Type**
 - When creating new controllers, they will come with one or more actions by default.
 - The Empty controller includes an `Index` action with a return value of type `ActionResult`

The Action Result Execution Process



Controller Action Methods

- Actions typically return an ActionResult/VIEWRESULT

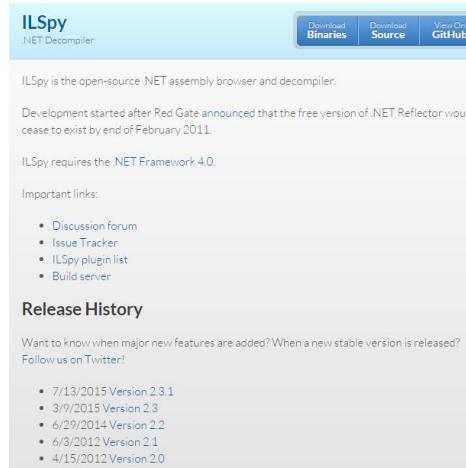
Name	FrameworkBehaviour	Producing Method
ContentResult	Returns a String literal	Content
EmptyResult	No response	
FileContentResult/ FilePathResult/FileStreamResult	Return the contents of a file	File
HttpUnauthorizedResult	Returns an HTTP 403 status	
JavaScript Result	Returns a script to execute	JavaScript
JSONResult	Returns data in JSON format	JSON
RedirectResult	Redirects the client to a new URL	Redirect
RedirectToRouteResult	Redirect to another action, or another controller's action	RedirectToRoute/RedirectToAction
ViewResult PartialViewResult	Response is the responsibility of view engine	View/PartialView

Please read terms and conditions of use

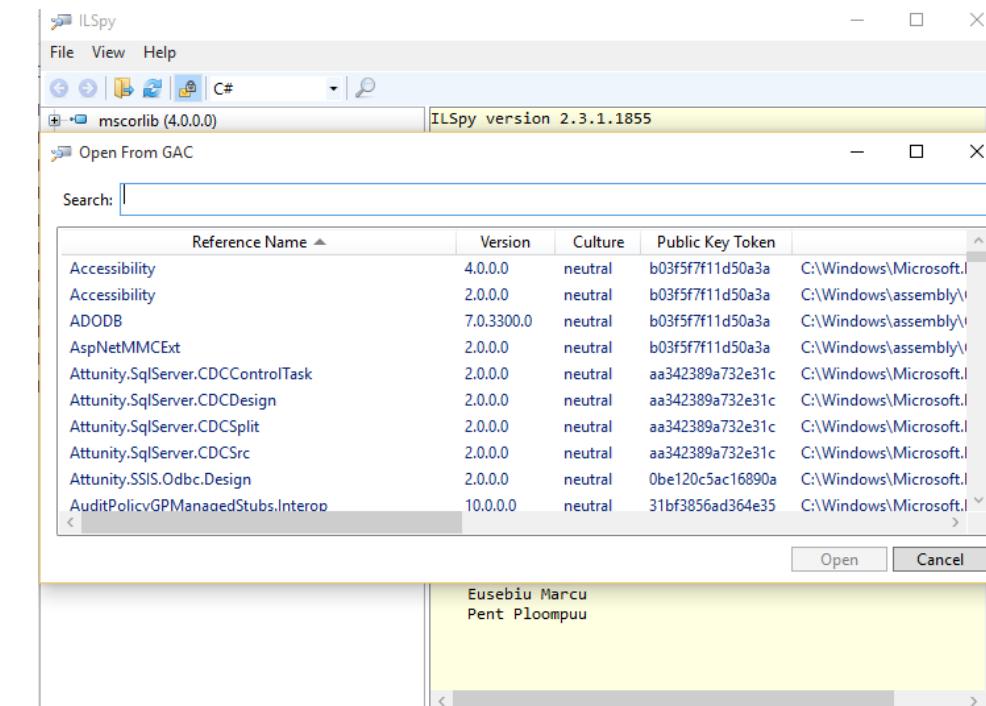
Original Series

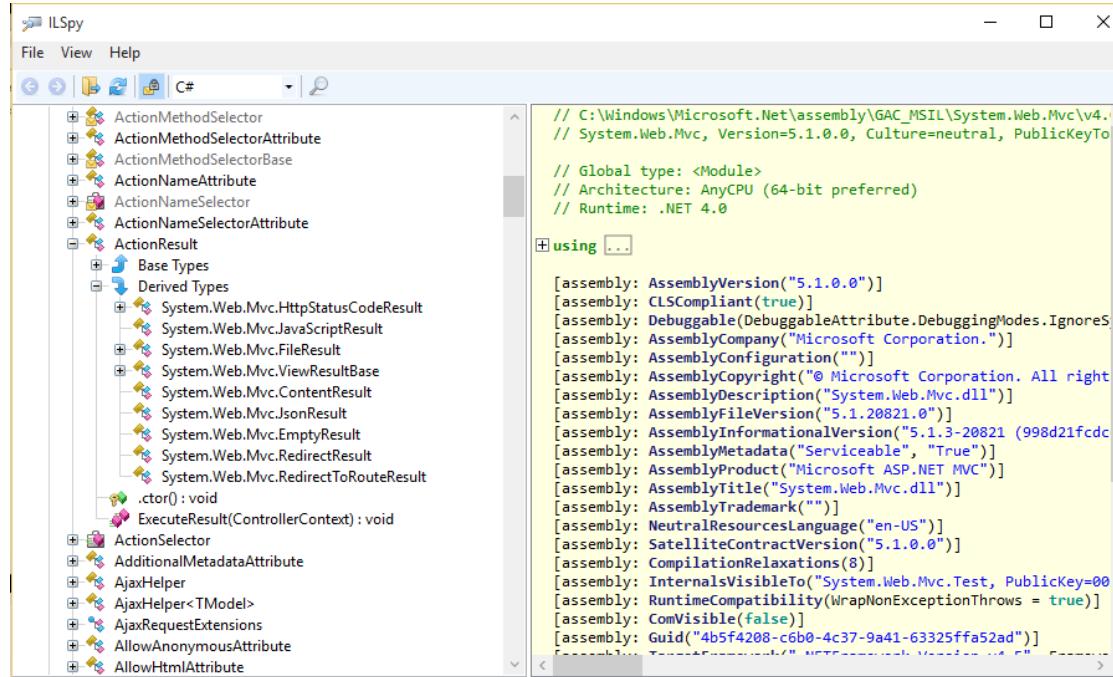
ILSPY

- An open source .NET assembly browser and decompiler



GAC – Global Assembly Cache





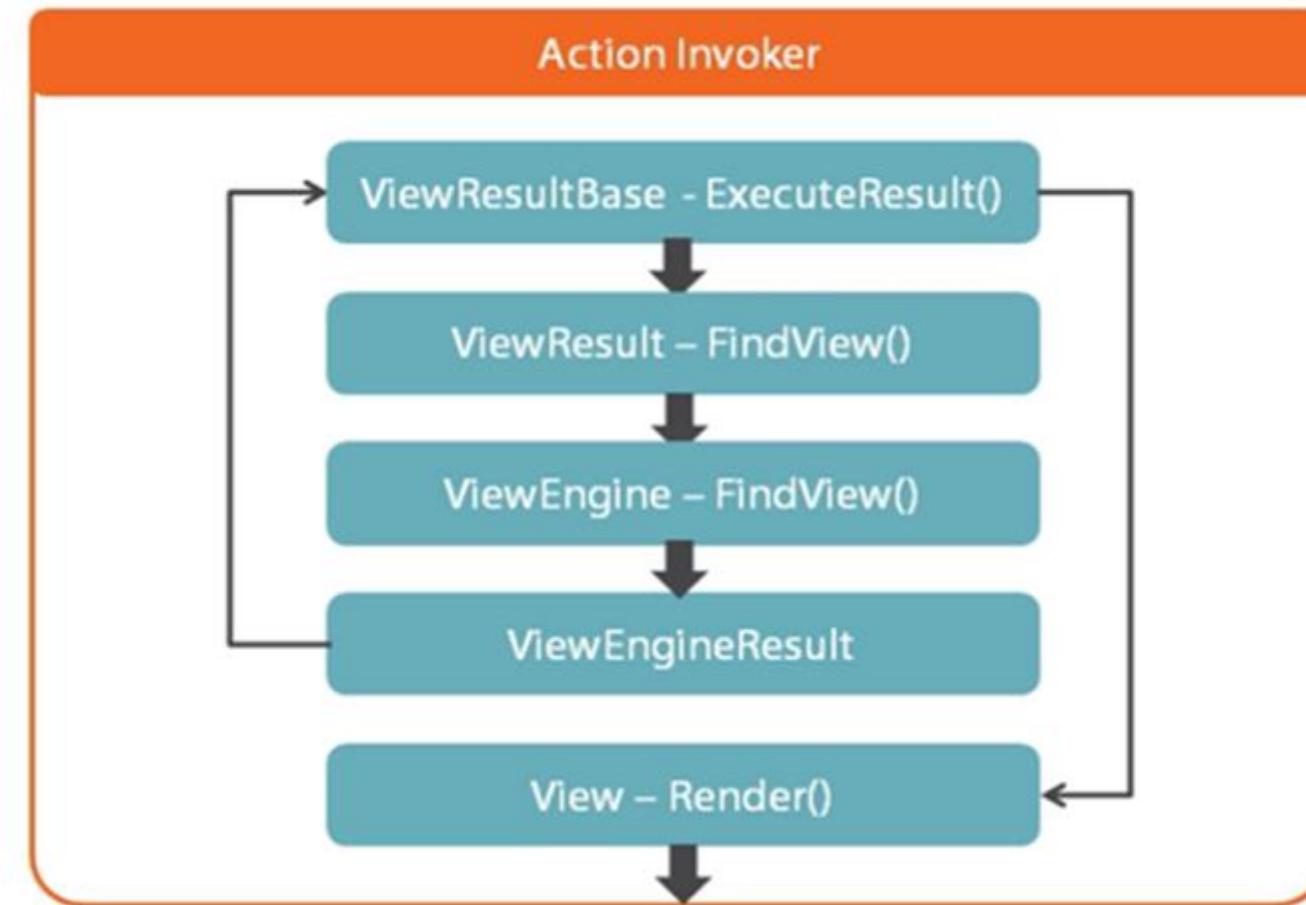
- Best Practice

- Return specific sub-types, but if different paths of the action method return different subtypes, then it is better to return an ActionResult Object

The View Result Execution Process

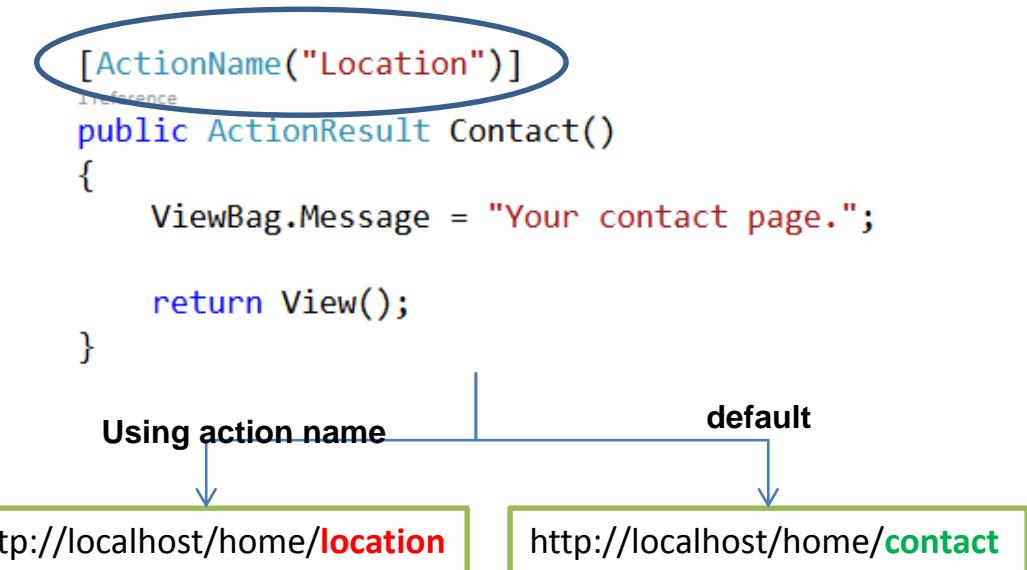
Please read terms and conditions of use

Original Series



Action Selectors

- Action selectors are **attributes** that can be applied *to action methods and are used to influence which action method gets invoked in response to a request.*
- ACTION NAME (ALIAS THE NAME OF ACTION METHOD)**
 - Used to **change the name to invoke an action method.**
- ACTION VERB**
 - used when we want to control the selection of the action method based on request type.



Why use the MVC AcceptVerbs attribute?

- Can be applied to action methods in a controller so that appropriate overloaded method is invoked for a given request.
- MVC will automatically **dispatch a request to appropriate action method based on the HTTP VERB.**
- Advantage of differentiating methods based on HTTP VERB is that the same URL can be used for multiple purposes (e.g. Display and edit).
 - Which is achieved with 2 separate URLs, but downside is that it makes bookmarking pages difficult.

```
[AcceptVerbs(HttpVerbs.Get)]
public ActionResult Edit(int id) {
    // code snipped
    // this is invoked when viewing the edit page
}

[AcceptVerbs(HttpVerbs.Post)]
public ActionResult Edit(int id, FormCollection formValues) {
    // code snipped
    // this is invoked when POSTing data to the edit page
}
```

Routes and Controllers

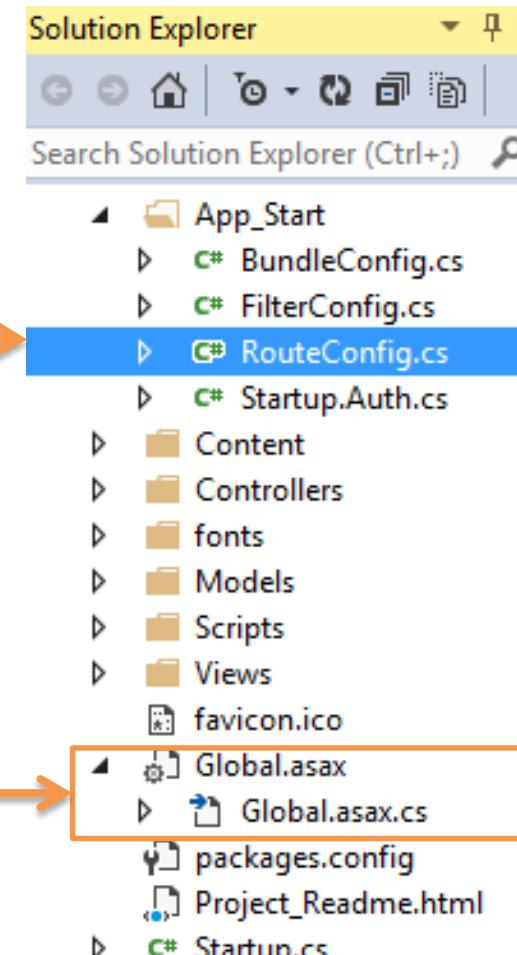
RouteData

```
public class RouteConfig
{
    public static void RegisterRoutes(RouteCollection routes)
    {
        routes.IgnoreRoute("{resource}.axd/{*pathInfo}");

        routes.MapRoute(
            name: "Default",
            url: "{controller}/{action}/{id}",
            defaults: new { controller = "Home", action = "Index", id =
                UrlParameter.Optional }
        );
    }
}
```

Insert this in Controller action method

```
ViewBag.PageBreadCrumbValue = string.Format("{0}>{1}>{2}",
    RouteData.Values["controller"],
    RouteData.Values["action"],
    RouteData.Values["id"]);
```



Define a New Route

```
routes.MapRoute(  
    name: "Default",  
    url: "{controller}/{action}/{id}",  
    defaults: new { controller = "Home", action = "Index", id =  
        UrlParameter.Optional }  
);
```

Action Filters

- Attributes can add pre/post processing to an action
- Or to an entire controller

```
[HandleError]
public ActionResult Index()
{
    return View();
}
```

```
[Authorize]
[HandleError]
public class HomeController:Controller
{
    //controller action methods go here
}
```

ACTION FILTERS

- An action filter is an **attribute** that you can **apply to individual controller action – or an entire controller – that modifies the way in which the action is executed.**
- **Users can create their own custom action filters**

```
[ActionName("Location")]
[Authorize(Roles="Admin")]
public ActionResult Contact()
{
    ViewBag.Message = "Your contact page.";

    return View();
}
```

Action Filter

Name	Description
OutputCache	Cache the output of the controller
ValidateInput	Turn Off request validation and allow dangerous input
Authorize	Restrict an action to authorized users or roles
ValidateAntiForgeryToken	Helps prevent cross site request forgeries
HandleError	Can specify a view to render in the event of an unhandled exception

ACTION FILTER

INDIVIDUAL Controller Action Method()

```
[ActionName("Location")]
[Authorize(Roles="Admin")]
1 reference
public ActionResult Contact()
{
    ViewBag.Message = "Your contact page.";

    return View();
}
```

GLOBAL

ENTIRE CONTROLLER

```
[Authorize(Roles="Admin")]
6 references
public class HomeController : Controller
{
```

ACTION FILTER TYPES

Please read terms and conditions of use

Original Series

ASP.NET MVC Framework supports four different types of filters:

If you want to control the order in [which filters of the same type are executed](#) then you can set a [filter's Order property](#).

Authorization filters	Implements the IAuthorizationFilter Attribute	used to implement authentication and authorization for controller actions.
Action Filters	Implements the IActionFilter attribute	contains logic that is executed before and after a controller action executes. You can use an action filter, for instance, to modify the view data that a controller action returns.
Result filters	Implements the IResultFilter attribute	Result filters contain logic that is executed before and after a view result is executed. For example, you might want to modify a view result right before the view is rendered to the browser.
Exception Filters	Implements the IExceptionFilter attribute	Exception filters are the last type of filter to run. You can use an exception filter to handle errors raised by either your controller actions or controller action results. You also can use exception filters to log errors.

Filters are executed in the order listed above. For example, authorization filters are always executed before action filters and exception filters are always executed after every other type of filter.

ACTION FILTER

Please read terms and conditions of use

Original Series

Name	Behaviour
Authorize	Actions are restricted to allow only authorized users.
ValidateInput	Validate all kinds of inputs
OutputCache	Caches controller's output
HandleError	User can specify custom views for handling different kind of exceptions
ValidateAntiForgeryToken	Prevent Cross Site Scripting (XSS)

Partial Views

- Similar to Web User Controls in ASP.NET Web Forms
- Reusable components
- Used to create HTML markups in separate files as a partial view
- Partial view enables you to define a view that will be rendered inside a parent view
- Generally Classified into two
 - Static
 - @Html.RenderPartial("_partialView")
 - @Html.Partial("PartialView")
 - Dynamic
 - @Html.RenderAction("PartialView")
 - @Html.Action("PartialView")

HTML.Partial

- Used to render the partial view in the view page
- Returns MVCHtmlString, which can be assigned to the variable and we are able to manipulate it if required or return it from a function
- Slower in loading when compared to the HTML.RenderPartial, since it is assigned to some variable and then used
- Used when displaying data is present in the corresponding view model.

HTML.RenderPartial

- Used to render the partial view in the view page
- Method returns void and the output will be directly written to the output stream
- Faster in loading as compared to @HTML.Partial as the output is written directly to the output stream
- Used when displaying data is present in the corresponding view model

HTML.Action

- Used to render the partial view in the view page
- Returns MVCHtmlString which can be assigned to the variable and manipulate it if required or return it from a function.
- Slower loading when compared to HTML.RenderAction.
- Child action method is required for rendering the partial view.

HTML.RenderAction

- Used to render the partial view in the view page
- Returns void and the output will be directly written to the output stream
- Faster loading when compared to HTML.Action as the output is directly written to the output stream
- Child action method is required for rendering the partial view.

Child Output Caching using Partial Views

Please read terms and conditions of use

- **OutputCache** can now apply to child actions using partial views

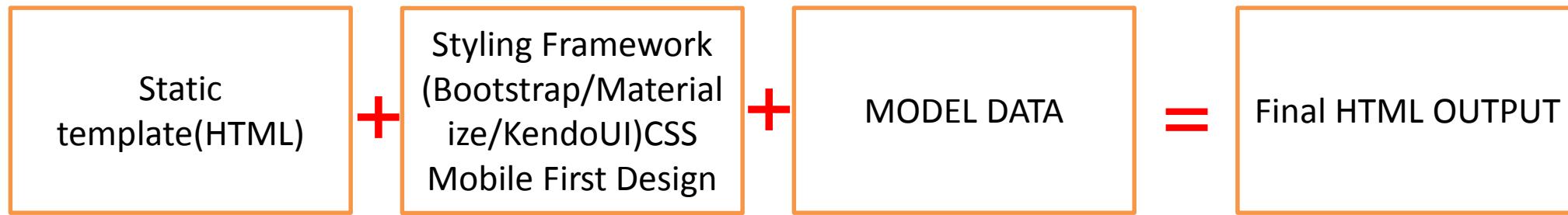
```
[OutputCache(Duration=40)]
1 reference | 0/0 passing
public ActionResult Index()
{
    var currenttime = DateTime.Now;
    return View(currenttime);
}
```

```
//partial view
[ChildActionOnly]
[OutputCache(Duration = 10)]
0 references
public PartialViewResult CurrentTime()
{
    var model = DateTime.Now;
    return PartialView(model);
}
```

Original Series

Summary

- Foundational knowledge about how MVC Works
 - MVC Controller
 - Action Methods and Action Parameters
 - Action Names and Action Verbs
 - Routing
 - Action Filters



SECTION -XIV

RAZOR VIEW ENGINE

Alternative to Razor View Engine are [NHaml](#)

Overview

- Razor Syntax
 - Transition between C# code and HTML code
- Layout
- HTML Helpers
- XSS (Cross Site Scripting) and CSRF (cross site request forgeries)
- Partial Views
- Introduced in ASP.NET MVC 2013.

Razor Templates



A General Purpose templating engine built upon [Microsoft's Razor parsing technology](#). The RazorEngine allows you to use Razor syntax to build robust templates

- Markup syntax for adding server-based code to web pages
- It has the power of traditional ASP.NET markup, but is easier to learn and easier to use.
- Razor supports C# and Visual Basic Programming Languages

Razor Goals

- Easy to use/easy to learn
- No ties to ASP.NET runtime
 - Unlike ASP.NET – lifecycle of ASPX files have dependency on ASP.NET runtime. While Razor does not have any dependency on ASP.NET runtime to execute its code.

Razor Expression

- Razor code blocks are enclosed in @{...}
- Inline expressions (variables and functions) start with @
- Code statements end with semicolon
- Variables are declared with the var keyword
- Strings are enclosed with quotation marks
- C# code is case sensitive
- C# files have the extension .cshtml

Razor code block

```
@{  
    ViewBag.Title = "BestReview";  
    Layout = "~/Views/Shared/_Layout.cshtml";  
}
```

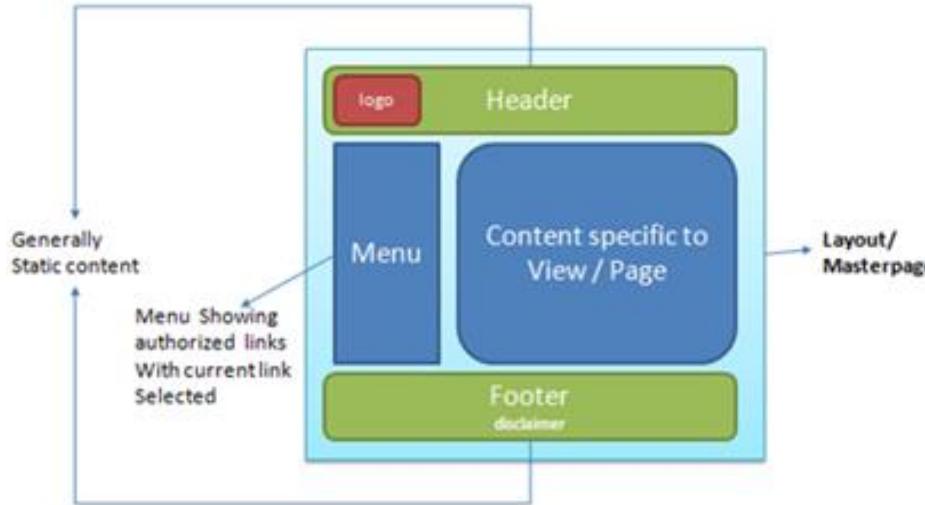
```
<!-- Inline expression or variable -->  
<p>The value of myMessage is: @myMessage</p>
```

```
<!-- Multi-statement block -->  
{@  
    var greeting = "Welcome to our site!";  
    var weekDay = DateTime.Now.DayOfWeek;  
    var greetingMessage = greeting + " Today is: " + weekDay;  
}  
<p>The greeting is: @greetingMessage</p>
```

ContentBlocks

Please read terms and conditions of use

Original Series



```
<html>
<body>
@RenderPage("header.cshtml")
<h1>Hello Web Pages</h1>
<p>This is a paragraph</p>
@RenderPage("footer.cshtml")
</body>
</html>
```

- With Web Pages you can use the `@RenderPage()` method to import content from separate files.
- Content block (from another file) can be imported anywhere in a web page, and can contain text, markup and code just like any regular web page
- Using common headers and footers as an example, this saves you a lot of work. You don't have to write the same content in every page, and when you change the header or footer files, the content is updated in all your pages.

Razor Syntax

Please read terms and conditions of use

Original Series

Syntax/Sample	Razor	Web Forms Equivalent (or remarks)
Code Block	@{ int x = 123; string y = "because."; }	<% int x = 123; string y = "because."; %>
Expression (Html Encoded)	@model.Message	<%: model.Message %>
Expression (Unencoded)	@Html.Raw(model.Message) 	<%= model.Message %>
Combining Text and markup	@foreach(var item in items) { @item.Prop }	<% foreach(var item in items) { %> <%: item.Prop %> <% } %>

Syntax/Sample	Razor	Web Forms Equivalent (or remarks)
Mixing code and Plain text	@if (foo) { <text>Plain Text</text> }	<% if (foo) { %> Plain Text <% } %>
Using block	@ using (Html.BeginForm()) { <input type="text" value="input here"> }	<% using (Html.BeginForm()) { %> <input type="text" value="input here"> <% } %>
Mixing code and plain text (alternate)	@if (foo) { @:Plain Text is @bar }	Same as above
Email Addresses	Hi philha@example.com	Razor recognizes basic email format and is smart enough not to treat the @ as a code delimiter

Syntax/Sample	Razor	Web Forms Equivalent (or remarks)
Explicit Expression	<code>ISBN@(isbnNumber)</code>	In this case, we need to be explicit about the expression by using parentheses.
Escaping the @ sign	<code>In Razor, you use the @@foo to display the value of foo</code>	@@ renders a single @ in the response.
Server side Comment	<code>@* This is a server side multiline comment *@</code>	<code><%-- This is a server side multiline comment --%></code>
Calling generic method	<code>@(MyClass.MyMethod<AType>())</code>	Use parentheses to be explicit about what the expression is.
Creating a Razor Delegate	<code>@{ Func<dynamic, object> b = @@item; } @b("Bold this")</code>	Generates a Func<T, HelperResult> that you can call from within Razor. See this blog post for more details.
Mixing expressions and text	Hello @title. @name.	Hello <%: title %>. <%: name %>.

NEW IN RAZOR v2.0/ASP.NET MVC 4

Please read terms and conditions of use

Original Series

Syntax/Sample	Razor	Web Forms Equivalent (or remarks)
Conditional attributes	<div class="@className"></div>	When className = null<div></div>When className = ""<div class=""></div>When className = "my-class"<div class="my-class"></div>
Conditional attributes with other literal values	<div class="@className foo bar"> </div>	When className = null<div class="foo bar"></div> <i>Notice the leading space in front of foo is removed.</i> When className = "my-class"<div class="my-class foo bar"> </div>
Conditional data-* attributes. <i>data-* attributes are always rendered.</i>	<div data-x="@xpos"></div>	When xpos = null or ""<div data-x=""></div>When xpos = "42"<div data-x="42"></div>
Boolean attributes	<input type="checkbox" checked="@isChecked" />	When isChecked = true<input type="checkbox" checked="checked" />When isChecked = false<input type="checkbox" />
URL Resolution with tilde	<script src("~/myscript.js")></script>	When the app is at /<script src="/myscript.js"></script>When running in a virtual application named MyApp<script src="/MyApp/myscript.js"></script>

Razor Layout

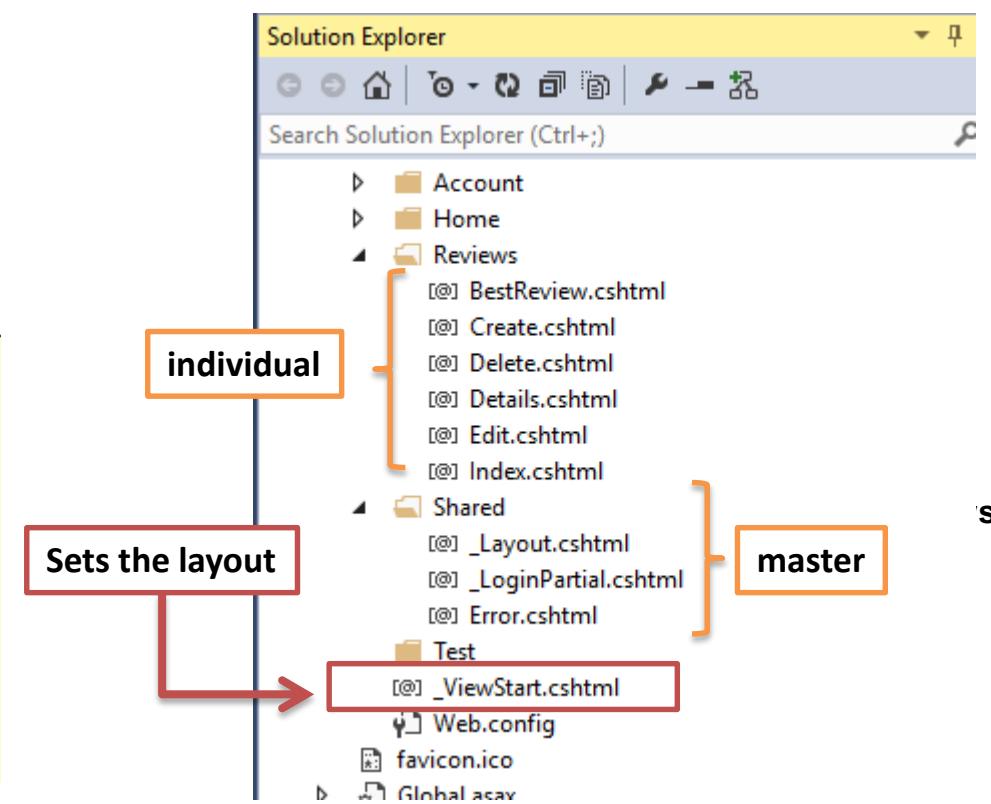
<http://razorcheatsheet.com/>

Please read terms and conditions of use

- Layout views are “master pages” for razor
- Use inherited method to specify content areas
 - RenderBody
 - RenderSection
- Common UI Structure for application

```
<!DOCTYPE html>
<html>
<head>
    <title>@ViewBag.Title</title>
    <script src="@Url.Content("~/Scripts/jquery-1.4.4.min.js")"
        type="text/javascript"></script>
</head>

<body>
    @RenderBody()
</body>
</html>
```



Razor Layout

```
<div id="logindisplay">
    @Html.Partial("_LogOnPartial")
</div>

<div id="menucontainer">

    <ul id="menu">
        <li>@Html.ActionLink("Home", "Index", "Home")</li>
        <li>@Html.ActionLink("About", "About", "Home")</li>
    </ul>

</div>
</div>

<div id="main">
    @RenderBody()
    <div id="footer"> I
        @RenderSection("Footer", true);
    </div>
</div>
</div>
</body>
</html>
```

HTML Helper

- HTML Helper extension method generates html elements based on **model properties**
- **Html.TextBox()** is loosely type method while **Html.TextBoxFor()** is strongly type (generic) extension method
- **It is advisable to use “For” extension methods for compile time type checking e.g. TextBoxFor, CheckBoxFor etc..**

HTML Helpers

- HTML Helpers are used to modify HTML output (easy to create small blocks of HTML)
- HTML is a property of the `ViewPage` base class
 - Create input
 - Create links
 - Create forms

HTML.ActionLink

```
Html.ActionLink(article.Title,
    "Item", // <-- ActionMethod
    "Login", // <-- Controller Name.
    new { article.ArticleID }, // <-- Route arguments.
    null // <-- htmlArguments .. which are none. You need this value
        // otherwise you call the WRONG method ...
        // (refer to comments, below).
)
```

```
routes.MapRoute(
    "Default", // Route name
    "{controller}/{action}/{id}", // URL with parameters
    new { controller = "Home", action = "Index", id = "" } // Parameter defaults
);
```

HTML.ActionLink()

- The easiest way to render an HTML link in is to use the **HTML.ActionLink()** helper.
- With MVC, the **Html.ActionLink()** does not link to a view. It creates a link to a controller action.

Property	Description
@Html.ActionLink()	
@Html.linkLabel	
@Html.action()	
@Html.AntiForgeryToken()	
@Html.RouteCollection()	

HTML Helper : FORM ELEMENTS

- HTML helpers are available for every kind of form control
- HTML label elements use descriptive text to form control and provide usability improvements
- Each helper method provides a shorthand way to render valid HTML for the specific control

Helper

```
Html.CheckBox  
Html.DropDownList  
Html.Hidden  
Html.Label  
Html.ListBox  
Html.Password  
Html.Radio  
Html.TextArea  
Html.TextBox
```

HTML Element

```
<input type="checkbox" />  
<select></select>  
<input type="hidden" />  
<label for="" />  
<select></select> or <select multiple></select>  
<input type="password" />  
<input type="radio" />  
<textarea></textarea>  
<input type="text" />
```

```
@using ( Html.BeginForm() )  
{  
    <fieldset>  
        <legend>Product</legend>  
  
        <div class="editor-label">  
            @Html.LabelFor(model => model.Name)  
        </div>  
        <div class="editor-field">  
            @Html.TextBoxFor(model => model.Name)  
        </div>  
  
        <div class="editor-label">  
            <div style="float: left;">  
                @Html.CheckBoxFor(model => model.Featured)  
            </div>  
            @Html.LabelFor(model => model.Featured)  
        </div>  
  
        <!-- Price, Featured, ... -->  
    </fieldset>  
}
```

Built-in ASP.NET MVC Framework: HTML HELPERS

Please read terms and conditions of use

Original Series

HTML Helper	Generates HTML Controls
Html.BeginForm	Form element
Html.EndForm	
Html.TextBox	Textbox
Html.TextArea	Text area
Html.Password	Password TextBox
Html.Hidden	Hidden field
Html.CheckBox	Checkbox
Html.RadioButton	Radio button
Html.DropDownList	Dropdownbox, combobox
Html.ListBox	Mult-select list box
Html.Display	HTML text
Html.Label	Label

HTML HELPERS FOR STRONGLY TYPED VIEWS

Please read terms and conditions of use

Original Series

HTML Helper	Generates Html Control
Html.TextBoxFor	Textbox
Html.TextAreaFor	TextArea
Html.PasswordFor	Password textbox
Html.HiddenFor	Hidden field
Html.CheckBoxFor	Checkbox
Html.RadioButtonFor	Radiobutton
Html.DropDownListFor	Dropdown, combobox
Html.ListBoxFor	Multiselect list box
Html.DisplayFor	Html text
Html.LabelFor	Label
Html.EditorFor	Generate Html controls based on data type of specified model property

```
@using (Html.BeginForm()) {
    @Html.ValidationSummary(true)
    <div class="editor-label">
        @Html.LabelFor(model => model.FirstName)
    </div>
    <div class="editor-field">
        @Html.EditorFor(model => model.FirstName)
        @Html.ValidationMessageFor(model => model.FirstName)
    </div>
```

TryUpdateModel()

- **TryUpdateModel()** allows you to bind parameters to your model inside your action.
- useful if you want to load your model from a database then update it based on user input rather than taking the entire model from user input.
- use this method to update the model that backs a particular view via the given controller.

```
public class Student {  
    public string studentID { get; set; }  
}  
  
// ... in the controller  
public ActionResult Save() {  
    var myStudent = new Student();  
    TryUpdateModel(myStudent);  
}
```

Custom Helpers

- can create custom HTML Helpers that you can use within your MVC views.
- By taking advantage of HTML Helpers, you can reduce the amount of tedious typing of HTML tags that you must perform to create a standard HTML page.

We need a custom helper for image tag as shown below

```
→ 
```

Custom Html helper tag in the view that generates above html snippet:

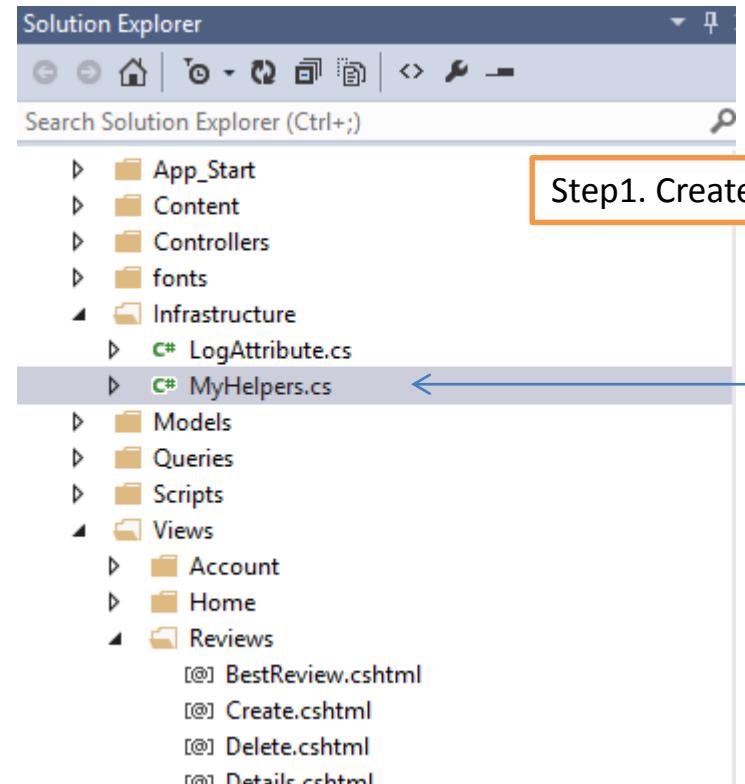
```
@Html.Image(item.Restaurant.ImageUrl, item.Restaurant.Name)
```

Introduced in C# 3.0 Extension Methods to create custom helpers

Steps to Create a Custom Helper

Please read terms and conditions of use

Original Series



Step1. Create a Helpers Class

Step2: Write the builder function

```
namespace ZomatoreviewApp.Infrastructure
{
    public class MyHelpers
    {
        //html required output
        //
        //custom helper
        //@html.Image(@item.Restaurant.ImageUrl, @item.Restaurant.Name)
        public static MvcHtmlString Image(this
            System.Web.WebPages.Html.HtmlHelper helper,
            string src, string altText)
        {
            var builder = new TagBuilder("img");
            builder.MergeAttribute("src", src);
            builder.MergeAttribute("alt", altText);
            return MvcHtmlString.Create(builder.ToString
                (TagRenderMode.SelfClosing));
        }
    }
}
```

Step3: include the namespace in the views

@ using zomatoreviewapp.Infrastructure (locally)

Step4: Use Html.Image() in View

@Html.Image(item.Restaurant.ImageUrl, item.Restaurant.Name)

Registering Custom Helper in NameSpace Globally

Please read terms and conditions of use

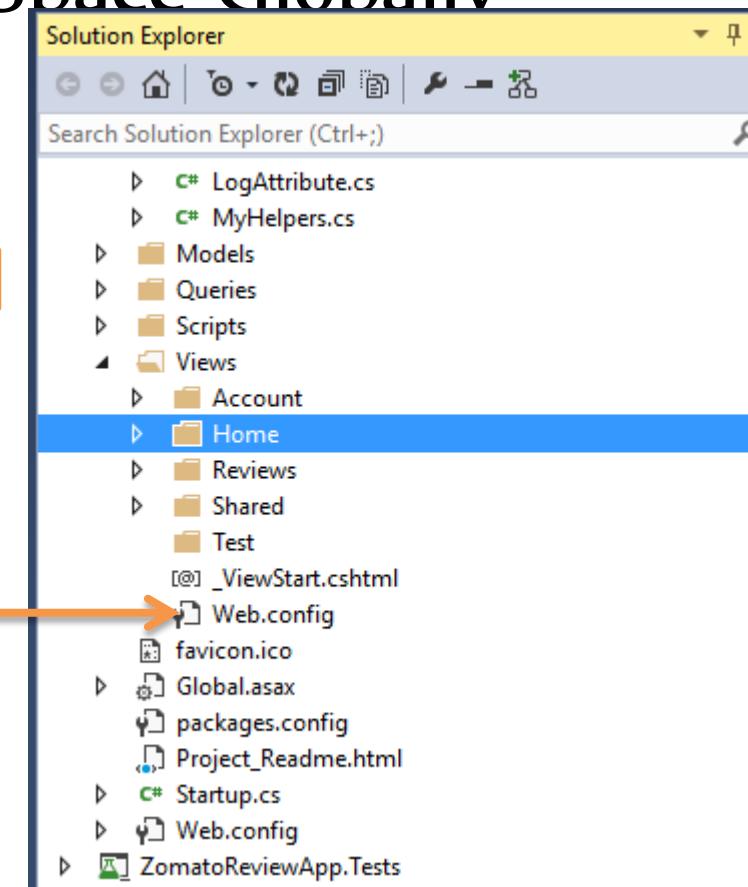
Original Series

**Now you can use the
@html.Image()
Custom helper globally in all the views**

1. Open Web.config

2.add

```
<system.web.webPages.razor>
  <host factoryType="System.Web.Mvc.MvcWebRazorHostFactory", System.Web.Mvc,
  <pages pageBaseType="System.Web.Mvc.WebViewPage">
    <namespaces>
      <add namespace="System.Web.Mvc" />
      <add namespace="System.Web.Mvc.Ajax" />
      <add namespace="System.Web.Mvc.Html" />
      <add namespace="System.Web.Optimization"/>
      <add namespace="System.Web.Routing" />
      <add namespace="ZomatoReviewApp" />
      <add namespace="ZomatoReviewApp.Infrastructure" />
    </namespaces>
  </pages>
</system.web.webPages.razor>
```



<http://www.asp.net/mvc/overview/older-versions-1/views/creating-custom-html-helpers-cs>

© Syed Awase 2015-16 - ASP.Net MVC Ground Up

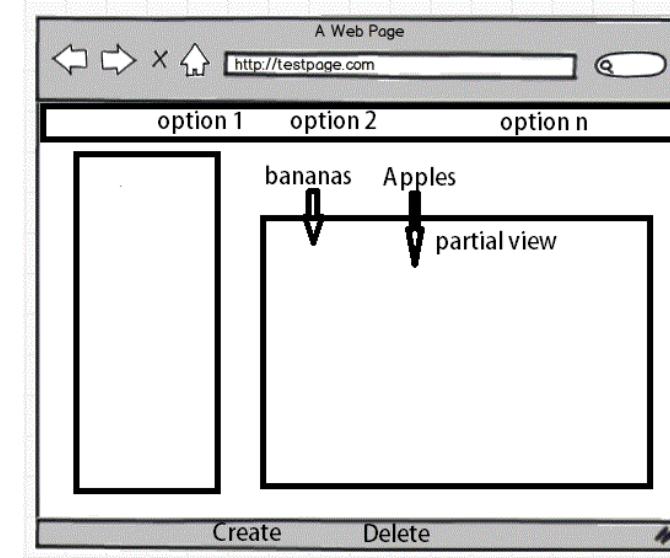
134

Partial Views

Please read terms and conditions of use

- Allows users to put html and C# code into file for code reusability.
- A partial view enables you to define a view that will be rendered inside a parent view.
- By using partial view we can render a view inside a parental view and to create reusable content in the project

```
<div id="main">
    @RenderBody()
    <div id="footer">
        @Html.Action("BestReview", "Reviews")
        @RenderSection("Footer", false)
    </div>
</div>
```



```
<h2>The Latest Reviews</h2>

@foreach (var item in Model)
{
    @Html.Partial("_Review", item)
}
<p>
    @Html.ActionLink("Create New", "Create")
</p>

@section Footer {
    <p>This is the footer</p>
}
```

Original Series

Security

- **Encoding**
 - Helps to avoid XSS(cross site scripting) attacks
 - Not encoding user input makes you particularly vulnerable

- **Html.AntiForgeryToken**
 - Helps to avoid CSRF attacks
 - Requires a ValidateAntiForgeryToken attribute on controller action
 - Valid only for POST operators
 - Cross Site Request forgery is a type of a hack where the hacker exploits the trust of a website on the user.
 - the site trusts the user (because they have authenticated themselves) and accepts data that turns out to be malicious.

Summary

- Razor Syntax – Implicit and Explicit expressions
- HTML Helpers and Creating Custom HTML Helpers
- Razor Layout
- Partial Views
- Introduced the concepts of XSS and CSRF

SECTION -XIV

WORKING WITH DATA IN ASP.NET MVC USING ENTITY FRAME WORK DB FIRST

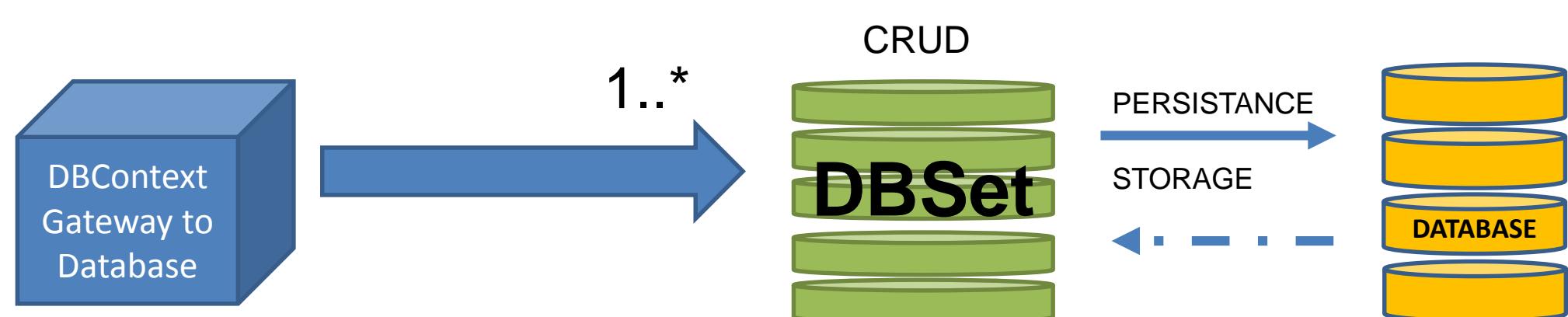
Object Relational Mapping Landscape

- ADO.NET
- LINQ TO SQL
- Entity Framework v1.0
- nHibernate
- Entity Framework 4
- Entity Framework 4.1 : DbContext
- Entity Framework 5
- Entity Framework 6
- Entity Framework 7 (Complete Rewrite)
- Dapper
- ORMLite
- PetaPocos

Entity Framework

Please read terms and conditions of use

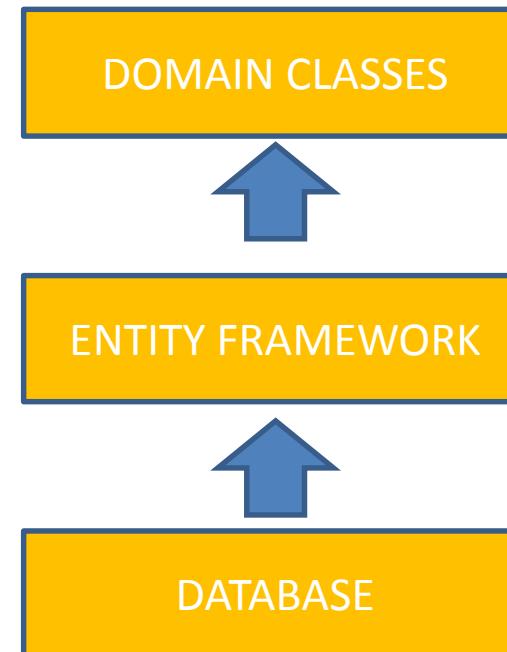
- VERSIONS
- EF 4.0
 - EF 5.0
 - EF 6.0
 - EF CORE



Approaches

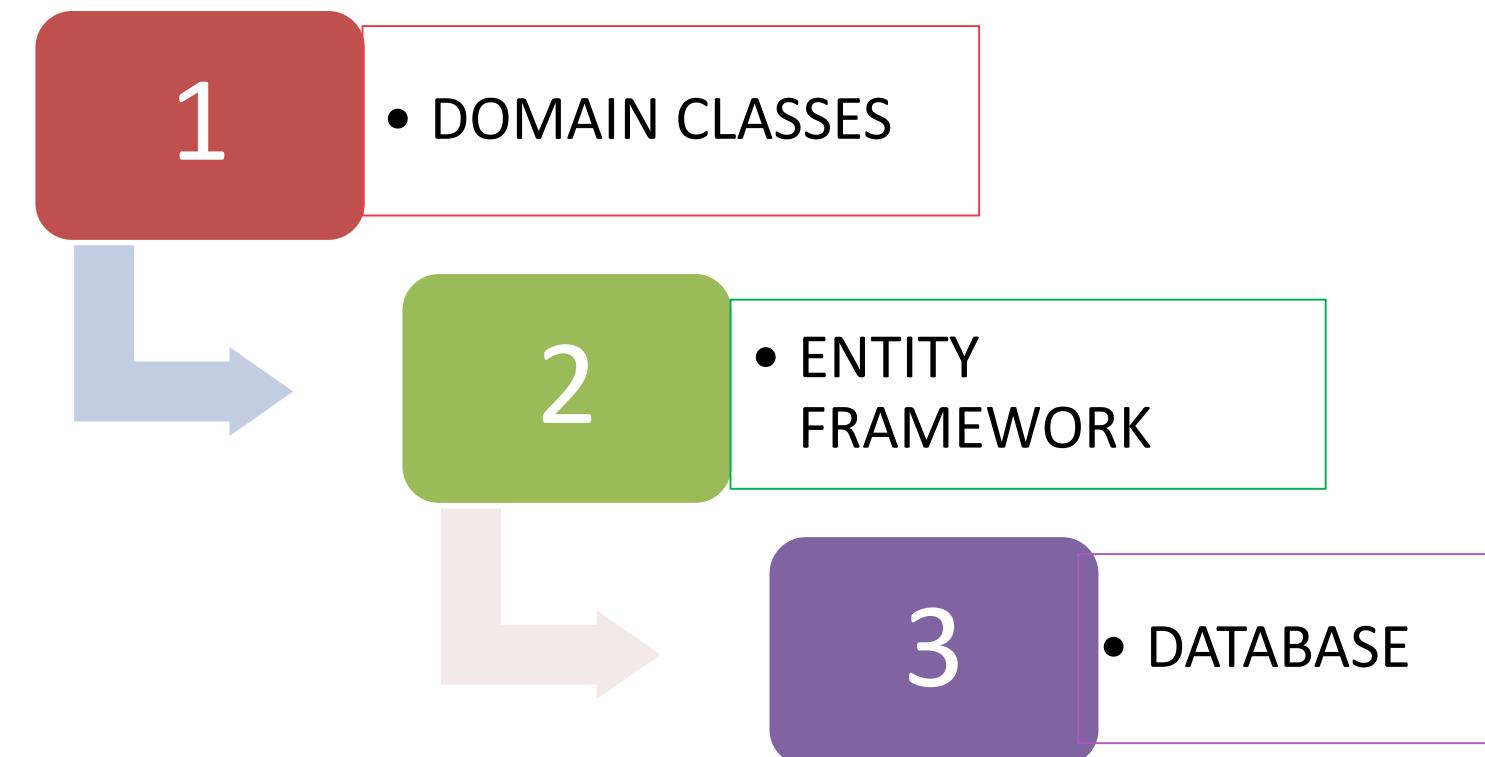
DBFIRST APPROACH

DBA FRIENDLY



CODE FIRST APPROACH

DEVELOPER FRIENDLY



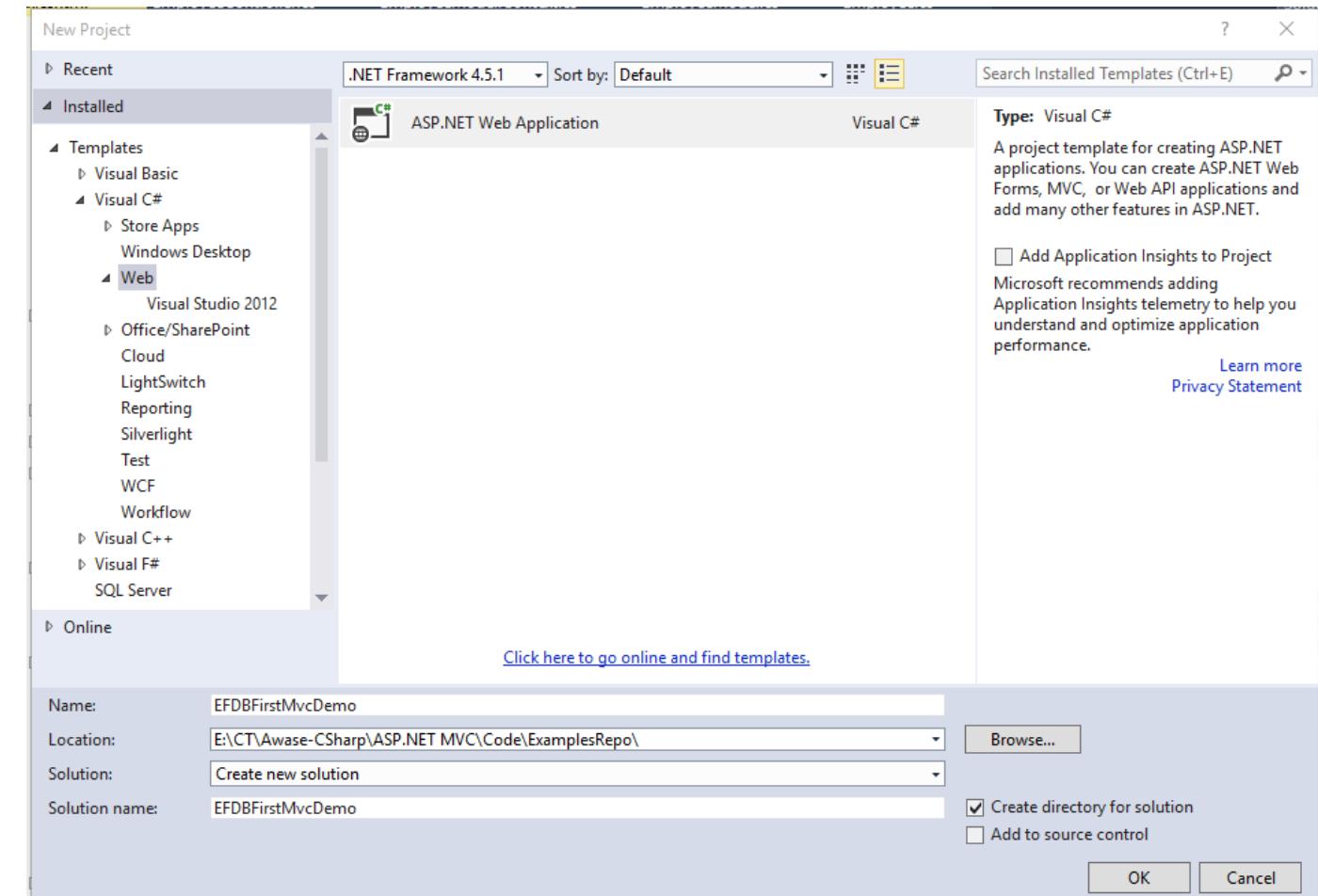
Please read terms and conditions of use

Original Series

1. Create ASP.NET Web Application

Please read terms and conditions of use

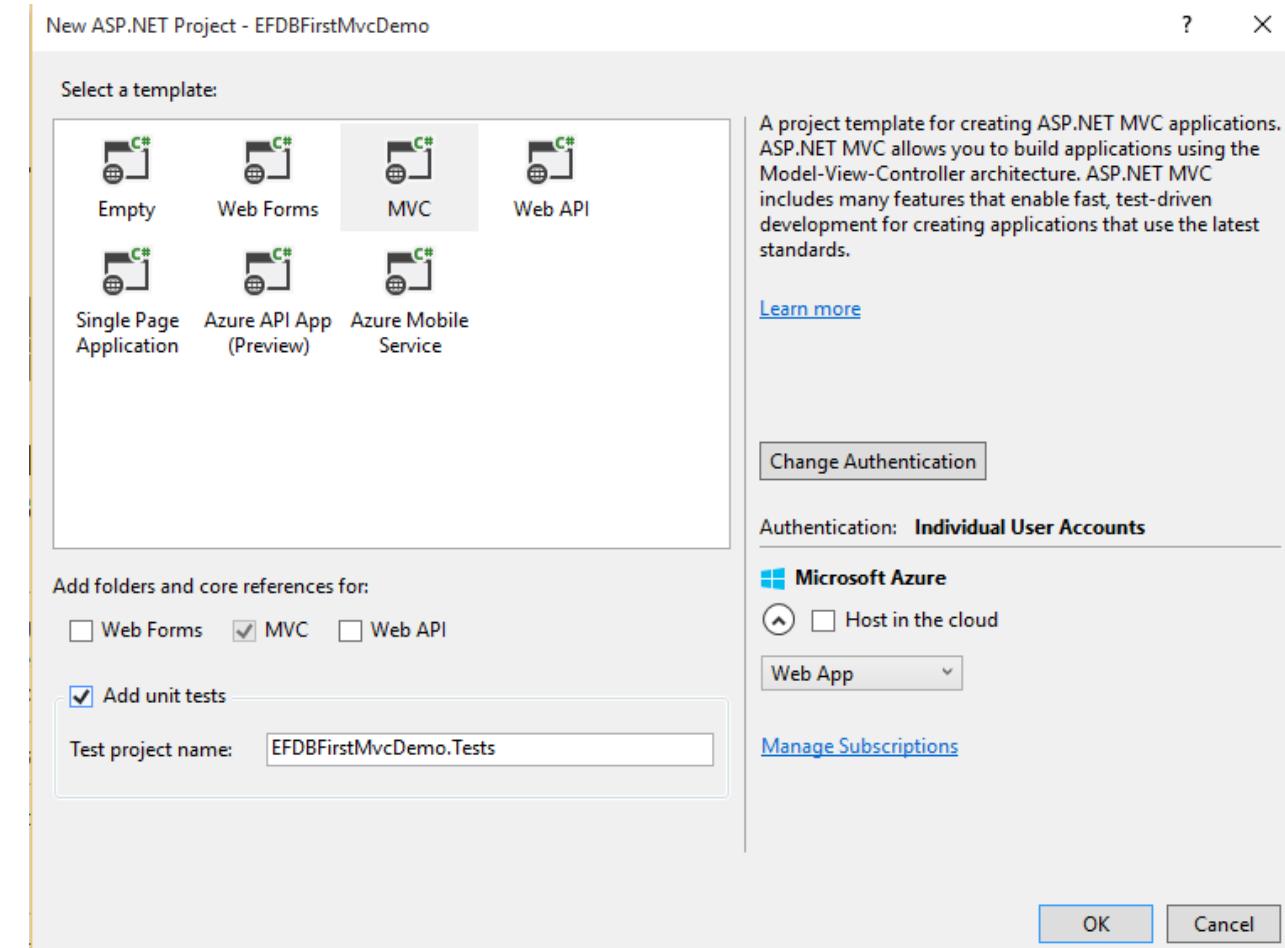
Original Series



2. Select MVC Web Application with Unit Testing

Please read terms and conditions of use

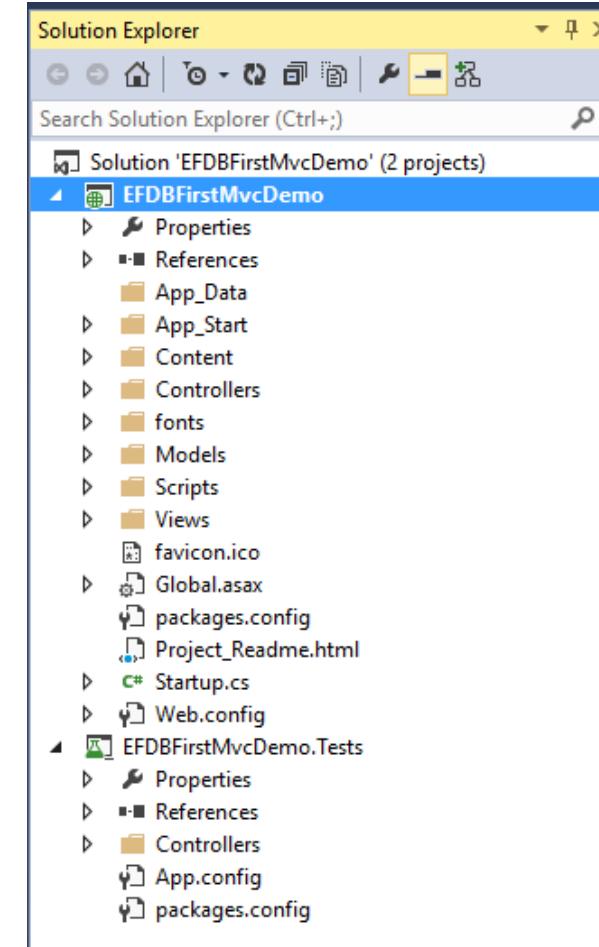
Original Series



3. Scaffolding MVC Application

Please read terms and conditions of use

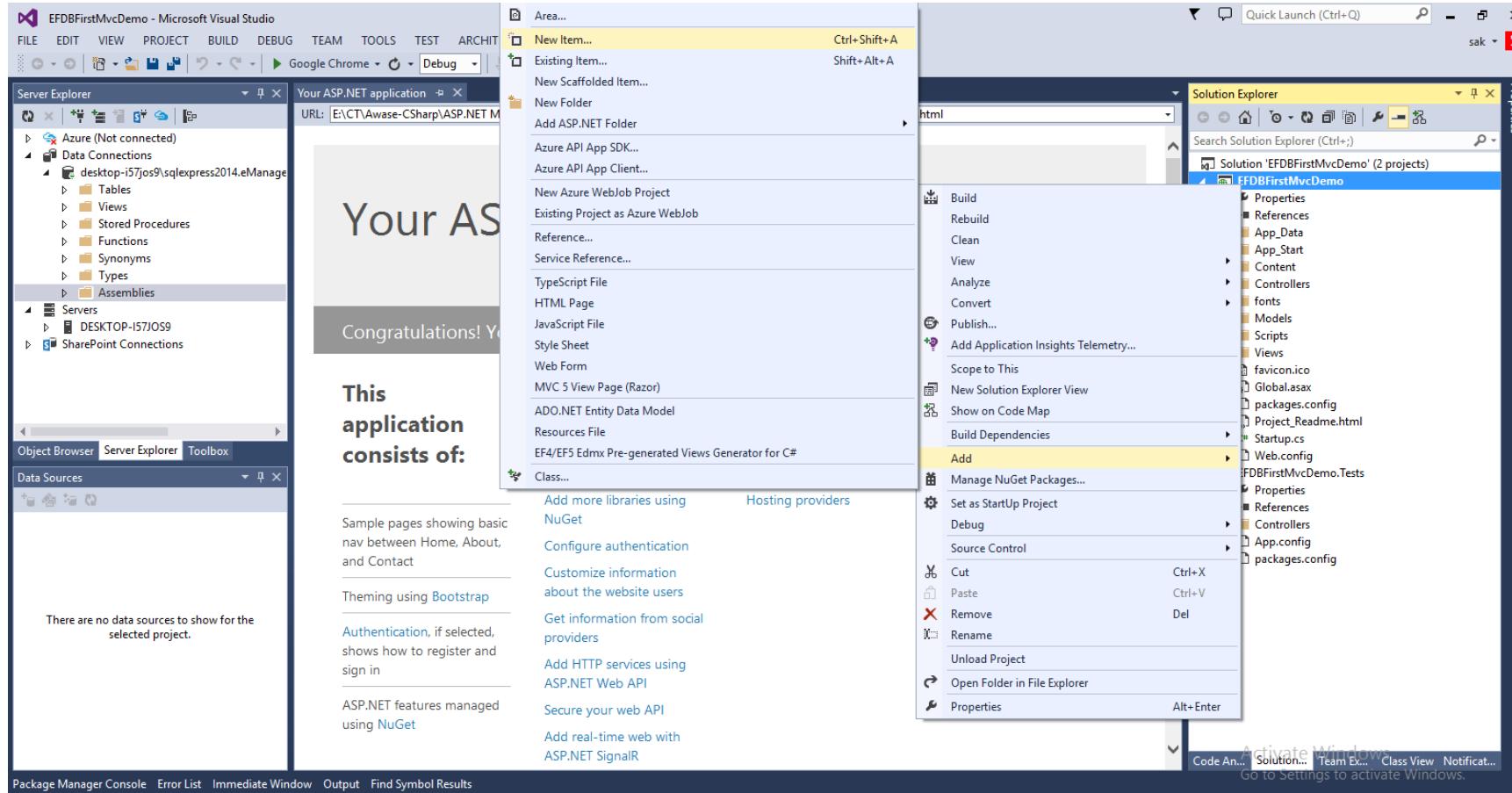
Original Series



4. Add -> New Item

Please read terms and conditions of use

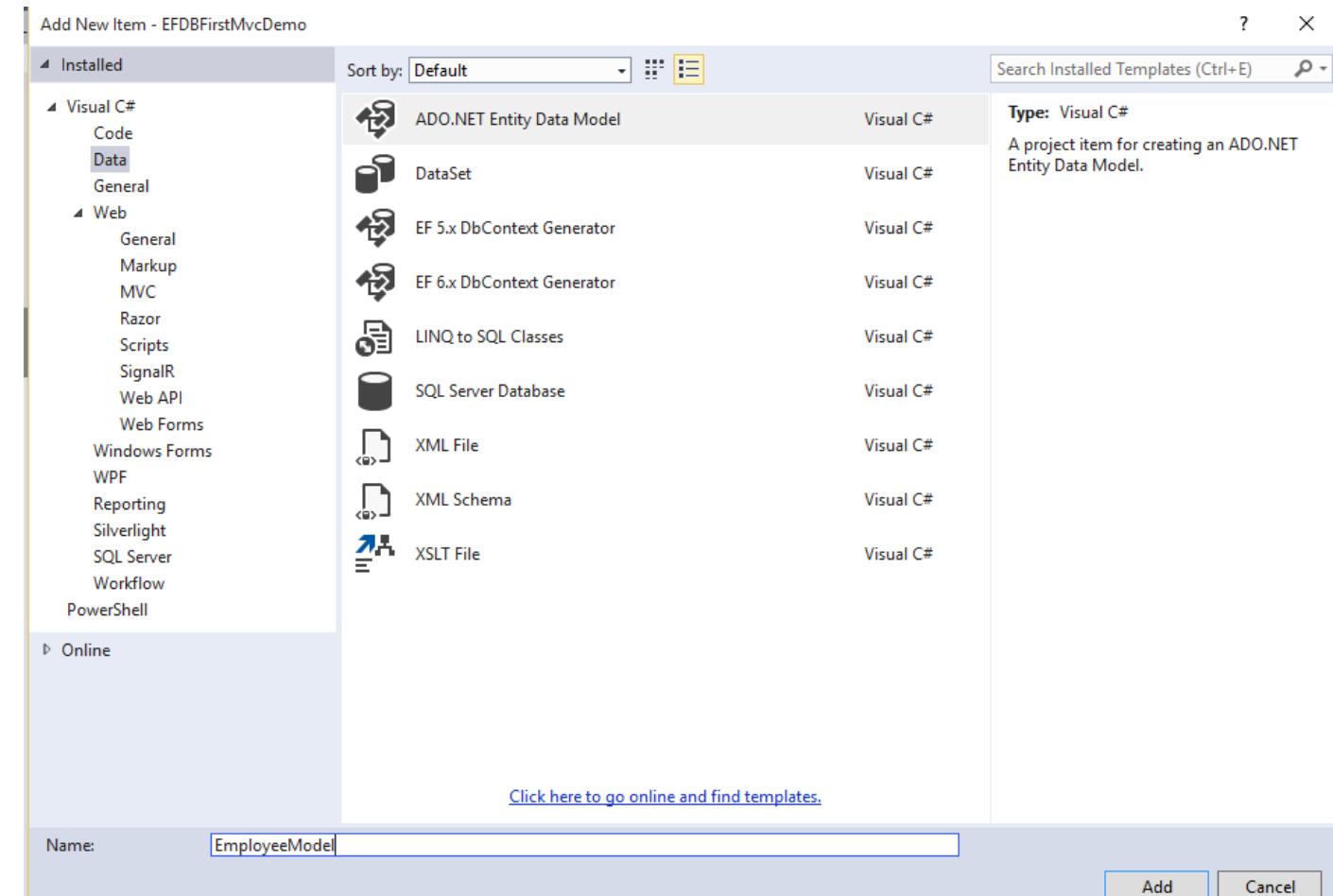
Original Series



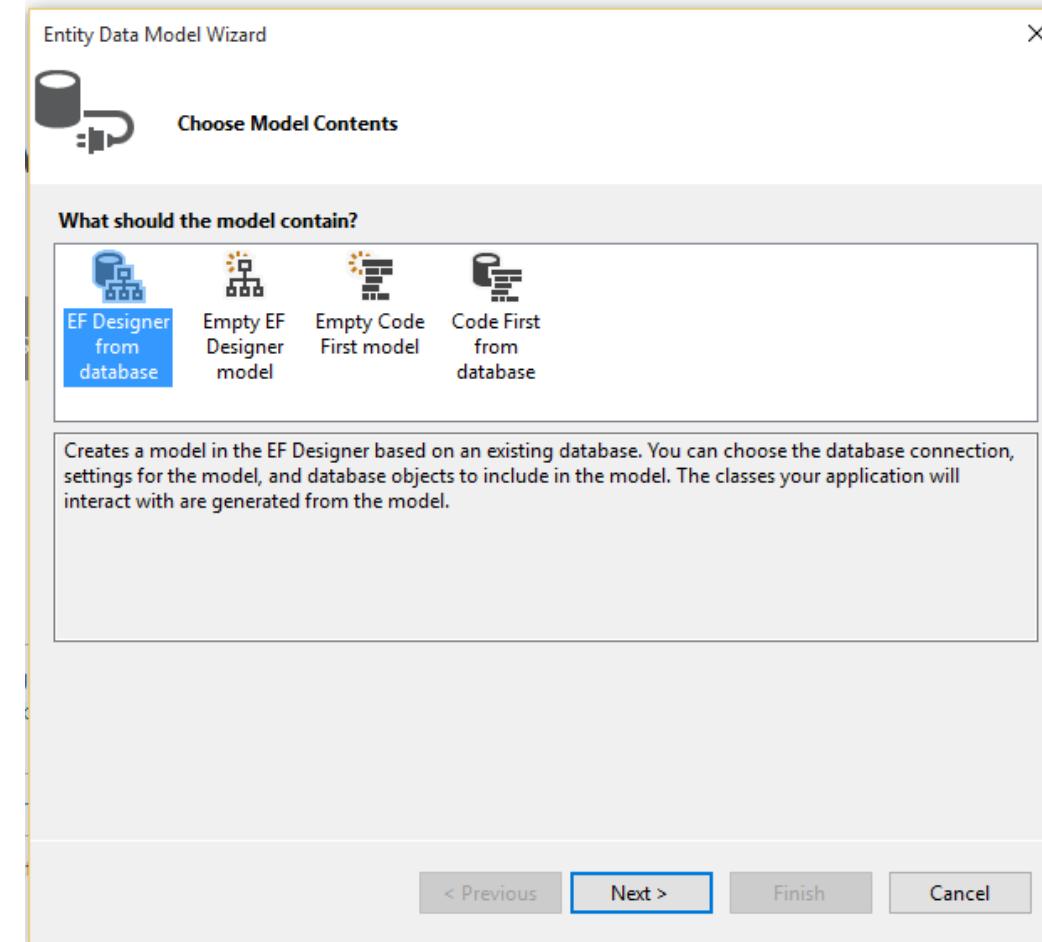
5. Create ADO.NET ENTITY DATA MODEL

Please read terms and conditions of use

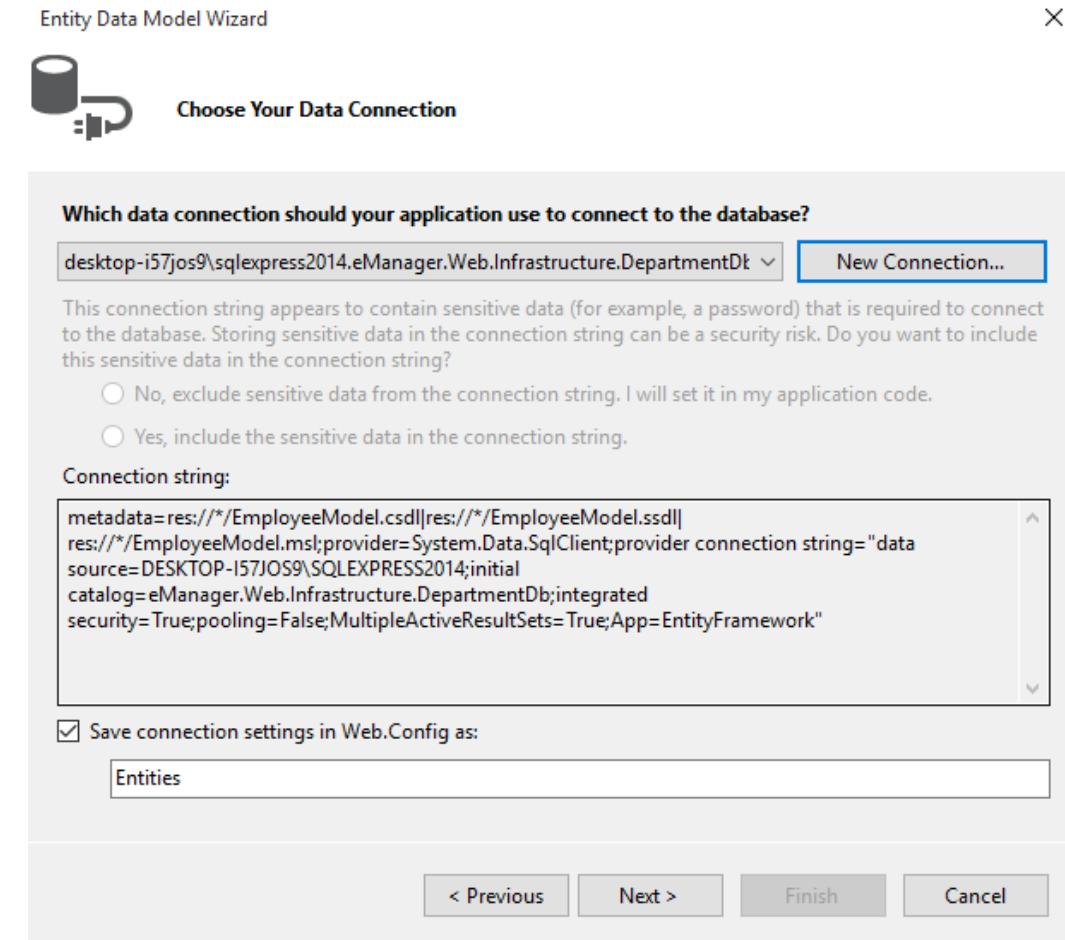
Original Series



6.EF Designer from Database



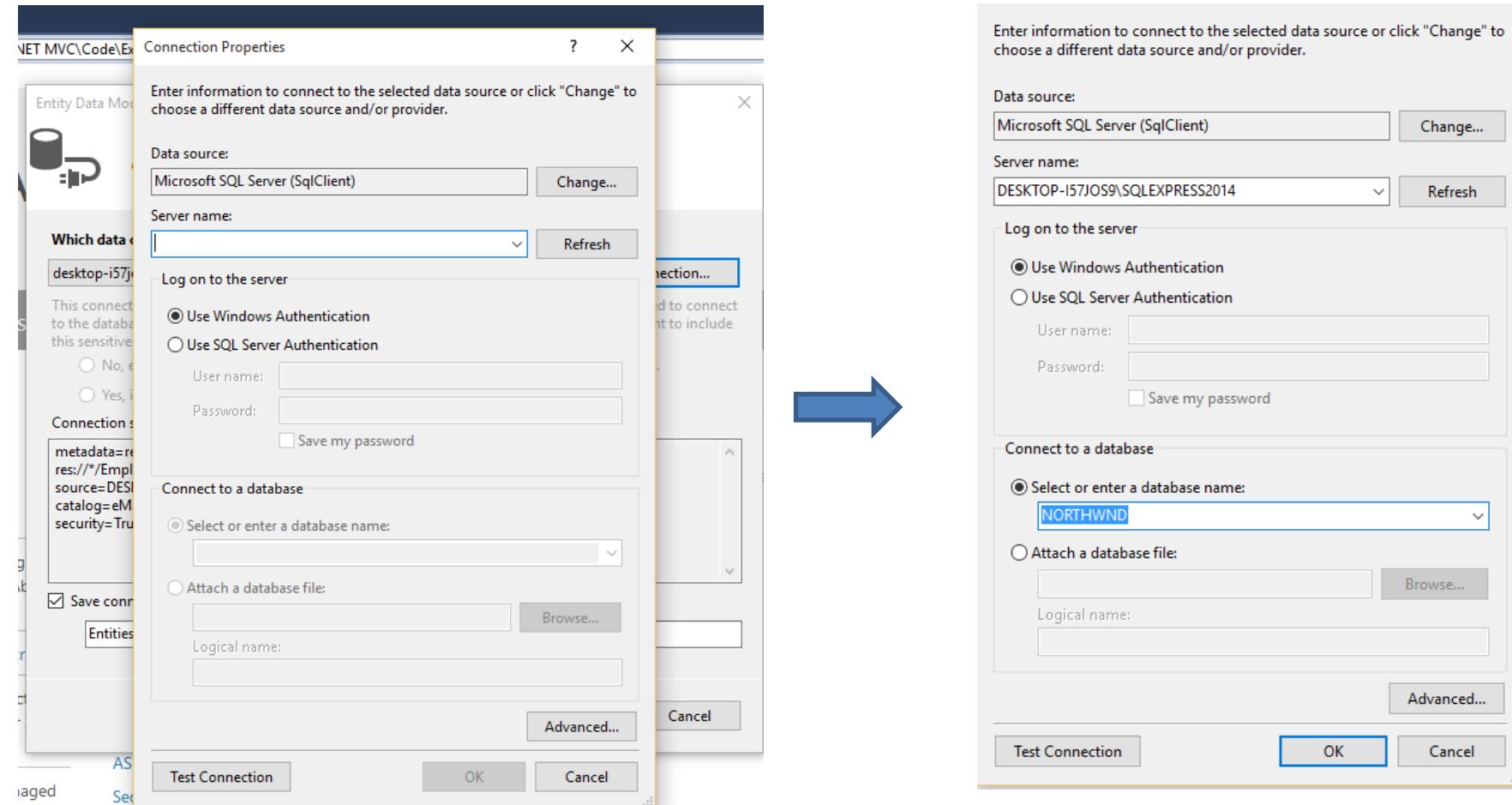
7. DATA CONNECTION



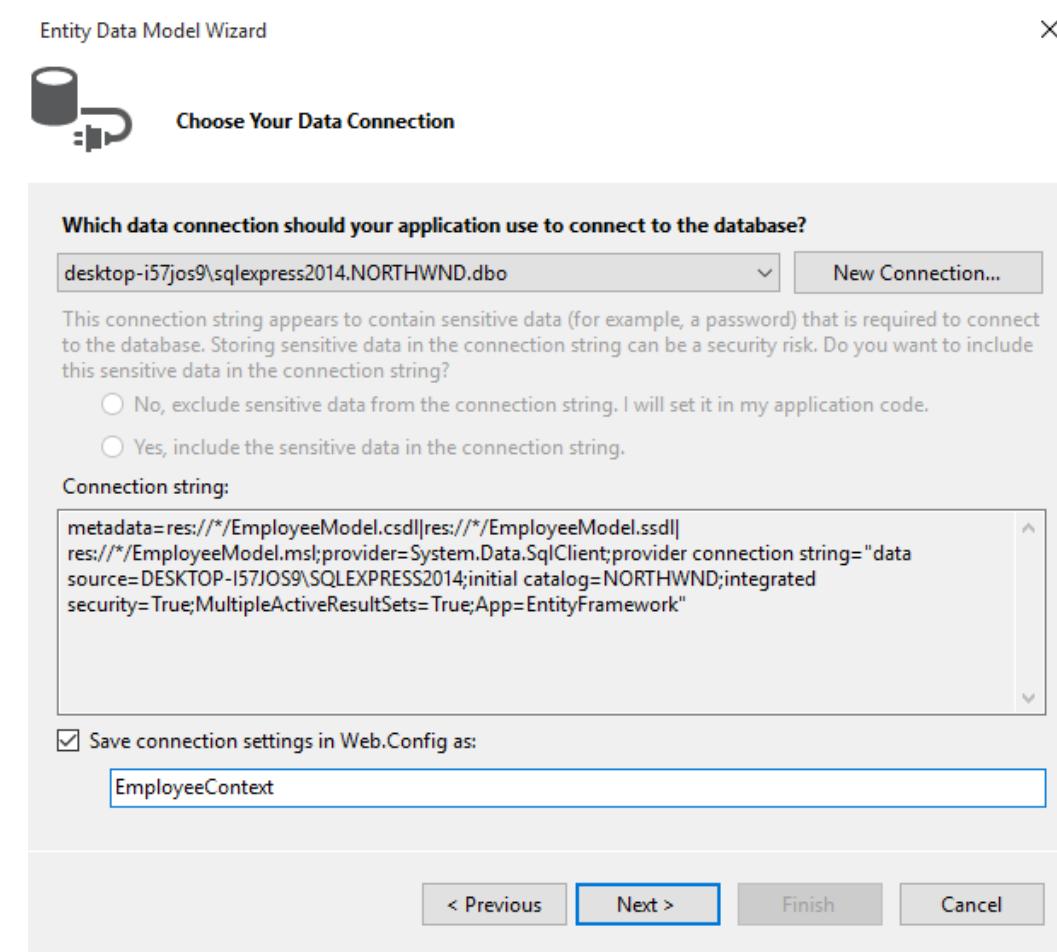
8. DEFINE CONNECTION PROPERTIES

Please read terms and conditions of use

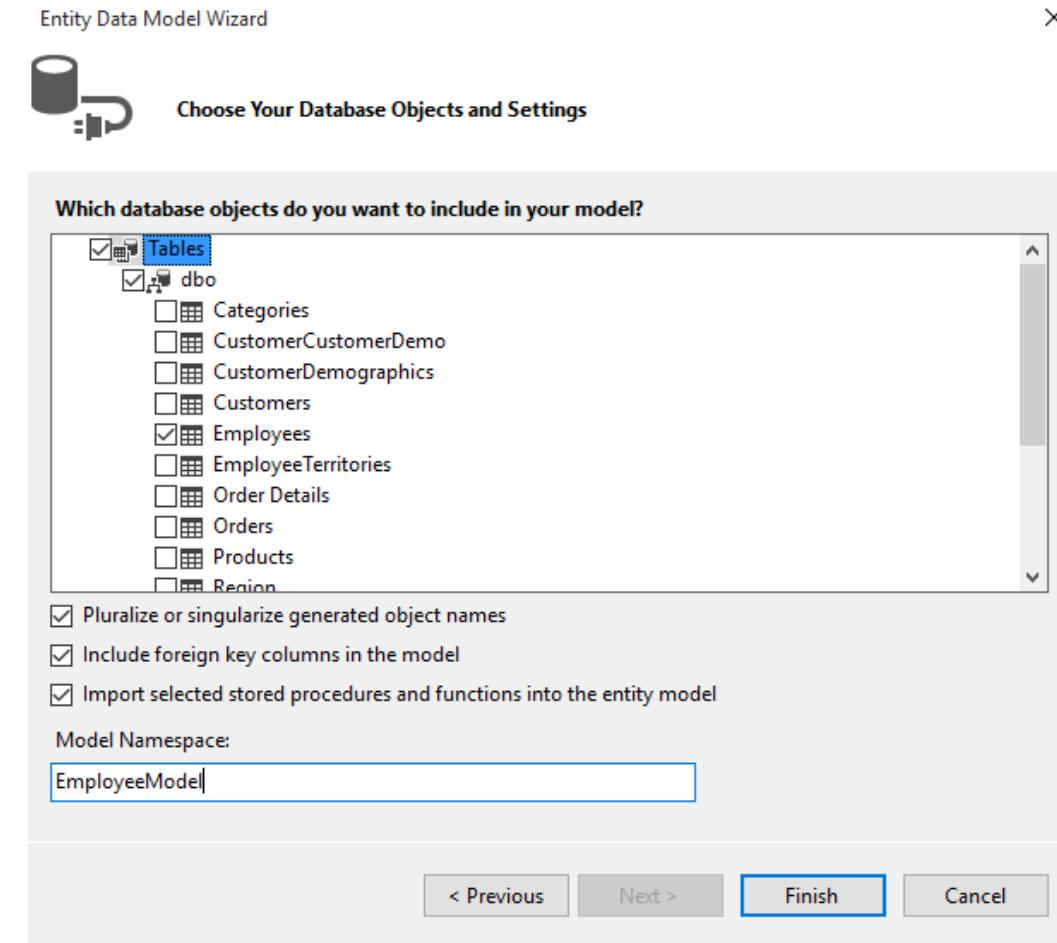
Original Series



9. Post Connection Setting



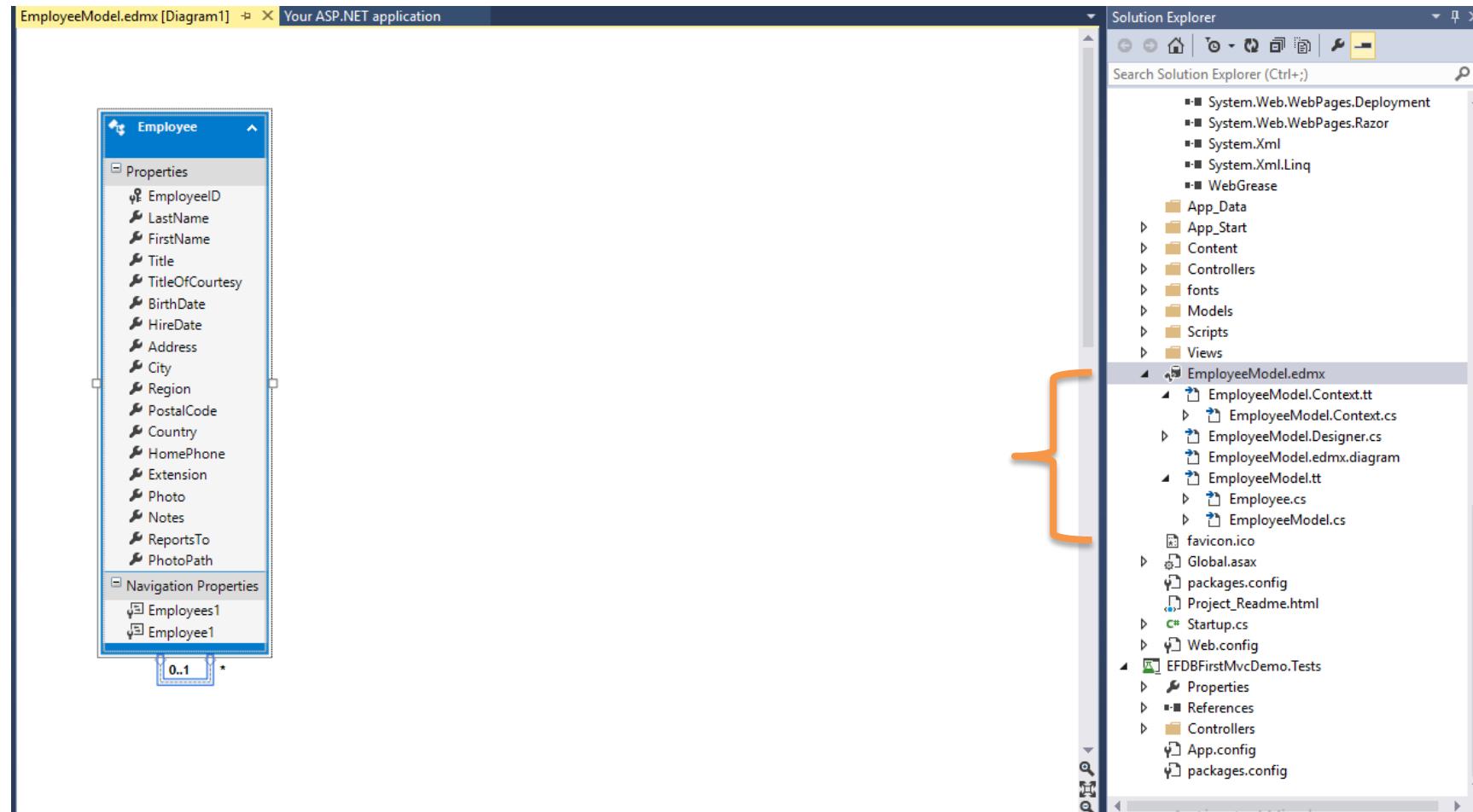
10. Choose Database Objects and Settings



11. Successful Creation of Entity Model

Please read terms and conditions of use

Original Series



11. Successful Creation of Entity Model

Please read terms and conditions of use

Original Series

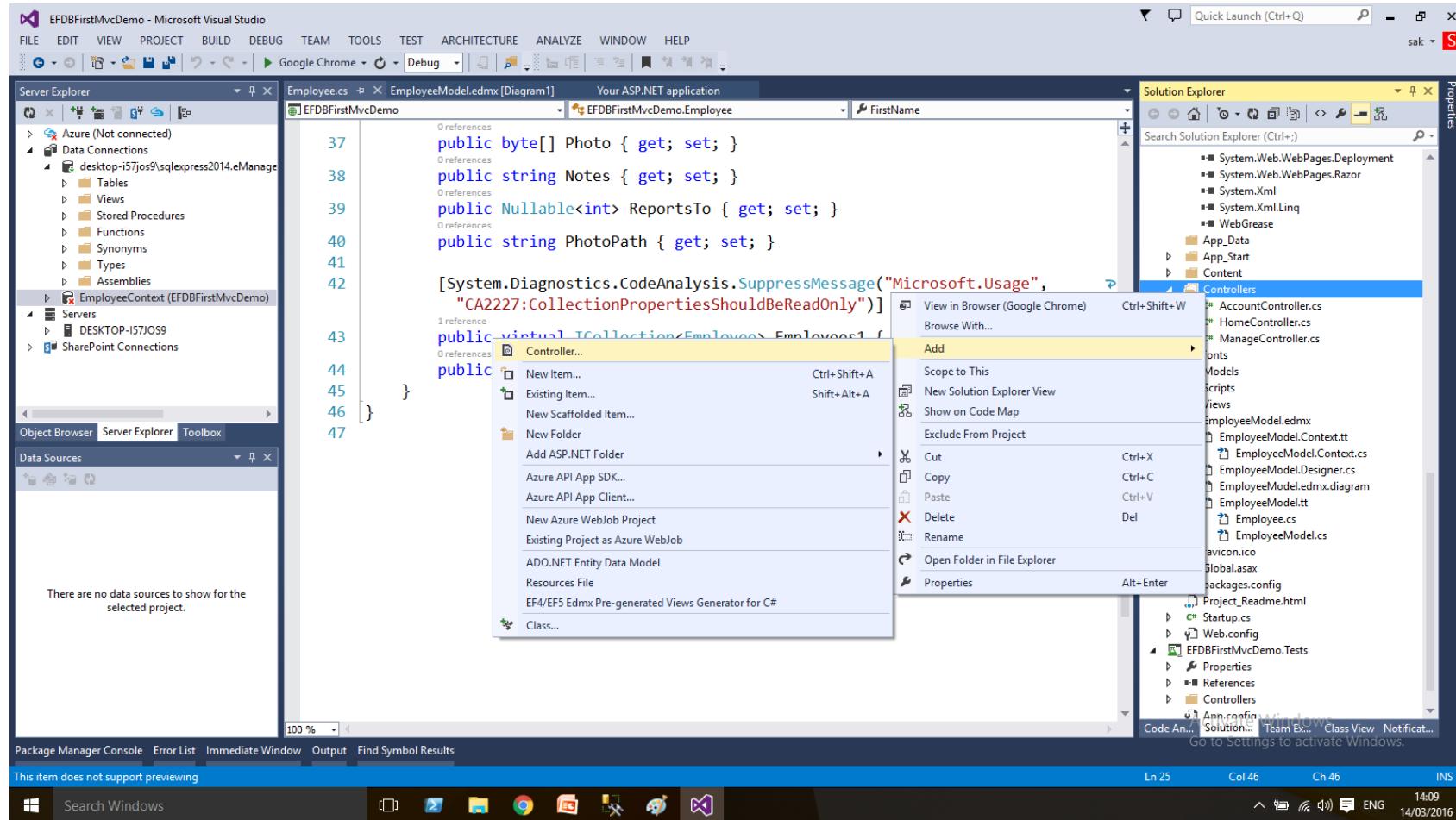
The screenshot shows the Microsoft Visual Studio IDE interface. On the left is the code editor window titled "Employee.cs" under "EmployeeModel.edmx [Diagram1]". The code defines a partial class "Employee" with properties EmployeeID, LastName, FirstName, Title, and TitleOfCourtesy. The Solution Explorer window on the right shows the project structure for "EFDBFirstMvcDemo", including files like "EmployeeModel.edmx", "EmployeeModel.tt", "Employee.cs", and "EmployeeModel.cs". The status bar at the bottom indicates "Activate Windows" and "Go to Settings to activate Windows".

```
6 //      Manual changes to this file will be overwritten if the code is
7 //      regenerated.
8 //
9
10 namespace EFDBFirstMvcDemo
11 {
12     using System;
13     using System.Collections.Generic;
14
15     public partial class Employee
16     {
17         [System.Diagnostics.CodeAnalysis.SuppressMessage("Microsoft.Usage",
18             "CA2214:DoNotCallOverridableMethodsInConstructors")]
19         public Employee()
20         {
21             this.Employees1 = new HashSet<Employee>();
22         }
23
24         public int EmployeeID { get; set; }
25         public string LastName { get; set; }
26         public string FirstName { get; set; }
27         public string Title { get; set; }
28         public string TitleOfCourtesy { get; set; }
```

12. Create EmployeeController

Please read terms and conditions of use

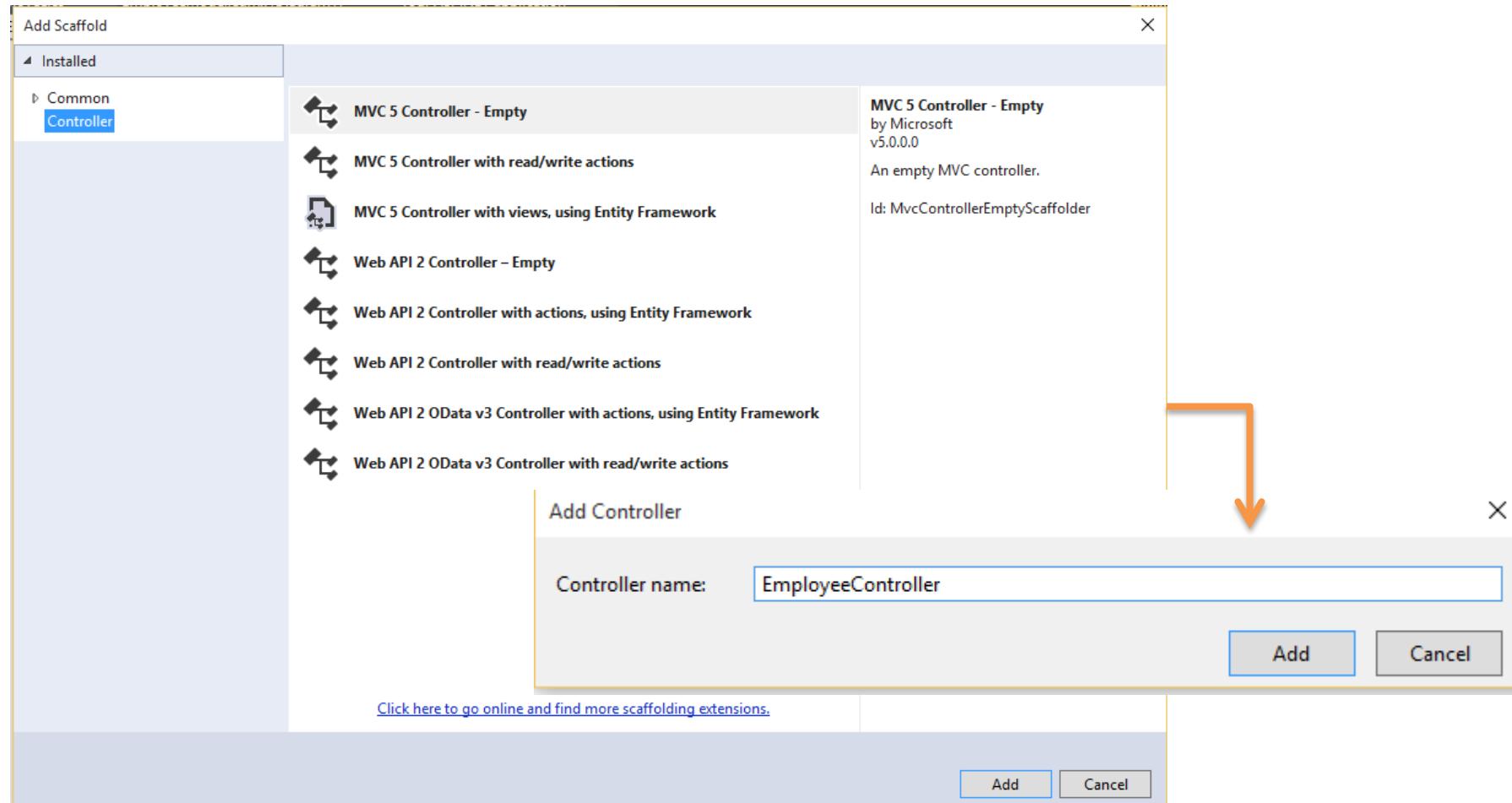
Original Series



13. Empty Controller

Please read terms and conditions of use

Original Series



14.EmployeeController

Please read terms and conditions of use

Original Series

The screenshot shows the Microsoft Visual Studio IDE interface. On the left, the code editor displays the `EmployeeController.cs` file:

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Web;
5  using System.Web.Mvc;
6
7  namespace EFDBFirstMvcDemo.Controllers
8  {
9      public class EmployeeController : Controller
10     {
11         // GET: Employee
12         public ActionResult Index()
13         {
14             return View();
15         }
16     }
17 }
```

The Solution Explorer on the right lists the project structure:

- System.Web.WebPages.Deployment
- System.Web.WebPages.Razor
- System.Xml
- System.Xml.Linq
- WebGrease
- App_Data
- App_Start
- Content
- Controllers
 - AccountController.cs
 - EmployeeController.cs** (selected)
 - HomeController.cs
 - ManageController.cs
- fonts
- Models
- Scripts
- Views
 - Account
 - Employee
 - Home
 - Manage
 - Shared
 - _ViewStart.cshtml
 - Web.config
- EmployeeModel.edmx
 - EmployeeModel.Context.tt
 - EmployeeModel.Context.cs
 - EmployeeModel.Designer.cs
 - EmployeeModel.edmx.diagram
 - EmployeeModel.tt
 - Employee.cs
 - EmployeeModel.cs
 - favicon.ico
 - Global.asax
 - packages.config

15. Retrieve the List of Employee in Controller

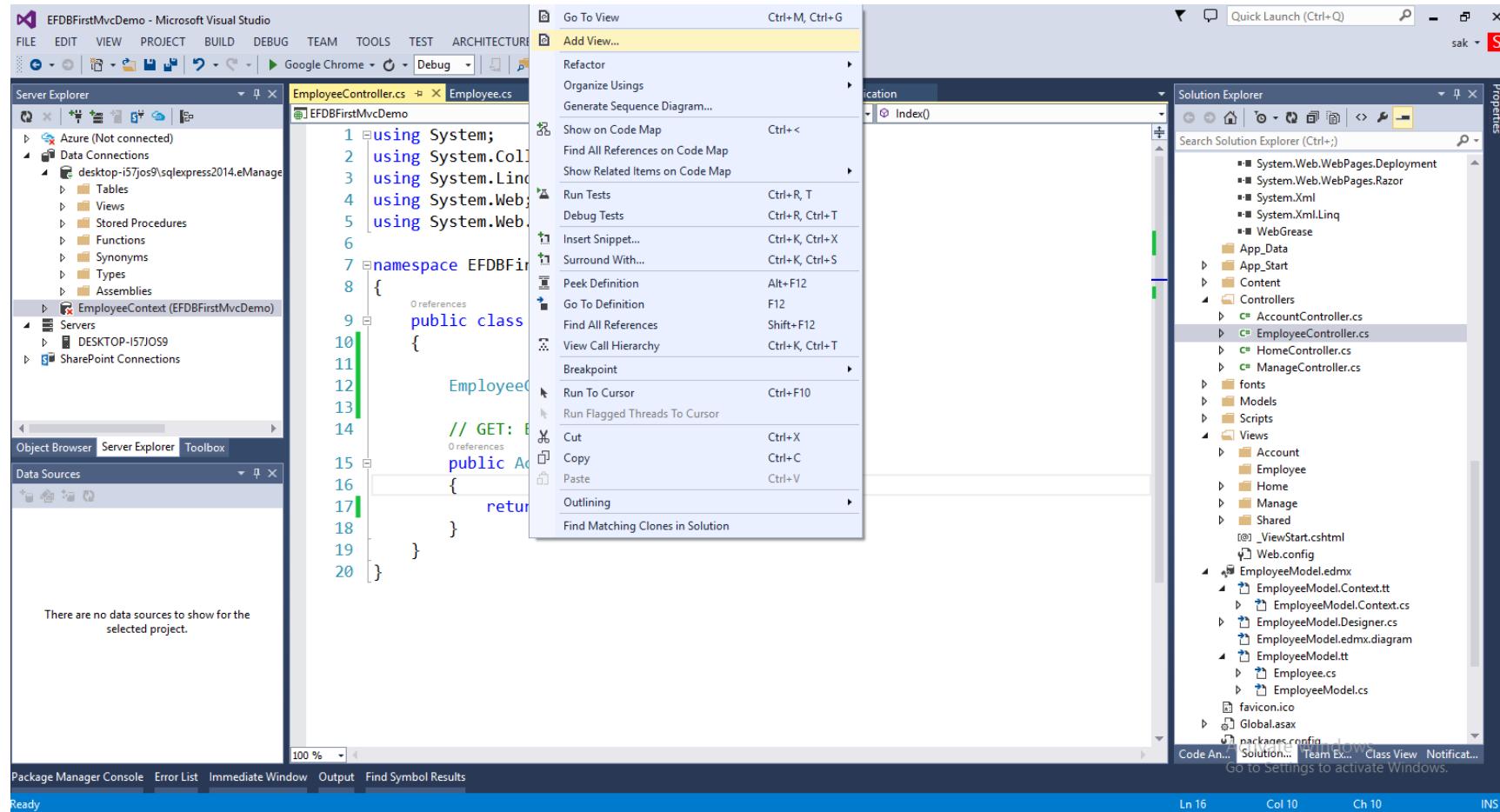
```
public class EmployeeController : Controller
{
    EmployeeContext db = new EmployeeContext();

    // GET: Employee
    public ActionResult Index()
    {
        return View(db.Employees.ToList());
    }
}
```

16. Generate View for Index

Please read terms and conditions of use

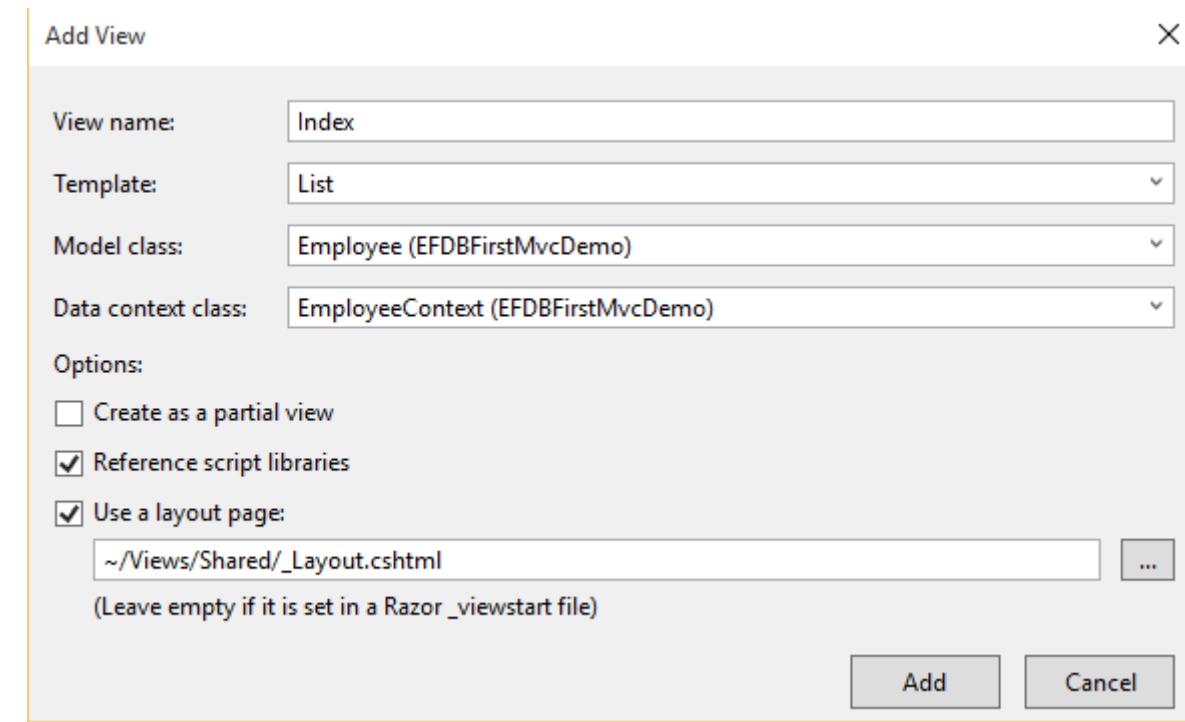
Original Series



17. Index View

Please read terms and conditions of use

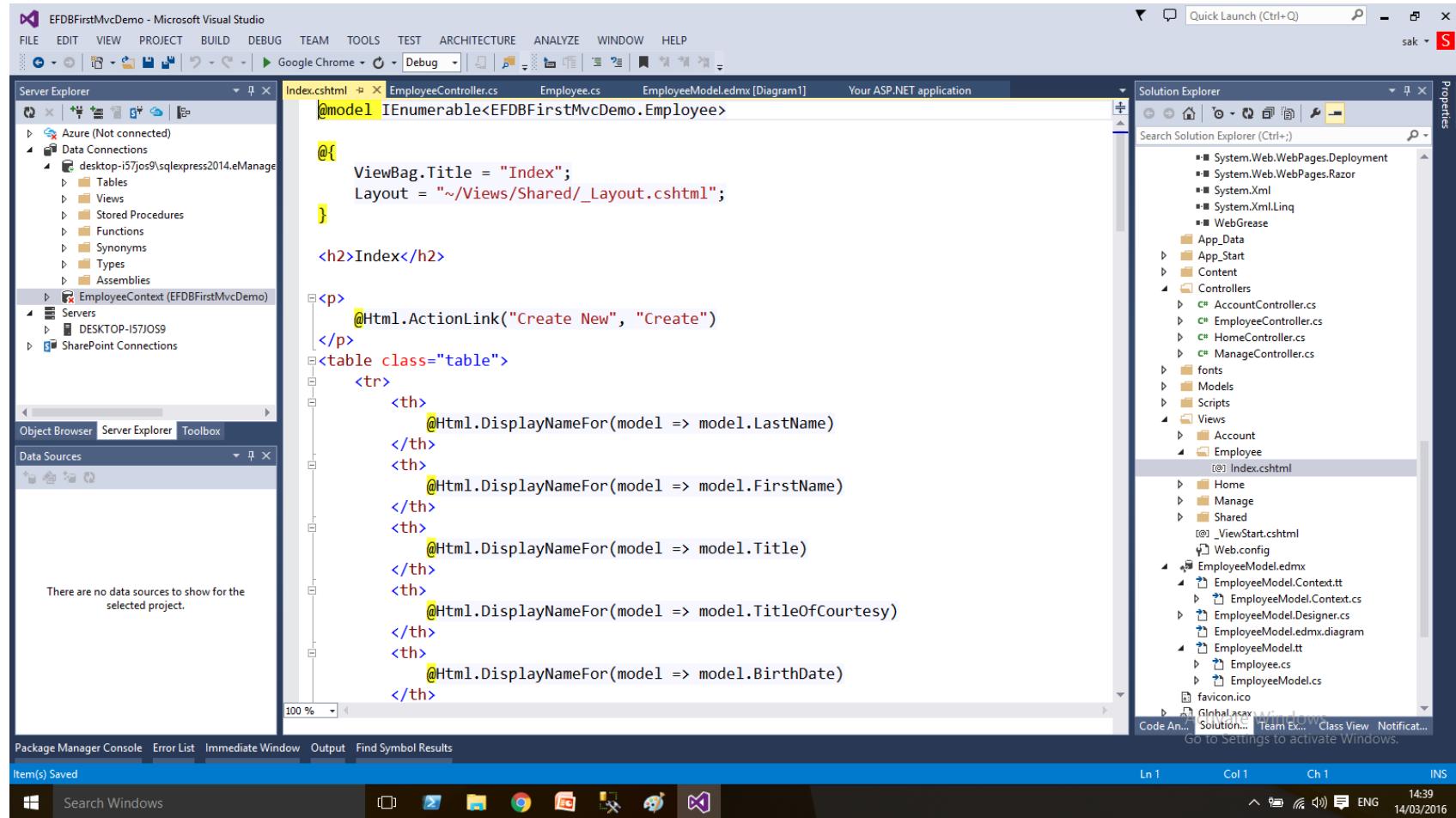
Original Series



18. Index View Scaffolding Template

Please read terms and conditions of use

Original Series



19. Result

List of All Employees

LastName	FirstName	Title	TitleOfCourtesy	BirthDate	HireDate	Address	City	Region	PostalCode	Country	HomePhone	Extension	Photo
Davolio	Nancy	Sales Representative	Ms.	08/12/1948 00:00:00	01/05/1992 00:00:00	507 - 20th Ave. E. Apt. 2A	Seattle	WA	98122	USA	(206) 555-9857	5467	21284702000130140200330
Fuller	Andrew	Vice President, Sales	Dr.	19/02/1952 00:00:00	14/08/1992 00:00:00	908 W. Capital Way	Tacoma	WA	98401	USA	(206) 555-9482	3457	21284702000130140200330

Activate Windows
Go to Settings to activate Windows.

ModelState

- ModelState is a property of a Controller, and can be accessed from those classes that inherit from [System.Web.Mvc.Controller](#).
 - The ModelState represents a collection of name and value pairs that were submitted to the server during a POST.
 - It also contains a collection of error messages for each value submitted.
 - Despite its name, it doesn't actually know anything about any model classes, it only has names, values, and errors.
-
- **Serves 2 purposes**
 - To store the value submitted to the server
 - To store the validation errors associated with those values
 - **ModelState represents the submitted values and errors in said values during a POST**

DataAnnotations

Please read terms and conditions of use

Original Series

- Data Validation is a key aspect for developing web application. In ASP.NET MVC, we can apply validation to web application by using **Data Annotation Attribute classes to model class.**
- **Data Annotation attribute classes are present in System.ComponentModel.DataAnnotations namespace.**
- They help us to define rules to the model classes or properties for data validation and displaying suitable message to end users.

Annotation Type	Description
Data Type	Specify the datatype of a property
DisplayName	Specify the display name for a property
DisplayFormat	Specify the display format for a property like different format for Date property
Required	Specify a property as required
RegularExpression	Validate the value of a property by specified regular expression pattern
Range	Validate the value of a property within a specified range of values
StringLength	Specify min and max length for a string property
MaxLength	Specify max length for a string property
Bind	Specify fields to include or exclude when adding parameter or form values to model properties
ScaffoldColumn	Specify fields for hiding from editor forms

Data Annotations

Advantages

- Enable you to perform validation simply by adding one or more attributes – such as required or StringLength attribute – to a class property

Update Model vs TryUpdateModel

Please read terms and conditions of use

UpdateModel()

- Throws an exception if validation fails
- Used to update the model with the form values and perform the validations

TryUpdateModel()

- Never throws an exception if a validation fails. But will redirect to the same page with error/validation error message displayed.
- Used to update the model with the form values and perform the validations

Original Series

MVC5 WITH EF CODE FIRST

Code First EF MVC5 Application

Step 1. Create MVC5 Application

Step 2. Create A folder Context

Step 3. Create a Model – ModelName.cs

```
public class RealEstateProperty
{
    [Key]
    public int EstatePropertyId { get; set; }

    [Required]
    [DisplayName("Natural Address Code 30 digit Alphanumeric code")]
    public string NACCCode { get; set; }

    public string EstatePropertyName { get; set; }

    public string OwnerName { get; set; }
    public string City { get; set; }

    public string Country { get; set; }

    public string Zip{get;set;}
```

- Step 4. Create a `ModelContext` class in Context Folder – `ModelContext.cs`

The screenshot shows a Visual Studio interface. The code editor displays the `RealEstateContext.cs` file with the following content:

```
1  using ExampleFourPropertyViewResult;
2  using System;
3  using System.Collections.Generic;
4  using System.Data.Entity;
5  using System.Linq;
6  using System.Web;
7
8  namespace ExampleFourPropertyViewResult.Context
9  {
10    public class RealEstateContext : DbContext
11    {
12      public DbSet<RealEstateProperty> RealEstateProperties { get; set; }
13    }
14 }
```

The Solution Explorer on the right shows the project structure for `ExampleFourPropertyViewResult`, which contains two projects: `ExampleFourPropertyViewResult` and `RealEstateProperties`. The `RealEstateProperties` project includes files such as `RealEstateContext.cs`, `RealEstateProperty.cs`, and several `ViewModels` and `Models`.

- Step 5. Add ConnectionString in web.config

```
<connectionStrings>
  <add name="DefaultConnection" connectionString="Data Source=(LocalDb)\v11.0;AttachDbName=RealEstateContext.mdf;Integrated Security=True;MultipleActiveResultSets=True" providerName="System.Data.SqlClient" />
  <add name="RealEstateContext" connectionString="Data Source=(LocalDb)\v11.0;AttachDbName=RealEstateContext.mdf;Integrated Security=True;MultipleActiveResultSets=True" providerName="System.Data.SqlClient" />
</connectionStrings>
```

- Step 6. Create a Controller

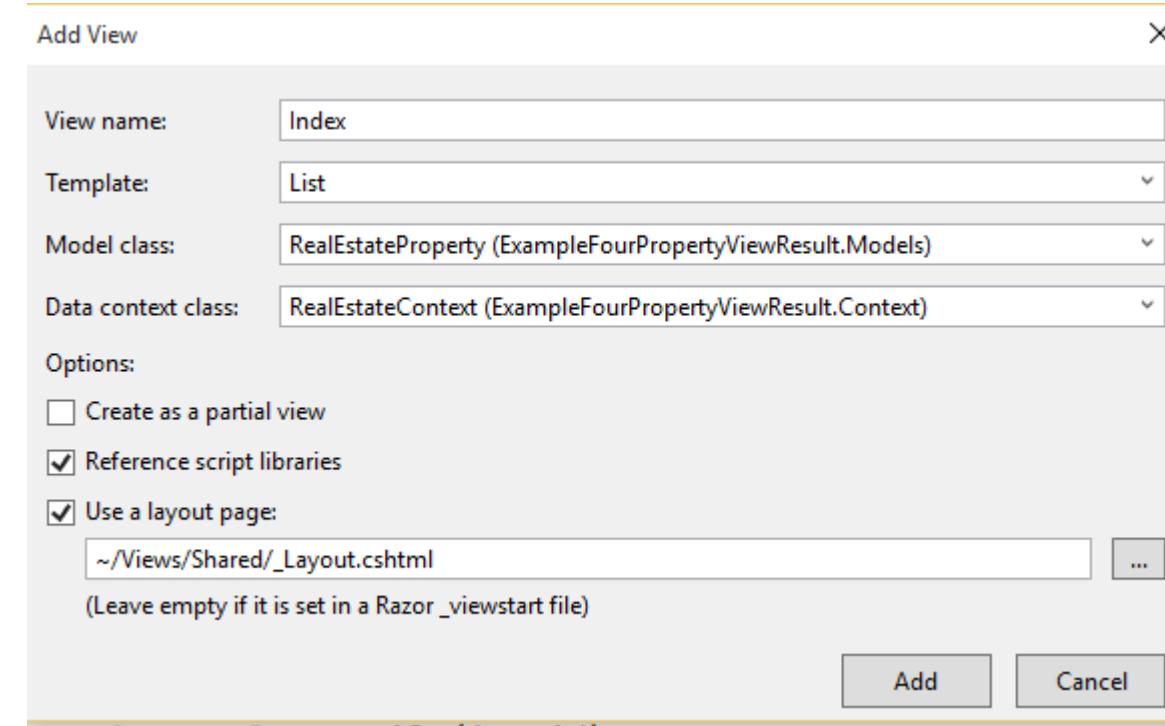
```
namespace ExampleFourPropertyViewResult.Controllers
{
    public class RealEstatePropertyController : Controller
    {
        // GET: RealEstateProperty
        public ActionResult Index()
        {
            return View();
        }
    }
}
```

- Step 7. Add DbContext to the Controller and return list to the conventional view

```
RealEstatePropertyController.cs    ~  web.config    RealEstateContext.cs    RealEstateProperty.cs    Your ASP.NET application
ExampleFourPropertyViewResult      ExampleFourPropertyViewResult.Controllers.RealEstat     Create()
1  using ExampleFourPropertyViewResult.Context;
2  using System;
3  using System.Collections.Generic;
4  using System.Linq;
5  using System.Web;
6  using System.Web.Mvc;
7
8  namespace ExampleFourPropertyViewResult.Controllers
9  {
10    public class RealEstatePropertyController : Controller
11    {
12      RealEstateContext db = new RealEstateContext();
13      // GET: RealEstateProperty
14      public ActionResult Index()
15      {
16        return View(db.RealEstateProperties.ToList());
17      }
18    }
19 }
```

- Step 8. Build the application with ctrl+shift+B

➤ Step 9. Generate Index View



- Step 10. Run the Application – **ctrl +f5**
- Step 11. Upon first access to the Controller Action Method, We have a DB generated from **the RealEstateProperty Model**

Summary

- Demonstrated how to create EF DB First and Code First approaches
- Demonstrated the use of data annotations for validations

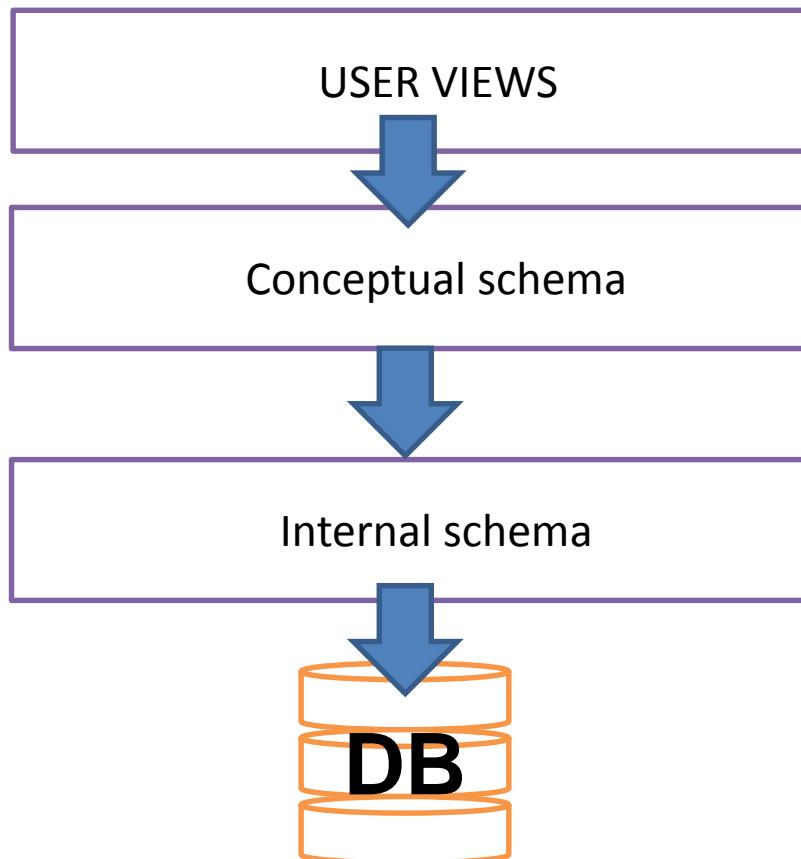
SECTION -XV

N-TIER ARCHITECTURE

Architecture

Please read terms and conditions of use

Original Series



Logical independence
Physical independence

- **Logical independence:** the ability to change the logical schema without changing the external schema or application programs.
 - Can add new fields, table without changing views.
 - Can change the structure of the table without changing the view.
- **Physical Independence:** the ability to change the physical schema without changing the logical schema
 - Storage space can change
 - Types of some data can change for optimizing the application

N-Tier Architectures

- Have
 - Presentation Layer
 - Business/Logic Layer
 - Data Access Layer
 - N-tier architectures try to separate the components into different tiers/layers
 - Tier : physical separation
 - Layer:logical separation

Repository Design Pattern

Martin Fowler: “An Repository Design Pattern mediates between the domain and data mapping layers, acting like an in-memory collection of domain objects”

SECTION -XV

WEB API 2.2

OWIN + KATANA

PORTABLE

MODULAR

LIGHTWEIGHT

DESIRE FOR FAST, LIGHTWEIGHT FRAMEWORKS AND SERVERS

OPEN WEB INTERFACE FOR .NET

SECTION -XVI

OWIN: MIDDLEWARE/KATANA

Open Web Interface for .Net

- The goal of OWIN is to decouple server and application and, by being an open standard, stimulate the open source ecosystem of .NET web development tools.

www.owin.org

Motivation for OWIN

- System.Web basis for All Web
- WebForms bundled
- Regression bugs (fix old bugs, and open new ones)
- Monolithic Architectures
- Slow release cycle
- Performance
- IIS (performance is better as long as we do not load System.Web*)
- Evolutionary steps: MVC, WEBAPI
- Moving from big server application to smaller client/device centric applications, microservices architectures, mobile clients, REST.
- Cloud, Docker, cross-platform
- Web based IDE, lightweight IDEs(sublime text, Atom)
- ASP.NET coupled to IIS
- Hard to do REST
- ASP.NET --- toooo heavy
- Drawn inspiration from Rack(Ruby) and WSGI(Python)

Solutions

- Boat
- Figment
- Fracture
- Frank
- FubuMVC
- KayakHTTP
- NancyFx
- Nina
- Nrack
- OpenRasta
- Suave
- WCF Web API
- Etc..

ASP.NET Evolution

- ASP.NET – released in early 2002 with .NET framework 1.0
- Optimized for 2 primary customers
 - Classis ASP developers
 - Desktop line of business developers (VB6 WINFORMS Developers)
- System.Web.dll (aka ASP.NET)
 - Considered heavy
 - Always executes lots of ASP.NET – specific code
- Result was a monolithic framework that included lots of concerns in a single package(System.Web)
 - HTTP intrinsics
 - Modules
 - Handlers
 - Session
 - Cache
 - Web Forms
 - Controls
- ASP.NET was designed to be run on IIS

Challenges

- ASP.NET shipped as a part of the .NET Framework – slow ship cycle
- As ASP.NET **evolved -> the size and complexity of System.Web also grew.**
- All of the features were turned on by default – you have to opt out
- Only one hosting option –IIS
- Performance Cost and Processing Costs increase.

- 2007-2008 ASP.NET MVC ships out of band from the .NET Framework
- Enables a more rapid ship cycle
- <http://asp.net/mvc>
- 2012 – ASP.NET WebAPI released
 - Designed with no dependencies on System.Web
 - Includes self-host capability
- **Self-host realization**
 - ASP.NET Web API enabled self host for Web API applications
 - Additionally, other frameworks had or were writing their own self-host servers
 - A modern web application needs the ability to compose multiple frameworks in a single server.

BareBones Http Server in NodeJS

- Basic http service written in Node.JS
- Full control on the content types and negotiation

```
var http= require('http');
http.createServer(function(req,res){
    res.writeHead(200,['Content-Type':'text/plain']);
    res.end("Hello Harman\n");
}).listen(9999,'127.0.0.1');
console.log('Server running at http://127.0.0.1:9999');
```

OWIN

- Open Web Interface for dot Net
- Defines interface between app components
 - Decouples app from framework, host and server
- Open standard
- Not revolution but evolution, influenced by other stacks
- A BAREBONES HTTP STACK FOR .NET FRAMEWORK

OWIN

- Specification for Open Web Interface for .Net
- Community standard
- No more System.Web, just dictionary of environment variables (request, response, etc)
- Async
- Microsofts implementation: Katana
 - v1-3 is in ASP.NET vCurrent
 - v4 is vNext
 - MVC 6, WebAPI, SignalR
- Helios, Kestrel, Nowin

- IIS and OS
 - Realease with the OS
 - released with new version of Windows (WebSockets are available only on the latest version of Windows)
- OWIN/Katana Support
 - Many application frameworks support OWIN/Katana
 - Web API
 - SignalR
 - Nancy
 - ServiceStack
 - FubuMVC
 - Simple.Web
 - RavenDB

OWIN Adoption and Support

- Adoption
 - Microsoft.OWIN also called as Katana
 - Fix
 - Freya
 - SimpleOWIN
 - Simple.OWIN
 - Suave
 - WebSharper
- Server Support
 - System.Web
 - HttpListener
 - IIS(Helios)
 - Nowin(cross-platform)
 - Suave(cross-platform)
- Middleware
 - CORS
 - Security
 - Routing(Superscribe)
 - Diagnostics

OWIN Specs:AppFunc

```
using AppFunc = func<IDictionary<string,object>,Task>;
```

```
using AppFunc = func<  
    IDictionary<string,object>, //Environment  
    Task>; //Done
```

OWIN Specs:Environment

- Keys in the dictionary (8 request, 5 response, 2 others)
 - Owin.RequestBody =>Stream
 - Owin.RequestHeaders =>IDictionary<string,string[]>
 - Owin.Request*
 - Owin.ResponseBody =>Stream
 - Owin.ResponseHeaders =>IDictionary<string,string[]>
 - Owin.ResponseStatusCode =>int
 - Owin.Response*
 - Owin.CallCancelled =>CancellationToken

OWIN Request Keys

Please read terms and conditions of use

Original Series

Owin.RequestBody	Stream
Owin.RequestHeaders	IDictionary<string,string[]>
Owin.RequestMethod	String
Owin.RequestPath	String
Owin.RequestPathBase	String
Owin.RequestProtocol	String
Owin.RequestQueryString	String
Owin.RequestScheme	String
Owin.RequestId*	Optional String
Owin.RequestUser*	Optional ClaimsPrincipal

OWIN Response Keys

Please read terms and conditions of use

Owin.ResponseBody	Stream
Owin.ResponseHeaders	IDictionary<string,string[]>
Owin.ResponseStatusCode	Optional int(default is 200)
Owin.ResponseReasonPhrase	Optional string(default set by server)
Owin.ResponseProtocol	Optional String

OWIN Other Data

Original Series

Owin.CallCancelled	CancellationToken
Owin.Version	String

OWIN Specs:Layers

Host

- Startup, Initialization and process management

Server

- Listens to socket and calls the first middle ware

Middleware

- Pass-through components

Application

- Application code

Please read terms and conditions of use

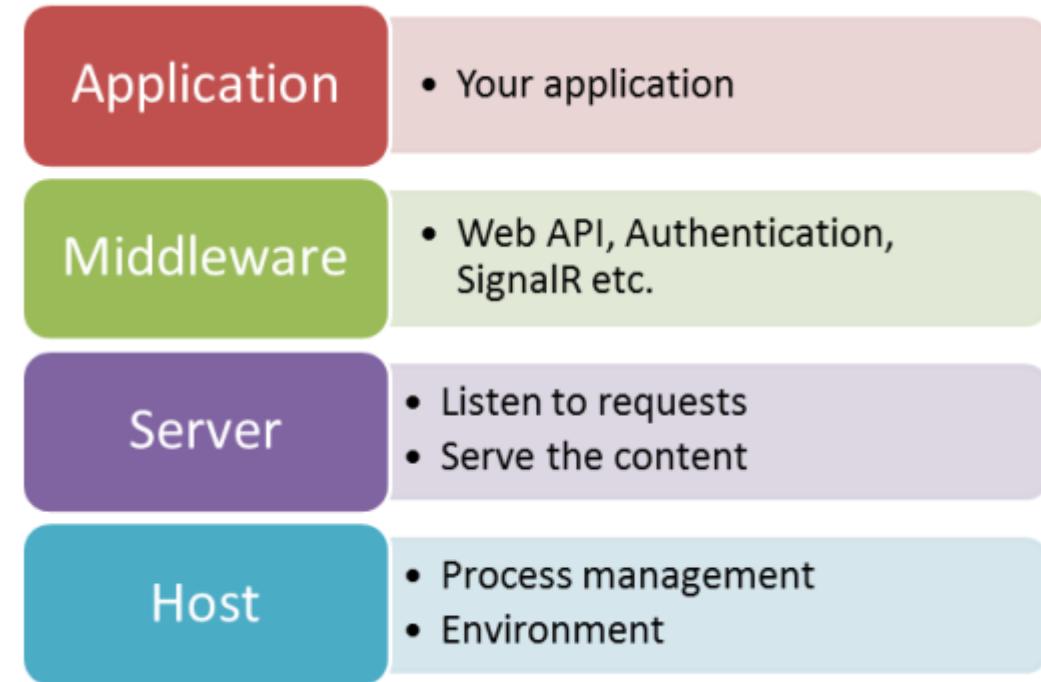
Original Series

Katana

Please read terms and conditions of use

Original Series

- Microsoft's OWIN implementation
 - <https://katanaproject.codeplex.com>
- Set of hosts and servers
 - IIS or self-hosting
- Set of convenience classes
 - OwinContext, OwinRequest,OwinResponse,etc
 - AppBuilderUseExtensions
 - AuthenticationManager
- Set of middleware for common features
 - Authentication
 - Hosting content(e.g.static files)
 - CORS



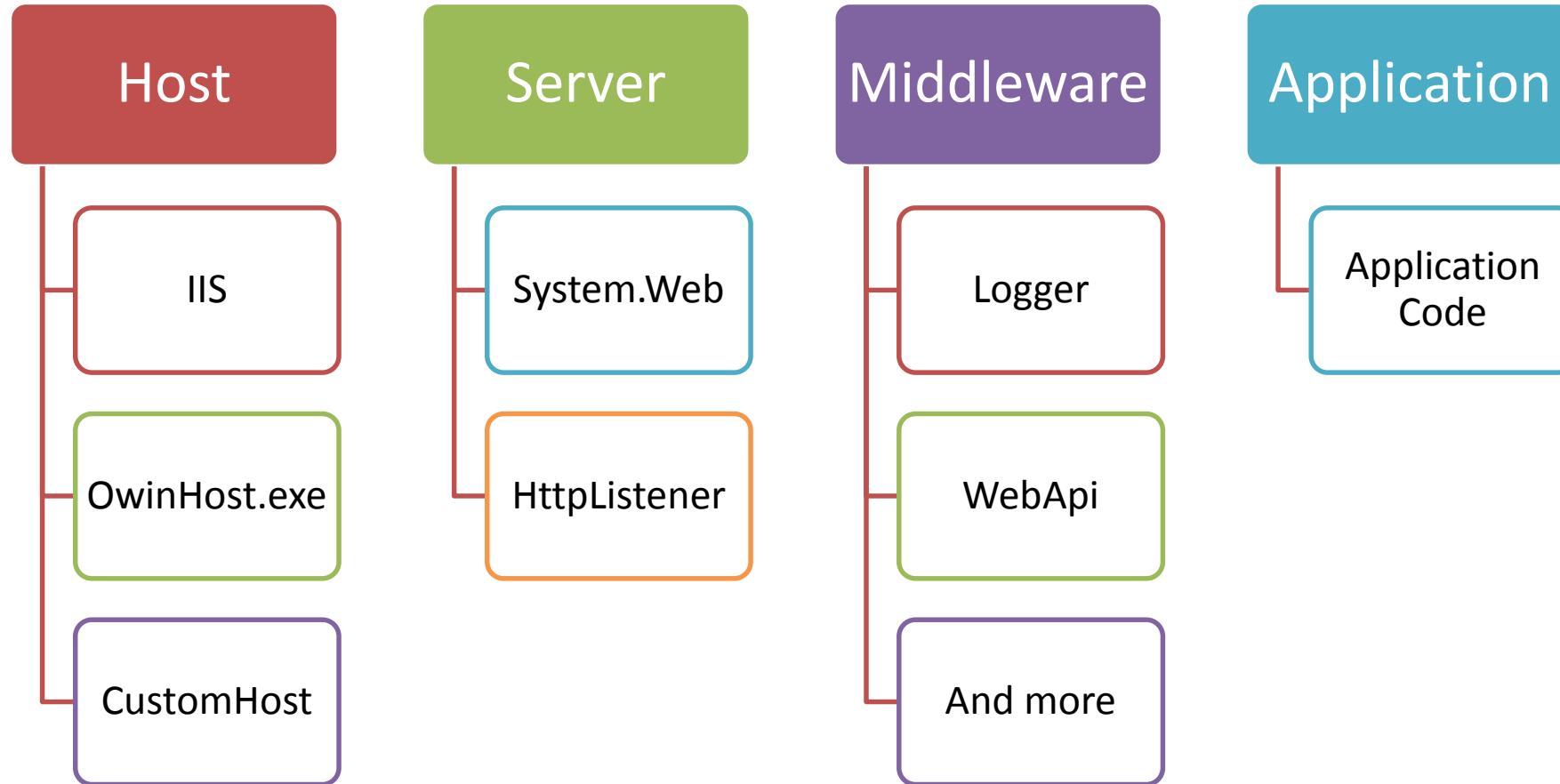
Removing the dependency on IIS

Katana pillars

- It's portable
 - Components should be able to easily substituted for new components as they become available
 - This includes all types of components, from the framework to the server and host
- Lightweight/performant/scalable
 - Fewer computing resources
 - Explicit decision to the application developer to include on-demand features from the underlying infrastructure to the OWIN pipeline.
- It's Modular/flexible
 - Unlike many frameworks which include a myriad of features that are turned on by default, katana project components should be **small and focused**, giving control over to the application developer in determining which components to use in their applications

Katana Pipeline

Please read terms and conditions of use
Original Series



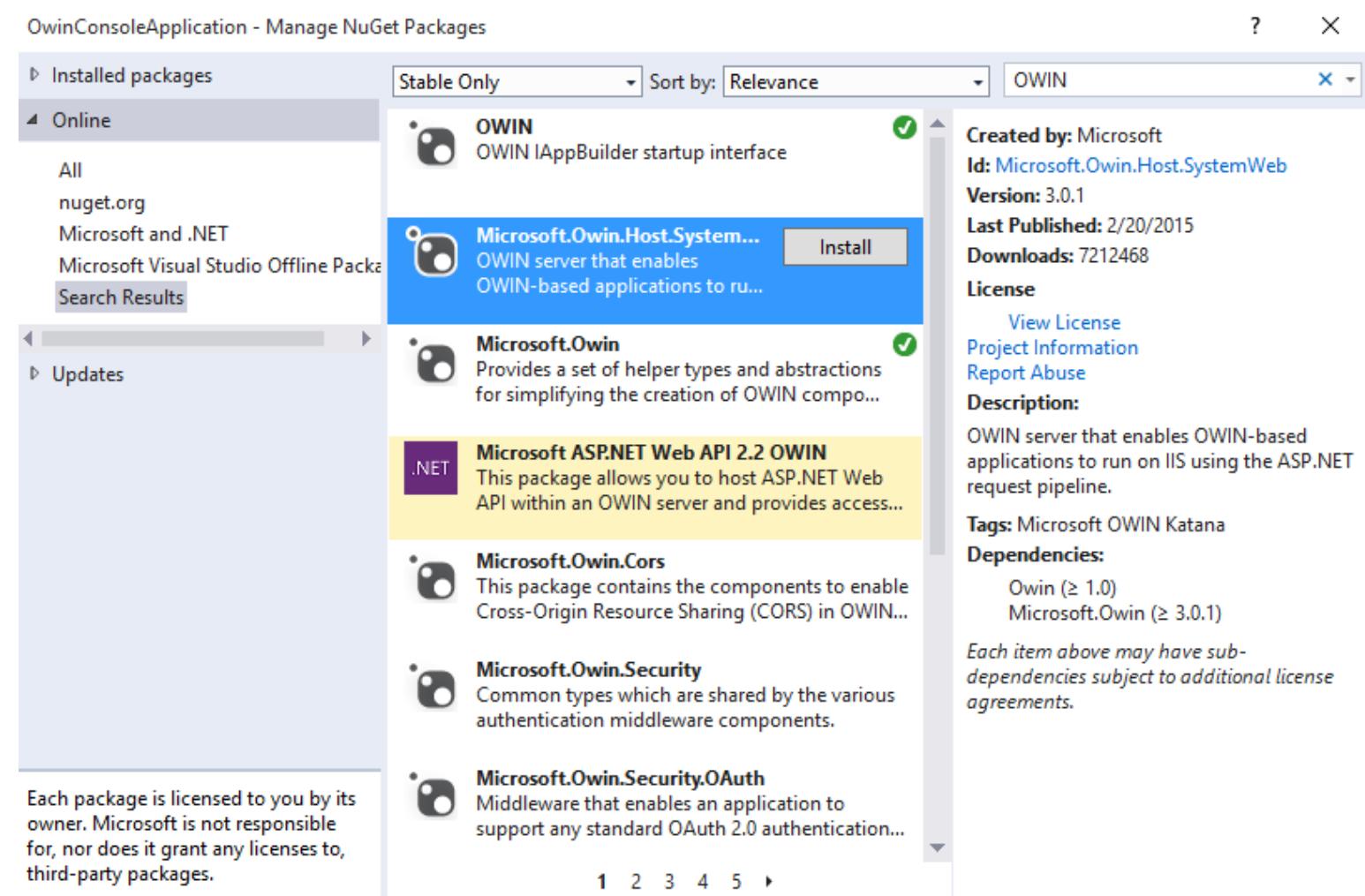
Microsoft ASP.Net Web API 2.2 OWIN

1. Create an empty web application and corresponding models and controllers.
2. Create OWIN startup.cs file
3. Create WebApiConfig.cs file
4. Import the Assembly references from Nuget Packages
 - a. Microsoft ASP.Net Web Api 2.2 OWIN
 - b. Microsoft.Owin
 - c. Microsoft.Owin.Host.SystemWeb
 - d. Microsoft.Owin.SelfHost
 - e. OwinHost

Hosting on IIS

Please read terms and conditions of use

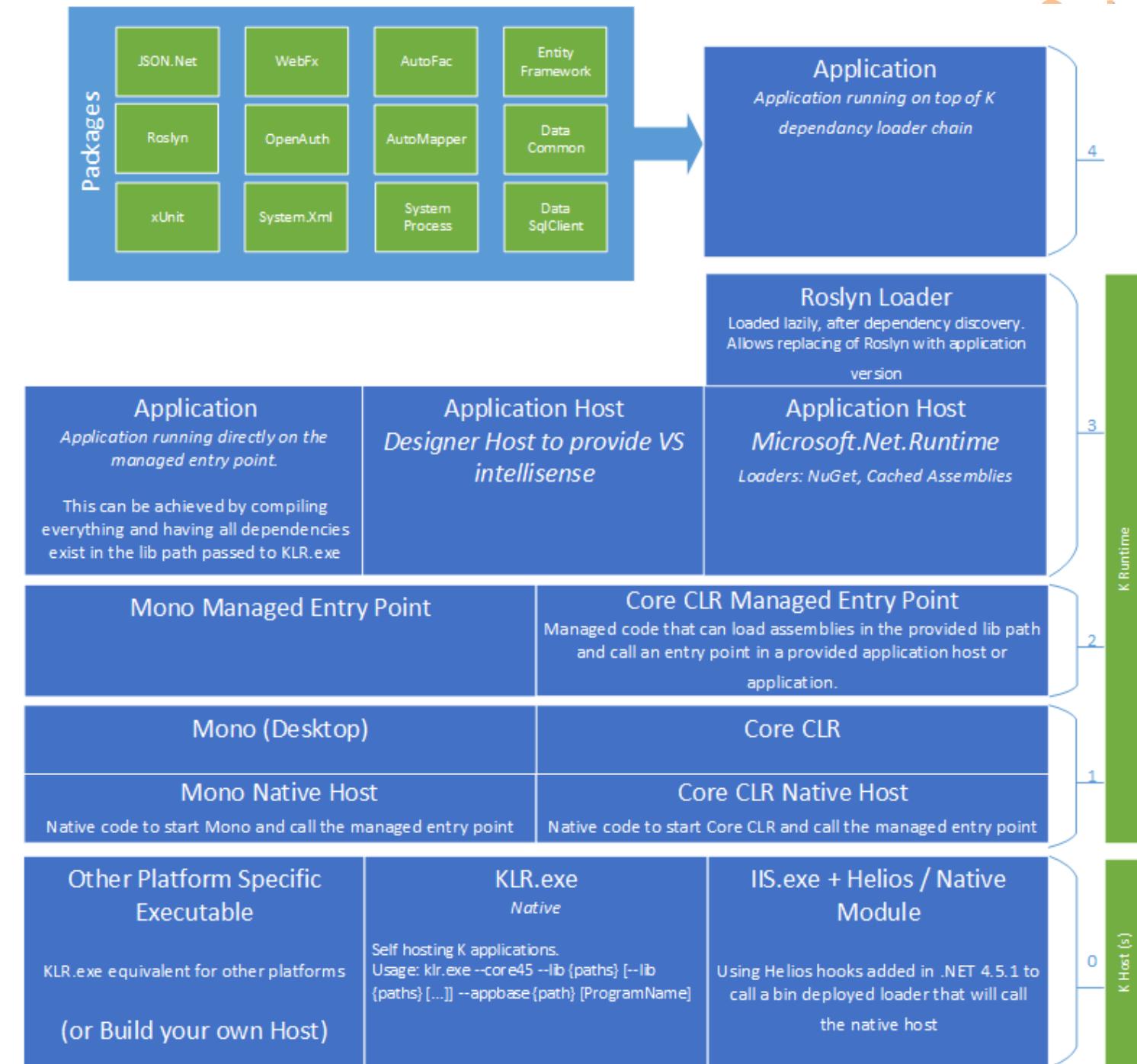
Original Series



VNEXT

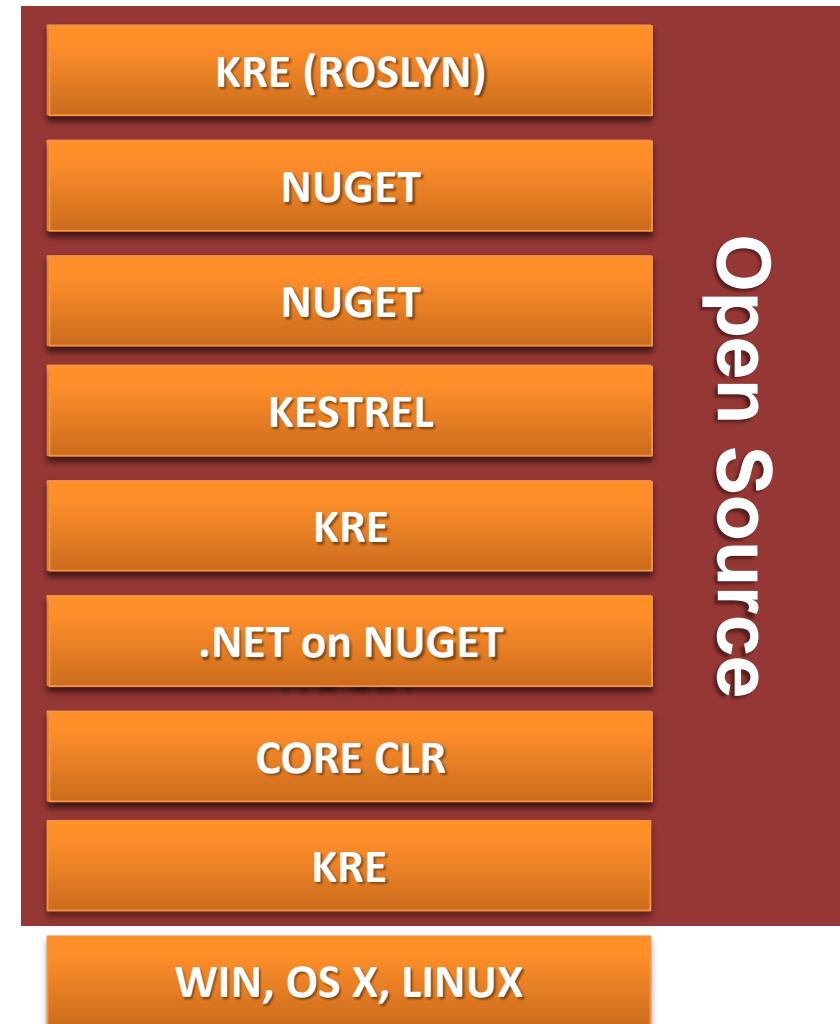
Please read terms and conditions of use

Original Series



Please read terms and conditions of use

Original Series

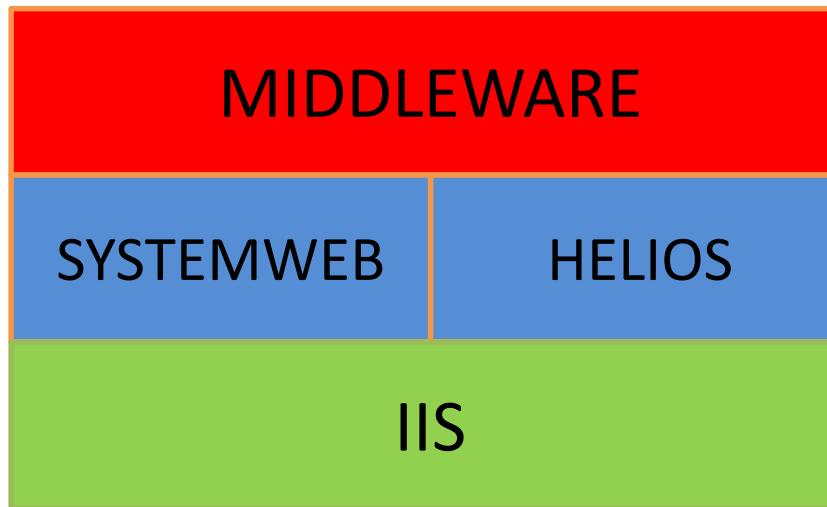


NOTE: KRE = XRE , K = DOTNET , KVM = DOTNET SDK, KPM = NUGET

KATANA HOSTS AND SERVERS

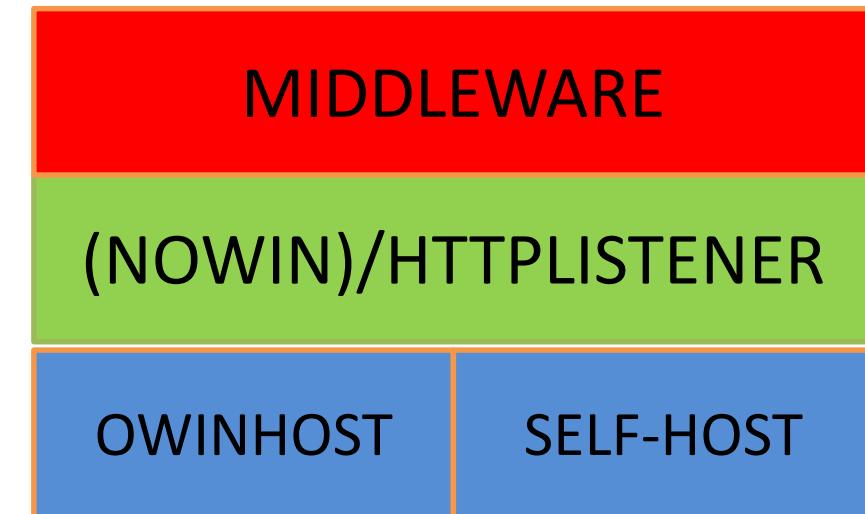
Please read terms and conditions of use

IIS HOST



- IIS BASED HOSTING

NON-IIS HOST



Original Series

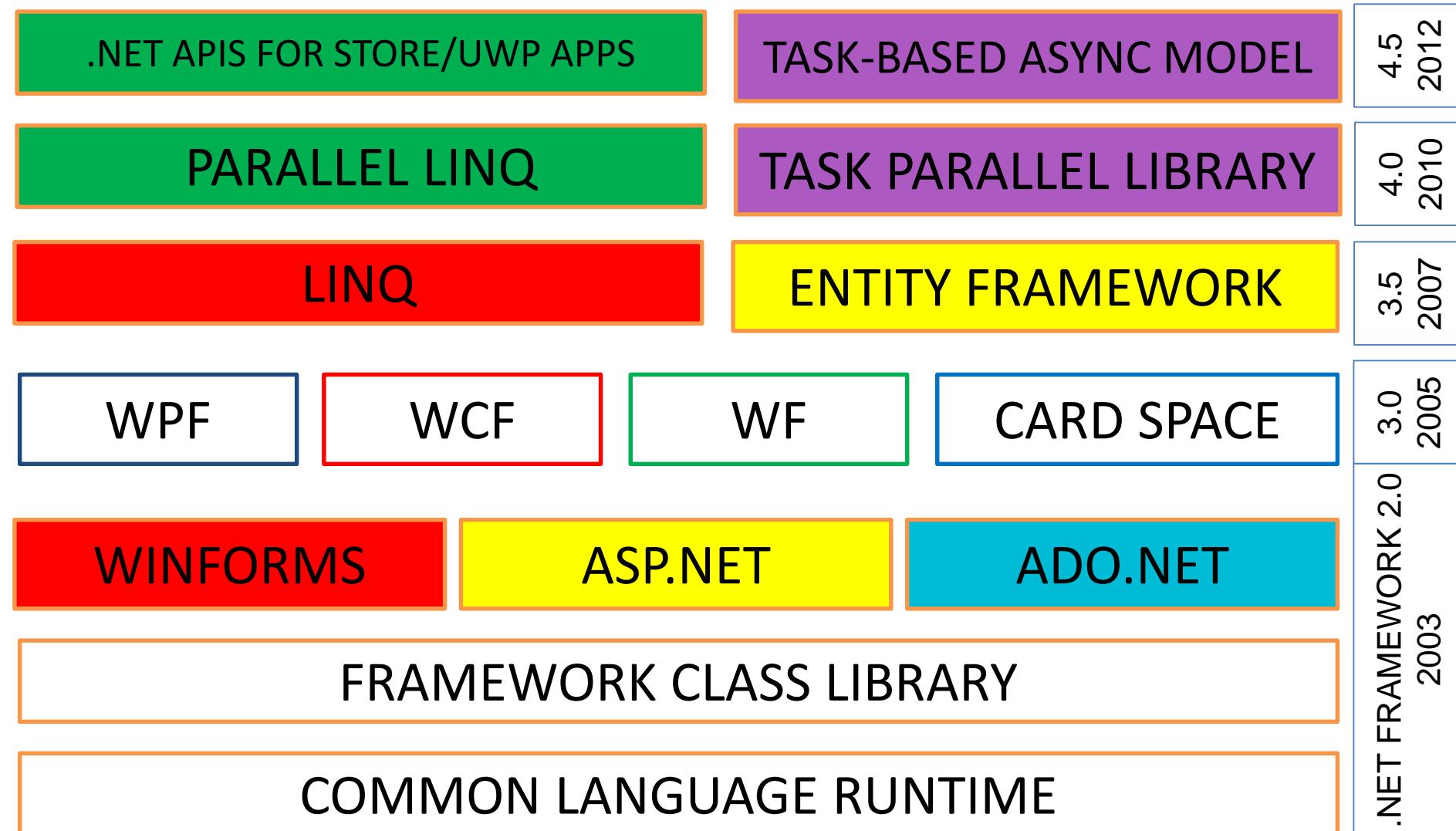


<https://dotnet.github.io>

Any App, Any Platform, Any Developer

Please read terms and conditions of use

Original Series



.NET RUNTIME

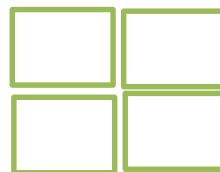
<https://www.ecma-international.org/publications/standards/Ecma-335.htm>

- COMMON LANGUAGE INFRASTRUCTURE (CLI)
 - FORMAL SPECIFICATION FOR .NET RUNTIME
 - ECMA 335
- SEVERAL DIFFERENT HISTORICAL IMPLEMENTATIONS
 - COMMON LANGUAGE RUNTIME
 - RUNTIME FOR THE .NET FRAMEWORK
 - MOST COMMONLY USED
 - COMPACT FRAMEWORK
 - USED WITH MOBILE DEVICES
 - SHARED SOURCE CLI(ROTOR)
 - MONO (LINUX PLATFORMS)
 - SILVERLIGHT
 - WINDOWS PHONE
 - XAMARIN (CROSS PLATFORM SUPPORT)

.NET CLASS LIBRARIES

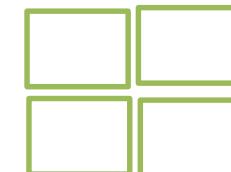
- FRAMEWORK CLASS LIBRARIES (FCL)
 - Libraries used by the .NET framework
- Silverlight
 - Deprecated subset of the FCL for building web and mobile UIs
- Portable class libraries (PCL)
 - Libraries for building cross platform applications
 - Target platforms
 - Windows
 - Windows Phone
 - Silverlight

.NET 2015



.NET FRAMEWORK

ASP.NET 5
ASP.NET 4.6
WPF
WINDOWS FORMS



.NET CORE

ASP.NET 5
.NET NATIVE
ASP.NET 5 FOR MAC
AND LINUX

COMMON

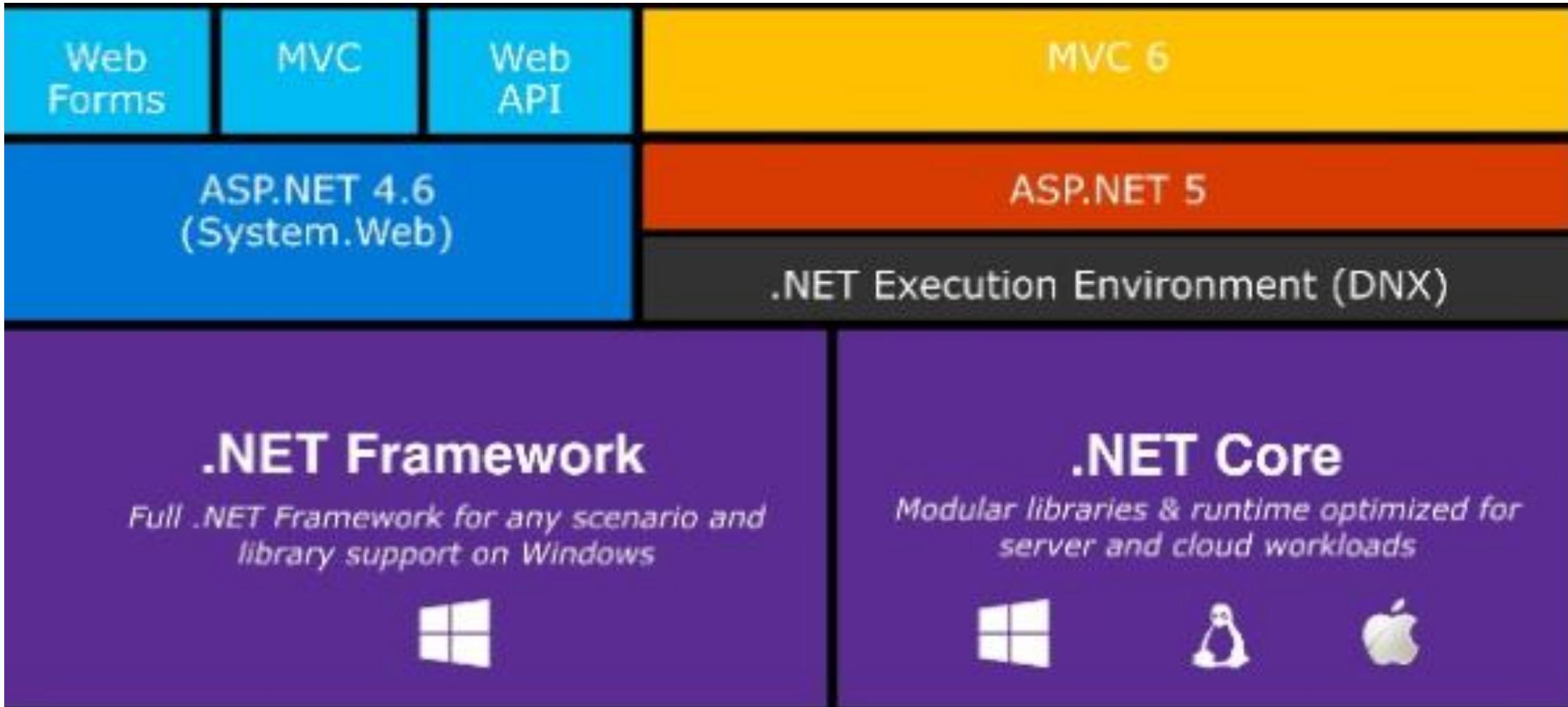
RUNTIME
-NEXT GEN JIT

COMPILERS
- .NET COMPILER PLATFORM
- LANGUAGE

NUGET PACKAGES
-.NET CORE 5 LIBRARIES
.NET FRAMEWORK 4.6 LIB

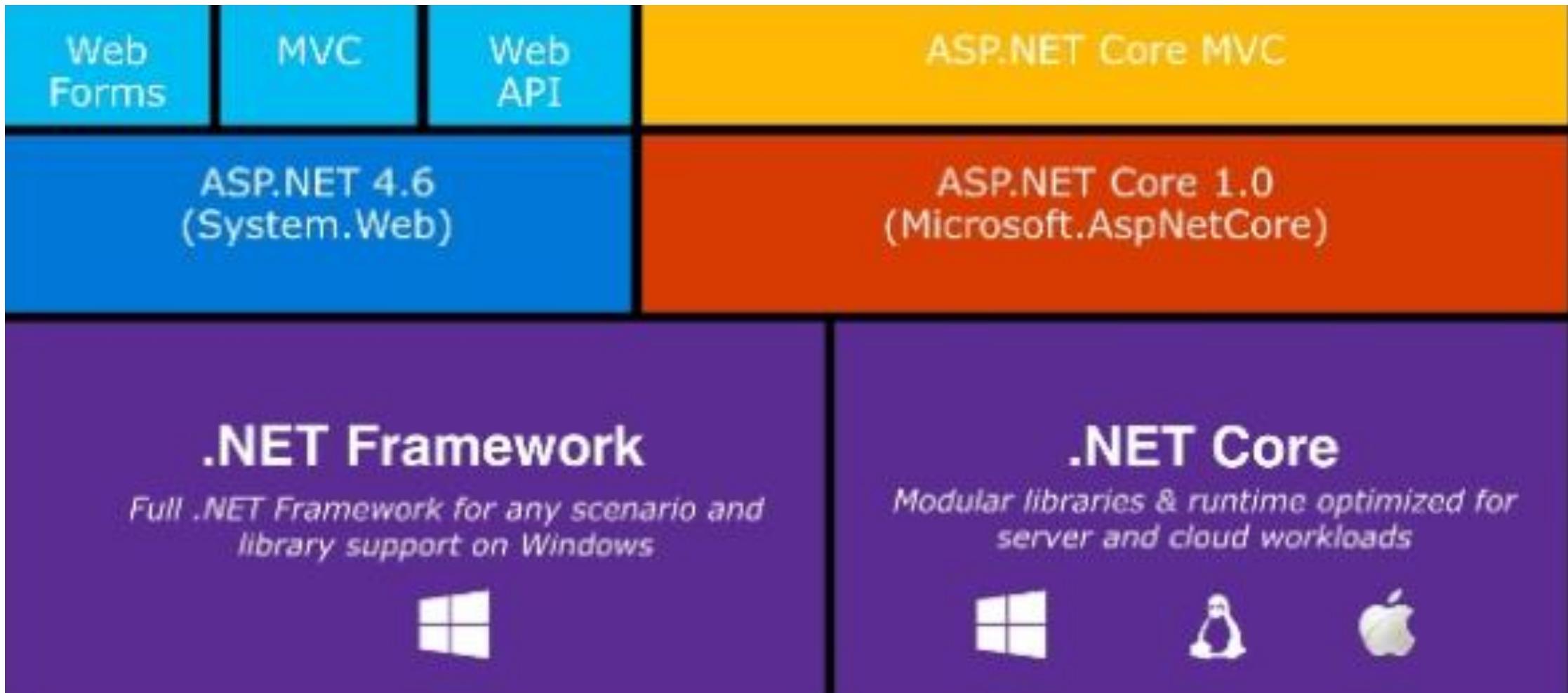
ASP .NET 5

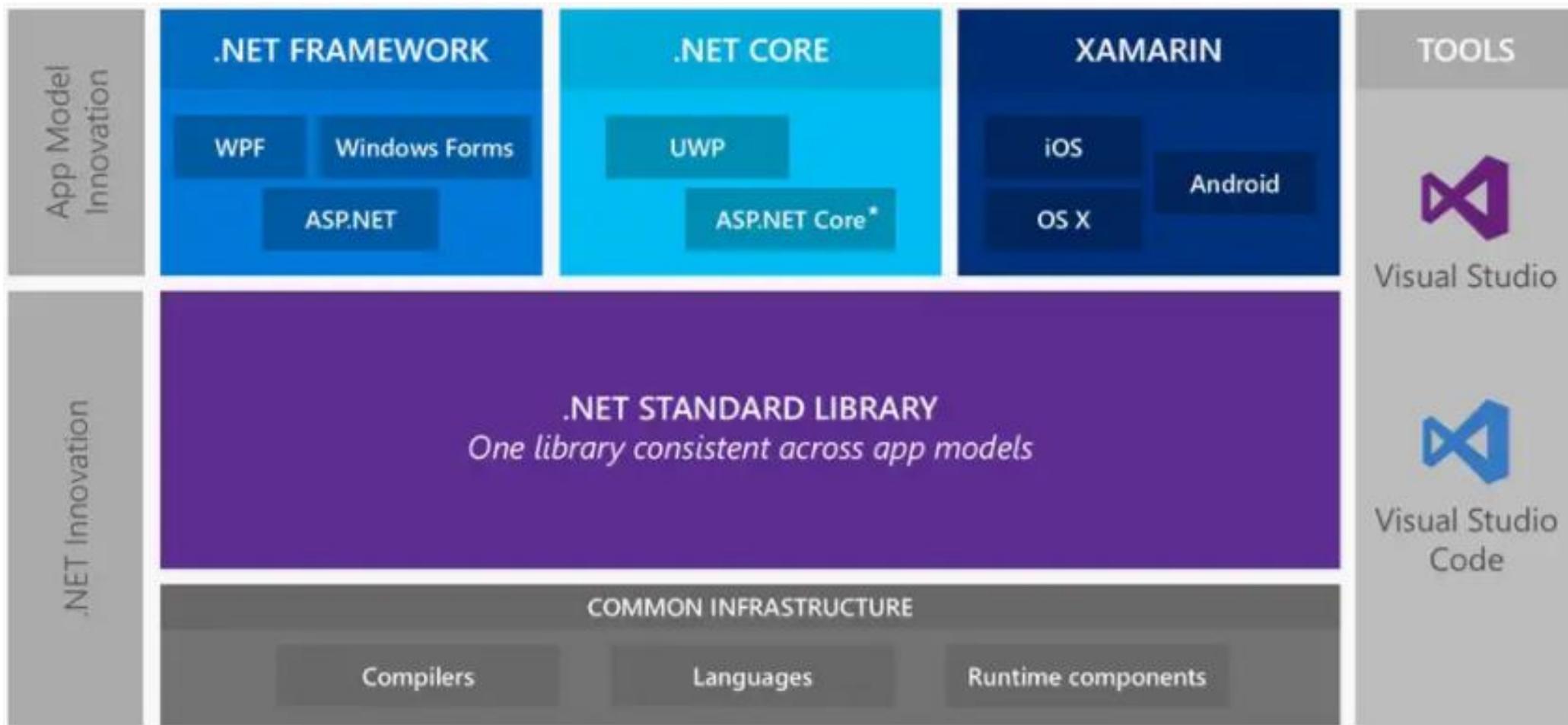
Please read terms and conditions of use



ASP.NET CORE

Please read terms and conditions of use





ASP.NET CORE FEATURE

Please read terms and conditions of use

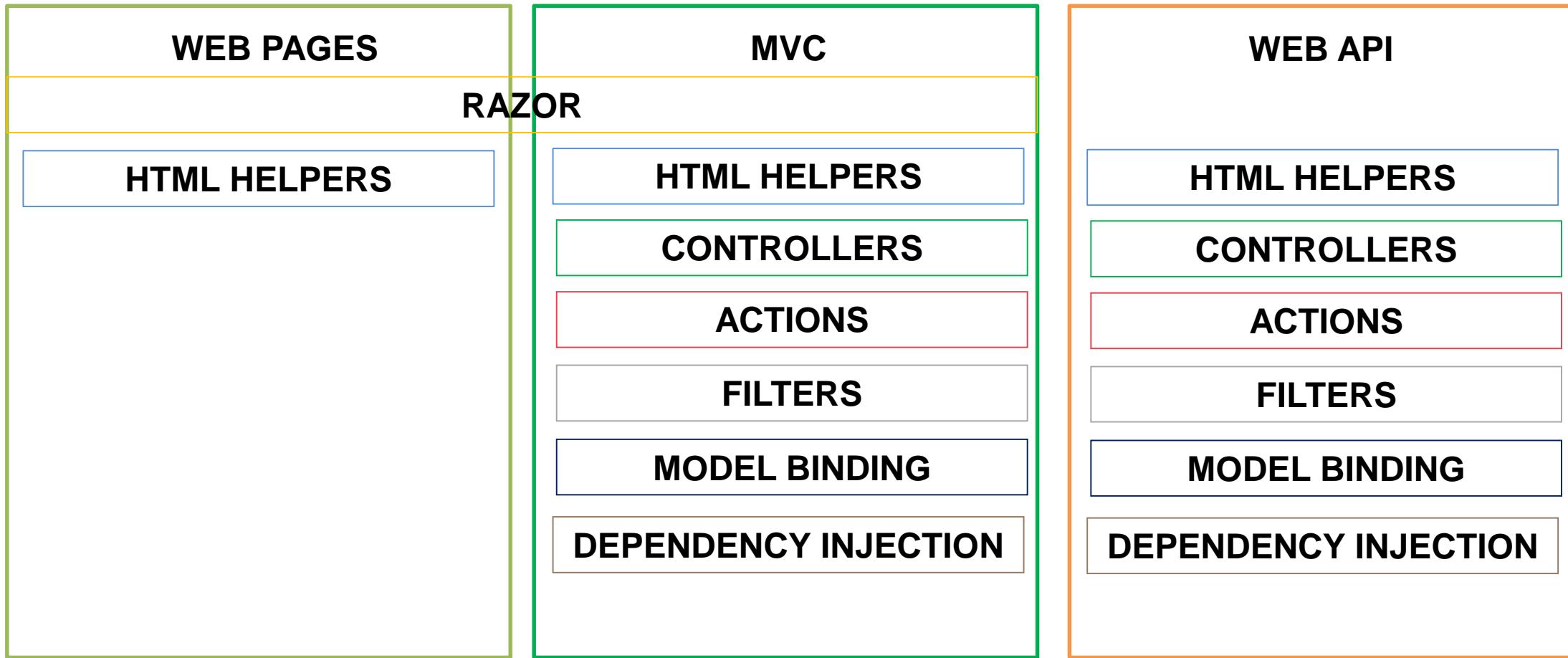
Original Series

- HOSTING
 - KESTREL,STARTUP
- MIDDLEWARE
 - ROUTING
 - AUTHENTICATION
 - STATIC FILE RENDERING
 - DIAGNOSTICS
 - ERROR HANDLING
 - SESSION MANAGEMENT
 - CORS
 - LOCALIZATION
 - CUSTOM
- DEPENDENCY INJECTION
- CONFIGURATION MANAGEMENT
- LOGGIN
- APPLICATION FRAMEWORKS
 - MVC
 - IDENTITY
 - SIGNALR

ASP.NET FRAMEWORKS

Please read terms and conditions of use

Original Series



Motivation for Change in .NET

- Shift in the EcoSystem from Windows to Azure Cloud
- IIS is tightly coupled with
 - System.Web
 - System.Net
- Legacy Baggage
 - XML
 - Remoting
 - Enterprise Services
- Lightweight, stripped down version of application that can run from desktop to IOT platforms.

.NET CORE 1.0

- CROSS PLATFORM
- LIGHT-WEIGHT
- NEW CLI TOOLING
- UNIT TESTING SUPPORT WITH XUNIT, NUNIT
- DOCKER DEPLOYMENT POSSIBLE.

- 2016
 - Container Ecosystems
 - Docker adoption in .NET Development Environments
 - Azure cloud support for containers with different services
 - Windows Server 2016.
 - Support for Native Windows Containers and Hyper V Containers
 - Created Separate Version for Containers – server core and nano server
 - Microservices
 - Application architecture tuned for smaller services.

.NET Foundation

<https://github.com/dotnet>

The screenshot shows the GitHub organization page for ".NET Foundation". The page has a purple header with the ".NET foundation" logo. Below the header, there are links for "This organization", "Search", "Pull requests", "Issues", "Marketplace", and "Explore". A notification bell icon shows 1 new notification. The main content area displays six pinned repositories:

- corefx**: This repo contains the .NET Core foundational libraries, called CoreFX. It includes classes for collections, file systems, console, XML, async and many others. We welcome contributions.
- coreclr**: This repo contains the .NET Core runtime, called CoreCLR, and the base library, called System.Private.CoreLib (or mscorlib). It includes the garbage collector, JIT compiler, base .NET data types and...
- standard**: This repo is building the .NET Standard
- roslyn**: The .NET Compiler Platform ("Roslyn") provides open-source C# and Visual Basic compilers with rich code analysis APIs.
- cli**: This repo contains the .NET Core command-line (CLI) tools, used for building .NET Core apps and libraries through your development flow (compiling, NuGet package management, running, testing, ...).
- orleans**: Orleans - Distributed Virtual Actor Model

Each repository card shows the language (C#), stars, and forks counts.

Please read terms and conditions of use

Original Series

.NET STANDARD VERSIONS

<http://immo.landwerth.net/netstandard-versions/#>



Please read terms and conditions of use

Original Series

.NET Standard	1.0	1.1	1.2	1.3	1.4	1.5	1.6	2.0
.NET Core	1.0	1.0	1.0	1.0	1.0	1.0	1.0	2.0
.NET Framework	4.5	4.5	4.5.1	4.6	4.6.1	4.6.1	4.6.2	4.6.1 vNext
Mono	4.6	4.6	4.6	4.6	4.6	4.6	4.6	5.4
Xamarin.iOS	10.0	10.0	10.0	10.0	10.0	10.0	10.0	10.14
Xamarin.Mac	3.0	3.0	3.0	3.0	3.0	3.0	3.0	3.8
Xamarin.Android	7.0	7.0	7.0	7.0	7.0	7.0	7.0	8.0
Universal Windows Platform	10.0	10.0	10.0	10.0	10.0	10.0.16299	10.0.16299	10.0.16299
Windows	8.0	8.0	8.1					
Windows Phone	8.1	8.1	8.1					
Windows Phone Silverlight	8.0							

.NET STANDARD VERSIONS

- 1.4
 - DEFAULT CHOSEN BY 1.X COMMAND LINE TOOLS AND VISUAL STUDIO
 - LOWEST COMPATIBLE VERSION WITH .NET 4.6.1
- 1.6
 - SHIPPED WITH .NET CORE 1.0
 - HIGHEST VERSION SUPPORTED BY .NET CORE 1.0
 - .NET CORE 1.X SDK ORIGINALLY INCOMPATIBLE WITH .NET 4.6.1
- 2.0
 - RELEASED ON AUG 9,2017
 - COMMAND LINE TOOLS
 - TARGET 2.0 BY DEFAULT AFTER .NET CORE 2.0 SDK INSTALLATION
 - VISUAL STUDIO
 - SET TARGET TO 2.0 AFTER .NET CORE 2.0 SDK AND VS TEMPLATE INSTALLATION.
 - SUPPORTED BY .NET CORE 2.0
 - **.NET FRAMEWORK 4.6.1 IS NOW COMPATIBLE WITH .NET STANDARD**

2.0

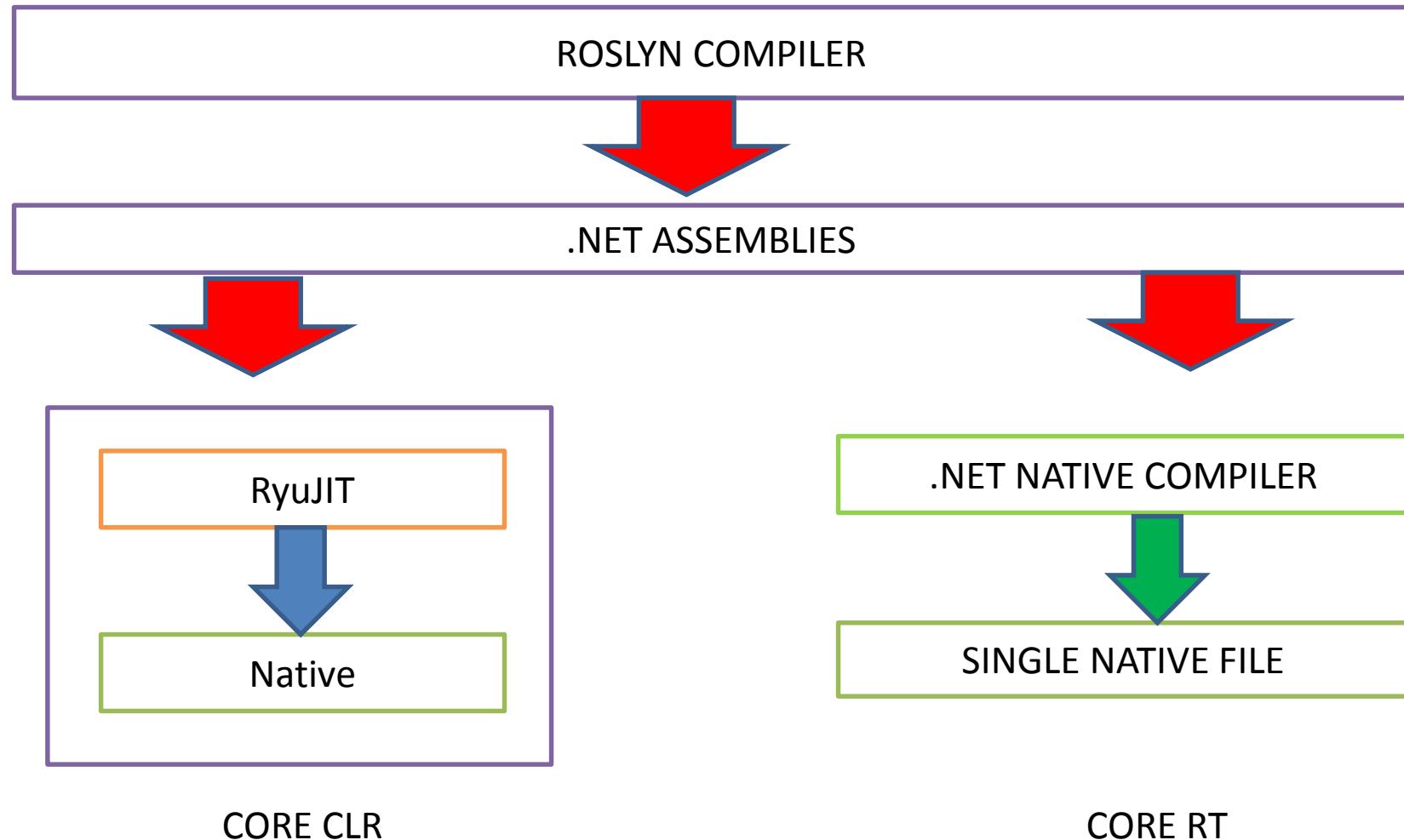
.NET CORE

- MICROSOFT'S CROSS PLATFORM IMPLEMENTATION OF .NET
- Supported multiple operating systems
 - Windows
 - Linux
 - macOS
- Supported on multiple processors
 - X64, X86 on windows, ARM64, ARM32
- Open Source contribution by Microsoft through .NET Development Foundation.
- Components
 - Uses the CoreCLR runtime
 - Implements the .NET standard library
- Languages C#,F#,Visual Basic

COMPILER

Please read terms and conditions of use

Original Series



CORE CLR

CORE RT

FRAMEWORK CHOICES

Please read terms and conditions of use

Original Series

.NET FRAMEWORK	MONO	.NET CORE
MACHINE WIDE	MACHINE WIDE	PER APPLICATION
EXISTING CODE	EXISTING CODE	NEW CODE
MANY TYPES	MANY TYPES	LIMITED TYPES
WINDOWS ONLY	CROSS-PLATFORM	CROSS-PLATFORM

.NET CORE PROJECTS

<https://blog.rendle.io/a-guide-to-the-net-projects-on-github/>

Please read terms and conditions of use

Original Series

CORE FX

- FOUNDATION LIBRARIES FOR .NET CORE
- PLATFORM NEUTRAL

CORE CLR

- .NET CORE RUNTIME IMPLEMENTATION
- WRITTEN IN C/C++
- PLATFORM SPECIFIC

CORE RT

- RUNTIME OPTIMIZED FOR AHEAD OF TIME (AOT) COMPILATION
- TRANSPILES .NET CODE INTO NATIVE C++
- ALLOWS .NET CODE TO RUN WITHOUT THE .NET FRAMEWORK

.NET CORE PROJECTS

<https://blog.rendle.io/a-guide-to-the-net-projects-on-github/>

Please read terms and conditions of use

Original Series

CLI

- COMMAND LINE INTERFACE
- PACKAGES TO CREATE PROJECTS, INSTALL PACKAGES, BUILD AND RUN APPLICATIONS.

LLILC

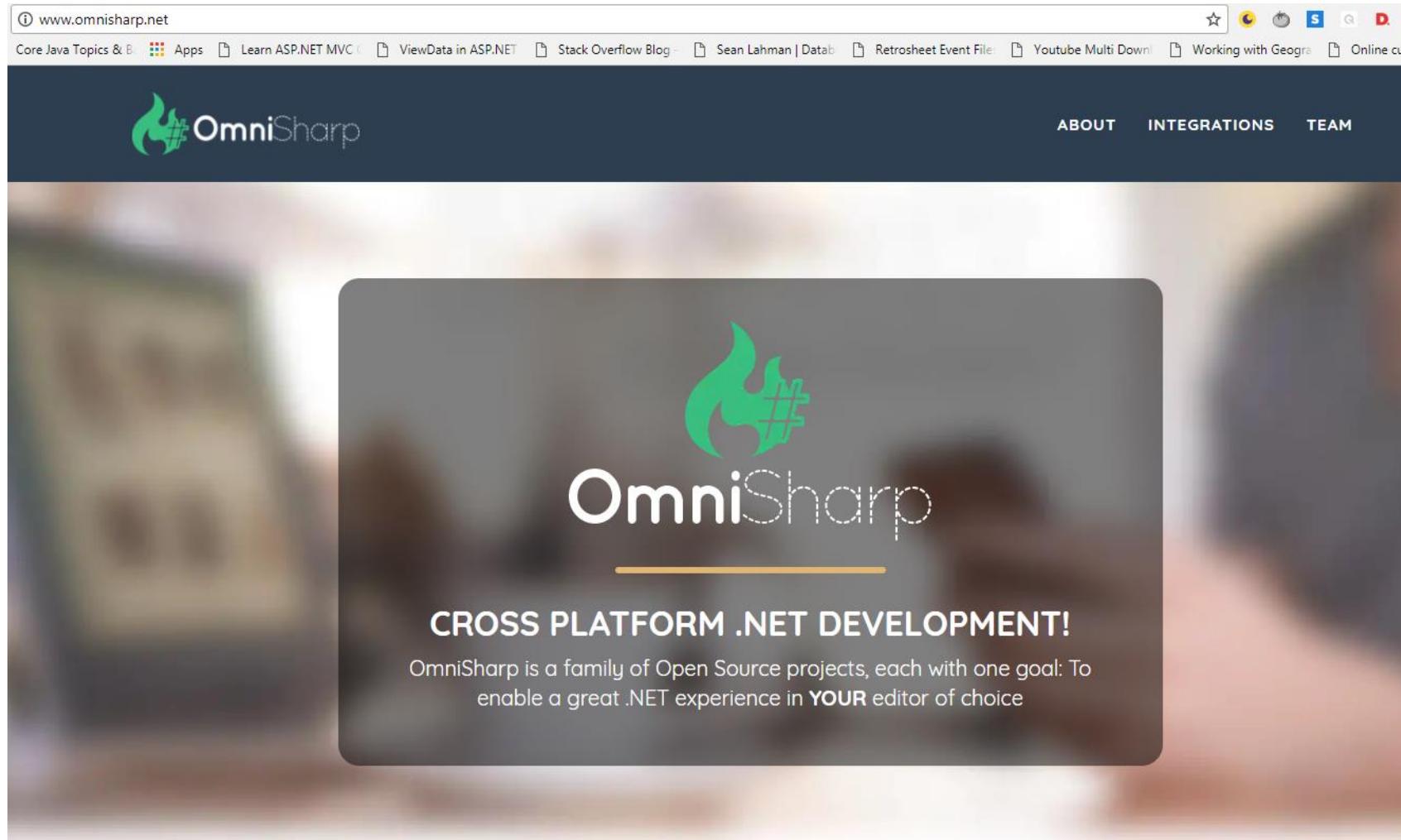
- LLVM MSIL COMPILER IS AN ALTERNATIVE COMPILER FOR TURNING .NET BYTECODE INTO MACHINE CODE USING THE LLVM COMPILER INFRASTRUCTURE.

ROSLYN

- THE C# AND VB COMPILERS AND TOOLS
- APACHE 2.0 LICENSE.
- AN OPEN ,CROSS-PLATFORM COMPILER FRAMEWORK IS FUNDAMENTAL TO ALL THE OTHER .NET CORE PROJECTS.

OMNISHARP

<http://www.omnisharp.net/>



Please read terms and conditions of use

Original Series

DOTNET.EXE

Please read terms and conditions of use

Original Series

- Driver for executing .NET Core commands
- Host for .NET core applications
 - Platform-specific exe for executing .NET core programs.
- MSBUILD is now the foundation
 - Visual Studio 2017 uses *csproj*
 - *Dotnet.exe commands often forward to MSBUILD equivalent.*

.NET CORE 2.0 SDK

<https://www.microsoft.com/net/download/windows>

The screenshot shows the Microsoft .NET Downloads page. At the top, there's a navigation bar with links like Microsoft 365, Azure, Office 365, Dynamics 365, SQL, Windows 10, and More. Below that is a purple header bar with links for .NET, Downloads, Learn, Architecture, Docs, Customers, Community, and Support. The main content area has a title "Downloads" and a sub-section for "Windows". It features logos for Visual Studio, .NET Core, and .NET Framework. Below each logo is a brief description and a "Download" button.

Downloads

Not sure where to start? See [Get started with .NET in 10 minutes.](#)

Windows

Linux

macOS



Visual Studio

Integrated Development Environment (IDE) for developing .NET apps on Windows. Includes .NET Core and .NET Framework.

[Download Visual Studio](#)



.NET Core SDK

Cross-platform .NET implementation. The smallest download to build .NET apps, using command line tools and any editor.

[Download .NET Core 2.1.4 SDK](#)



.NET Framework

Windows-only .NET implementation. Run existing .NET apps on Windows.

[Download .NET Framework 4.7.1](#)

Other Windows Downloads

© TPRI/SYCLIQ -Syed Awase 2017 C# GROUND UP SERIES

.NET CORE SDK INSTALLATION

C:\Program Files\dotnet

The screenshot shows the Windows File Explorer interface with the path 'Computer > Local Disk (C:) > Program Files > dotnet'. The table lists the following files and folders:

	Name	Date modified	Type	Size
	additionalDeps	2/12/2018 3:52 PM	File folder	
	host	1/22/2018 11:49 PM	File folder	
	sdk	2/12/2018 3:49 PM	File folder	
	shared	1/22/2018 11:49 PM	File folder	
	store	2/12/2018 3:52 PM	File folder	
	swidtag	2/12/2018 3:49 PM	File folder	
	dotnet	12/20/2017 4:45 PM	Application	140 KB
	LICENSE	12/20/2017 4:30 PM	Text Document	10 KB
	ThirdPartyNotices	12/20/2017 4:30 PM	Text Document	9 KB

The screenshot shows a Windows Command Prompt window titled 'CMD.exe' with the path 'C:\Windows\system32\CMD.exe'. The output of the 'dotnet' command is displayed:

```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\syedawase>dotnet

Usage: dotnet [options]
Usage: dotnet [path-to-application]

Options:
  -h|--help           Display help.
  --version          Display version.

path-to-application:
  The path to an application .dll file to execute.

C:\Users\syedawase>
```

- Dotnet /?
- Dotnet –version
- Dotnet –info
- Dotnet --diagnostics

Dotnet sdk commands

Please read terms and conditions of use

Original Series

Commands	description
New	Initialize .NET projects
Restore	Restore dependencies specified in the .NET project
Run	Compiles and immediately executes a .NET Project
Build	Builds a .NET Project
Publish	Publishes a .NET project for deployment including the runtime.
Test	Runs unit tests using the test runner specified in the project.
Pack	Creates a NuGet package
Migrate	Migrates a project.json based project to a msbuild based project.

Dotnet sdk commands

Please read terms and conditions of use

Original Series

Commands	description
Clean	Clean build output
Sln	Modify solution files
Add	Add reference to the project
Remove	Remove reference from the project
List	List reference in the project
Nuget	Provides additional NuGet Commands
Msbuild	Runs Microsoft build engine <MSBUILD>
Vstest	Runs Microsoft test execution command line tool
-v -verbosity	Set the verbosity level of the command, normal/diagnostic

Dotnet new

```
PS C:\Users\syedawase> dotnet new  
Usage: new [options]
```

Options:

-h, --help	Displays help for this command.
-l, --list	Lists templates containing the specified name. If no name is specified, lists all t
-n, --name	The name for the output being created. If no name is specified, the name of the cur
is used.	
-o, --output	Location to place the generated output.
-i, --install	Installs a source or a template pack.
-u, --uninstall	Uninstalls a source or a template pack.
--type	Filters templates based on available types. Predefined values are "project", "item"
--force	Forces content to be generated even if it would change existing files.
-lang, --language	Specifies the language of the template to create.

Templates

	Short Name	Language	Tags
Console Application	console	[C#], F#, VB	Common/Console
Class library	classlib	[C#], F#, VB	Common/Library
Unit Test Project	mstest	[C#], F#, VB	Test/MSTest
xUnit Test Project	xunit	[C#], F#, VB	Test/xUnit
ASP.NET Core Empty	web	[C#], F#	Web/Empty
ASP.NET Core Web App (Model-View-Controller)	mvc	[C#], F#	Web/MVC
ASP.NET Core Web App	razor	[C#]	Web/MVC/Razor Pages
ASP.NET Core with Angular	angular	[C#]	Web/MVC/SPA
ASP.NET Core with React.js	react	[C#]	Web/MVC/SPA
ASP.NET Core with React.js and Redux	reactredux	[C#]	Web/MVC/SPA
ASP.NET Core Web API	webapi	[C#], F#	Web/WebAPI
global.json file	globaljson		Config
NuGet Config	nugetconfig		Config
Web Config	webconfig		Config
Solution File	sln		Solution
Razor Page	page		Web/ASP.NET
MVC ViewImports	viewimports		Web/ASP.NET
MVC ViewStart	viewstart		Web/ASP.NET

Examples:

```
dotnet new mvc --auth Individual  
dotnet new razor --auth Individual  
dotnet new --help
```

```
PS C:\Users\syedawase>
```

KESTREL:HTTP SERVER

- Open source web server for ASP.NET based on libUV (also used by NodeJS)
- Fast and production ready
- Fully featured web server.
- Recommended to run it behind a more fully featured webserver like IIS or NGNIX
- Can be run behind IIS using the ASP.NET Core Module (Native IIS Module)

- Maps a HTTP Request to the HttpContext.

	IIS	Kestrel
Platform Support	Windows	Windows/Linux/Mac
Static Files	Yes	Yes
HTTP Access Logs	Yes	No
Port Sharing / Multiple apps*	Yes	No
SSL Certificates	Yes	Internal**
Windows Authentication	Yes	No
Management Console	Yes	No
Process Activation (start it up)	Yes	No
Application Initialization (warm it up)	Yes	No
Configuration API	Yes	No
Request Filtering & Limits	Yes	No
IP & Domain Restrictions	Yes	No
HTTP Redirect Rules	Yes	No
WebSocket Protocol	Yes	Middleware
Response Output Caching	Yes	No
Compression	Optional	Optional
FTP Server	Yes	No

EXERCISE DEMO: 1.1

LEARNING OUTCOMES

- Creating a basic dotnet core 2 **console application**
- Understanding the folder structure
- Building dotnet and deploying a dotnet core 2 console application.

- Use case 1
- Use case 2
- Use case 3
- Use case 4

USE CASES

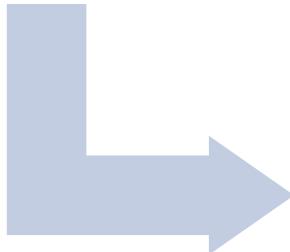
.NET CORE CONSOLE APP:CREATE/RESTORE

Please read terms and conditions of use

Original Series

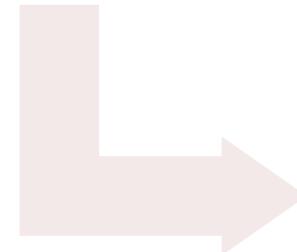
1

- Create a new console application using dotnet core 2.0
- **Dotnet new console**



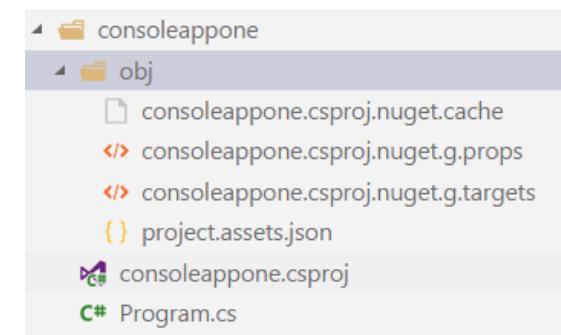
2

- Runs post-creation actions
- **Dotnet restore (in dotnet 1.x we had to explicitly call this)**



3

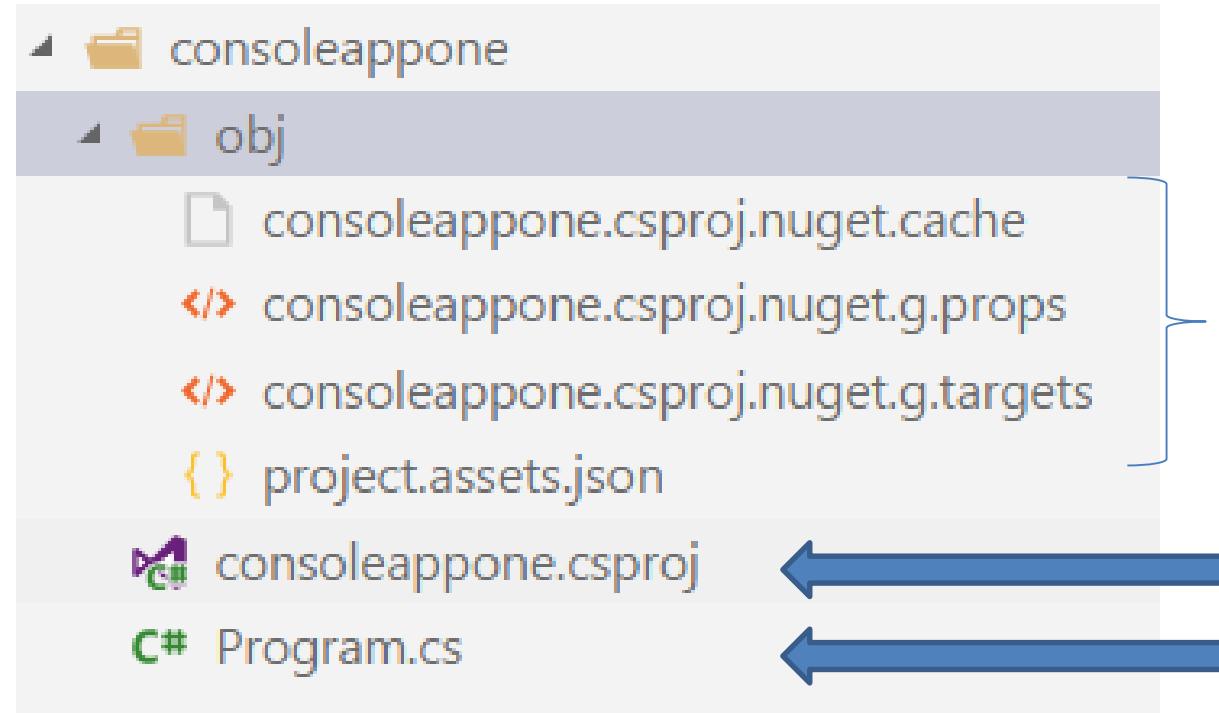
- Creates the following folder structure
 - Obj
 - projectName.csproj
 - Program.cs



.NET CORE 2.0 CONSOLE APP

Please read terms and conditions of use

Original Series



- Project manifest files
- Nuget project properties
- Indicates output type
- Target framework information

```
using System;

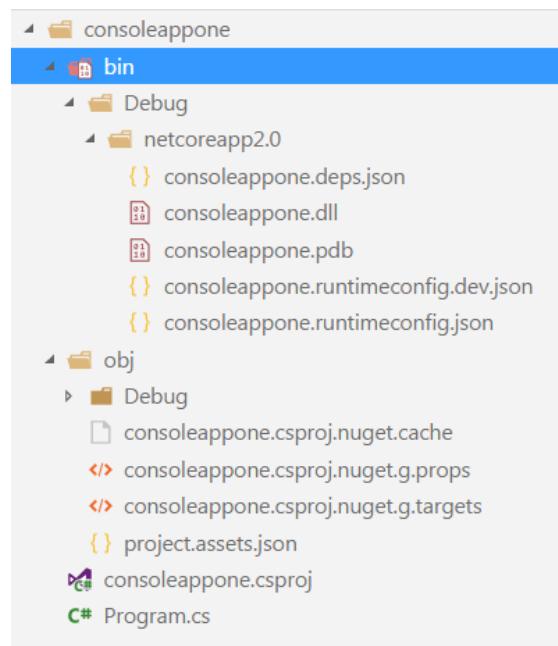
namespace consoleappone
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hello World!");
        }
    }
}
```

.NET CORE 2.0 CONSOLE APP:BUILD

Please read terms and conditions of use

Original Series

- Building your dotnet core application
 - dotnet build



Created post build operation

```
PS H:\awase-harddisk\CT\C#2017\DOTNETCORE\dotnetcore-appscenarios\ConsoleAppOne> dotnet build
Microsoft (R) Build Engine version 15.5.180.51428 for .NET Core
Copyright (C) Microsoft Corporation. All rights reserved.

Restore completed in 42.23 ms for H:\awase-harddisk\CT\C#2017\DOTNETCORE\dotnetcore-appscenarios\ConsoleAppOne.csproj.
  consoleappone -> H:\awase-harddisk\CT\C#2017\DOTNETCORE\dotnetcore-appscenarios\ConsoleAppOne\ConsoleAppOne.dll

Build succeeded.

  0 Warning(s)
  0 Error(s)

Time Elapsed 00:00:12.55
```

- Performs a debug build of the code.
- Use “-c” to build a “release build” of the application.
- .NET core application is a dll

.NET CORE 2.0 CONSOLE APP:RUN

- Running your dotnet console app using
 - dotnet
./bin/Debug/netcoreapp2.0/app.dll
- Alternatively navigating to the project folder and running it
 - dotnet run

```
PS H:\awase-harddisk\CT\C#2017\DOTNETCORE\dotnetcore-appscenarios\consoleappone> dotnet ./bin\Debug\netcoreapp2.0\consoleappone.dll
Hello World!
```

```
PS H:\awase-harddisk\CT\C#2017\DOTNETCORE\dotnetcore-appscenarios\consoleappone> dotnet run
Hello World!
```

EXERCISE DEMO: 1.2

LEARNING OUTCOMES

- Creating a dotnet core 2 **class library**
- Understand the basic structure of the dotnet core 2 class library application
- Building and executing your application

- Use case 1
- Use case 2
- Use case 3
- Use case 4

USE CASES

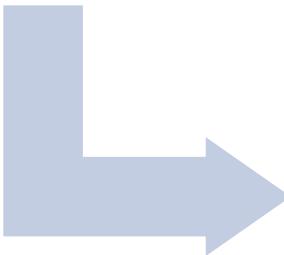
.NET CORE CLASS LIBRARY:CREATE

Please read terms and conditions of use

Original Series

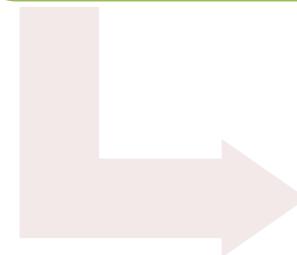
1

- Create a new class library using dotnet core 2.0
- **Dotnet new classlib**



2

- Runs post-creation actions
- **Dotnet restore (in dotnet 1.x we had to explicitly call this)**



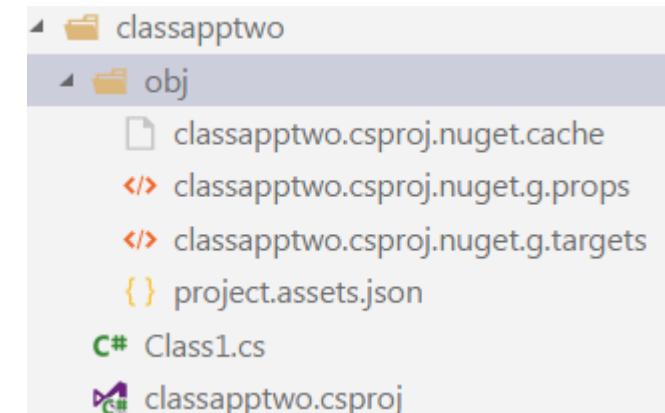
3

- Creates the following folder structure
 - Obj
 - projectName.csproj
 - Class1.cs

```
PS H:\awase-harddisk\CT\C#2017\DOTNETCORE\dotnetcore-appscenarios\classapptwo> dotnet new classlib  
The template "Class library" was created successfully.
```

```
Processing post-creation actions...  
Running 'dotnet restore' on H:\awase-harddisk\CT\C#2017\DOTNETCORE\dotnetcore-appscenarios\classapptwo\classapptwo.csproj...  
Restoring packages for H:\awase-harddisk\CT\C#2017\DOTNETCORE\dotnetcore-appscenarios\classapptwo\classapptwo.csproj...  
Generating MSBuild file H:\awase-harddisk\CT\C#2017\DOTNETCORE\dotnetcore-appscenarios\classapptwo\obj\classapptwo.csproj.nuget.g.props.  
Generating MSBuild file H:\awase-harddisk\CT\C#2017\DOTNETCORE\dotnetcore-appscenarios\classapptwo\obj\classapptwo.csproj.nuget.g.targets.  
Restore completed in 619.79 ms for H:\awase-harddisk\CT\C#2017\DOTNETCORE\dotnetcore-appscenarios\classapptwo\classapptwo.csproj.
```

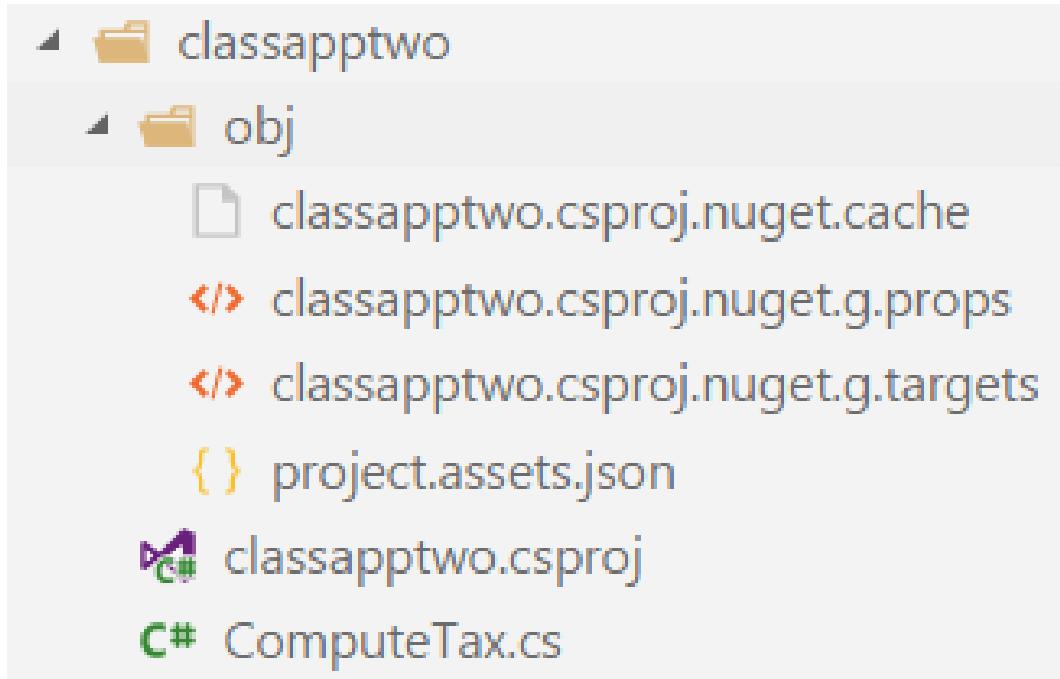
```
Restore succeeded.
```



.NET Core Class: Compute Tax

Please read terms and conditions of use

Original Series



```
using System;

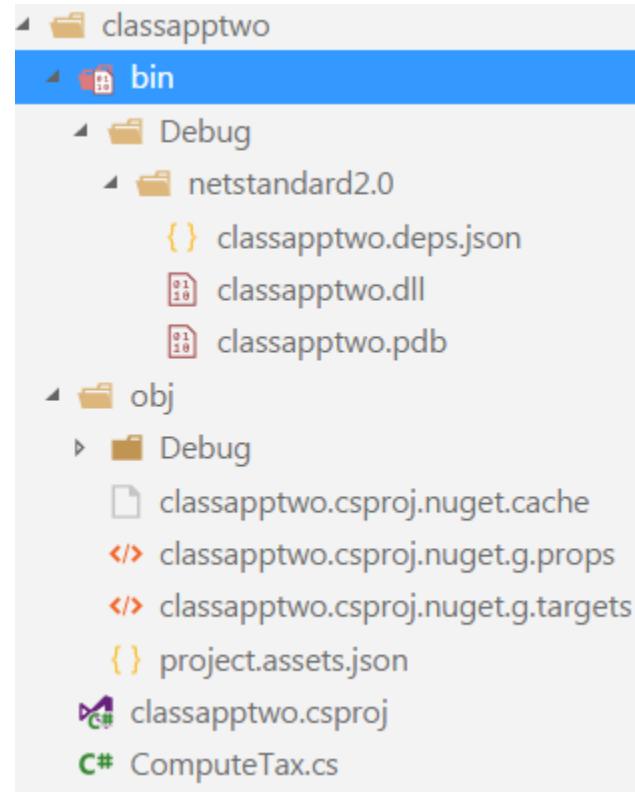
namespace classapptwo
{
    public class ComputeTax
    {
        public int StateTax(int amount, int sgstrate){
            return amount*sgstrate/100;
        }

        public int CentralTax(int amount; int cgstrate){
            return amount*cgstrate/100;
        }
    }
}
```

- Build using => dotnet build

Building your class library

- Building the class library
 - dotnet build



```
PS H:\awase-harddisk\CT\C#2017\DOTNETCORE\dotnetcore-appscenarios\classapptwo> dotnet build
Microsoft (R) Build Engine version 15.5.180.51428 for .NET Core
Copyright (C) Microsoft Corporation. All rights reserved.

Restore completed in 33.49 ms for H:\awase-harddisk\CT\C#2017\DOTNETCORE\dotnetcore-appscenarios\classapptwo\proj.csproj.

classapptwo -> H:\awase-harddisk\CT\C#2017\DOTNETCORE\dotnetcore-appscenarios\classapptwo\classapptwo.dll

Build succeeded.
  0 Warning(s)
  0 Error(s)

Time Elapsed 00:00:04.52
```

EXERCISE DEMO: 1.3

LEARNING OUTCOMES

- Creating a dotnet core 2 **solution to consume the class library created.**
- Understand the process and structure of the solution
- Build and execute the solution.

- Use case 1
- Use case 2
- Use case 3
- Use case 4

USE CASES

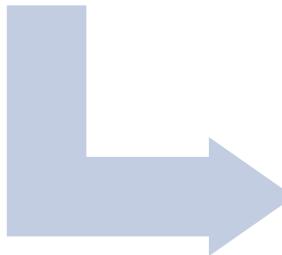
.NET CORE SOLUTION:CREATE

Please read terms and conditions of use

Original Series

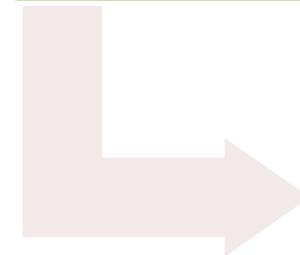
1

- Create a new solution using dotnet core 2.0
- **Dotnet new sln**



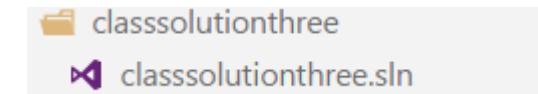
2

- Creates a solution file



3

- Just creates a solution file



Adding Projects to the Solution

- Adding existing projects (console application and class library) to the solution
 - `dotnet sln add ..\project\project.csproj`

```
PS H:\awase-harddisk\CT\C#2017\DOTNETCORE\dotnetcore-appscenarios\classsolutionthree> dotnet sln add ..\consoleappone\consoleappone.csproj
Project `..\consoleappone\consoleappone.csproj` added to the solution.
PS H:\awase-harddisk\CT\C#2017\DOTNETCORE\dotnetcore-appscenarios\classsolutionthree> dotnet sln add ..\classapptwo\classapptwo.csproj
Project `..\classapptwo\classapptwo.csproj` added to the solution.
PS H:\awase-harddisk\CT\C#2017\DOTNETCORE\dotnetcore-appscenarios\classsolutionthree> █
```

Adding a reference to the library

- Adding reference to the solution.
 - dotnet add reference ./prj/prj.csproj

Usage: dotnet add <PROJECT> reference [options] <args>

```
PS H:\awase-harddisk\CT\C#2017\DOTNETCORE\dotnetcore-appscenarios\classsolutionthree> dotnet add ..\consoleappone reference ..\classapptwo\classapptwo.csproj
Reference `..\classapptwo\classapptwo.csproj` added to the project.
```

```
<Project Sdk="Microsoft.NET.Sdk">
  <ItemGroup>
    <ProjectReference Include="..\classapptwo\classapptwo.csproj" />
  </ItemGroup>
  <PropertyGroup>
    <OutputType>Exe</OutputType>
    <TargetFramework>netcoreapp2.0</TargetFramework>
  </PropertyGroup>
</Project>
```

Consoleapp.csproj

Consuming the class library

Please read terms and conditions of use

Original Series

```
using System;
using classapptwo;

namespace consoleappone
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hello World!");
            ComputeTax ct=new ComputeTax();
            var mytax =ct.StateTax(20000,18);
            Console.WriteLine(mytax);
        }
    }
}
```

- Building the solution
 - dotnet build
- Executing the project
 - Cd consoleappone
 - Dotnet run

```
PS H:\awase-harddisk\CT\C#2017\DOTNETCORE\dotnetcore-appscenarios
\consoleappone> dotnet run
Hello World!
3600
```

Adding package to existing project

```
dotnet add package Microsoft.AspNetCore
```

- Used to add additional functionality to the existing project to render web application functionality.

EXERCISE DEMO: 1.4

LEARNING OUTCOMES

- Creating .NET core apps with Visual Studio 2017.
- Install .NET core 2.0 SDK
 - <http://www.Microsoft.com/net/download/core>
- Install project templates through Visual Studio Installer
 - .NET Core Cross-platform development
 - ASP.NET and Web Development

- Use case 1
- Use case 2
- Use case 3
- Use case 4

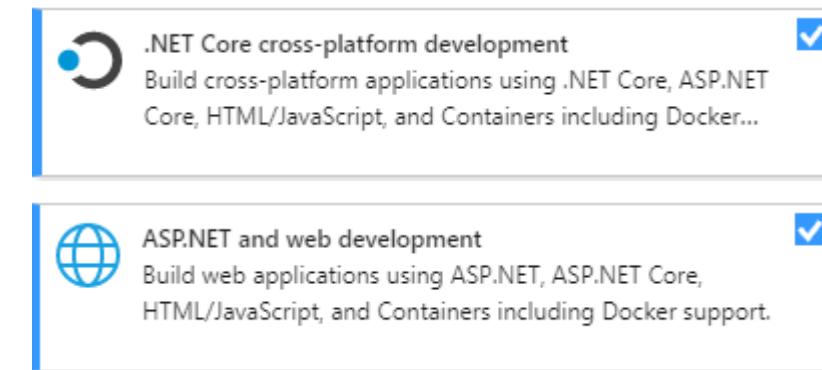
USE CASES

Installing .NET CORE for Visual Studio 2017

Please read terms and conditions of use

Original Series

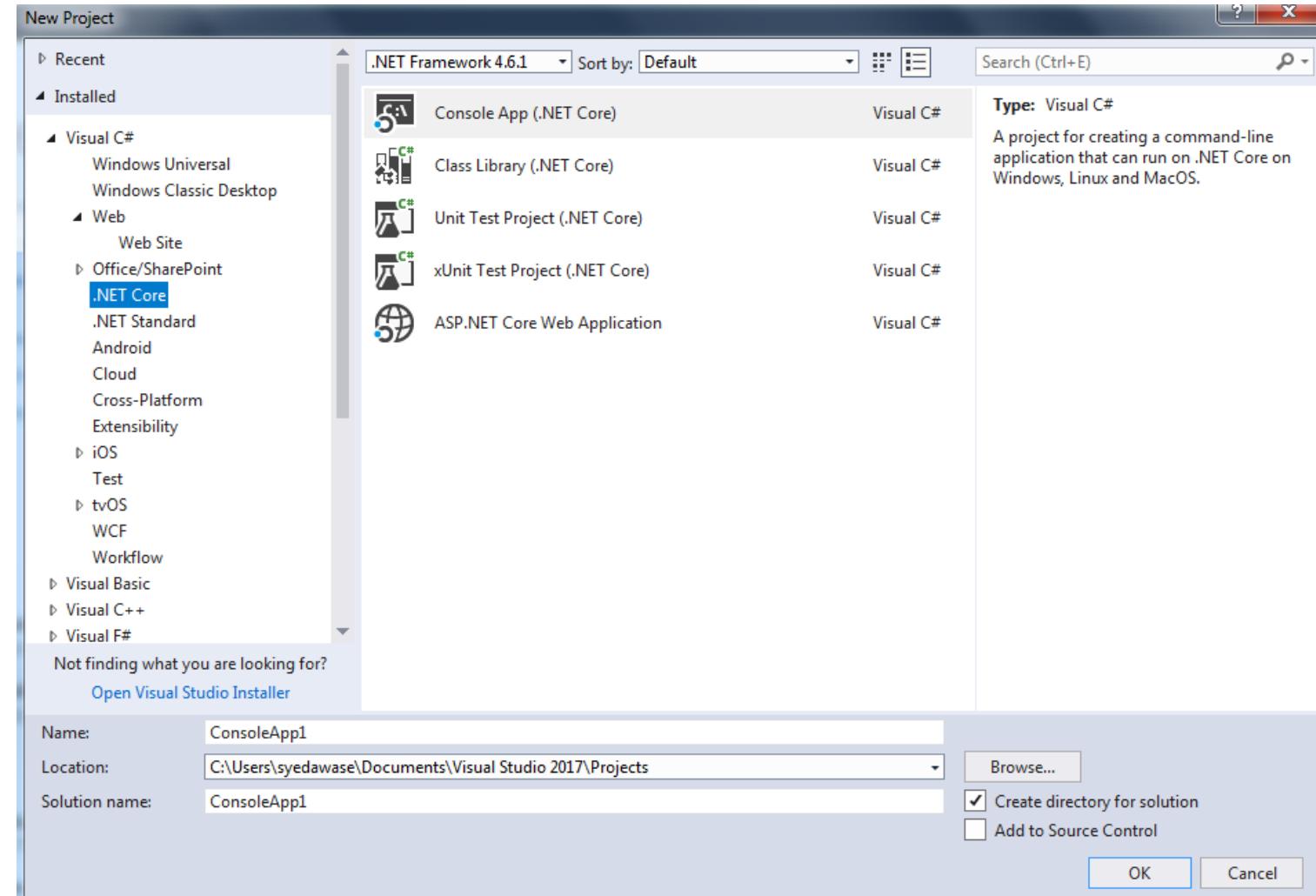
1. Open Visual Studio Installer
2. Modify Installation
3. Select
 1. .NET CORE CROSS-PLATFORM DEVELOPMENT
 2. ASP.NET and Web Development



Creating a new .NET Core Application in VS2017

Please read terms and conditions of use

Original Series



NuGet Packages

- Fine-grained packages
 - Represent one assembly whose name is the same as package.
 - Mostly independent of other packages.
 - Support for different platforms
- Packages support multiple frameworks.
- NuGet Metapackages
 - Represents a set of related packages.
- Each framework implicitly references a metapackage
- .NETStandard.Library
 - Implicitly referenced by .NET standard framework
 - Ability to run on multiple runtimes.
- Microsoft.NETCore.App
 - Implicitly referenced by .NET CORE
 - Dependent on NETStandard.Library

Packaging your library as NuGet Package.

<https://docs.microsoft.com/en-us/nuget/quickstart/create-and-publish-a-package-using-visual-studio>

- Create a NuGet package
 - A standard approach for distribution of libraries in .NET Core
 - **dotnet pack**
 - Creates a zip file with **.nupkg** extension
- Publishing the NuGet package
 - Register/Sign in to www.nuget.org
 - Upload the nugget package file
- Configure the owners of the package.

EXERCISE DEMO: 1.5

LEARNING OUTCOMES

- Deployment in .Net Core application

- Use case 1
- Use case 2
- Use case 3
- Use case 4

USE CASES

Two types of Deployment in .NET CORE

Framework-dependent Deployment FDD

- Assumes that .NET Core installed on the target machine.
- Contains only the application and third party library dependency
- dotnet publish –c Release
 - Creates a publish folder in bin/Release
 - Contains .dlls for app and dependent assemblies.
 - Contains pdbs for app and dependent assemblies.

Self-contained Deployment (SCD)

- Contains .NET Core libraries and runtime.
- Contains the application and third party library dependencies.
- Developers need to specify the target platform where the app will run
 - Add RuntimeIdentifier element
 - Add to the PropertyGroup section of csproj file
 - Indicate platform using Runtime Identifier(RID)
 - [os].[version]-[architecture]

Self Contained Deployment

- Execute the publish command for each RID
 - Dotnet publish –c Release –r [RID]
- Creates a folder named publish in bin/release/netcoreapp2.0/RID with
 - Renamed version of dotnet.exe
 - Which is used to launch the apps
 - Named the same as the application name on Windows.
 - .NET Core binaries and runtime.
 - Dlls for app and dependent assemblies
 - Pdbs for app and dependent assemblies.

FDD vs SCD

Framework-dependent Deployment (FDD)

- Advantages
 - Smaller deployment package
 - Single .NET Core installation shared by multiple apps on target machine
 - Target platform does not need to be specified in advance.
- Disadvantages
 - Correct version of .NET Core must be present on the target machine.
 - Application behavior is dependent on the version.

Self-contained Deployment (SCD)

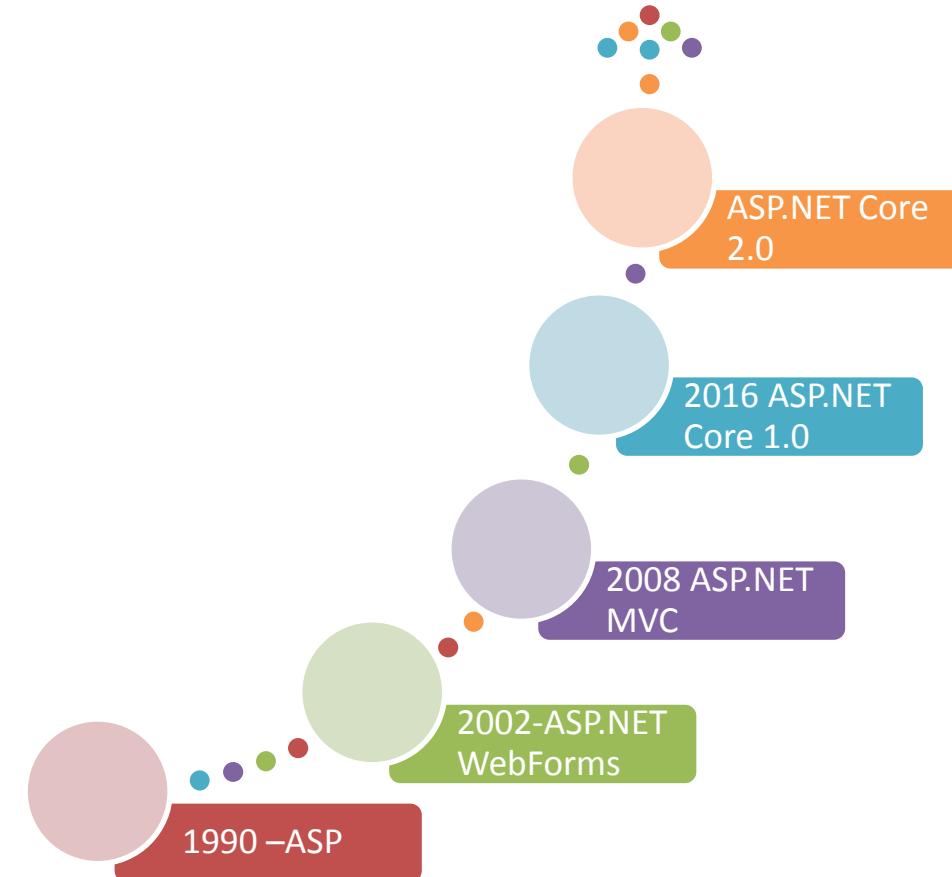
- Advantages
 - Not dependent on what is installed on the target machine
 - Guaranteed that application will run as intended.
- Disadvantages
 - Large deployment package
 - No sharing of .NET Core installations by multiple apps on target machine.
 - Target platform must be specified in advance.

SECTION -XVIII

ASP.NET CORE 2.0

ASP.NET Core Evolution

Please read terms and conditions of use



Original Series

ASP.NET CORE 2 Features

- Cross-platform support
- Micro service architecture
- Working with docker and containers
- Performance and scalability
- Side-by-side deployments
- Technology restrictions.
- ASP.NET MVC and Web API have been combined into a single framework.
- Runtime Store : it contains compiled packages, which were compiled using the native machine language and it is key for improved performance.
- Fully supports Razor engine.
- Seamless integration with client-side frameworks including Angular, Knockout and Bootstrap.

ASP.NET CORE Features

- Environment based configuration system ready for cloud hosting
- Built-in dependency injection functionalities.
- Light-weight and modular HTTP request pipeline.
- Host the same application in IIS, self-host, Docker, Cloud
- Side-by-side deployments with older versions.

Not found currently in ASP.NET core

- ASP.NET Web Forms applications.
- ASP.NET Web Pages applications.
- WCF Services , it only has support for WCF Client
- WorkFlow services: Windows Workflow foundation, workflow Services and WCF data services are not supported.
- WPF and Windows forms applications.

When to choose ASP.NET CORE 2.0

- Migrating your existing application to ASP.NET CORE 2.0 straight away is not easy. As there is not much support.
- But for new application development, ASP.NET Core 2.0 is ideal choice.
- IOT and Cross Platform Application development ASP.NET CORE would be an ideal choice.
- Specifically targeted microservices.
- Docker containers.
- High performance and highly scalable applications.
- Multiple applications with different .NET versions side by side.

ASP.NET Core 2.0 Empty Web App

EXERCISE DEMO: 2.1

LEARNING OUTCOMES

- Creating a ASP.NET Core Empty web application using CLI

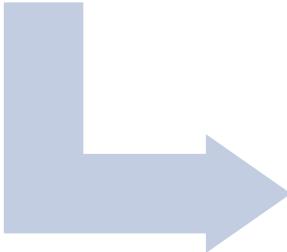
ASP.NET Empty Web Application:Create

Please read terms and conditions of use

Original Series

1

- Create a new solution using dotnet core 2.0
- **Dotnet new web**



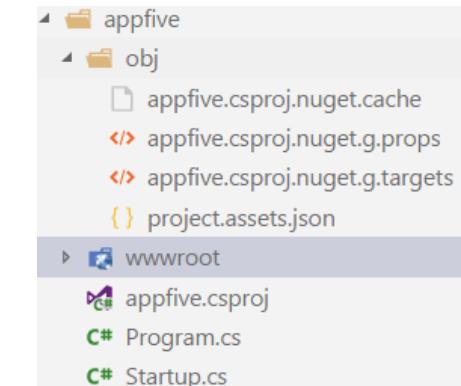
2

- Runs post-creation actions
- **Dotnet restore (in dotnet 1.x we had to explicitly call this)**



3

- Creates an application structure with nuget packages
- wwwroot
- Project.csproj
- Program.cs
- Startup.cs(Global.asax)



```
PS H:\awase-hddisk\CT\C#2017\DOTNETCORE\dotnetcore-appscenarios\appfive> dotnet new web  
The template "ASP.NET Core Empty" was created successfully.
```

```
This template contains technologies from parties other than Microsoft, see https://aka.ms/template-3 for details.
```

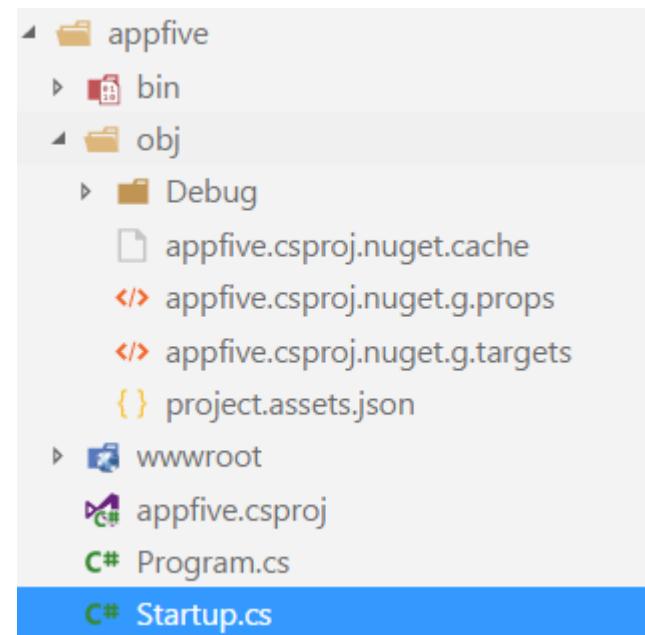
```
Processing post-creation actions...  
Running 'dotnet restore' on H:\awase-hddisk\CT\C#2017\DOTNETCORE\dotnetcore-appscenarios\appfive\appfive.csproj...  
Restoring packages for H:\awase-hddisk\CT\C#2017\DOTNETCORE\dotnetcore-appscenarios\appfive\appfive.csproj...  
Generating MSBuild file H:\awase-hddisk\CT\C#2017\DOTNETCORE\dotnetcore-appscenarios\appfive\appfive.csproj...  
Generating MSBuild file H:\awase-hddisk\CT\C#2017\DOTNETCORE\dotnetcore-appscenarios\appfive\appfive.csproj...  
Restore completed in 29.38 sec for H:\awase-hddisk\CT\C#2017\DOTNETCORE\dotnetcore-appscenarios\appfive\appfive.csproj...
```

```
Restoring packages for H:\awase-hddisk\CT\C#2017\DOTNETCORE\dotnetcore-appscenarios\appfive\appfive.csproj...  
Generating MSBuild file H:\awase-hddisk\CT\C#2017\DOTNETCORE\dotnetcore-appscenarios\appfive\appfive.csproj...  
Generating MSBuild file H:\awase-hddisk\CT\C#2017\DOTNETCORE\dotnetcore-appscenarios\appfive\appfive.csproj...  
Restore completed in 29.38 sec for H:\awase-hddisk\CT\C#2017\DOTNETCORE\dotnetcore-appscenarios\appfive\appfive.csproj...
```

```
Restore succeeded.
```

Building your ASP.NET Core: build

- Building your ASP.NET Core application
 - dotnet build



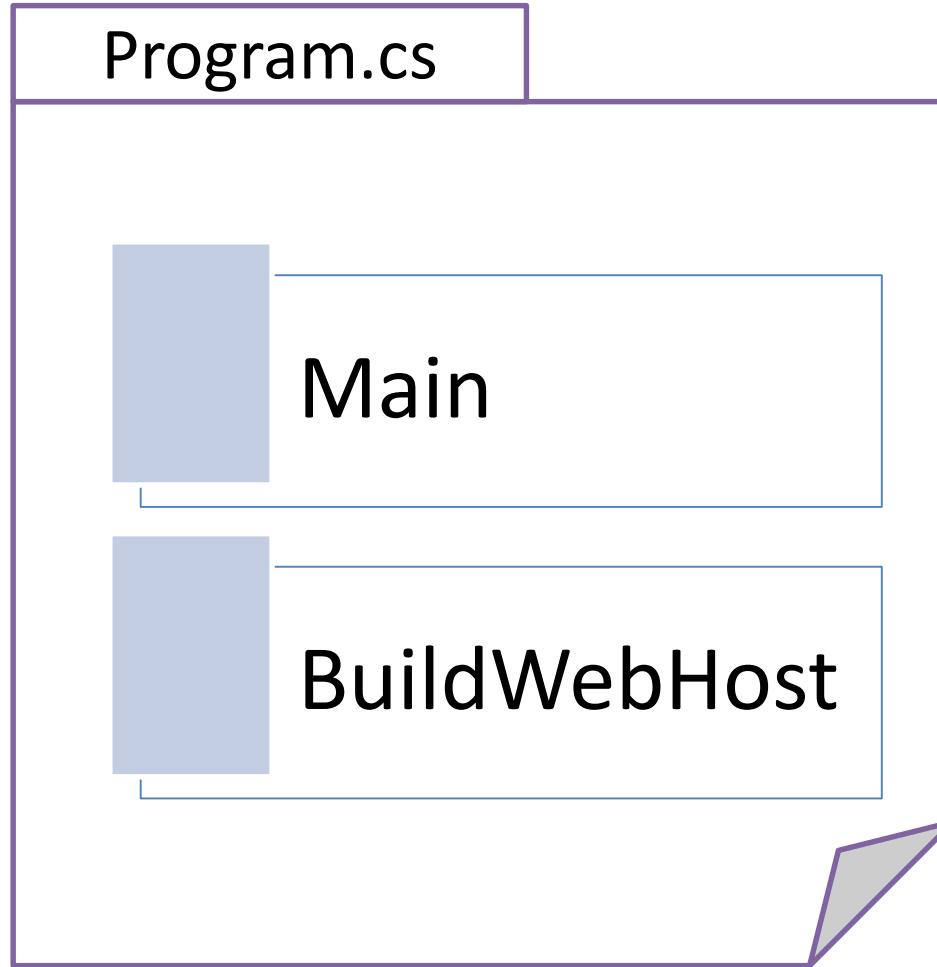
```
PS H:\awase-harddisk\CT\C#2017\DOTNETCORE\dotnetcore-appscenarios\appfive> dotnet build  
Microsoft (R) Build Engine version 15.5.180.51428 for .NET Core  
Copyright (C) Microsoft Corporation. All rights reserved.
```

```
Restore completed in 259.12 ms for H:\awase-harddisk\CT\C#2017\DOTNETCORE\dotnetcore-app  
appfive -> H:\awase-harddisk\CT\C#2017\DOTNETCORE\dotnetcore-appscenarios\appfive\bin\De
```

```
Build succeeded.
```

```
0 Warning(s)  
0 Error(s)
```

Please read terms and conditions of use
Original Series



```
namespace appfive
{
    public class Program
    {
        public static void Main(string[] args)
        {
            BuildWebHost(args).Run();
        }

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

Startup.cs

ConfigureServices

- Register dependencies

Configure

- Request pipeline setup

```
namespace appfive
{
    public class Startup
    {
        // This method gets called by the runtime. Use this method to add services to the container.
        // For more information on how to configure your application, visit https://go.microsoft.com/fwlink/?LinkID=398940
        public void ConfigureServices(IServiceCollection services)
        {

        }

        // This method gets called by the runtime. Use this method to configure the HTTP request pipeline.
        public void Configure(IApplicationBuilder app, IHostingEnvironment env)
        {
            if (env.IsDevelopment())
            {
                app.UseDeveloperExceptionPage();
            }

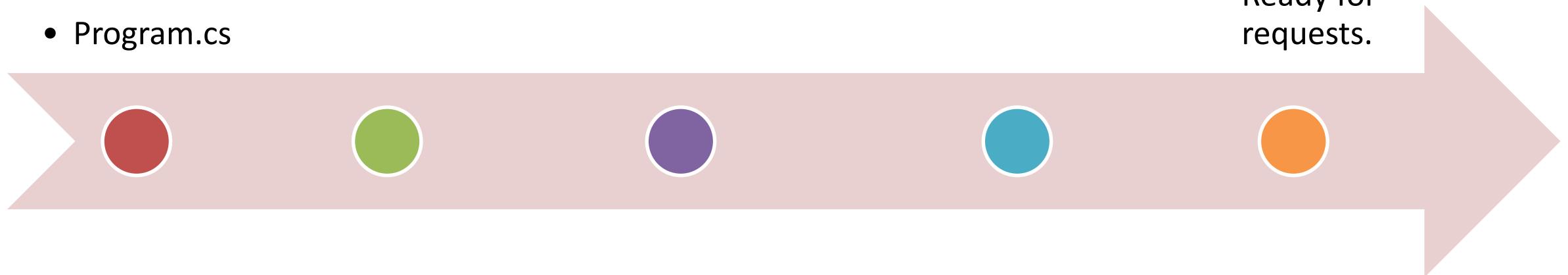
            app.Run(async (context) =>
            {
                await context.Response.WriteAsync("Hello World!");
            });
        }
    }
}
```

Application flow

Please read terms and conditions of use

Original Series

- Program.cs



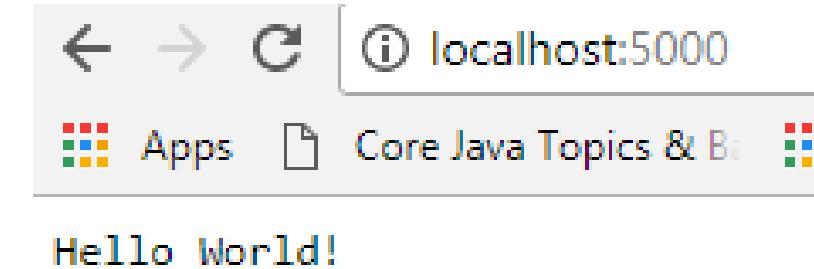
- Startup.cs

- ConfigureServices method
 - Registering Services
- Ready for requests.

- ConfigureMethod
 - Pipeline is created.

Running your ASP.NET Core: run

- Running your ASP.NET Core
 - Dotnet run



```
PS H:\awase-harddisk\CT\C#2017\DOTNETCORE\dotnetcore-appscenarios\appfive> dotnet run
Hosting environment: Production
Content root path: H:\awase-harddisk\CT\C#2017\DOTNETCORE\dotnetcore-appscenarios\appfive
Now listening on: http://localhost:5000
Application started. Press Ctrl+C to shut down.
info: Microsoft.AspNetCore.Hosting.Internal.WebHost[1]
      Request starting HTTP/1.1 GET http://localhost:5000/
info: Microsoft.AspNetCore.Hosting.Internal.WebHost[2]
      Request finished in 144.5773ms 200
info: Microsoft.AspNetCore.Hosting.Internal.WebHost[1]
```

ASP.NET MVC Core 2.0

EXERCISE DEMO: 2.2

LEARNING OUTCOMES

- Creating an ASP.NET Core MVC 2.0 Application
- Understanding the application structure
- Building and executing the ASP.NET Core MVC 2.0 Application

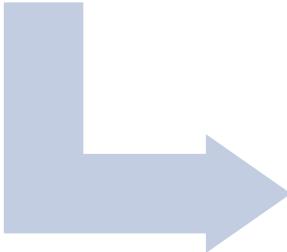
ASP.NET MVC 2.0 Application :Create

Please read terms and conditions of use

Original Series

1

- Create a new solution using dotnet core ASP.NET MVC 2.0
- **Dotnet new mvc**



2

- Runs post-creation actions
- **Dotnet restore (in dotnet 1.x we had to explicitly call this)**



3

- Creates the following application structure

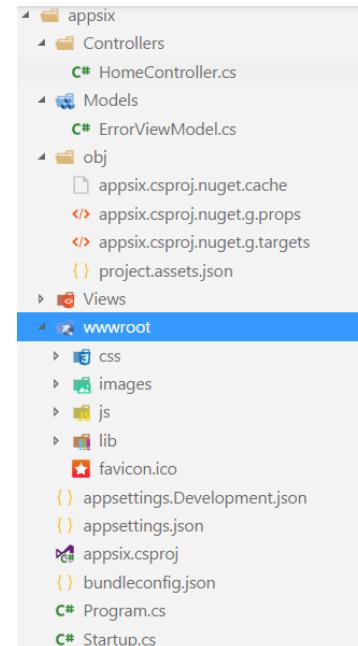
```
PS H:\awase-harddisk\CT\C#2017\DOTNETCORE\dotnetcore-appscenarios\appsix> dotnet new mvc
The template "ASP.NET Core Web App (Model-View-Controller)" was created successfully.
This template contains technologies from parties other than Microsoft, see https://aka.ms/template-3pn for details.

Processing post-creation actions...
Running 'dotnet restore' on H:\awase-harddisk\CT\C#2017\DOTNETCORE\dotnetcore-appscenarios\appsix\appsix.csproj...
Restoring packages for H:\awase-harddisk\CT\C#2017\DOTNETCORE\dotnetcore-appscenarios\appsix\appsix.csproj...
Restoring packages for H:\awase-harddisk\CT\C#2017\DOTNETCORE\dotnetcore-appscenarios\appsix\appsix.csproj...
Generating MSBuild file H:\awase-harddisk\CT\C#2017\DOTNETCORE\dotnetcore-appscenarios\appsix\obj\appsix.csproj.nuget.g.props

Generating MSBuild file H:\awase-harddisk\CT\C#2017\DOTNETCORE\dotnetcore-appscenarios\appsix\obj\appsix.csproj.nuget.g.targets

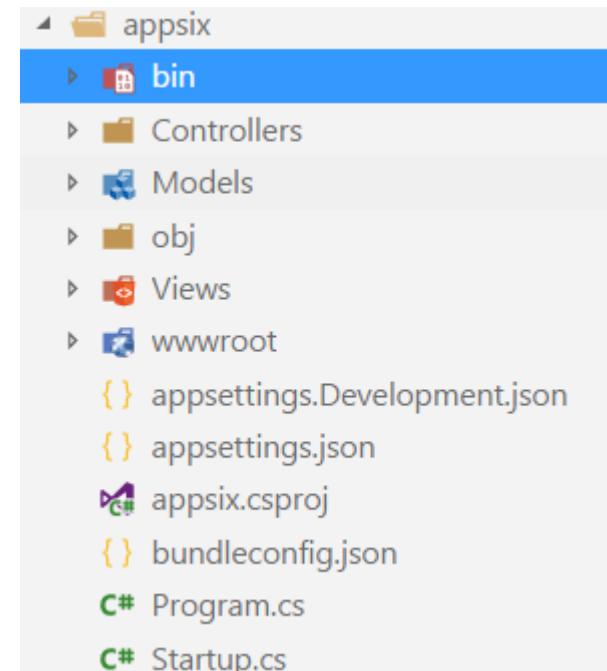
Restore completed in 11.15 sec for H:\awase-harddisk\CT\C#2017\DOTNETCORE\dotnetcore-appscenarios\appsix\appsix.csproj.
Restore completed in 11.42 sec for H:\awase-harddisk\CT\C#2017\DOTNETCORE\dotnetcore-appscenarios\appsix\appsix.csproj.

Restore succeeded.
```



ASP.NET MVC Core 2.0:Build

- Building your ASP.NET MVC Core 2.0 application
 - Dotnet build



```
PS H:\awase-harddisk\CT\C#2017\DOTNETCORE\dotnetcore-appscenarios\appsix> dotnet build  
Microsoft (R) Build Engine version 15.5.180.51428 for .NET Core  
Copyright (C) Microsoft Corporation. All rights reserved.
```

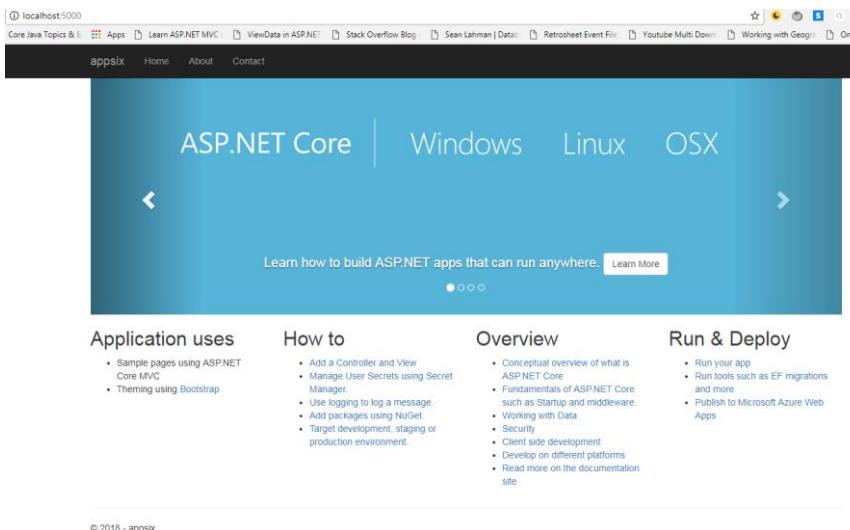
```
Restore completed in 266.57 ms for H:\awase-harddisk\CT\C#2017\DOTNETCORE\dotnetcore  
Restore completed in 189.68 ms for H:\awase-harddisk\CT\C#2017\DOTNETCORE\dotnetcore  
appsix -> H:\awase-harddisk\CT\C#2017\DOTNETCORE\dotnetcore-appscenarios\appsix\bin\
```

Build succeeded.

0 Warning(s)
0 Error(s)

ASP.NET Core MVC 2.0:Run

- Running your ASP.NET Core MVC 2.0 application
 - Dotnet run
 - <http://localhost:5000/>



```
PS H:\awase-harddisk\CT\C#2017\DOTNETCORE\dotnetcore-appscenarios\appsix> dotnet run
Hosting environment: Production
Content root path: H:\awase-harddisk\CT\C#2017\DOTNETCORE\dotnetcore-appscenarios\appsix
Now listening on: http://localhost:5000
Application started. Press Ctrl+C to shut down.
```

ASP.NET MVC Core 2.0 (Razor) EXERCISE DEMO: 2.3

LEARNING OUTCOMES

- Creating an ASP.Net Core 2.0 with razor view engine
- Understanding the application structure
- Building and running your application.

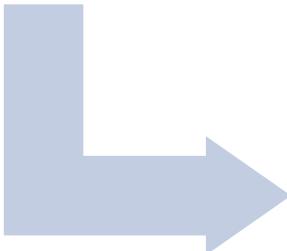
ASP.NET MVC 2.0 Razor Application :Create

Please read terms and conditions of use

Original Series

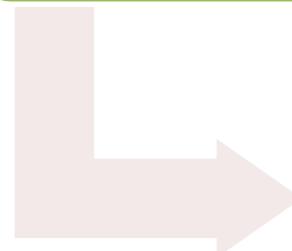
1

- Create a new solution using dotnet core ASP.NET MVC 2.0
- **Dotnet new razor**



2

- Runs post-creation actions
- **Dotnet restore (in dotnet 1.x we had to explicitly call this)**

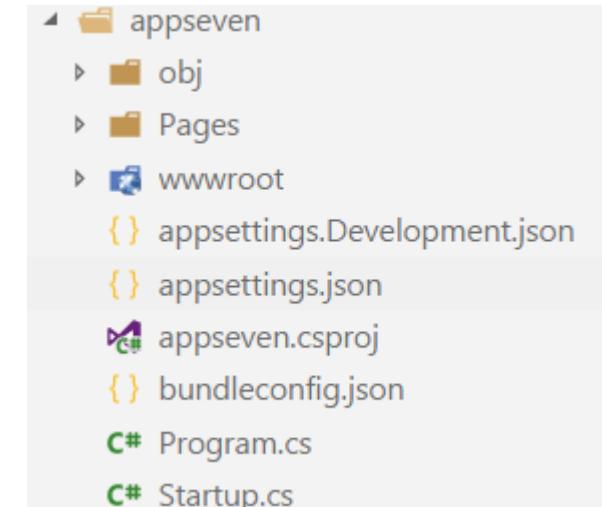


3

- Creates the following application structure

```
PS H:\awase-hddisk\CT\C#2017\DOTNETCORE\dotnetcore-appscenarios\appseven> dotnet new razor
The template "ASP.NET Core Web App" was created successfully.
This template contains technologies from parties other than Microsoft, see https://aka.ms/templa...
Processing post-creation actions...
Running 'dotnet restore' on H:\awase-hddisk\CT\C#2017\DOTNETCORE\dotnetcore-appscenarios\appse...
Restoring packages for H:\awase-hddisk\CT\C#2017\DOTNETCORE\dotnetcore-appscenarios\appseven
Restore completed in 172.03 ms for H:\awase-hddisk\CT\C#2017\DOTNETCORE\dotnetcore-appscenar...
proj.
Generating MSBuild file H:\awase-hddisk\CT\C#2017\DOTNETCORE\dotnetcore-appscenarios\appse...
get.g.props.
Generating MSBuild file H:\awase-hddisk\CT\C#2017\DOTNETCORE\dotnetcore-appscenarios\appse...
get.g.targets.
Restore completed in 4.01 sec for H:\awase-hddisk\CT\C#2017\DOTNETCORE\dotnetcore-appscenar...
roj.

Restore succeeded.
```

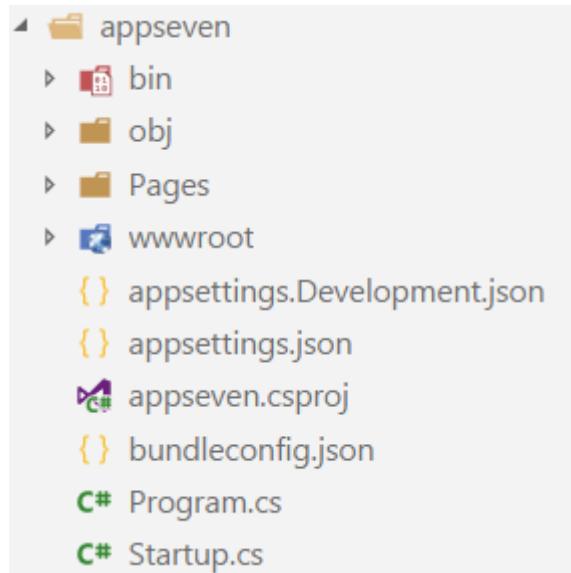


ASP.NET MVC 2.0 Razor Application :Build

Please read terms and conditions of use

Original Series

- Building your ASP.NET MVC 2.0 Razor Application
 - Dotnet build



```
PS H:\awase-harddisk\CT\C#2017\DOTNETCORE\dotnetcore-appscenarios\appseven> dotnet build  
Microsoft (R) Build Engine version 15.5.180.51428 for .NET Core  
Copyright (C) Microsoft Corporation. All rights reserved.
```

```
Restore completed in 186.59 ms for H:\awase-harddisk\CT\C#2017\DOTNETCORE\dotnetcore-ap  
proj.
```

```
Restore completed in 150.59 ms for H:\awase-harddisk\CT\C#2017\DOTNETCORE\dotnetcore-ap  
proj.
```

```
appseven -> H:\awase-harddisk\CT\C#2017\DOTNETCORE\dotnetcore-appscenarios\appseven\bin  
11
```

```
Build succeeded.
```

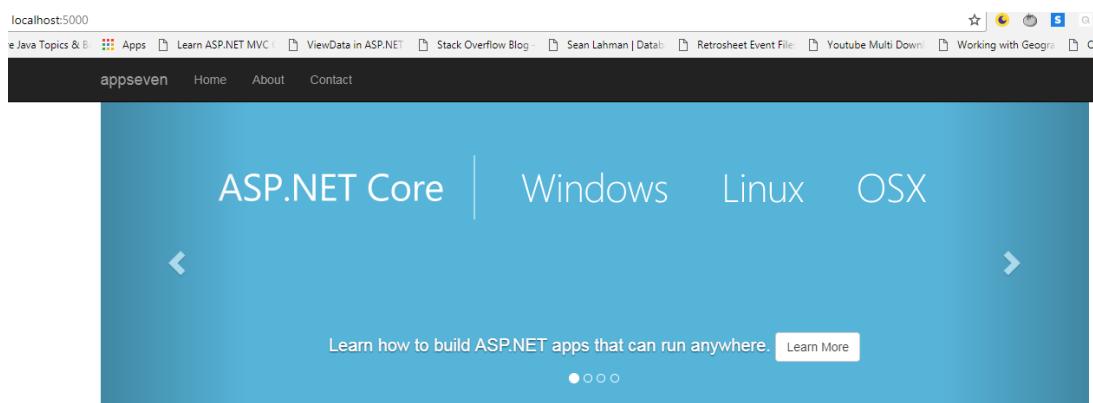
```
0 Warning(s)  
0 Error(s)
```

ASP.NET MVC 2.0 Razor Application :run

Please read terms and conditions of use

- Running your ASP.NET MVC 2.0 razor application
 - Dotnet run

```
PS H:\awase-harddisk\CT\C#2017\DOTNETCORE\dotnetcore-appscenarios\appseven> dotnet run
Hosting environment: Production
Content root path: H:\awase-harddisk\CT\C#2017\DOTNETCORE\dotnetcore-appscenarios\appseven
Now listening on: http://localhost:5000
Application started. Press Ctrl+C to shut down.
```



Original Series

- Sample pages using ASP.NET Core Razor Pages
- Theming using Bootstrap

How to

- Working with Razor Pages
- Manage User Secrets using Secret Manager
- Use logging to log a message
- Add packages using NuGet
- Target development, staging or production environment

Overview

- Conceptual overview of what is ASP.NET Core
- Fundamentals of ASP.NET Core such as Startup and middleware
- Working with Data
- Security
- Client side development

Run & Deploy

- Run your app
- Run tools such as EF migrations and more
- Publish to Microsoft Azure App Service

ASP.NET MVC Core 2.0 (Angular)

EXERCISE DEMO: 2.4

LEARNING OUTCOMES

- Creating an ASP.NET Core 2.0 with Angular
- Understanding the application structure
- Building and running your application.

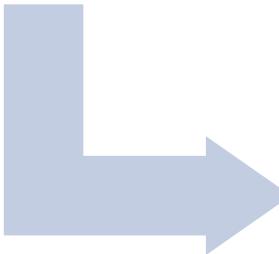
ASP.NET MVC 2.0 Angular Application :Create

Please read terms and conditions of use

Original Series

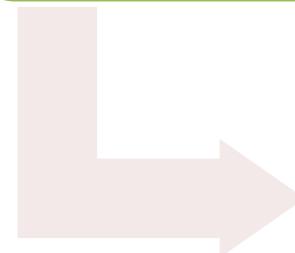
1

- Create a new solution using dotnet core ASP.NET Core with Angular
- **Dotnet new angular**



2

- Runs post-creation actions
- **Dotnet restore (in dotnet 1.x we had to explicitly call this)**

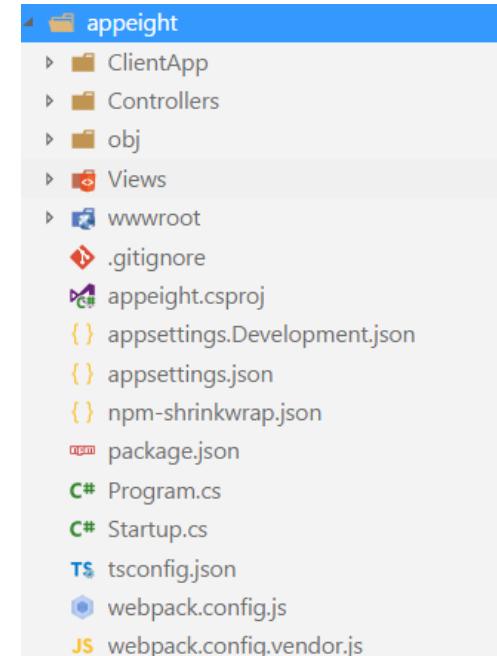


3

- Creates the following application structure
- Now run npm install -save

```
PS H:\awase-harddisk\CT\C#2017\DOTNETCORE\dotnetcore-appscenarios\appeight> dotnet new angular
The template "ASP.NET Core with Angular" was created successfully.

Processing post-creation actions...
Running 'dotnet restore' on H:\awase-harddisk\CT\C#2017\DOTNETCORE\dotnetcore-appscenarios\appe
Restoring packages for H:\awase-harddisk\CT\C#2017\DOTNETCORE\dotnetcore-appscenarios\appeigh...
```

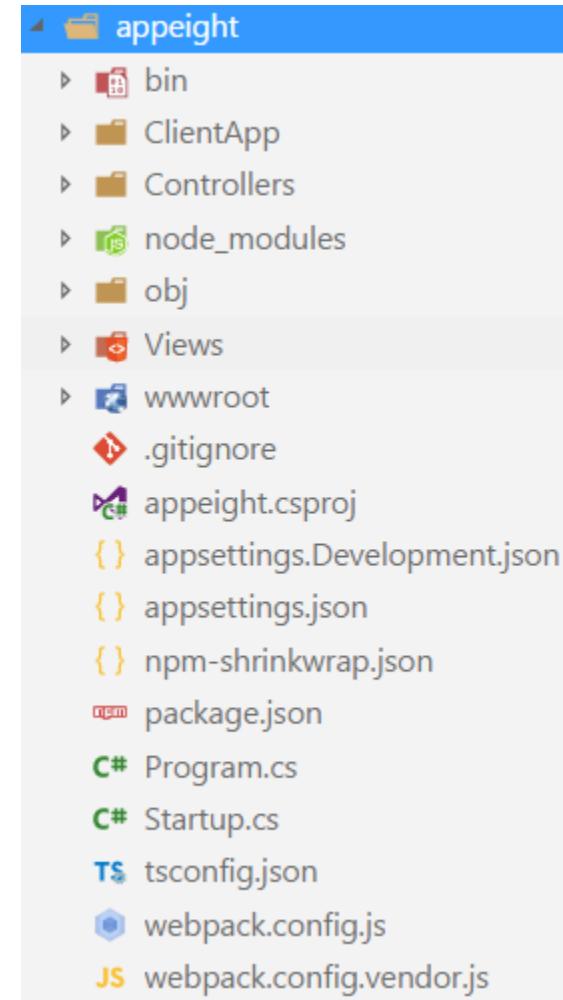


ASP.NET MVC 2.0 Angular Application :build

- Building your ASP.NET MVC 2.0 Angular Application
 - Dotnet build

```
PS H:\awase-harddisk\CT\C#2017\DOTNETCORE\dotnetcore-appscenarios\appeight> dotnet build
Microsoft (R) Build Engine version 15.5.180.51428 for .NET Core
Copyright (C) Microsoft Corporation. All rights reserved.

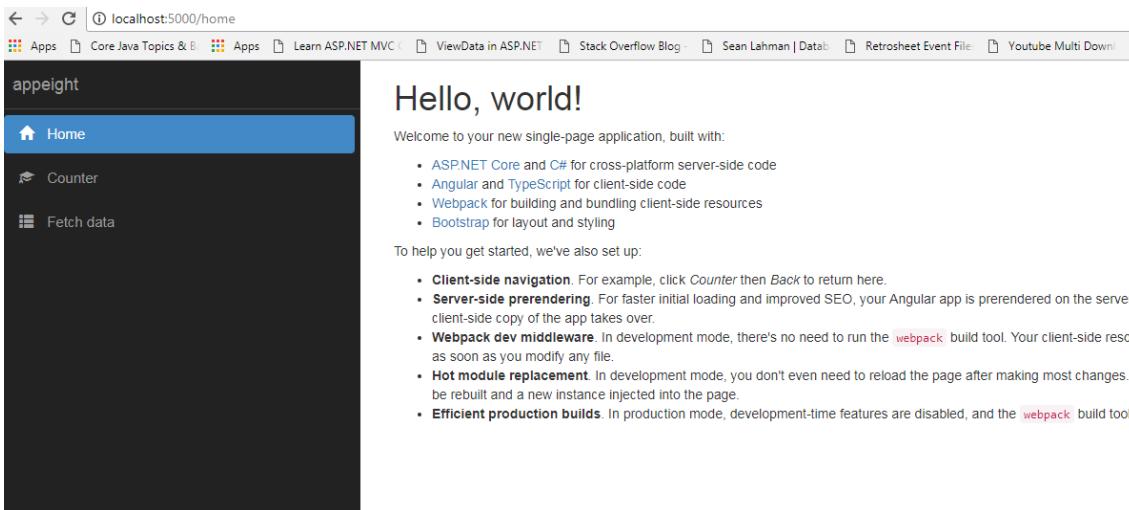
    Restore completed in 235.44 ms for H:\awase-harddisk\CT\C#2017\DOTNETCORE\dotnetcore-app
proj.
    Restore completed in 130.41 ms for H:\awase-harddisk\CT\C#2017\DOTNETCORE\dotnetcore-app
proj.
    appeight -> H:\awase-harddisk\CT\C#2017\DOTNETCORE\dotnetcore-appscenarios\appeight\bin\
11
    v8.1.3
    Performing first-run Webpack build...
    Hash: 4adf5b975b06d7f766a231699721357037744fe
    Version: webpack 2.5.1
```



ASP.NET Core 2.0 Angular: run

- Executing ASP.NET Core 2.0 Angular
 - Dotnet run

```
PS H:\awase-harddisk\CT\C#2017\DOTNETCORE\dotnetcore-appscenarios\appeight> dotnet run
Hosting environment: Production
Content root path: H:\awase-harddisk\CT\C#2017\DOTNETCORE\dotnetcore-appscenarios\appeight
Now listening on: http://localhost:5000
Application started. Press Ctrl+C to shut down.
```



ASP.NET MVC Core 2.0 (React)

EXERCISE DEMO: 2.5

LEARNING OUTCOMES

- Creating your application with ASP.NET Core with React.js
- Understanding the application structure
- Building and executing your application.

- Use case 1
- Use case 2
- Use case 3
- Use case 4

USE CASES

ASP.NET MVC 2.0 React Application :Create

Please read terms and conditions of use

Original Series

1

- Create a new solution using dotnet core ASP.NET Core with ReactJS
- **Dotnet new react**

```
PS H:\awase-harddisk\CT\C#2017\DOTNETCORE\dotnetcore-appscenarios\appnine> dotnet new react  
The template "ASP.NET Core with React.js" was created successfully.
```

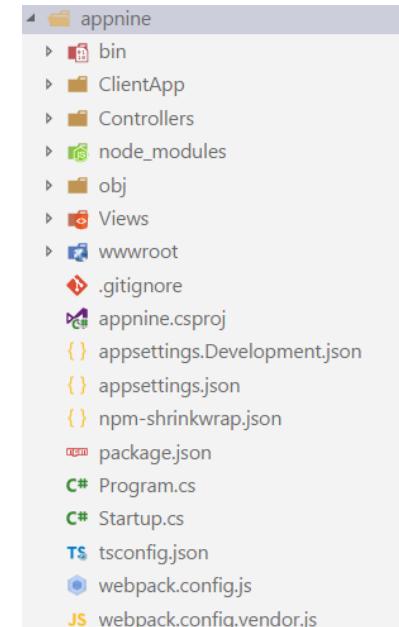
2

- Runs post-creation actions
- **Dotnet restore (in dotnet 1.x we had to explicitly call this)**

```
Processing post-creation actions...  
Running 'dotnet restore' on H:\awase-harddisk\CT\C#2017\DOTNETCORE\dotnetcore-appscenarios\  
Restoring packages for H:\awase-harddisk\CT\C#2017\DOTNETCORE\dotnetcore-appscenarios\app
```

3

- Creates the following application structure
- Now run npm install -save



ASP.NET CORE 2.0 React Application:build

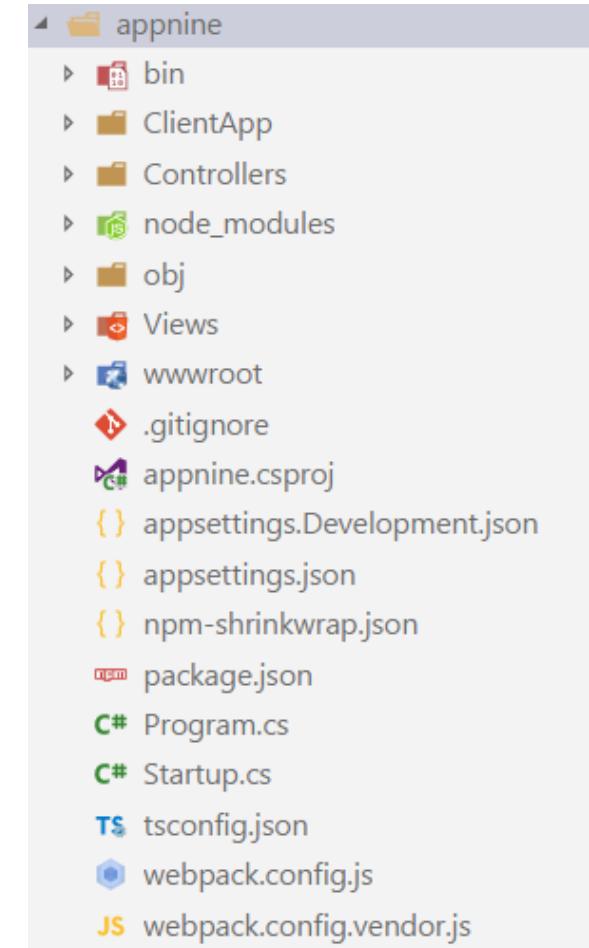
Please read terms and conditions of use

- Building your ASP.NET CORE 2.0 React Application
 - Dotnet build

```
PS H:\awase-harddisk\CT\C#2017\DOTNETCORE\dotnetcore-appscenarios\appninel> dotnet build
Microsoft (R) Build Engine version 15.5.180.51428 for .NET Core
Copyright (C) Microsoft Corporation. All rights reserved.
```

```
Restore completed in 209.03 ms for H:\awase-harddisk\CT\C#2017\DOTNETCORE\dotnetcore-ap
oj.
```

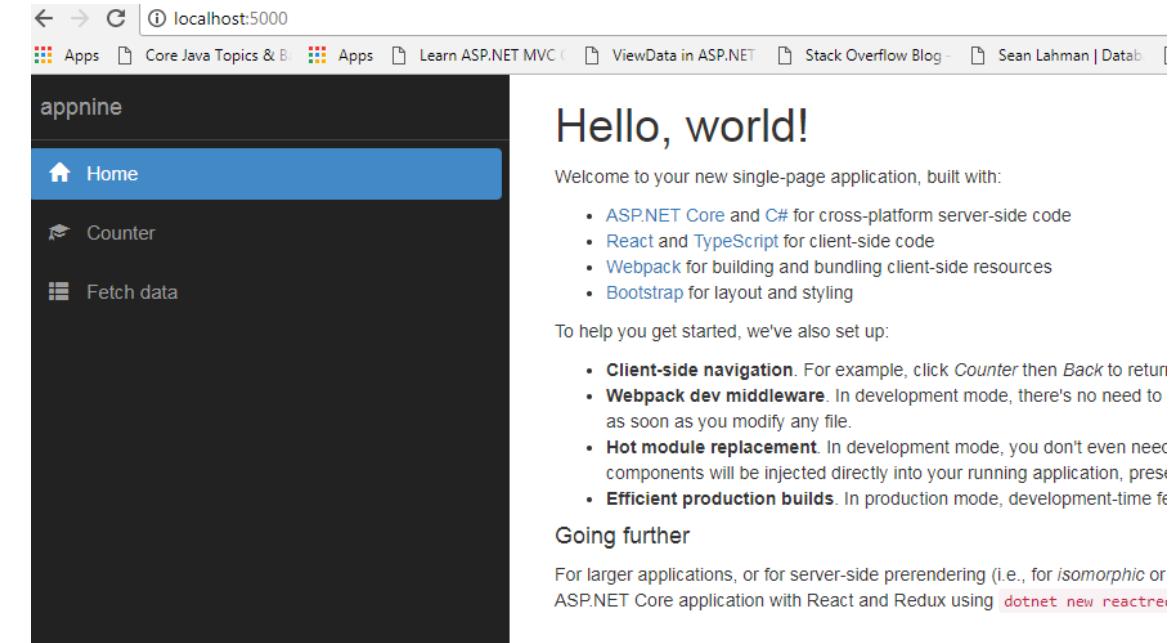
```
Restore completed in 160.58 ms for H:\awase-harddisk\CT\C#2017\DOTNETCORE\dotnetcore-ap
oj.
```



Original Series

ASP.NET CORE 2.0 React Application:run

- Executing your ASP.NET Core 2.0 React Application
 - Dotnet run



```
PS H:\awase-harddisk\CT\C#2017\DOTNETCORE\dotnetcore-appscenarios\appnne> dotnet run
Hosting environment: Production
Content root path: H:\awase-harddisk\CT\C#2017\DOTNETCORE\dotnetcore-appscenarios\appnne
Now listening on: http://localhost:5000
Application started. Press Ctrl+C to shut down.
```

ASP.NET MVC Core 2.0 (ReactJS +Redux)

EXERCISE DEMO: 2.6

LEARNING OUTCOMES

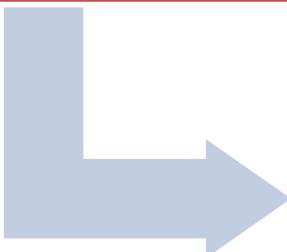
- Creating your application with ASP.NET Core with React.js with Redux
- Understanding the application structure
- Building and executing your application.

ASP.NET CORE MVC 2.0 ReactRedux Application

:Create

1

- Create a new solution using dotnet core ASP.NET Core with ReactJS Redux
- **Dotnet new reactredux**



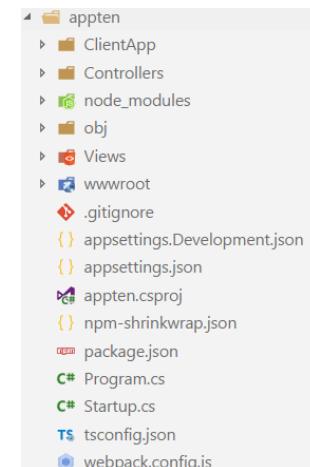
2

- Runs post-creation actions
- **Dotnet restore (in dotnet 1.x we had to explicitly call this)**



3

- Creates the following application structure
- Now run **npm install -save**



Please read terms and conditions of use

Original Series

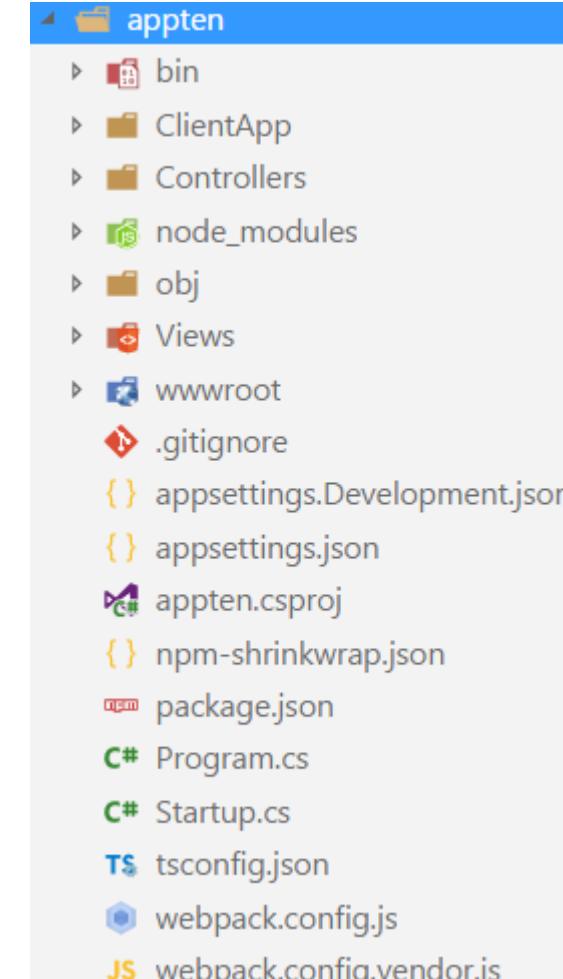
ASP.NET CORE MVC 2.0 ReactRedux Application

:build

- Building your ASP.NET MVC 2.0 ReactRedux Application
 - Dotnet build

```
PS H:\awase-hddisk\CT\C#2017\DOTNETCORE\dotnetcore-appscenarios\appten> dotnet build  
Microsoft (R) Build Engine version 15.5.180.51428 for .NET Core  
Copyright (C) Microsoft Corporation. All rights reserved.
```

```
Restore completed in 146.56 ms for H:\awase-hddisk\CT\C#2017\DOTNETCORE\dotnetcore-app  
Restore completed in 153.93 ms for H:\awase-hddisk\CT\C#2017\DOTNETCORE\dotnetcore-app  
appten -> H:\awase-hddisk\CT\C#2017\DOTNETCORE\dotnetcore-appscenarios\appten\bin\Debu  
v8.1.3  
Performing first-run Webpack build...  
Hash: 029fc5b0d9182e0674fbbfd69e4414711995fa51
```

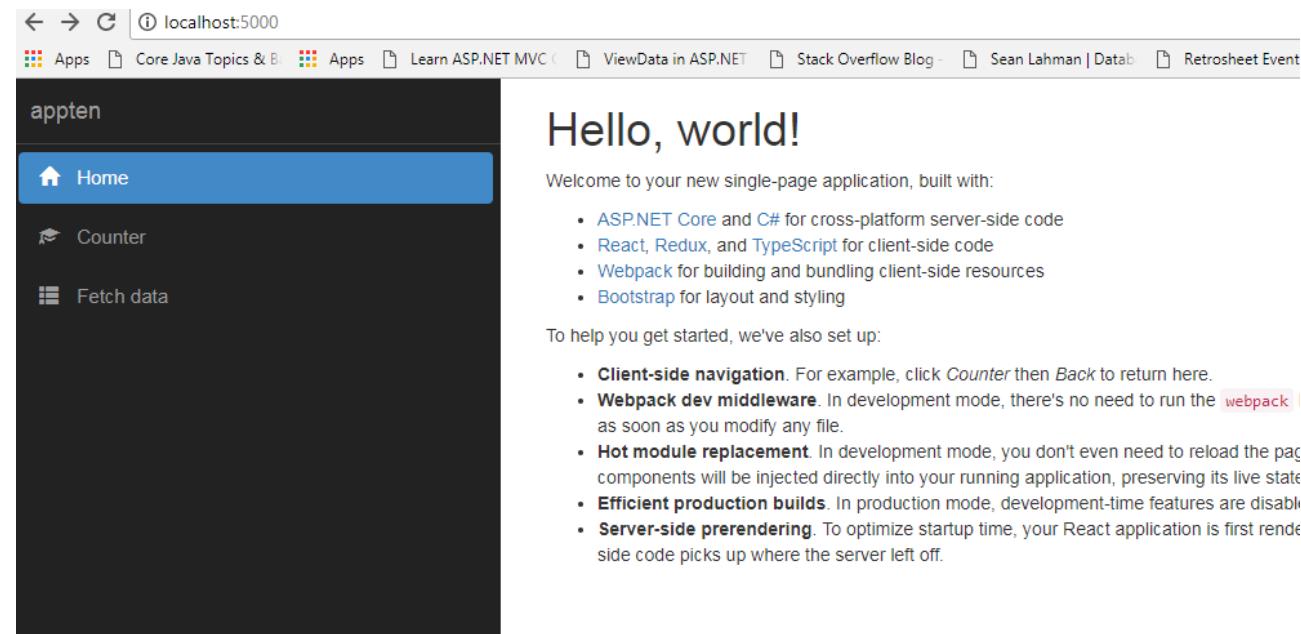


ASP.NET CORE MVC 2.0 REACTREDUX:RUN

Please read terms and conditions of use

- Executing your ASP.NET CORE 2.0 reactredux
 - Dotnet run

```
PS H:\awase-harddisk\CT\C#2017\DOTNETCORE\dotnetcore-appscenarios\apten> dotnet run
Hosting environment: Production
Content root path: H:\awase-harddisk\CT\C#2017\DOTNETCORE\dotnetcore-appscenarios\apten
Now listening on: http://localhost:5000
Application started. Press Ctrl+C to shut down.
```



Original Series

ASP.NET WEBAPI Core 2.0 (Razor)

EXERCISE DEMO: 2.7

LEARNING OUTCOMES

- Creating an ASP.NET Core WebAPI 2.0 application
- Understanding the application structure
- Building and executing the ASP.NET Core WebAPI Application

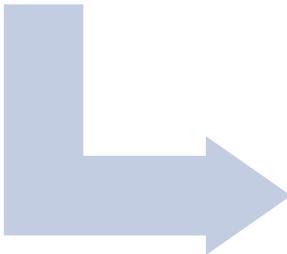
ASP.NET CORE 2.0 WEBAPI:CREATE

Please read terms and conditions of use

Original Series

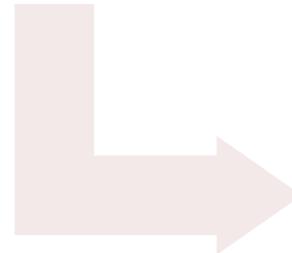
1

- Create a new solution using dotnet core ASP.NET Core with web api
- **Dotnet new webapi**



2

- Runs post-creation actions
- **Dotnet restore (in dotnet 1.x we had to explicitly call this)**



3

- Creates the following application structure

```
PS H:\awase-harddisk\CT\C#2017\DOTNETCORE\dotnetcore-appscenarios\appeleven> dotnet new webapi  
The template "ASP.NET Core Web API" was created successfully.  
This template contains technologies from parties other than Microsoft, see https://aka.ms/template
```

```
Processing post-creation actions...  
Running 'dotnet restore' on H:\awase-harddisk\CT\C#2017\DOTNETCORE\dotnetcore-appscenarios\appeleven>
```

```
Restoring packages for H:\awase-harddisk\CT\C#2017\DOTNETCORE\dotnetcore-appscenarios\appeleven>  
Restore completed in 182.77 ms for H:\awase-harddisk\CT\C#2017\DOTNETCORE\dotnetcore-appscenarios\appeleven.csproj.
```

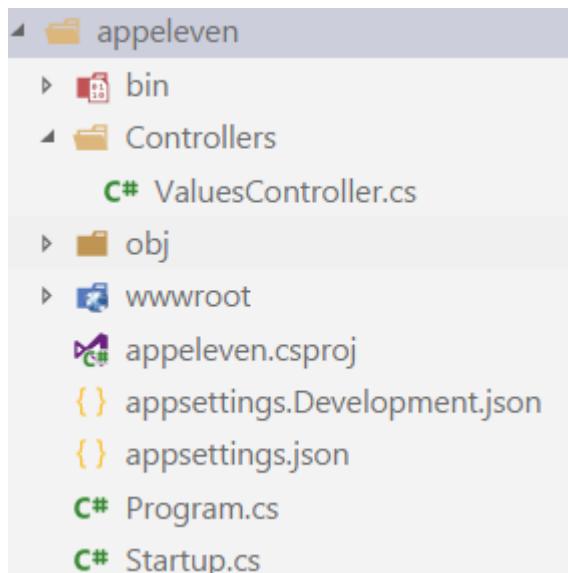
📁	appeleven
📁	Controllers
C#	ValuesController.cs
▶	obj
▶	wwwroot
➤	appeleven.csproj
{}	appsettings.Development.json
{}	appsettings.json
C#	Program.cs
C#	Startup.cs

ASP.NET CORE 2.0 WEBAPI:BUILD

Please read terms and conditions of use

Original Series

- Building your ASP.NET CORE 2.0 webapi
 - Dotnet build



```
PS H:\awase-harddisk\CT\C#2017\DOTNETCORE\dotnetcore-appscenarios\appeleven> dotnet build  
Microsoft (R) Build Engine version 15.5.180.51428 for .NET Core  
Copyright (C) Microsoft Corporation. All rights reserved.
```

```
Restore completed in 212.5 ms for H:\awase-harddisk\CT\C#2017\DOTNETCORE\dotnetcore-appscenarios\appeleven.csproj.
```

```
Restore completed in 226.98 ms for H:\awase-harddisk\CT\C#2017\DOTNETCORE\dotnetcore-appscenarios\appeleven.csproj.
```

```
appeleven -> H:\awase-harddisk\CT\C#2017\DOTNETCORE\dotnetcore-appscenarios\appeleven\built\bin.dll
```

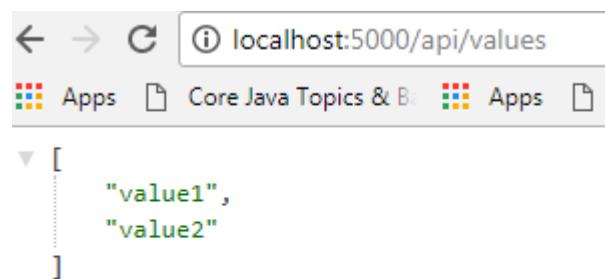
```
Build succeeded.
```

ASP.NET CORE 2.0 WEBAPI:RUN

Please read terms and conditions of use

- Executing your ASP.NET Core 2.0 webapi
 - Dotnet run

```
PS H:\awase-harddisk\CT\C#2017\DOTNETCORE\dotnetcore-appscenarios\appeleven> dotnet run
Hosting environment: Production
Content root path: H:\awase-harddisk\CT\C#2017\DOTNETCORE\dotnetcore-appscenarios\appeleven
Now listening on: http://localhost:5000
Application started. Press Ctrl+C to shut down.
```



Original Series

ASP.NET MVC Core 2.0 (Authentication)

EXERCISE DEMO: 2.8

LEARNING OUTCOMES

- Creating an ASP.NET Core MVC with Authentication Individual
- Understanding the application structure
- Building and executing the ASP.NET Core MVC 2.0 Application

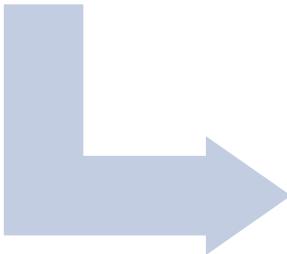
ASP.NET CORE 2.0 MVC WITH INDIVIDUAL AUTH :CREATE

Please read terms and conditions of use

Original Series

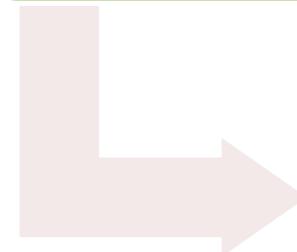
1

- Create a new solution using dotnet core ASP.NET Core MVC with individual authentication
- **dotnet new mvc --auth Individual**



2

- Runs post-creation actions
- **Dotnet restore (in dotnet 1.x we had to explicitly call this)**

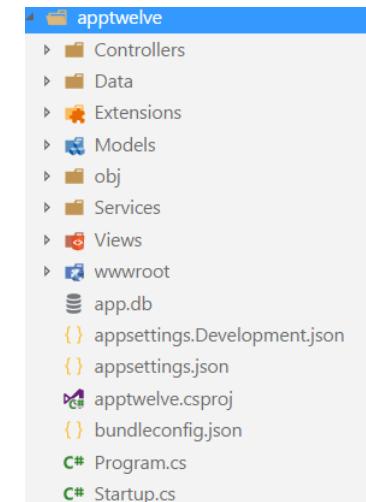


3

- Creates the following application structure
- Now run npm install -save

```
PS H:\awase-hddisk\CT\C#2017\DOTNETCORE\dotnetcore-appscenarios\apptwelve> dotnet new mvc --auth Individual
The template "ASP.NET Core Web App (Model-View-Controller)" was created successfully.
This template contains technologies from parties other than Microsoft, see https://aka.ms/template-3pn for details.
```

```
Processing post-creation actions...
Running 'dotnet restore' on H:\awase-hddisk\CT\C#2017\DOTNETCORE\dotnetcore-appscenarios\apptwelve\apptwelve.csproj...
Restoring packages for H:\awase-hddisk\CT\C#2017\DOTNETCORE\dotnetcor
```



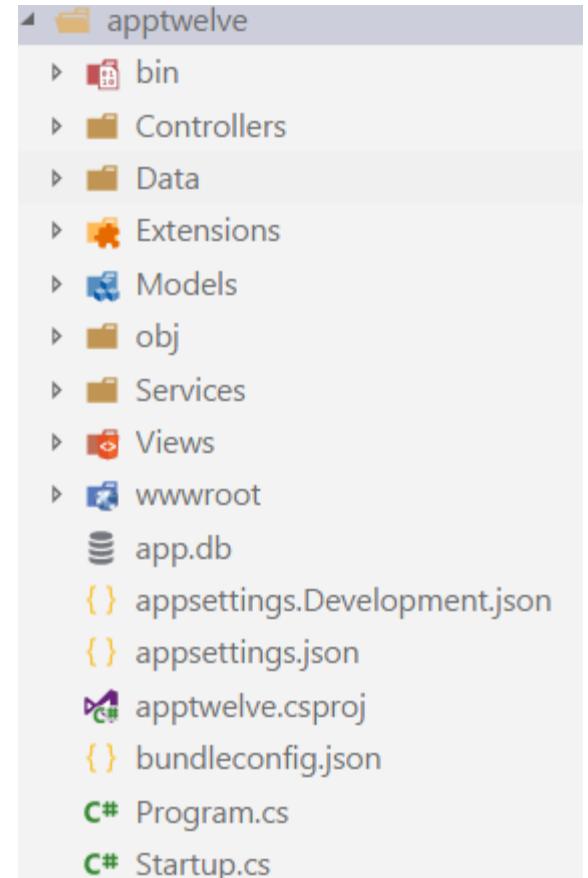
ASP.NET CORE 2.0 MVC WITH INDIVIDUAL AUTH :BUILD

Please read terms and conditions of use

- Building your ASP.NET Core 2.0 MVC application with individual authentication
 - `dotnet new mvc --auth Individual`

```
PS H:\awase-harddisk\CT\C#2017\DOTNETCORE\dotnetcore-appscenarios\apptwelve> dotnet build
Microsoft (R) Build Engine version 15.5.180.51428 for .NET Core
Copyright (C) Microsoft Corporation. All rights reserved.
```

```
Restore completed in 256.89 ms for H:\awase-harddisk\CT\C#2017\DOTNETCORE\dotnetcore-appscenarios\apptwelve.csproj.
```



ASP.NET CORE 2.0 MVC WITH INDIVIDUAL AUTH :RUN

Please read terms and conditions of use

- Executing your ASP.NET Core 2.0 MVC with individual Authentication

- Dotnet run

localhost:5000/Account/Register

PS H:\awase-harddisk\CT\C#2017\DOTNETCORE\dotnetcore-appscenarios\apptwelve> dotnet run
Hosting environment: Production
Content root path: H:\awase-harddisk\CT\C#2017\DOTNETCORE\dotnetcore-appscenarios\apptwelve
Now listening on: http://localhost:5000
Application started. Press Ctrl+C to shut down.

Original Series

[ASP.NET MVC Core 2.0 \(Razor with Authentication\)](#)

EXERCISE DEMO: 2.9

LEARNING OUTCOMES

- Creating an ASP.NET Core MVC 2.0 Razor View Engine with Authentication Individual
- Understanding the application structure
- Building and executing the ASP.NET Core MVC 2.0 Application

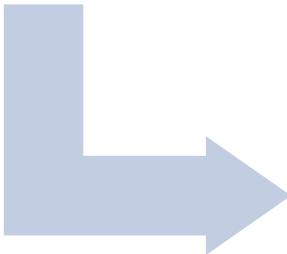
ASP.NET CORE 2.0 Razor WITH INDIVIDUAL AUTH :CREATE

Please read terms and conditions of use

Original Series

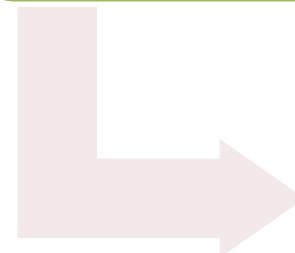
1

- Create a new solution using dotnet core ASP.NET Core with razor view and individual authentication.
- **dotnet new razor --auth Individual**



2

- Runs post-creation actions
- **Dotnet restore (in dotnet 1.x we had to explicitly call this)**

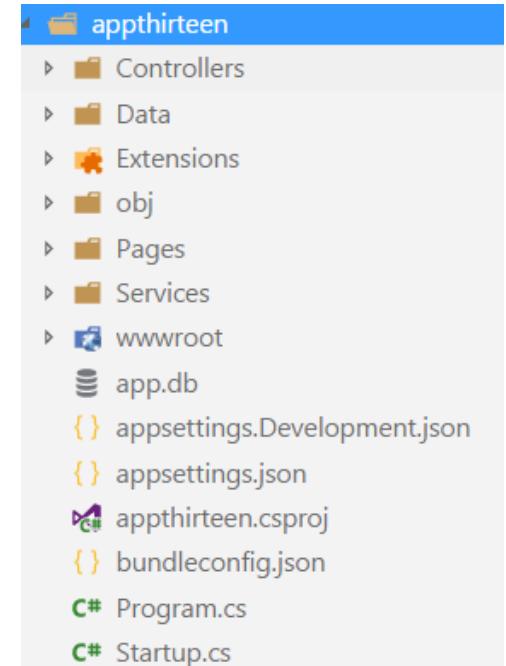


3

- Creates the following application structure
- Now run npm install -save

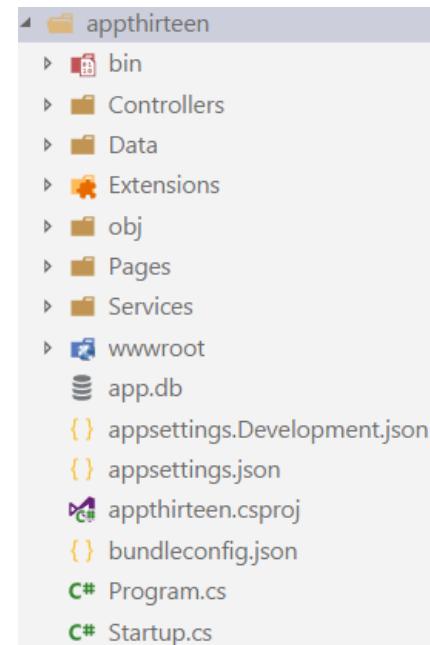
```
PS H:\awase-harddisk\CT\C#2017\DOTNETCORE\dotnetcore-appscenarios\appthirteen> dotnet new razor --auth Individual  
The template "ASP.NET Core Web App" was created successfully.  
This template contains technologies from parties other than Microsoft, see https://aka.ms/template-3pn for details.
```

```
Processing post-creation actions...  
Running 'dotnet restore' on H:\awase-harddisk\CT\C#2017\DOTNETCORE\dotnetcore-appscenarios\appthirteen\appthirteen.csproj...  
Restoring packages for H:\awase-harddisk\CT\C#2017\DOTNETCORE\dotnetcore-appscenarios\appthirteen\appthirteen.csproj...
```



ASP.NET CORE 2.0 Razor WITH INDIVIDUAL AUTH :BUILD

- Building your ASP.NET CORE 2.0 razor with individual authentication
 - Dotnet build



```
PS H:\awase-harddisk\CT\C#2017\DOTNETCORE\dotnetcore-appscenarios\appthirteen> dotnet build  
Microsoft (R) Build Engine version 15.5.180.51428 for .NET Core  
Copyright (C) Microsoft Corporation. All rights reserved.
```

```
Restore completed in 173.47 ms for H:\awase-harddisk\CT\C#2017\DOTNETCORE\dotnetcore-appscenarios\appthirteen.csproj.
```

```
Restore completed in 173.53 ms for H:\awase-harddisk\CT\C#2017\DOTNETCORE\dotnetcore-appscenarios\appthirteen.csproj.
```

ASP.NET CORE 2.0 Razor WITH INDIVIDUAL AUTH :RUN

Please read terms and conditions of use

- Executing ASP.NET Core 2.0 Razor with individual authentication
 - Dotnet run

```
PS H:\awase-hddisk\CT\C#2017\DOTNETCORE\dotnetcore-appscenarios\appthirteen> dotnet run
Hosting environment: Production
Content root path: H:\awase-hddisk\CT\C#2017\DOTNETCORE\dotnetcore-appscenarios\appthirteen
Now listening on: http://localhost:5000
Application started. Press Ctrl+C to shut down.
```

The screenshot shows a browser window with the following details:

- URL:** localhost:5000/Account/Register
- Header Bar:** Shows various links like Java Topics & Books, Apps, Learn ASP.NET MVC, ViewData in ASP.NET, Stack Overflow Blog, Sean Lahman | Database, Retrosheet Event File, Youtube Multi Downloader, Working with Geography, and Online.
- Navigation Bar:** appthirteen Home About Contact Register Log in
- Content Area:**
 - ## Register
 - Create a new account.
 - Email:**
 - Password:**
 - Confirm password:**
 - Register** button
- Footer:** © 2017 - appthirteen

Original Series

Web Server Implementation in ASP.NET CORE

EXERCISE DEMO: 2.10

LEARNING OUTCOMES

- Creating a basic webserver using kestrel to render data to the web
- <https://github.com/aspnet/KestrelHttpServer>

Web Server Implementations

- Kestrel : A cross platform HTTP Server based on libuv, a cross platform asynchronous I/O library
 - <https://docs.microsoft.com/en-us/aspnet/core/fundamentals/servers/kestrel?tabs=aspnetcore2x>
- HTTP.sys is a windows-only HTTP server based on HTTP.SYS Kernel driver.
 - <https://docs.microsoft.com/en-us/aspnet/core/fundamentals/servers/http.sys>
- Kestrel can be used by itself or with a reverse proxy server such as IIS, Nginx or Apache.
- A reverse proxy server receive HTTP requests from the internet and forwards them to Kestrel after some preliminary handling.
- A reverse proxy is required when we have multiple applications that share the same IP and port running on a single server.

libuv

<http://libuv.org/>

The screenshot shows the libuv website homepage. At the top, there is a navigation bar with links to "Java Topics & Books", "Apps", "Learn ASP.NET MVC", "ViewData in ASP.NET", "Stack Overflow Blog", "Sean Lahman | Database", "Retrosheet Event Files", "Youtube Multi Downloader", "Work", and a star icon. Below the navigation bar, the libuv logo is displayed, followed by links to "Documentation", "Code", "Releases", "Mailing List", and "IRC". The main content area features a large circular logo with a scorpion and the text "Asynchronous I/O made simple.". Below this, a subtitle reads "libuv is a multi-platform support library with a focus on asynchronous I/O." A button labeled "Show me the code!" is visible. At the bottom of the page, a footer note states "Made with love by the libuv team. — libuv is for C projects that miss the joy of javascript callback hell."

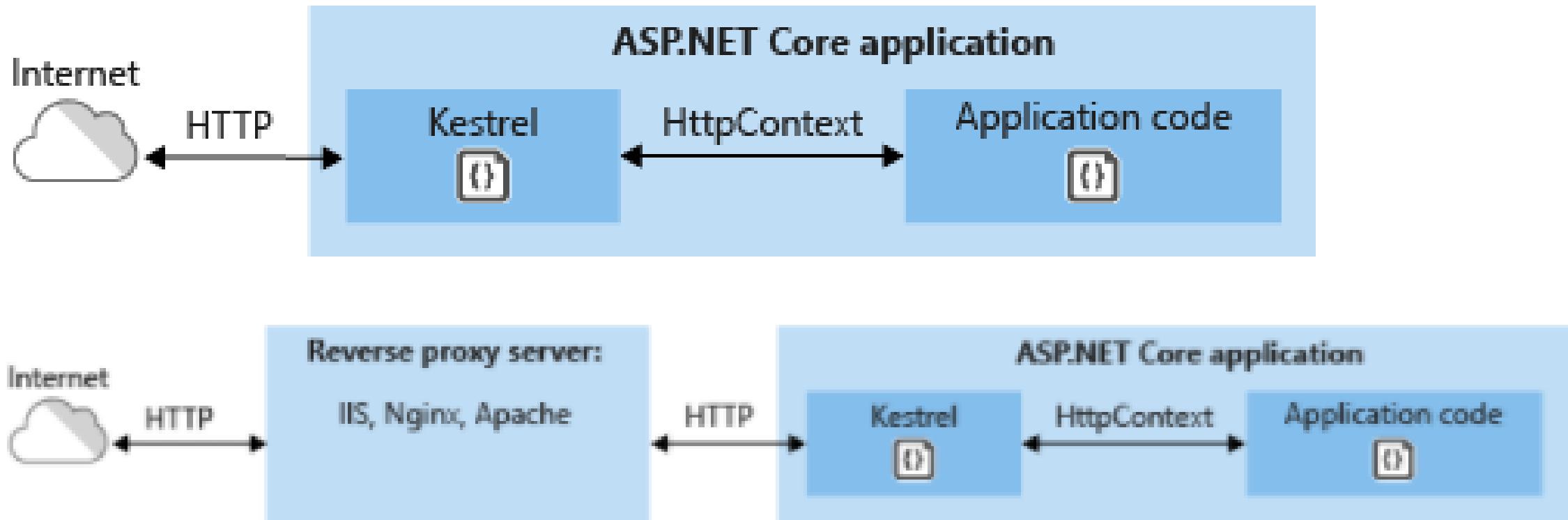
Please read terms and conditions of use

Original Series

KESTREL

<https://github.com/aspnet/KestrelHttpServer>

Please read terms and conditions of use

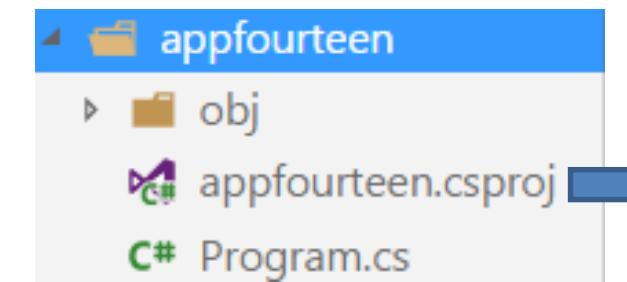


Source: <https://docs.microsoft.com/en-us/aspnet/core/fundamentals/servers/kestrel?tabs=aspnetcore2x>

Original Series

Steps to create a basic web server

1. Dotnet new console
2. Creates a project structure
3. Adding additional package
 - dotnet add package Microsoft.AspNetCore
4. Note that the project.csproj file is updated with the package reference of “Microsoft.AspNetCore”



Earlier version of .NET is called **project.json**

```
<Project Sdk="Microsoft.NET.Sdk">  
  
  <PropertyGroup>  
    <OutputType>Exe</OutputType>  
    <TargetFramework>netcoreapp2.0</TargetFramework>  
  </PropertyGroup>  
  
</Project>
```

Steps to create a basic web server

5. Create a startup.cs

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

namespace appfourteen
{
    public class Startup{
        public void Configure(IApplicationBuilder app){
            app.Run(context =>{
                return context.Response.WriteAsync("Hello!
TPRI/SYCLIQ!!");
            });
        }
    }
}
```

Steps to create a basic web server

6. Bind your Startup to WebHostBuilder application

For rendering multiple URL's

```
var host = new
Microsoft.AspNetCore.Hosting.WebHostBuilder()
.UseKestrel()
.UseUrls("http://*:1000", "http://*:5000")
//rendering many urls
.UseStartup<Startup>()
.Build();
```

```
using System;
using Microsoft.AspNetCore.Hosting;
namespace appfourteen
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hello World!");
            //create an instance of WebHostbuilder and bind the
            //kestrel
            var host = new Microsoft.AspNetCore.Hosting.WebHostBuilder()
                .UseKestrel()
                .UseStartup<Startup>()
                .Build();
            // to run your web host.
            host.Run();
        }
    }
}
```

Steps to create a basic web server

7. Now lets build the application using
dotnet build

8. Now execute your application using
dotnet run

```
PS H:\awase-harddisk\CT\C#2017\DOTNETCORE\dotnetcore-appscenarios\appfourteen> dotnet run
Hello World!
Hosting environment: Production
Content root path: H:\awase-harddisk\CT\C#2017\DOTNETCORE\dotnetcore-appscenarios\appfourteen\bin\Debug\netcoreapp2.0\
Now listening on: http://localhost:5000
Application started. Press Ctrl+C to shut down.
```

```
PS H:\awase-harddisk\CT\C#2017\DOTNETCORE\dotnetcore-appscenarios\appfourteen> dotnet build
Microsoft (R) Build Engine version 15.5.180.51428 for .NET Core
Copyright (C) Microsoft Corporation. All rights reserved.
```

```
Restore completed in 154.43 ms for H:\awase-harddisk\CT\C#2017\DOTNETCORE\dotnetcore-appscenarios\appfourteen\appfourteen.csproj.
appfourteen -> H:\awase-harddisk\CT\C#2017\DOTNETCORE\dotnetcore-appscenarios\appfourteen\bin\Debug\netcoreapp2.0\appfourteen.dll
```

Build succeeded.

<http://localhost:5000/>



Hello! TPRI/SYCLIQ!!

Steps to create a basic web server

9. Developer interlooping by adding the “**Microsoft.DotNet.Watcher.Tools**” in project.csproj
10. Run your application with **dotnet watch run**

```
<Project Sdk="Microsoft.NET.Sdk">  
  <PropertyGroup>  
    <OutputType>Exe</OutputType>  
    <TargetFramework>netcoreapp2.0</TargetFramework>  
  </PropertyGroup>  
  
  <ItemGroup>  
    <PackageReference Include="Microsoft.AspNetCore" Version="2.0.1" />  
  </ItemGroup>  
  
  <ItemGroup>  
    <DotNetCliToolReference Include="Microsoft.DotNet.Watcher.Tools" Version="2.0.0" />  
  </ItemGroup>  
</Project>
```

HTTP.sys

- Runs only on Windows. It's built on the HTTP.Sys Kernel mode driver.
- It is an alternative to Kestrel
- HTTP.Sys cannot be used with IIS or IIS Express.
- Used when we need to expose the server directly to the internet without using IIS.
- Protects against many kinds of attacks and provides the robustness, security and scalability of a full-featured web server.
- It supports
 - Windows authentication
 - Port sharing
 - HTTPS with SNI
 - HTTP/2 over TLS
 - Direct file transmission
 - Response caching
 - WebSockets(Windows 8)

ASP.NET CORE WITH VISUAL STUDIO 2017

EXERCISE DEMO: 2.11

LEARNING OUTCOMES

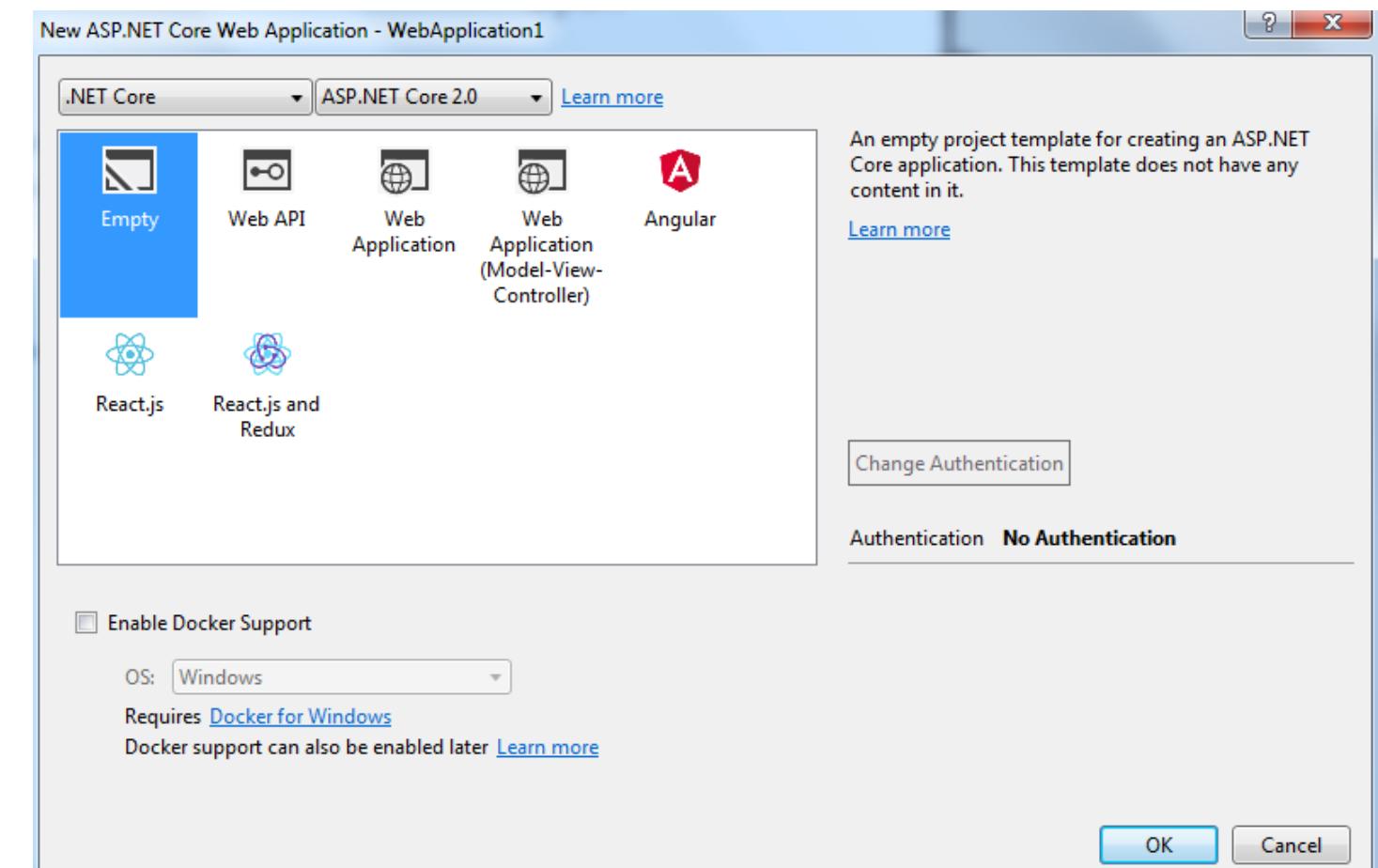
- Creating an ASP.NET CORE 2.0 Empty application with Visual Studio 2017
- Creating an ASP.NET Core 2.0 MVC application with Visual Studio 2017
- Creating an ASP.NET Core 2.0 MVC application with razor view engine using Visual Studio 2017
- Creating an ASP.NET Core 2.0 MVC application with Angular using Visual Studio 2017
- Creating an ASP.NET Core 2.0 MVC application with ReactJS using Visual Studio 2017
- Creating an ASP.NET Core 2.0 MVC application with ReactRedux using Visual Studio 2017

Creating an Empty ASP.NET Core 2.0 with Visual Studio 2017

- Creating an empty ASP.NET Core 2.0 application

Please read terms and conditions of use

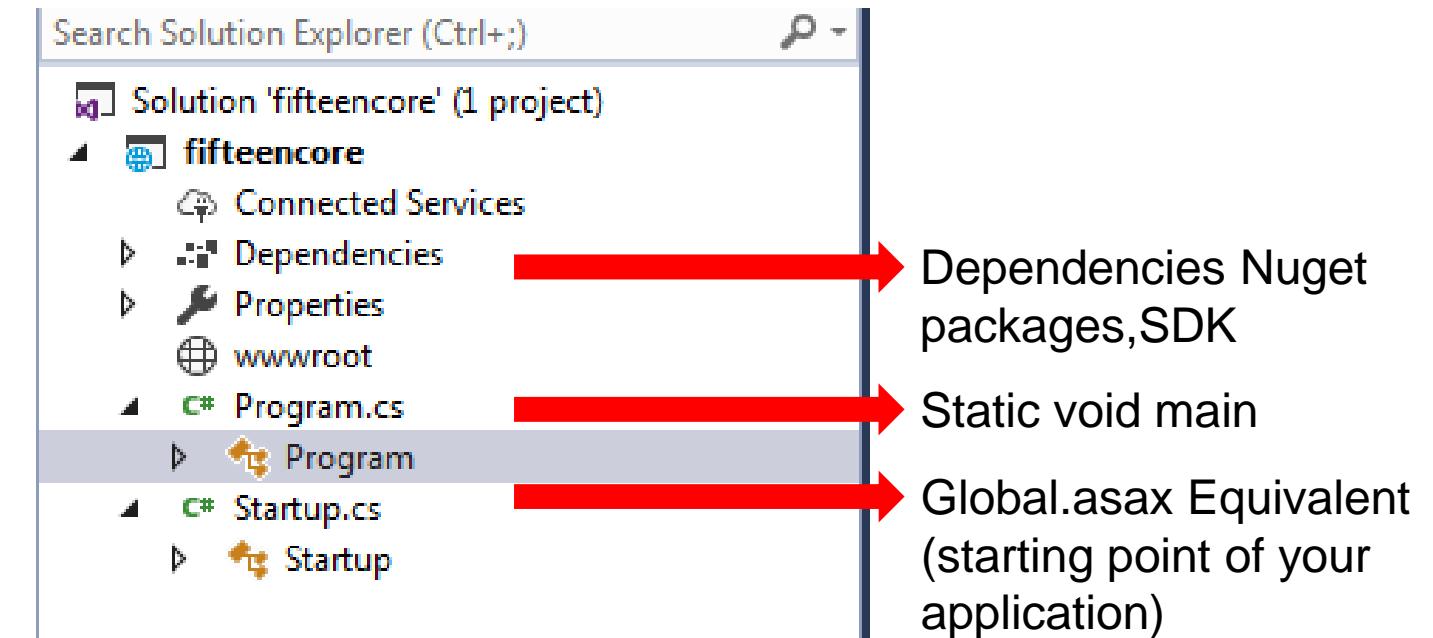
Original Series



Creating an Empty ASP.NET Core 2.0 with Visual Studio 2017

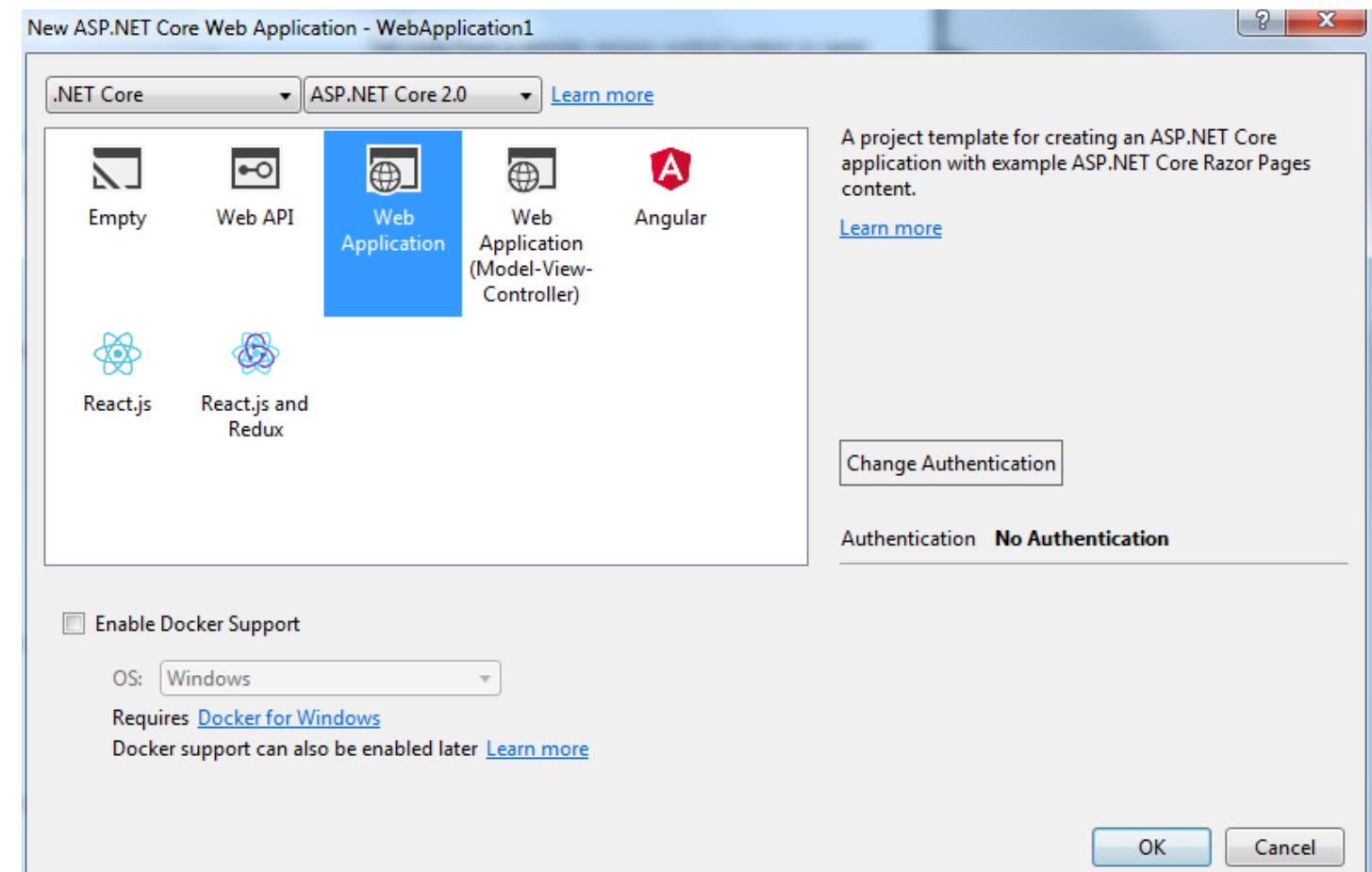
Please read terms and conditions of use

Original Series



Creating an ASP.NET Core 2.0 Web Application with Visual Studio 2017

- Select ASP.NET Core 2.0 web application

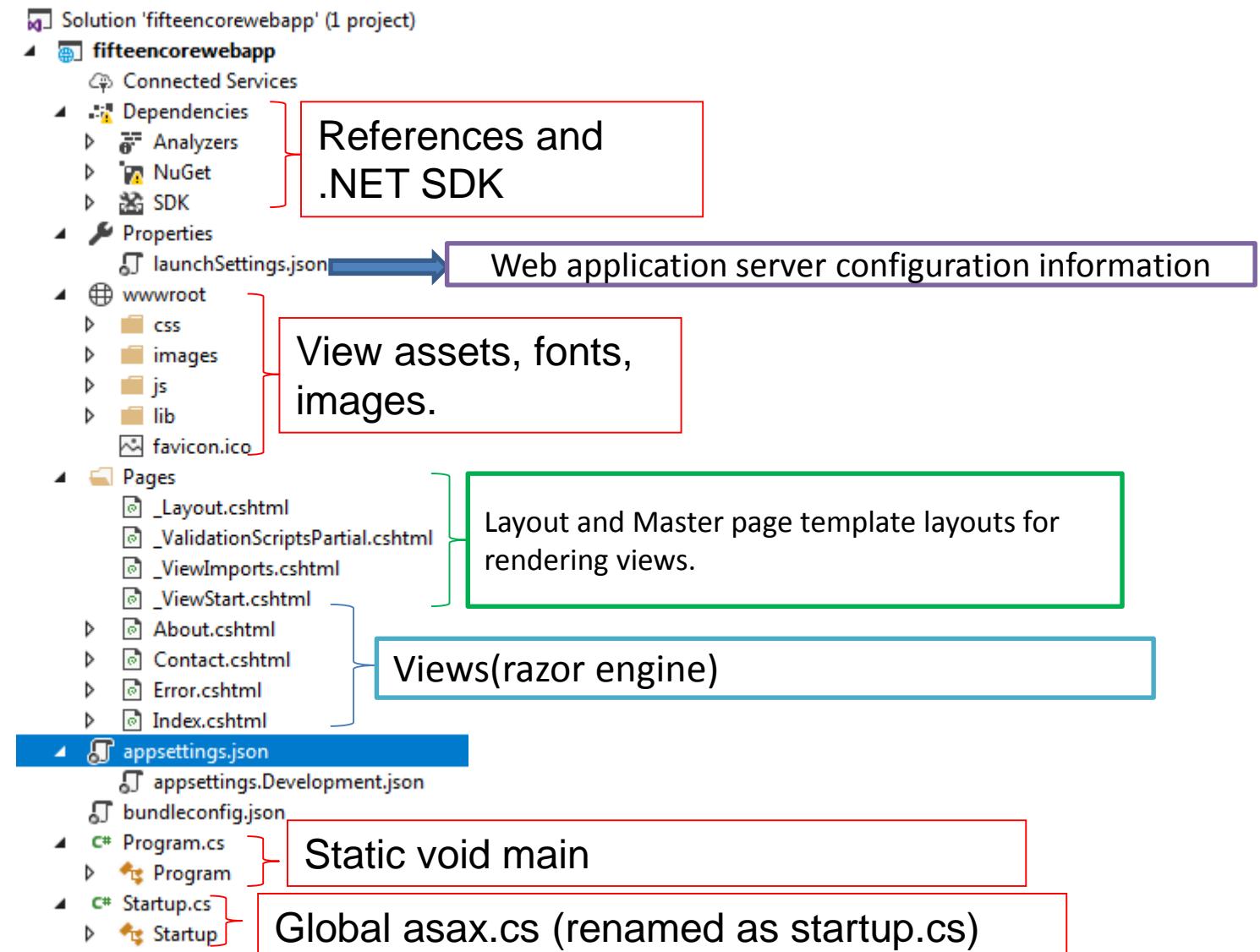


Please read terms and conditions of use

Original Series

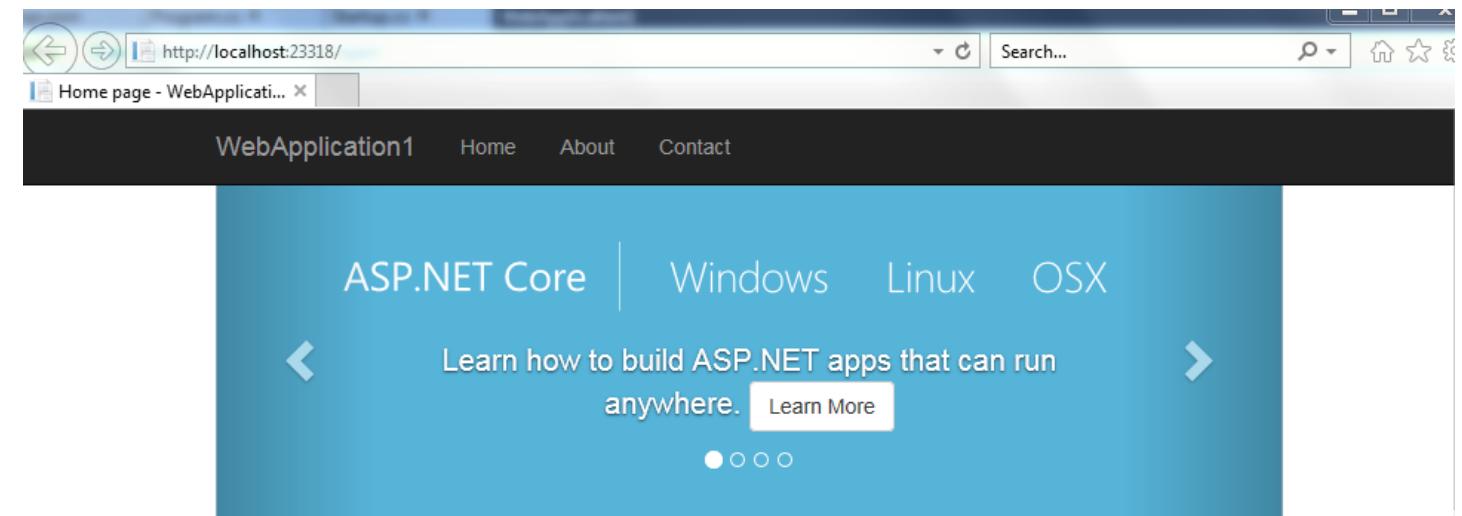
Creating an ASP.NET Core 2.0 Web Application with Visual Studio 2017

- A scaffolded ASP.NET Core 2.0 web application has the following structure outlined here.



Creating an ASP.NET Core 2.0 Web Application with Visual Studio 2017

- Upon running your application it renders this view.



Application uses

- Sample pages using ASP.NET Core Razor Pages
- Theming using [Bootstrap](#)

How to

- Working with Razor Pages.
- Manage User Secrets using [Secret Manager](#).
- Use logging to log a message.

ASP.NET Core Packages

Please read terms and conditions of use

Original Series

1.	Microsoft.AspNetCore	
2.	Microsoft.AspNetCore.Antiforgery	CSRF Support
3.	Microsoft.AspNetCore.ApplicationInsights.HostingStartup	WebServer and Startup.cs
4	Microsoft.AspNetCore.Authentication	Authentication
	Microsoft.AspNetCore.AzureAppServices	Azure Services
	Microsoft.AspNetCore.CookiePolicy	Cookie
	Microsoft.AspNetCore.Cors	Cross Origin Resource Sharing
	Microsoft.AspNetCore.Cryptography	Encryption/Decryption
	Microsoft.AspNetCore.DataProtection	Data Protection/Encryption
	Microsoft.AspNetCore.Diagnostics	Diagnostics Package
	Microsoft.AspNetCore.Hosting	Hosting Information

ASP.NET Core Packages

Please read terms and conditions of use

Original Series

	Microsoft.AspNetCore.Html	Html Helper Library
	Microsoft.AspNetCore.Identity	Identity management
	Microsoft.AspNetCore.JsonPatch	JSON Support package
	Microsoft.AspNetCore.Localization	Internationalization
	Microsoft.AspNetCore.MiddlewareAnalysis	Middleware Support
	Microsoft.AspNetCore.Mvc	MVC Support
	Microsoft.AspNetCore.NodeServices	Nodejs Service Support
	Microsoft.AspNetCore.Owin	
	Microsoft.AspNetCore.Razor	
	Microsoft.AspNetCore.ResponseCaching	
	Microsoft.AspNetCore.Rewrite	

ASP.NET Core Packages

Please read terms and conditions of use

Original Series

	Microsoft.AspNetCore.Routing	Routing Engine
	Microsoft.AspNetCore.Server.HttpSys	HttpSys Web Server
	Microsoft.AspNetCore.Server.IISIntegration	IIS Server Support
	Microsoft.AspNetCore.Server.Kestrel	Kestrel web server
	Microsoft.AspNetCore.Session	Session Management
	Microsoft.AspNetCore.SpaServices	
	Microsoft.AspNetCore.StaticFiles	Static File Rendering Support
	Microsoft.AspNetCore.WebSockets	Web Socket Support
	Microsoft.AspNetCore.WebUtilities	Web Utility Package
	Microsoft.CodeAnalysis.Razor	Razor Package

ASP.NET Core Packages

Please read terms and conditions of use

Original Series

	Microsoft.Data.Sqlite	Sqlite package
	Microsoft.EntityFrameworkCore	EntityFramework Core
	Microsoft.Extensions.Caching	Caching Support
	Microsoft.Extensions.Configuration	Configuration Management
	Microsoft.Extensions.DependencyInjection	Dependency Management
	Microsoft.Extensions.DiagnosticAdapter	
	Microsoft.Extensions.FileProviders	
	Microsoft.Extensions.FileSystemGlobbing	
	Microsoft.Extensions.Hosting.Abstractions	
	Microsoft.Extensions.Identity	
	Microsoft.Extensions.Logging	

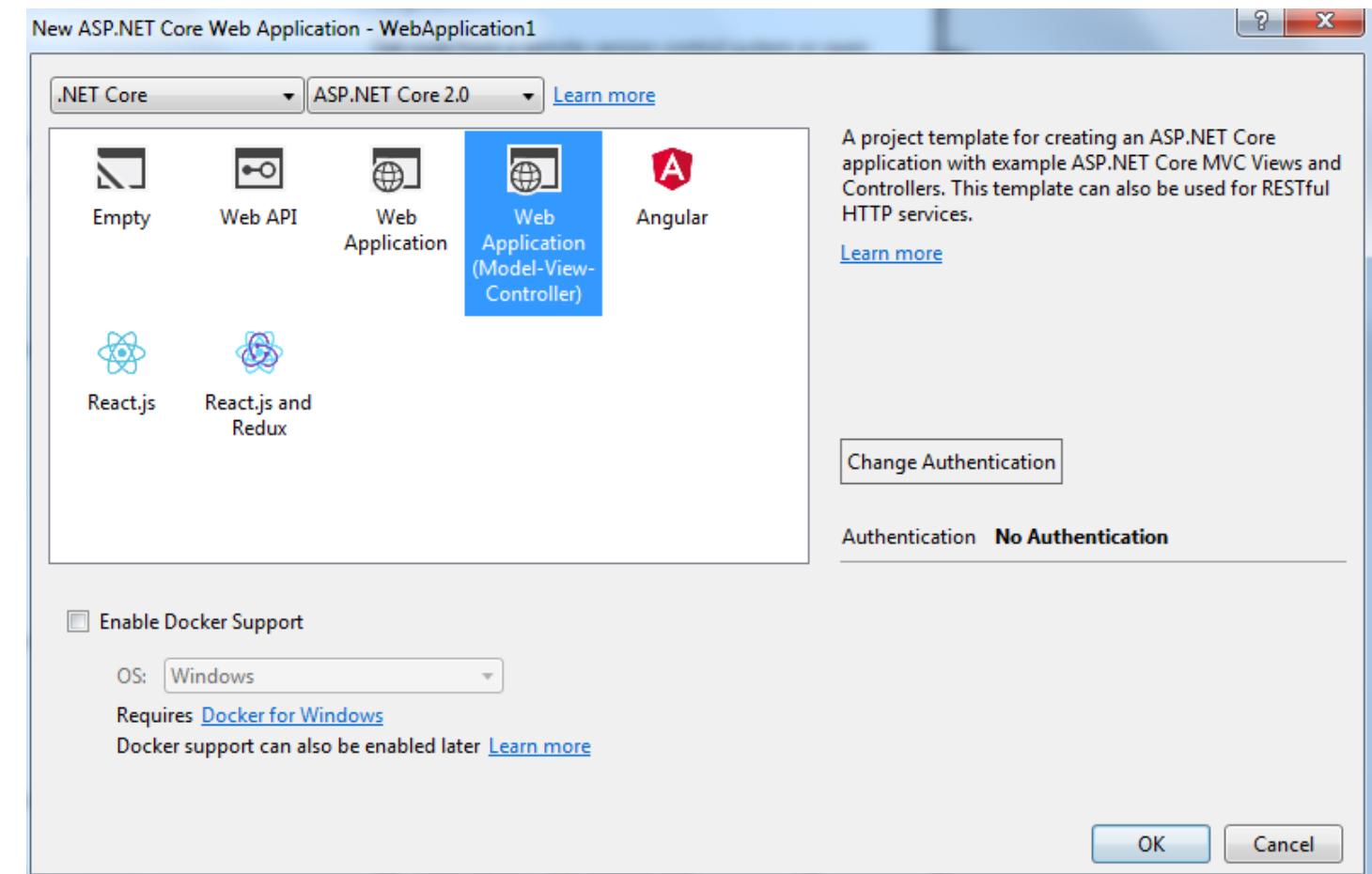
ASP.NET Core Packages

Please read terms and conditions of use

Original Series	NET Core Packages
	Microsoft.Extensions.Logging
	Microsoft.Extensions.ObjectPool
	Microsoft.Extensions.Options.
	Microsoft.Extensions.WebEncoders
	Microsoft.Net.Http.Headers
	Microsoft.VisualStudio.Web.BrowserLink
	Microsoft.NETCore.App
	Microsoft.CodeAnalysis.Analyzers
	Microsoft.CodeAnalysis.Csharp.Analyzers

Creating an ASP.NET Core 2.0 Web Application (MVC) with Visual Studio 2017

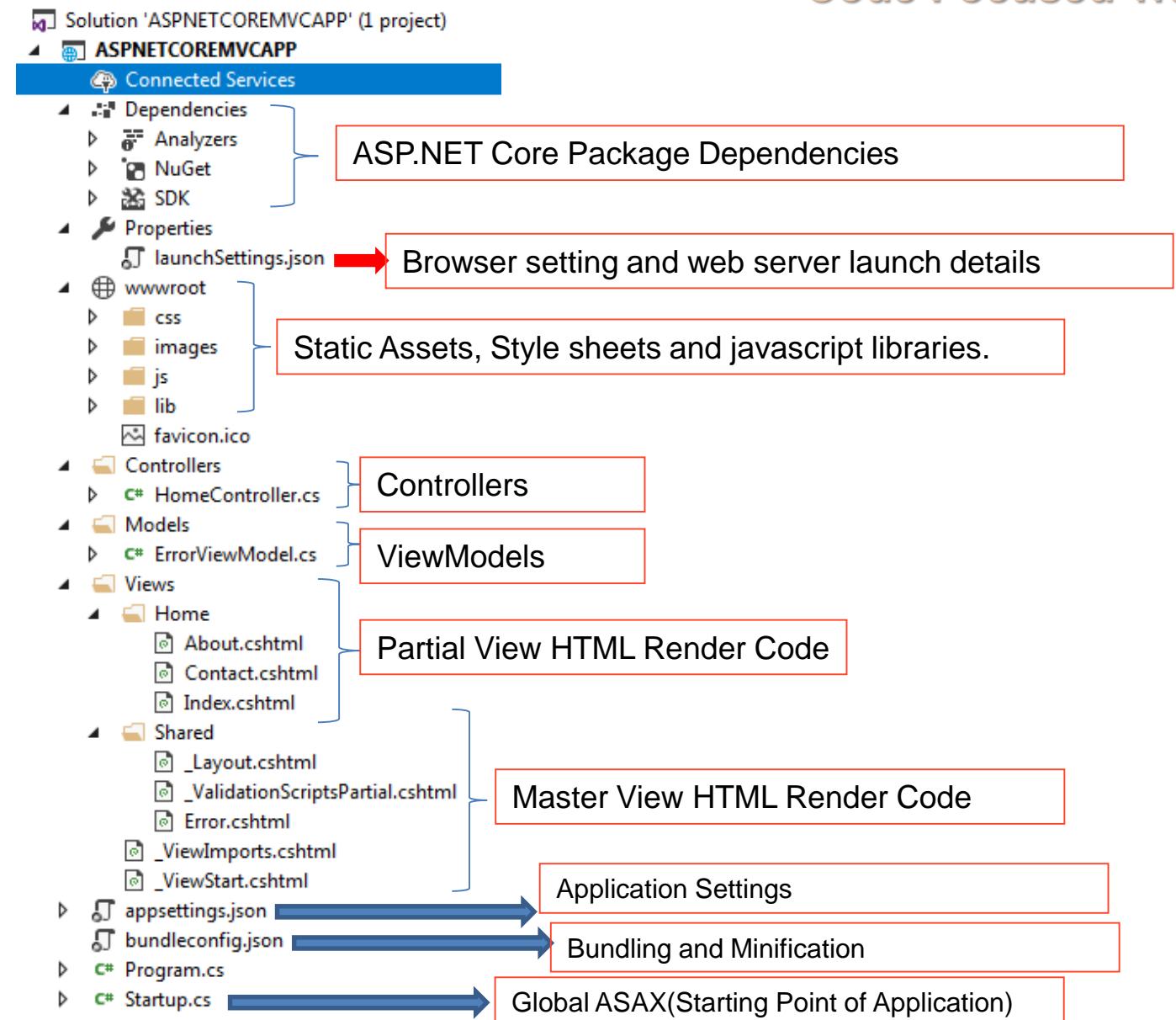
- Creating an ASP.NET Core 2.0 web application with model view controller



ASP.NET Core MVC Application structure

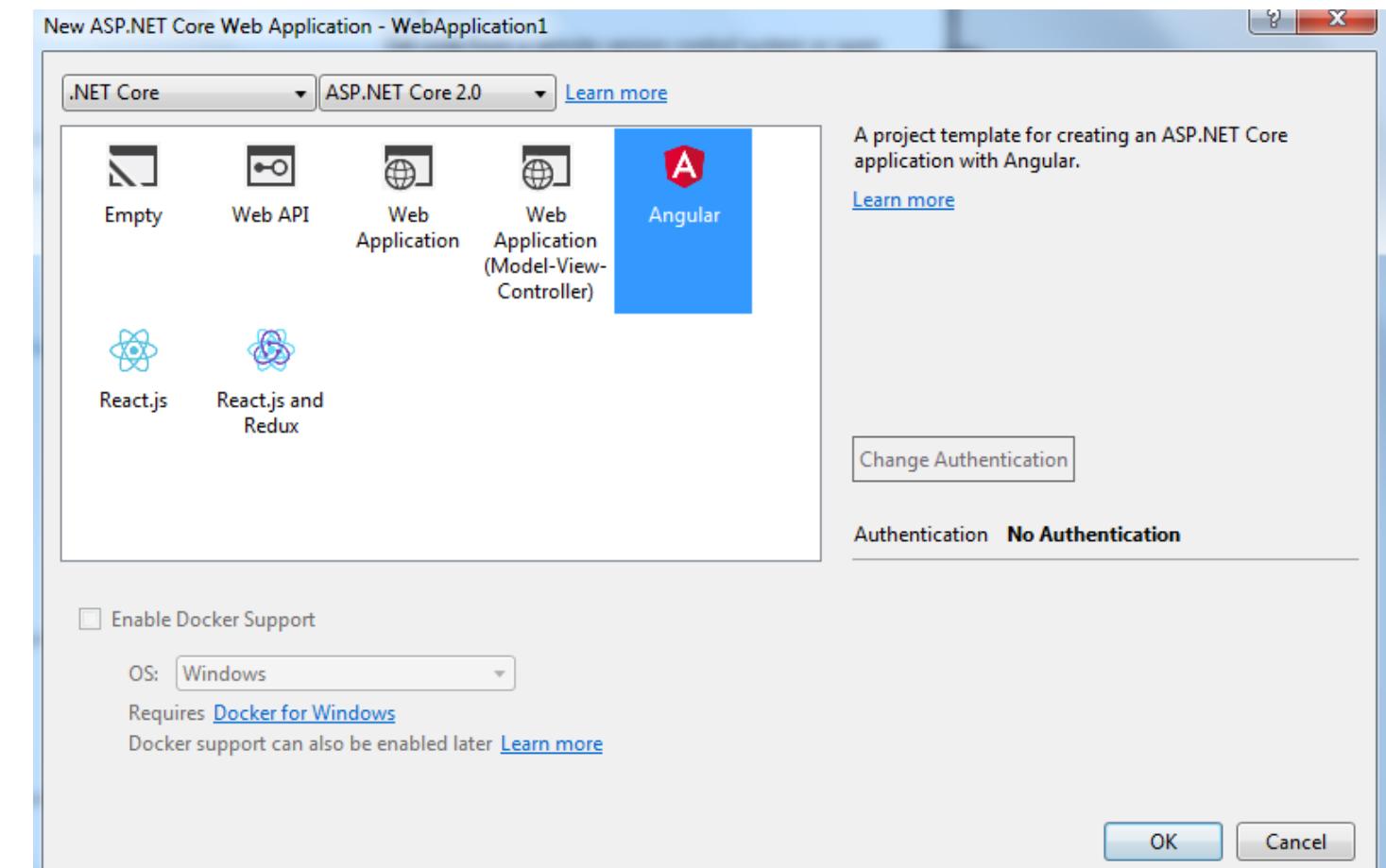
Please read terms and conditions of use

Original Series



Creating an ASP.NET Core 2.0 Web Application with Angular using Visual Studio 2017

- Creating an ASP.NET Core 2.0 web application with Angular

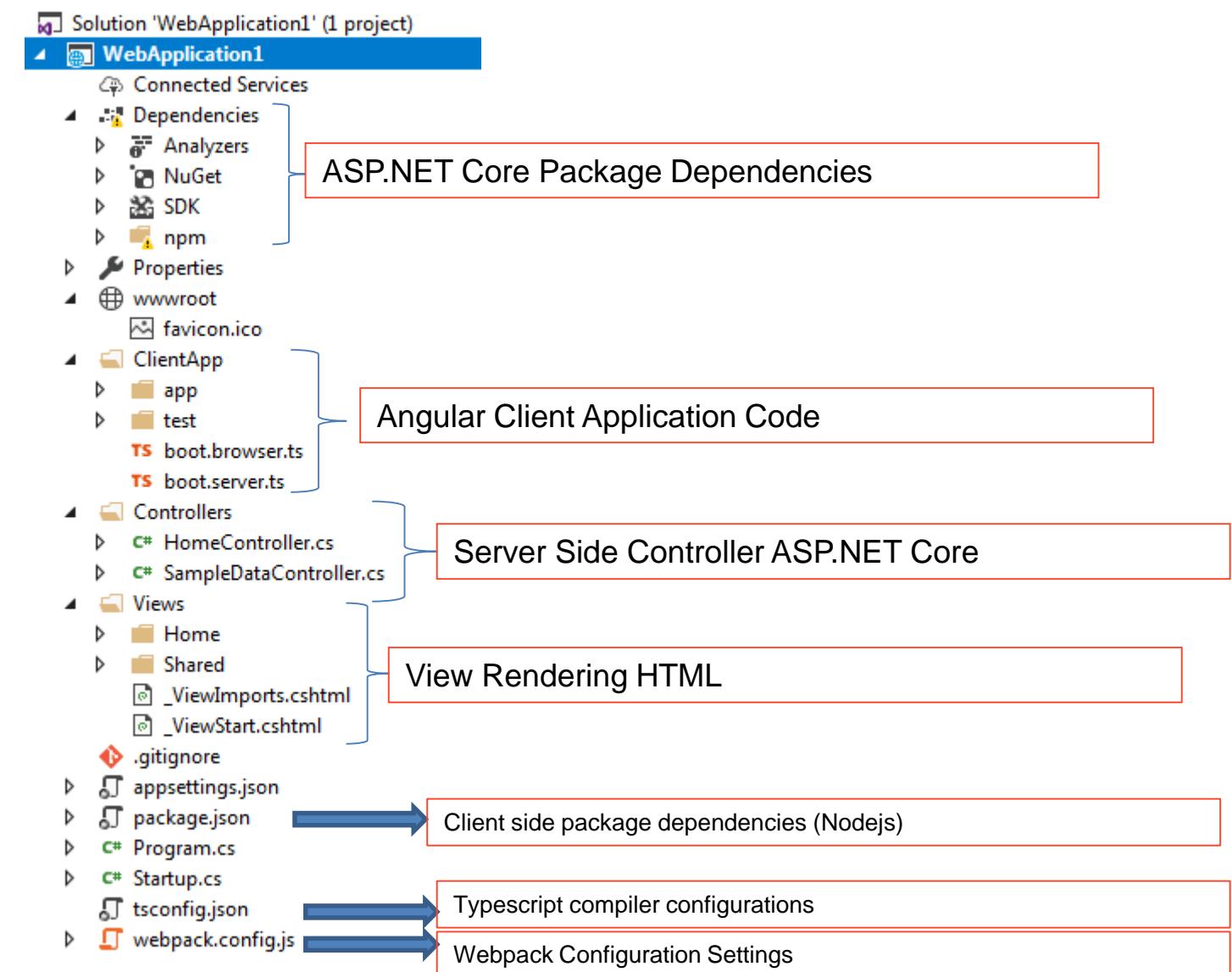


C# ASP.NET Core MVC with .NET Core Angular Application structure

Code Focused Training

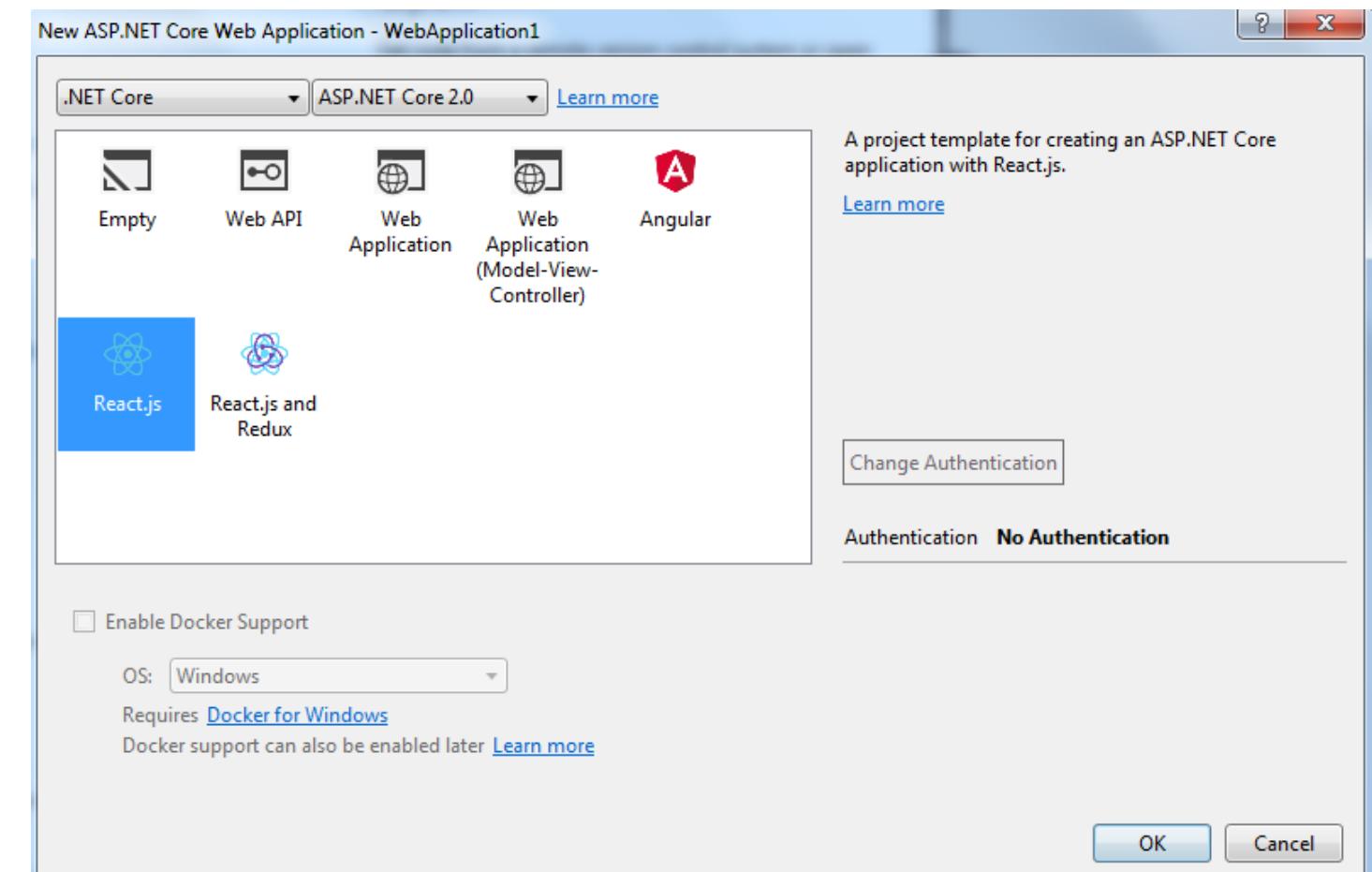
Please read terms and conditions of use

Original Series



Creating an ASP.NET Core 2.0 Web Application with ReactJS using Visual Studio 2017

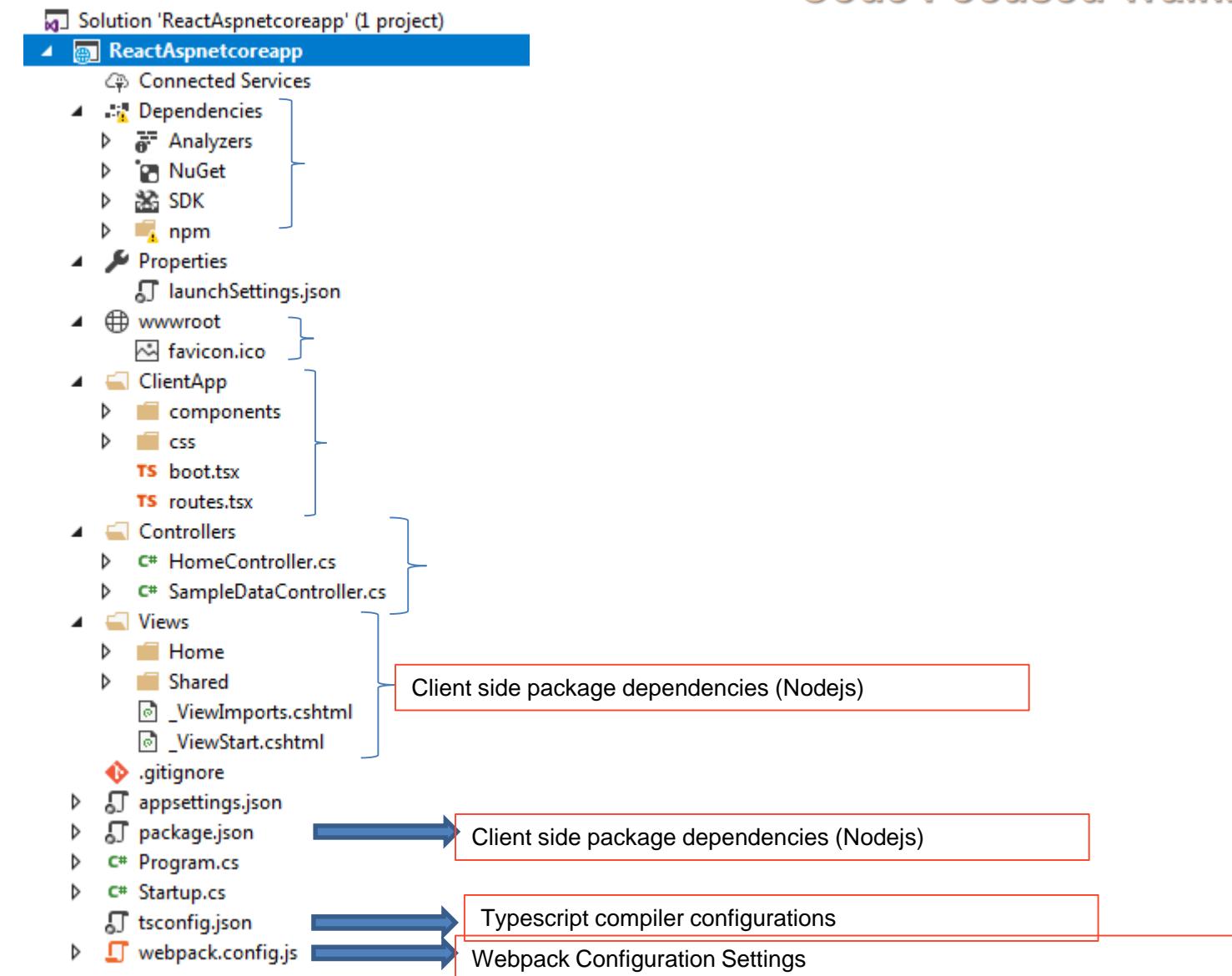
- Creating an ASP.NET Core 2.0 web application with ReactJS



ASP.NET Core MVC with ReactJS Application structure

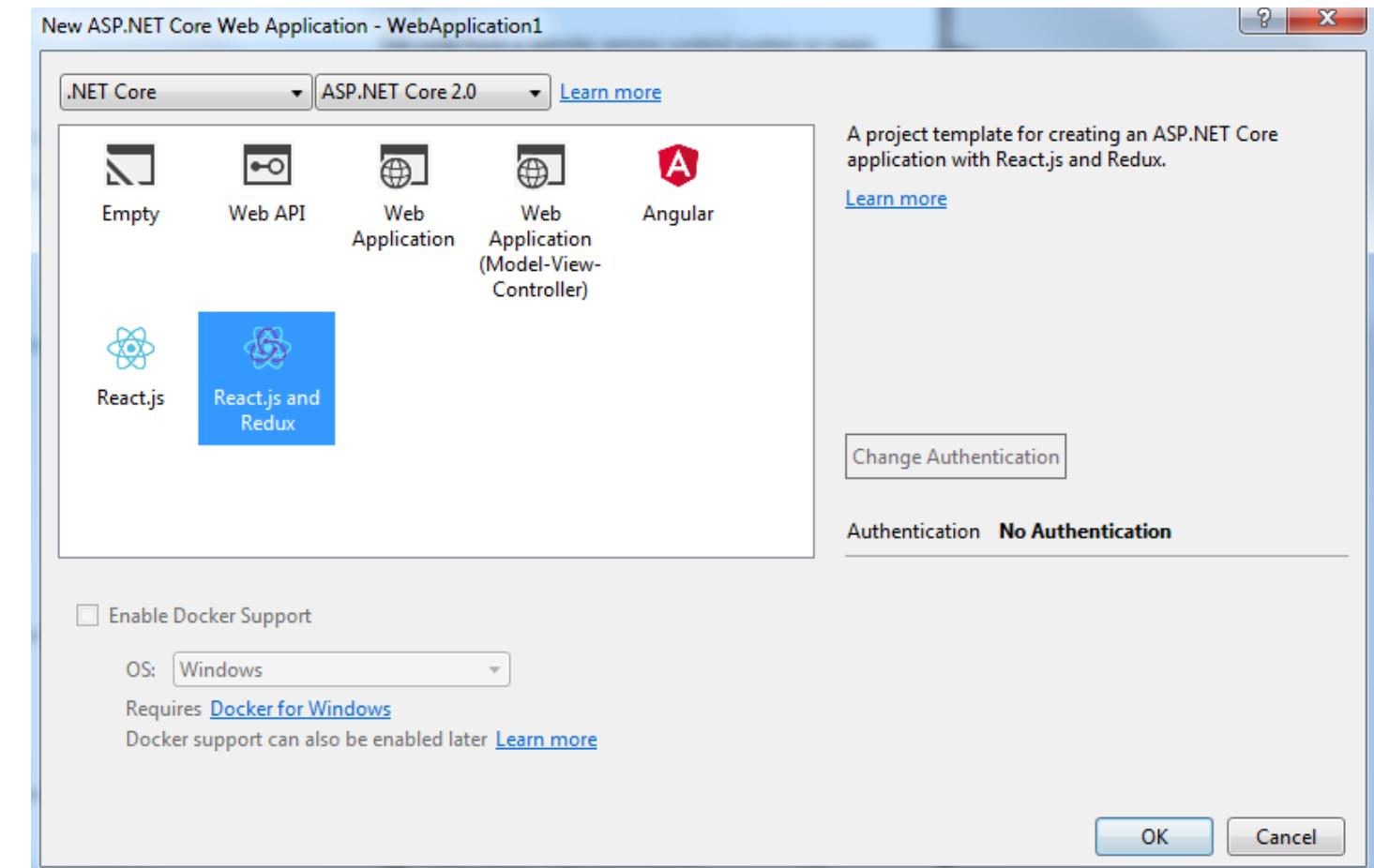
Please read terms and conditions of use

Original Series



Creating an ASP.NET Core 2.0 Web Application with React-Redux using Visual Studio 2017

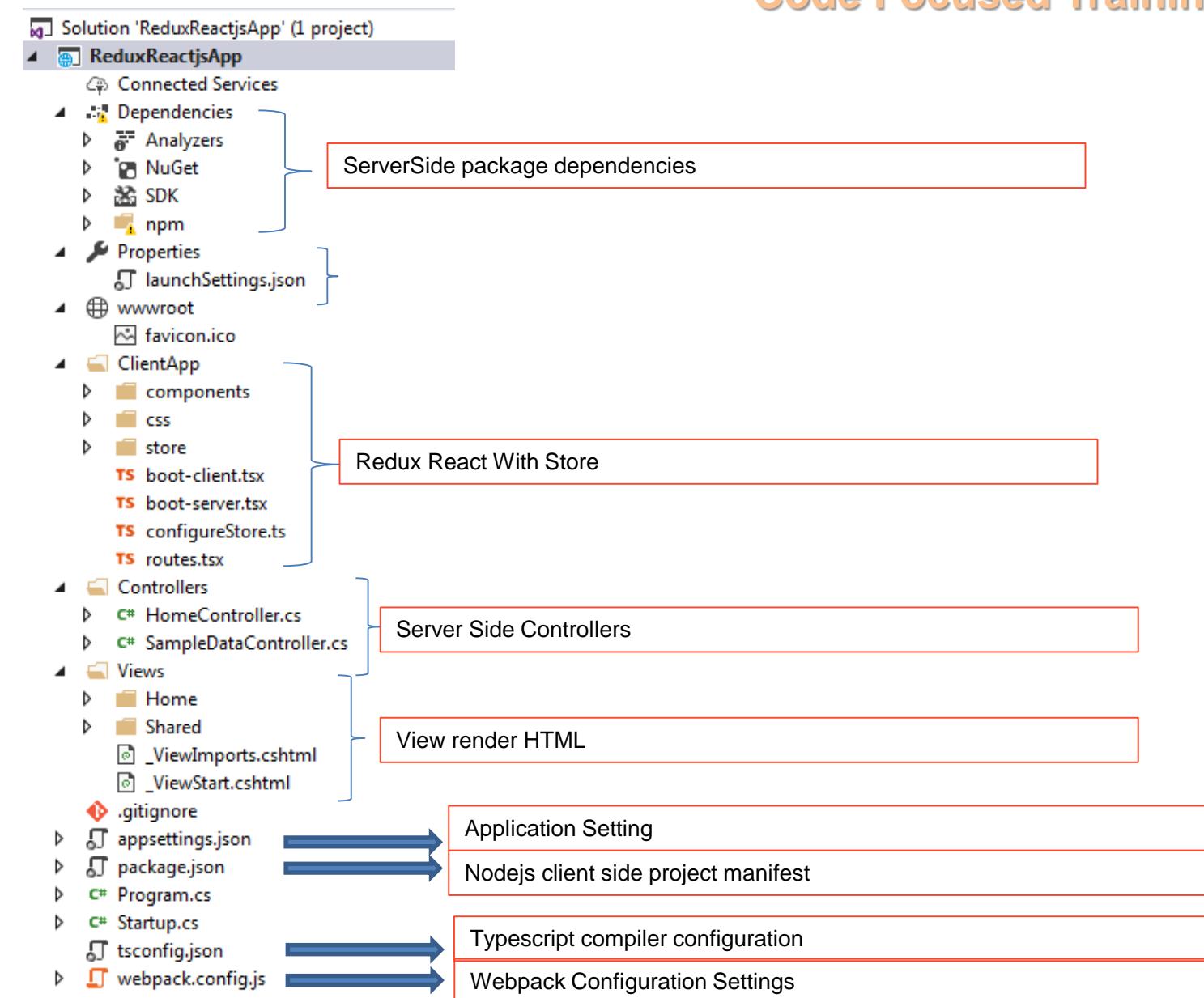
- Creating an ASP.NET Core 2.0 web application with ReactJS-Redux



ASP.NET Core MVC with ReactJS-Redux Application structure

Please read terms and conditions of use

Original Series



ZOMATO REVIEW APPLICATION -ASP.NET CORE WITH VISUAL STUDIO 2017

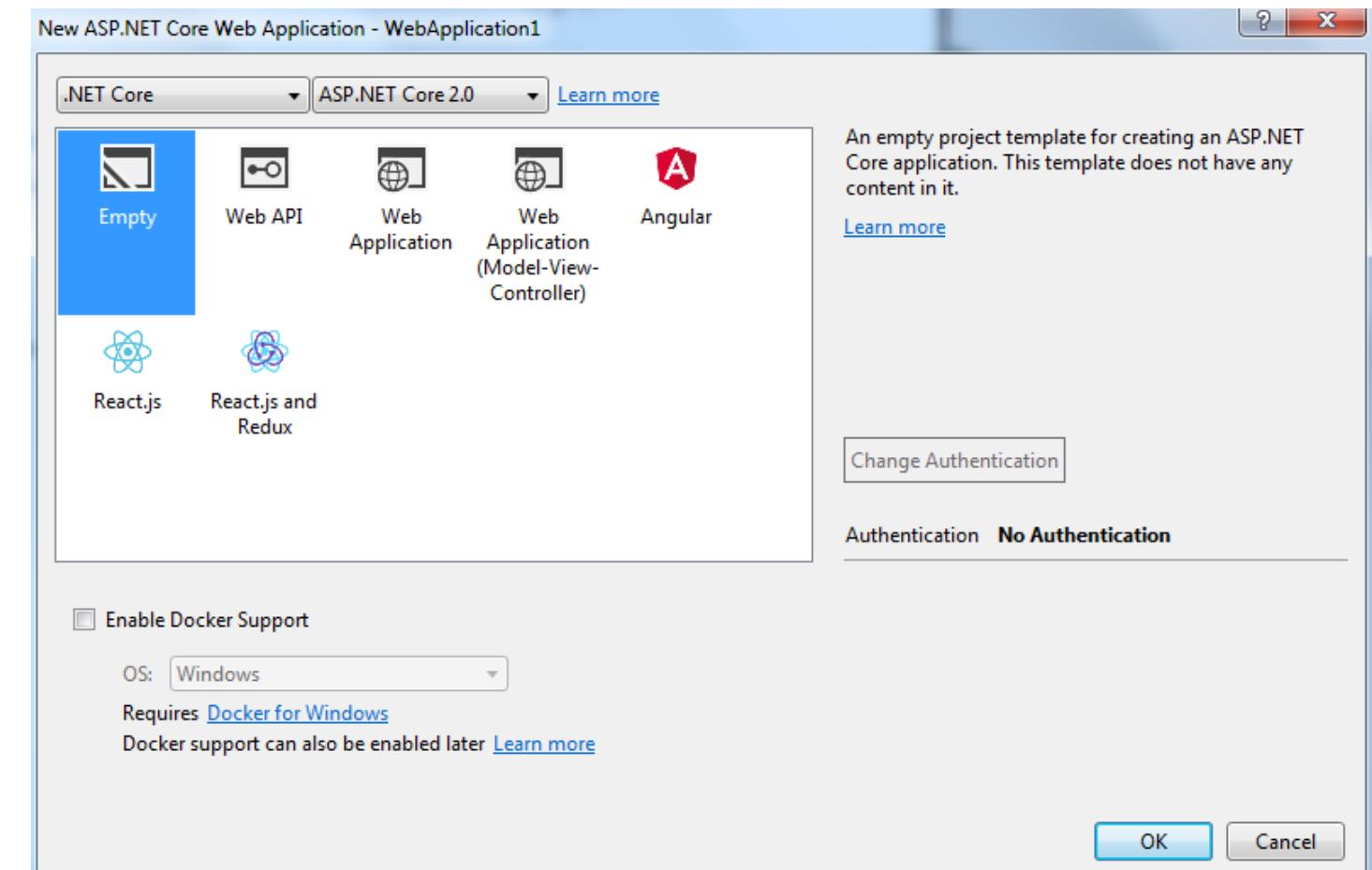
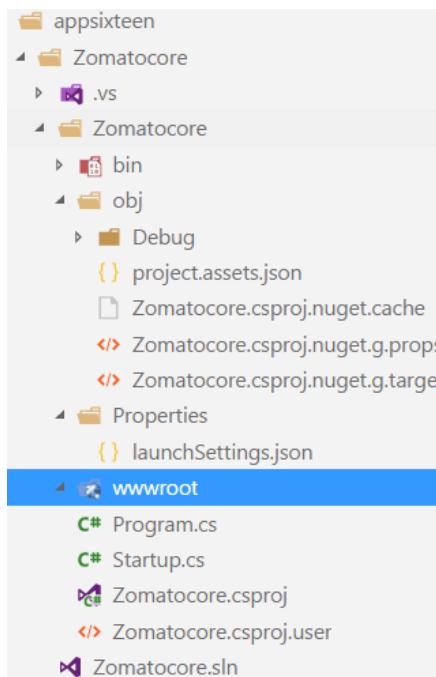
EXERCISE DEMO: 2.12

LEARNING OUTCOMES

- Creating an ASP.NET CORE 2.0 Empty application with Visual Studio 2017
- Using Configuration Builder to read appsetting.json (application configuration) and rendering the data to the view.
- Creating a service provider to read appsetting.json and rendering the data to the view.

Step 1: Creating an Empty ASP.NET Core 2.0 with Visual Studio 2017

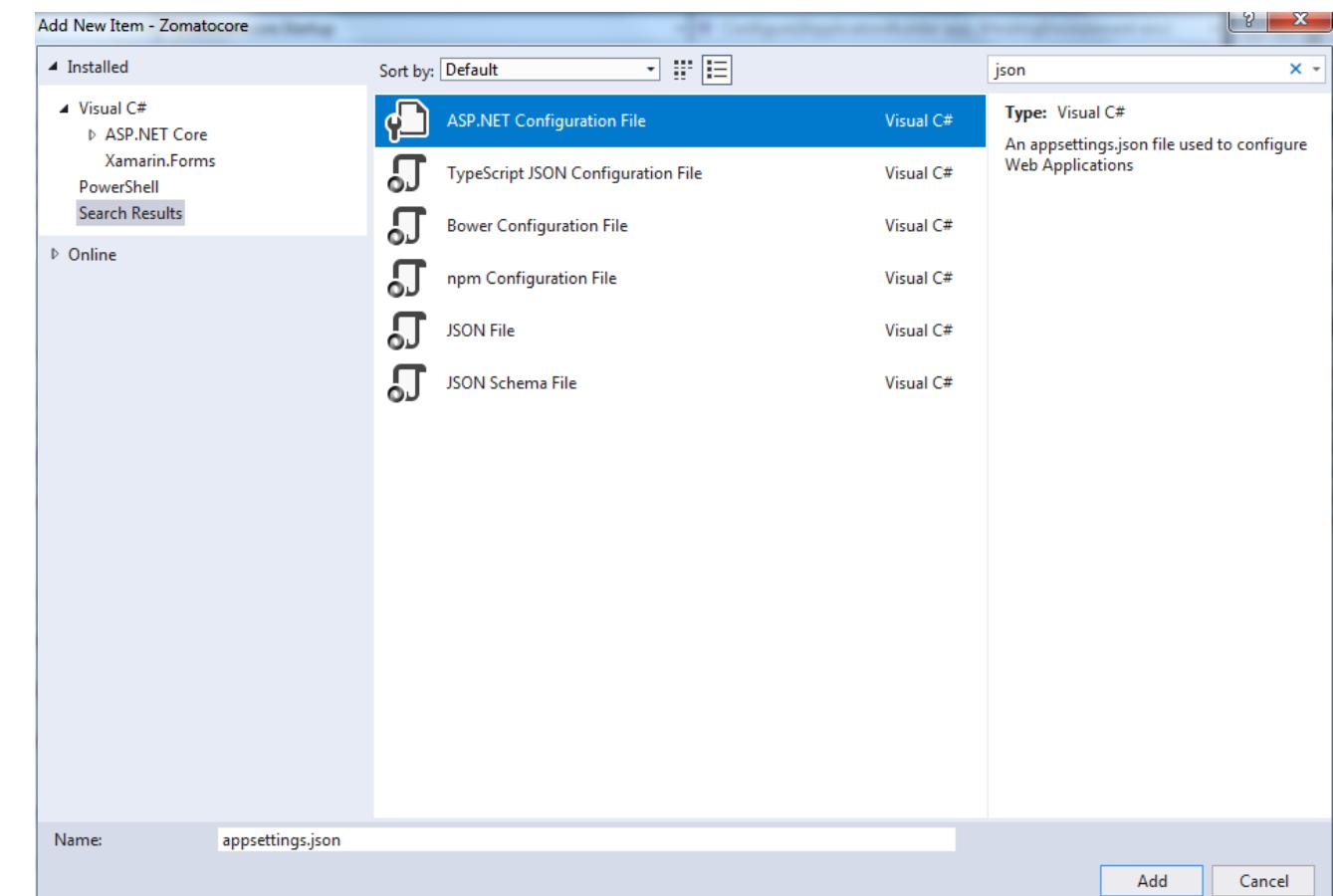
- Creating an empty
ASP.NET Core 2.0
application



Step 2:

Programming Application Configuration in ASP.NET Core 2.0

- Select the project and add a new item=> ASP.NET Configuration File



appsettings.json

```

{
  "ConnectionStrings": {
    "DefaultConnection": "Server=(localdb)\\MSSQLLocalDB;Database=_CHANGE_ME;Trusted_Connection=True;MultipleActiveResultSets=true"
  },
  "AppDataSet": "This is Application Specific Configuration Information-TPRI.SYCLIQ",
  "AppDbInfo": "This is your Application DB Configuration Information-TPRI.SYCLIQ"
}
  
```

Programming Application Configuration in ASP.NET Core 2.0

- Update Startup.cs (Global asax.cs) file with instance of ConfigurationBuilder();
- Run your application (ctrl+f5).
- You should be able to extract information from appsettings.json

```
public class Startup
{
    //ctor
    /// <summary> Import Microsoft.Extensions.Configuration IHostingEnvironment env ...
    0 references | 0 exceptions
    public Startup(IHostingEnvironment env)
    {
        var zomatoAppBuilder = new ConfigurationBuilder()
            .SetBasePath(env.ContentRootPath)
            .AddJsonFile("appsettings.json")
            .AddEnvironmentVariables();
        //building my application
        Configuration = zomatoAppBuilder.Build();
    }

    3 references | 0 exceptions
    public IConfiguration Configuration { get; set; }
    // This method gets called by the runtime. Use this method to add services to the container.
    // For more information on how to configure your application, visit https://go.microsoft.com/fwlink/?LinkID=398940
    0 references | 0 exceptions
    public void ConfigureServices(IServiceCollection services)
    {

        // 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.Run(async (context) =>
            {
                var appDataSetMessage = Configuration["AppDataSet"];
                var appDbInfoMsg = Configuration["AppDbInfo"];
                await context.Response.WriteAsync(appDataSetMessage);
            });
        }
    }
}
```



This is Application Specific Configuration Information-TPRI.SYCLIQ

Programming Service Configuration in ASP.NET Core 2.0

- Create InfoService.cs class which implements an interface IInfoService

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace Zomatocore
{
    1 reference
    public interface IInfoService
    {
        1 reference | 0 exceptions
        string GetRestaurantInfo();
    }
    0 references
    public class InfoService : IInfoService
    {
        1 reference | 0 exceptions
        public string GetRestaurantInfo()
        {
            return "User! You have requested for Restaurant Information";
        }
    }
}
```

Step 5: Registering the service in Startup.cs=> configure method

- Register the service in configure method of Startup.cs.

```
0 references | 0 exceptions
public Startup(IHostingEnvironment env)
{
    var zomatoAppBuilder = new ConfigurationBuilder()
        .SetBasePath(env.ContentRootPath)
        .AddJsonFile("appsettings.json")
        .AddEnvironmentVariables();
    //building my application
    Configuration = zomatoAppBuilder.Build();
}

3 references | 0 exceptions
public IConfiguration Configuration { get; set; }
// This method gets called by the runtime. Use this method to add services to the container.
// For more information on how to configure your application, visit https://go.microsoft.com/fwlink/?LinkID=398940
0 references | 0 exceptions
public void ConfigureServices(IServiceCollection services)
{
    //lifetime service lifetimes
    //AddTransient ,AddScoped, AddTransietnt
    services.AddSingleton<IInfoService, InfoService>(); ←
}

// 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,IInfoService myservice)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }
    app.Run(async (context) =>
    {
        //fetching data from service
        var svcMessage = myservice.GetRestaurantInfo(); ←
        var appDataSetMessage = Configuration["AppDataSet"];
        var appDbInfoMsg = Configuration["AppDbInfo"];
        // await context.Response.WriteAsync(appDataSetMessage);
        await context.Response.WriteAsync(svcMessage); ←
    });
}
```

Step 6: Registering the service in Startup.cs=> configure method

- Let us revisit the service again and rewrite the service to use configuration builder interface that enables us to read the appsettings.json.
- Instead of hardcoding some message, now our service is able to read from the appsettings.json file using the IConfiguration Builder

```
namespace Zomatocore
{
    3 references
    public interface IInfoService
    {
        2 references | 0 exceptions
        string GetRestaurantInfo();
    }
    2 references
    public class InfoService : IInfoService
    {
        private string _appsettingMessage; ←
        0 references | 0 exceptions
        public InfoService(IConfiguration configuration) ←
        {
            _appsettingMessage = configuration["AppDataSet"]; ←
        }
        2 references | 0 exceptions
        public string GetRestaurantInfo()
        {
            // return "User! You have requested for Restaurant Information";
            return _appsettingMessage; ←
        }
    }
}
```

Step 7: Registering the service in Startup.cs=> configure method

- Let us update the startup.cs with changes that have been made in step 6. lets bind it to the configuration service.
- Lets run our

```
public Startup(IHostingEnvironment env)
{
    var zomatoAppBuilder = new ConfigurationBuilder()
        .SetBasePath(env.ContentRootPath)
        .AddJsonFile("appsettings.json")
        .AddEnvironmentVariables();
    //building my application
    Configuration = zomatoAppBuilder.Build();
}

4 references | 0 exceptions
public IConfiguration Configuration { get; set; } ←
// This method gets called by the runtime. Use this method to add services to the container.
// For more information on how to configure your application, visit https://go.microsoft.com/fwlink/?LinkID=398940
0 references | 0 exceptions
public void ConfigureServices(IServiceCollection services)
{
    //binding with configuration service
    services.AddSingleton(Configuration);
    //lifetime service lifetimes
    //AddTransient ,AddScoped, AddTransietnt
    services.AddSingleton<IInfoService, InfoService>(); ←

    // 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,IInfoService myservice)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }
    app.Run(async (context) =>
    {
        ///fetching data from service
        var svcMessage = myservice.GetRestaurantInfo();
        var appDataSetMessage = Configuration["AppDataSet"];
        var appDbInfoMsg = Configuration["AppDbInfo"];
        // await context.Response.WriteAsync(appDataSetMessage);
        await context.Response.WriteAsync(svcMessage);
    });
}
```

Middleware Functions Demo

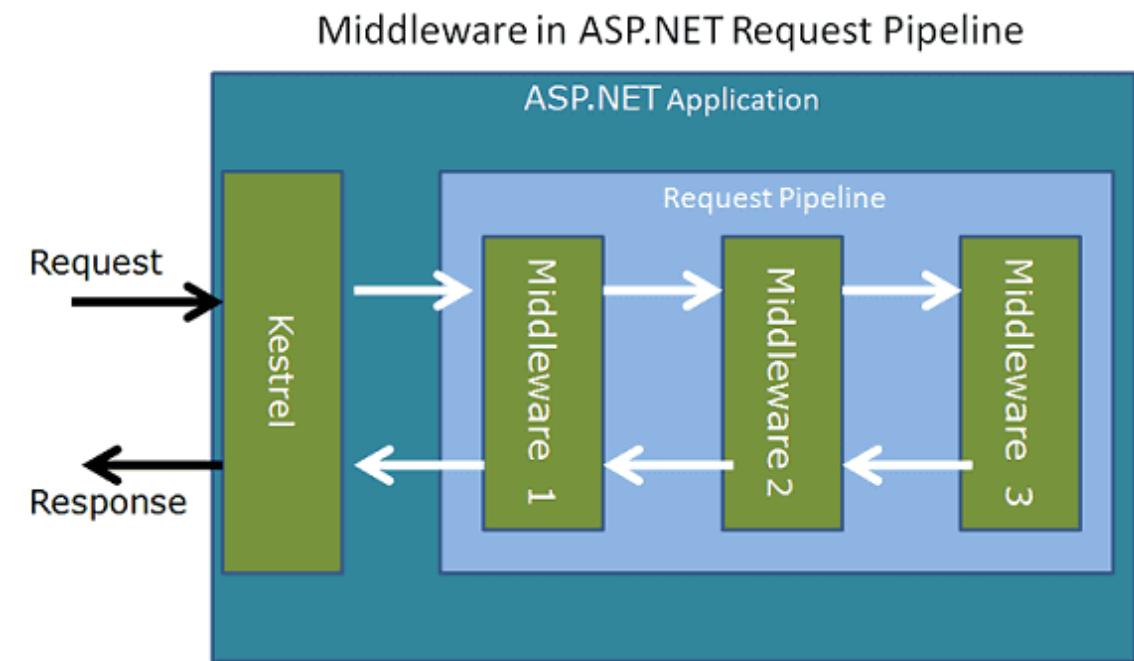
EXERCISE DEMO: 2.13

LEARNING OUTCOMES

- Explain middleware and http request pipeline
- Middleware entirely defines how the application behaves and responds to HTTP Requests
- Write a custom middleware functionality
- Demonstrate how to consume various middleware functions.

Request Pipeline

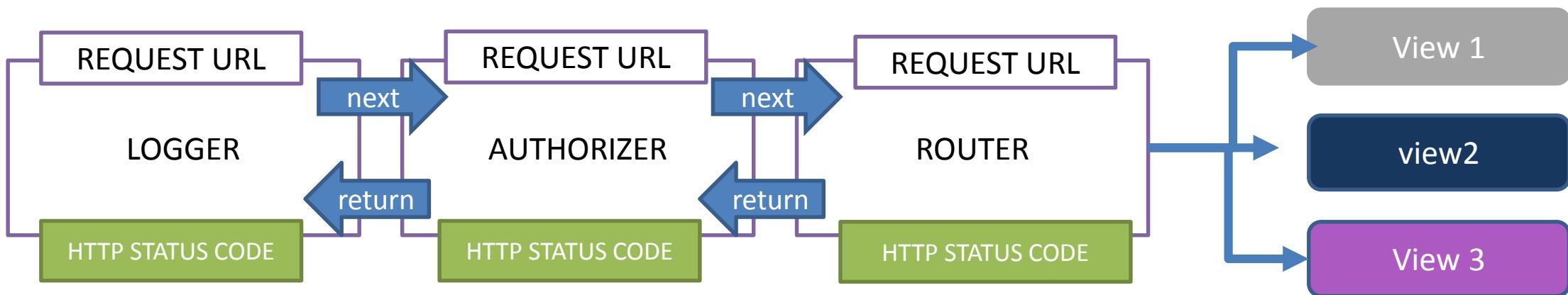
- A mechanism by which requests are processed beginning with a request (HTTP request) and ending with a response (HTTP response).
- The pipeline specifies how the application should respond to the HTTP request. The request arriving from the browser goes through the pipeline and back.
- Individual components that make up the pipeline are called Middleware.



Middleware

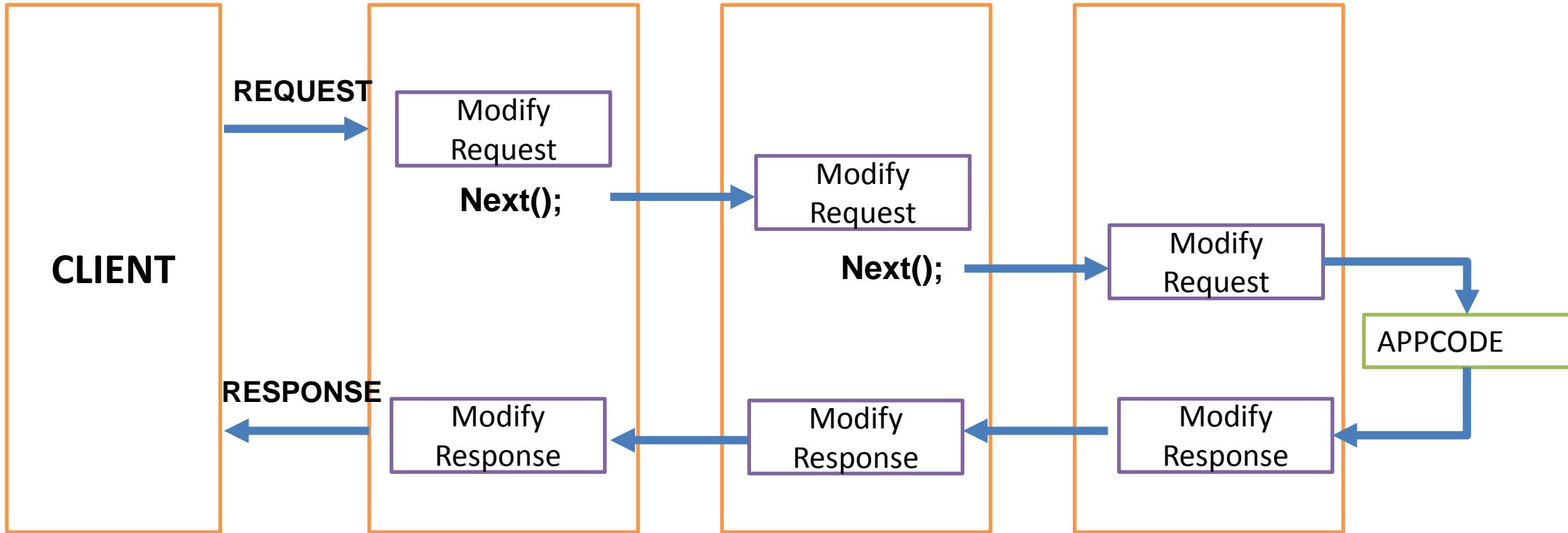
Please read terms and conditions of use

- A software component that hooks into the request pipeline to handle web requests and generates responses.
- Each middleware process and manipulates the request as it is received from the previous middle ware.
- It may decide to call the next middleware in the pipeline or send the response back to the previous middleware(terminating the pipeline)



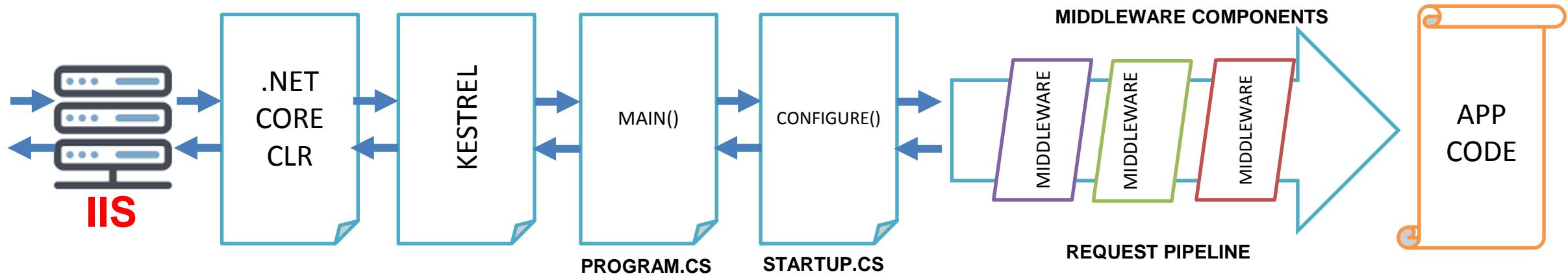
Middleware pipeline

Please read terms and conditions of use
Original Series



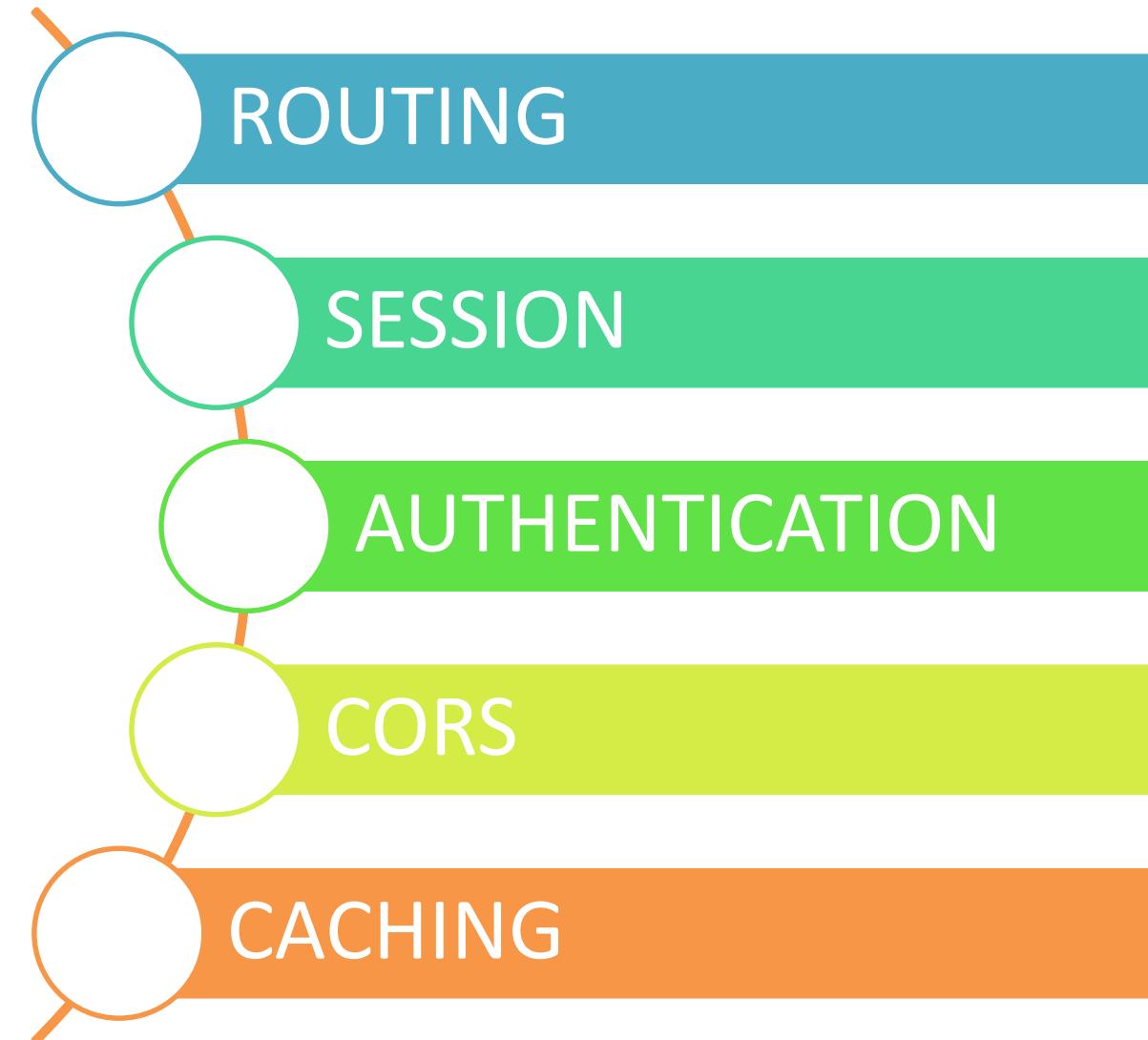
Application Flow

Please read terms and conditions of use



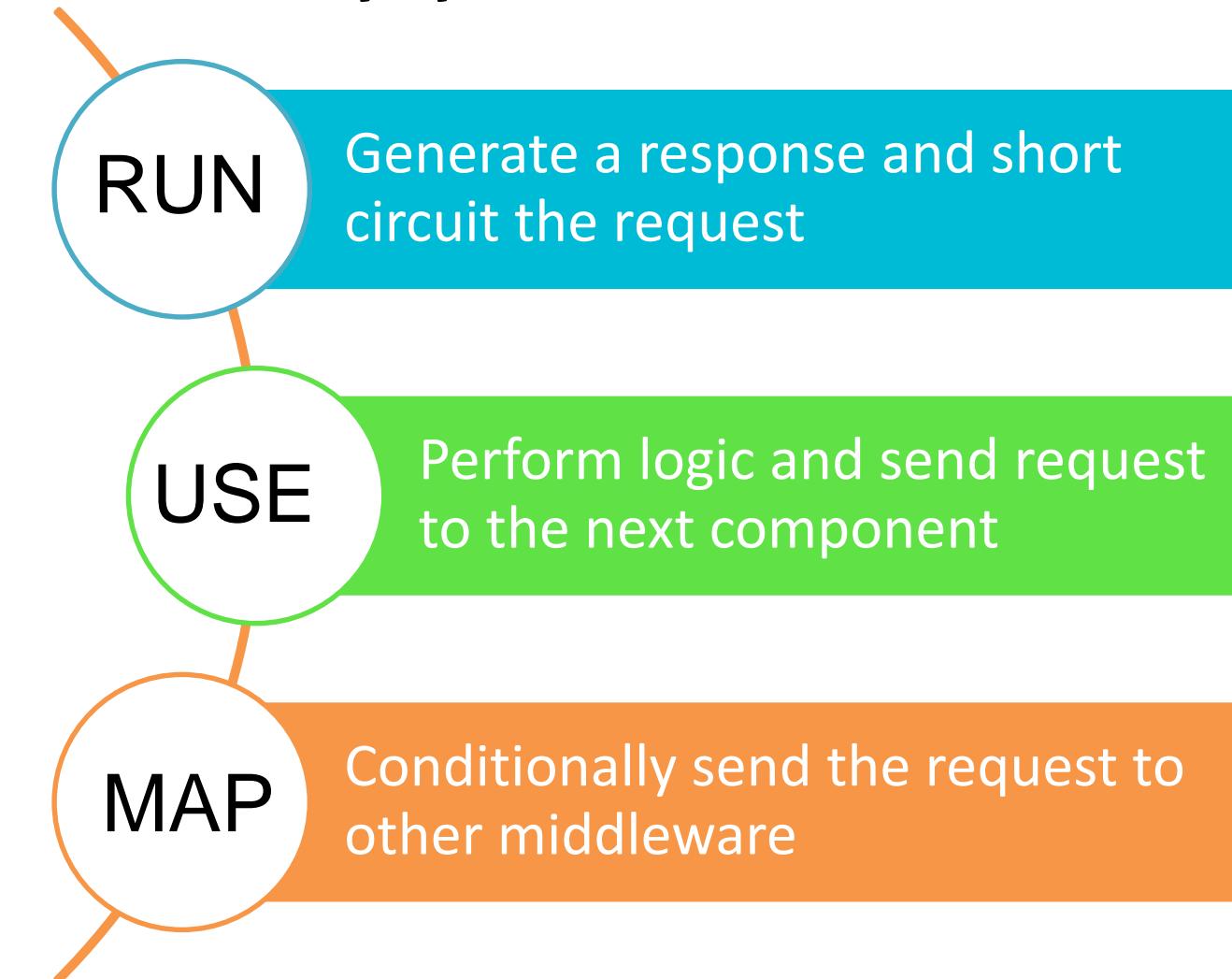
Original Series

MIDDLEWARE COMPONENTS



Configuring the middleware pipeline

MIDDLEWARE



Request Delegates

- When dealing with a request we use `IApplicationBuilder`
- It offers 4 methods to interact with a request
 - Use
 - Run
 - Map
 - MapWhen
- Use: adds a middleware to the application pipeline and it can either pass the request to the next delegate or it can end the request.
- Map: used to connect a request path with another middleware. It can use any of the other request delegates.
- MapWhen: Similar to Map except that we can specify the detailed condition by using a `HttpContext` Object. We could check for URL,headers, query strings, cookies, etc.
- Run: Once application executes this middleware it will reach the end of the request pipeline. Consequently, after that response pipeline starts executing in reverse order.

Use

Please read terms and conditions of use

Original Series

- Adds a middle ware to the application pipeline and it can either pass the request to the next delegate or it can end the request.
- Most commonly used method to interact with middleware.

Map

Please read terms and conditions of use

Original Series

- Used to connect a request path with another middleware. The middleware can use any of the other mentioned request delegates.

MapWhen

- It behaves the same as Map except that we can specify the detailed condition by using a HttpContext Object. We could check for URL, headers, query strings, cookies, etc.
- It helps us to split the middleware pipeline into two completely separate branches by specifying a predicate.

UseWhen

- UseWhen is used when some conditional pieces- require specific middleware that should only run for certain request.
- It uses a predicate to determine if the middleware should run.
- It continues to execute later middleware regardless of whether the UseWhen predicate was true or false.
- It helps in configuring the middleware workflow. This is widely used in the following scenarios
 - Restrict output caching to anonymous users.
 - Add diagnostic headers for a certain IP subnet
 - Handle errors differently for Web API or MVC actions.
 - Restrict certain requests from analytics

Run

Please read terms and conditions of use

Original Series

- Once application executes this middleware it will reach the end of the request pipeline. Consequently, after that response pipeline starts executing in reverse order.

Middleware functions

Please read terms and conditions of use

Original Series

App.UseApplicationInsightsExceptionTelemetry()	
App.UseApplicationInsightsRequestTelemetry()	
App.UseAuthentication()	Authentication Support
App.UseCookieAuthentication()	Cookie based Authentication
App.UseCookiePolicy()	Cookie Policy
App.Cors()	Cross Origin Resource Sharing
App.UseDatabaseErrorPage()	Database Connectivity
App.UseDefaultFiles()	Default Files

App.UseDeveloperExceptionPage()	Error Page
App.UseDirectoryBrowser()	Browsing Directory
App.UseFacebookAuthentication();	Facebook Authentication
App.UseExceptionHandler()	Exception Handler
App.UseFileServer();	File Server
App.UseForwardedHeaders()	Redirection/Forwarding
App.UseGoogleAuthentication()	Google Authentication

Middleware functions

Please read terms and conditions of use

Original Series

App.UseHttpMethodOveride()	
App.UseIdentity()	
App.UseJwtBearerAuthentication()	JWT Authentication
App.UseMicrosoftAccountAuthentication()	
App.UseMiddleware()	
App.UseMigrationsEndPoint()	
App.UseMvcWithDefaultRoute()	

App.UseWebpackDevMiddlware()	
App.UseOAuthAuthentication()	
App.UseOpenIdConnectAuthentication()	
App.useOwin()	
App.UsePathBase()	
App.UseRouter()	
App.UseRequestLocalization()	
App.UseResponseCaching()	

Middleware functions

Please read terms and conditions of use

Original Series

App.UseResponseCompression()	
App.UseRewriter()	
App.UseStaticFiles()	
App.UseStatusCodePages()	
App.UseStatusCodePagesWithRedirects();	
App.UseStatusCodePagesWithReExecute()	
App.UseTwitterAuthentication()	
App.UseWebpackDevMiddleware()	

App.UseWebSockets()	
App.UseWelcomePage()	Simple Welcome Page rendering
App.UseWhen()	

Note: The order in which we invoke the Middleware functions has significant impact on your application

Middleware Components

Please read terms and conditions of use

Original Series

Component	Project	Component	Project
AnalysisMiddleware	Diagnostics	ElmPageMiddleware	Diagnostics
AntiforgeryMiddleware	Antiforgery	ExceptionHandlerMiddleware	Diagnostics
AuthenticationMiddleware	Security	FacebookMiddleware	Security
ClaimsTransformationMiddleware	Security	ForwardedHeadersMiddleware	BasicMiddleware
CookieAuthenticationMiddleware	Security	GoogleMiddleware	Security
CookiePolicyMiddleware	Security	HttpMethodOverrideMiddleware	BasicMiddleware
CorsMiddleware	CORS	IISPlatformHandlerMiddleware	IISIntegration
DatabaseErrorPageMiddleware	Diagnostics	JwtBearerMiddleware	Security
DefaultFilesMiddleware	StaticFiles	MapMiddleware	HttpAbstractions
DeveloperExceptionPageMiddleware	Diagnostics	MapWhenMiddleware	HttpAbstractions
DirectoryBrowserMiddleware	StaticFiles	MicrosoftAccountMiddleware	Security

Middleware Components

Please read terms and conditions of use

Original Series

Component	Project	Component	Project
MigrationsEndPointMiddleware	Diagnostics	StatusCodePagesMiddleware	Diagnostics
OAuthMiddleware	Security	TwitterMiddleware	Security
OpenIdConnectMiddleware	Security	WebSocketMiddleware	WebSockets
ProxyMiddleware	Proxy	WelcomePageMiddleware	Diagnostics
RequestLocalizationMiddleware	Localization		
RequestServicesContainerMiddleware	Hosting		
ResponseBufferingMiddleware	BasicMiddleware		
RouterMiddleware	Routing		
RuntimeInfoMiddleware	Diagnostics		
SessionMiddleware	Session		
StaticFileMiddleware	StaticFiles		

Step 1:

Writing a custom middleware function.

- Add a custom middleware function in the startup.cs => configure method
- We have used delegate to direct to the next function in the request pipe line.
- The order in which we invoke the middleware function has significant impact on your application.

```
//CustomMiddleware  
///to render a middleware we use app.Use app is reference to IApplicationBuilder  
/// inject ILogger<Startup> logger (using Microsoft.Extensions.Logging)  
app.Use(next =>  
{  
    return async context =>  
    {  
        logger.LogInformation('Incoming HTTP Request');  
        if (context.Request.Path.StartsWithSegments("/sycliq"))  
        {  
            await context.Response.WriteAsync("You are using SycliQ Custom Middleware Route");  
            logger.LogInformation("Http Request has been handled");  
        }  
        else  
        {  
            await next(context);  
            logger.LogInformation("Outgoing Http Response");  
        }  
    };  
});
```

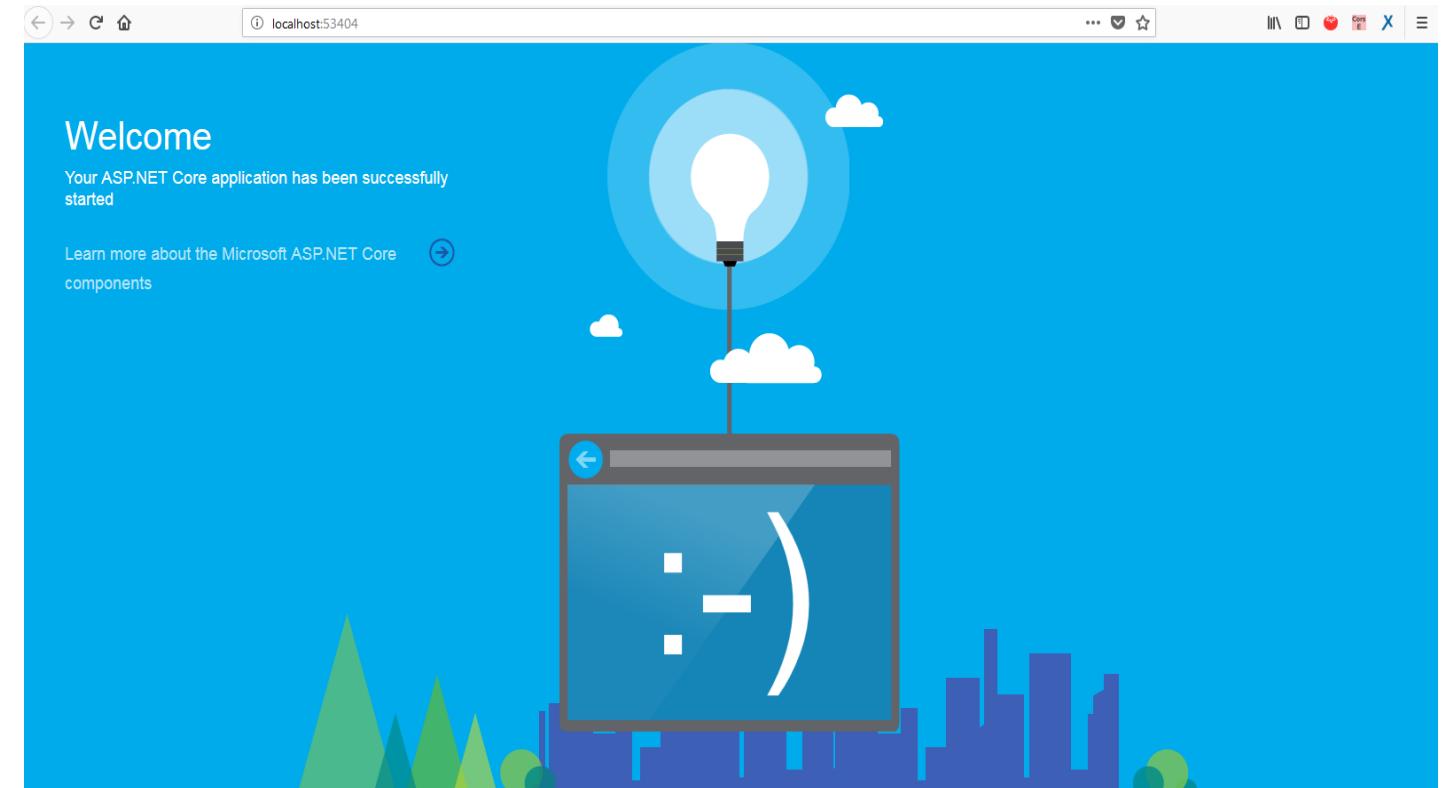


UseWelcomePage middleware function.

- It adds welcome page when your application starts.
- App.UseWelcomePage();
- Additional Options, for redirecting to a specific path to render the welcome page is also possible.

Add this to startup.cs => configure method

```
//WelcomePage Middleware => adds a welcome page to the default route.  
app.UseWelcomePage();
```

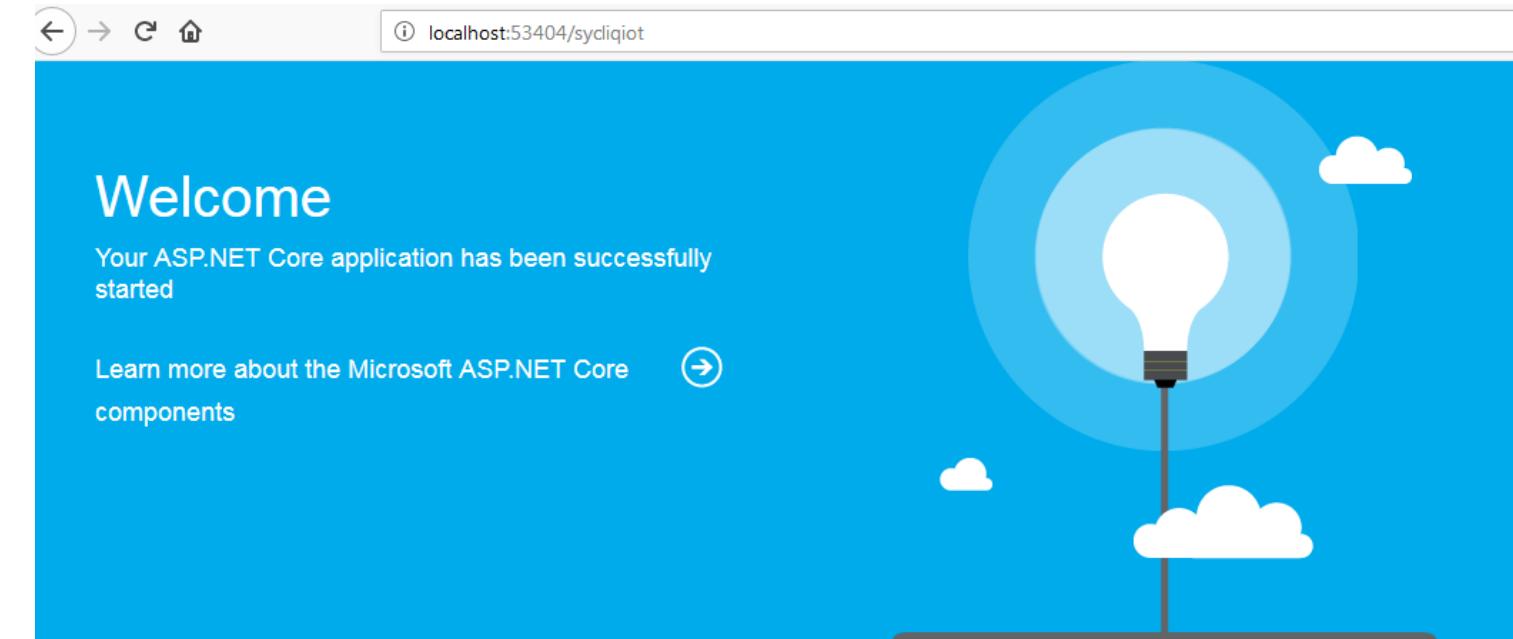


Step 2:
UseWelcomePage**middleware function with options.**

- WelcomePageOptions can be customized to a specific path to render the WelcomePage().
- App.UseWelcomePage(new WelcomePageOptions{ Path="/sycliqiot" });

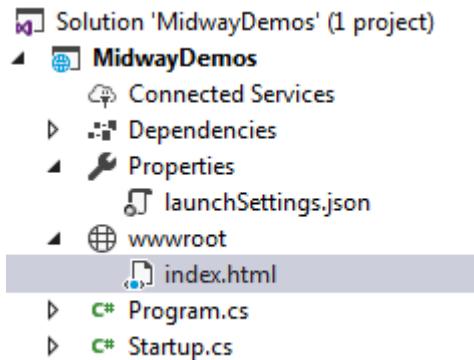
Add this to startup.cs => configure method

```
app.UseWelcomePage(new WelcomePageOptions {  
    Path="/sycliqiot"  
});
```



UseStaticFiles() middleware function with options.

- Used to render staticfiles from the wwwroot folder.

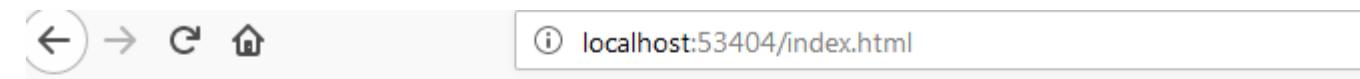


- Serving files that reside outside the webroot.

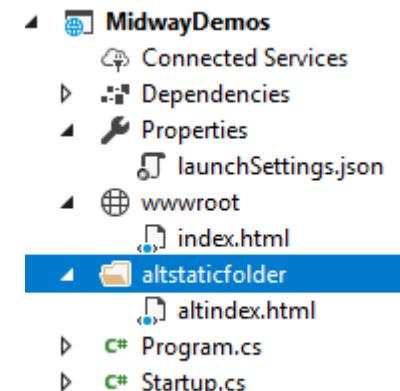
Add this to startup.cs => configure method

```
//defines the environment for static files to render
app.UseDefaultFiles();

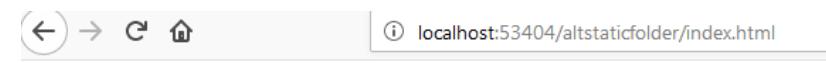
//used to render staticfiles created in wwwroot/index.html
app.UseStaticFiles();
```



Test page to test static file middleware rendering



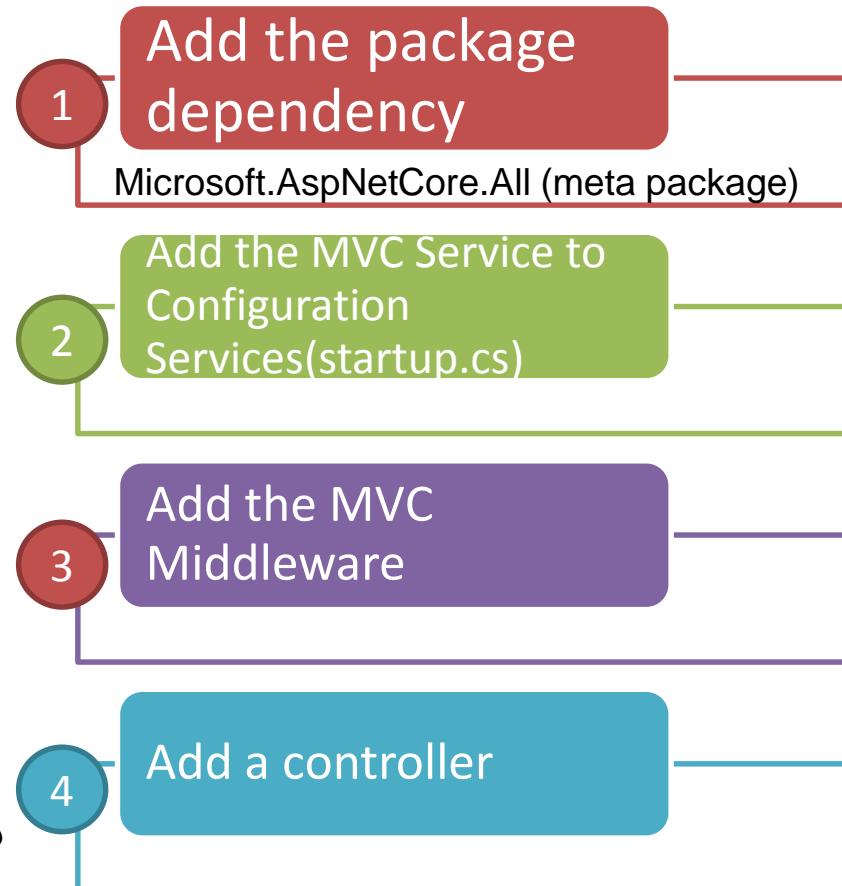
```
//custom folder
app.UseStaticFiles(new StaticFileOptions
{
  FileProvider = new PhysicalFileProvider(
    Path.Combine(Directory.GetCurrentDirectory(), "altstaticfolder")),
  RequestPath = "/altstaticfolder"
});
```



Alternative Static File rendering

Setting up ASP.NET MVC Middleware

Please read terms and conditions of use



2

```
public void ConfigureServices  
(IServiceCollection services)  
{  
    services.AddMvc();  
}
```

Add this to `startup.cs` => `configure` method

3

```
//include MVCwith Default Router  
app.UseMvcWithDefaultRoute();
```

4

Solution 'MidwayDemos' (1 project)

- MidwayDemos
 - Connected Services
 - Dependencies
 - Properties
 - launchSettings.json
 - wwwroot
 - index.html
 - altstaticfolder
 - index.html
 - Controllers
 - HomeController.cs
 - Program.cs
 - Startup.cs

namespace MidwayDemos.Controllers

```
public class HomeController  
{  
    // GET: /<controller>/  
    public String Index()  
    {  
        return "SycliQ MVC Controller Middleware Barebones";  
    }  
}
```

localhost:53404

SycliQ MVC Controller Middleware Barebones

Microsoft.Extensions.Logging

Please read terms and conditions of use

Original Series

Log Level	Severity	Extension Method	Description
Trace	0	LogTrace()	Log messages only for tracing purpose for the developers
Debug	1	LogDebug()	Log messages for short-term debugging purpose
Information	2	.LogInformation()	Log messages for the flow of the application.
Warning	3	.LogWarning()	Log messages for abnormal or unexpected events in the application flow.
Error	4	.LogError()	Log error messages.
Critical	5	LogCritical()	Log messages for the failures that require immediate attention

Summary

- Middleware forms the request handling pipeline
- Application configuration occurs inside of Program and Startup classes.
- An application can have as many middleware components.
- **Order of the registered middleware components matter.**
- Middleware components can short-circuit the request and generate the request immediately.
- Middleware helps us to write lightweight and modular apps.
- Three ways of configuring the middleware pipeline.
 - Run: generate a response and short-circuit the request.
 - Use-Perform logic and send request to next part
 - Map-conditionally send the request to other middleware.

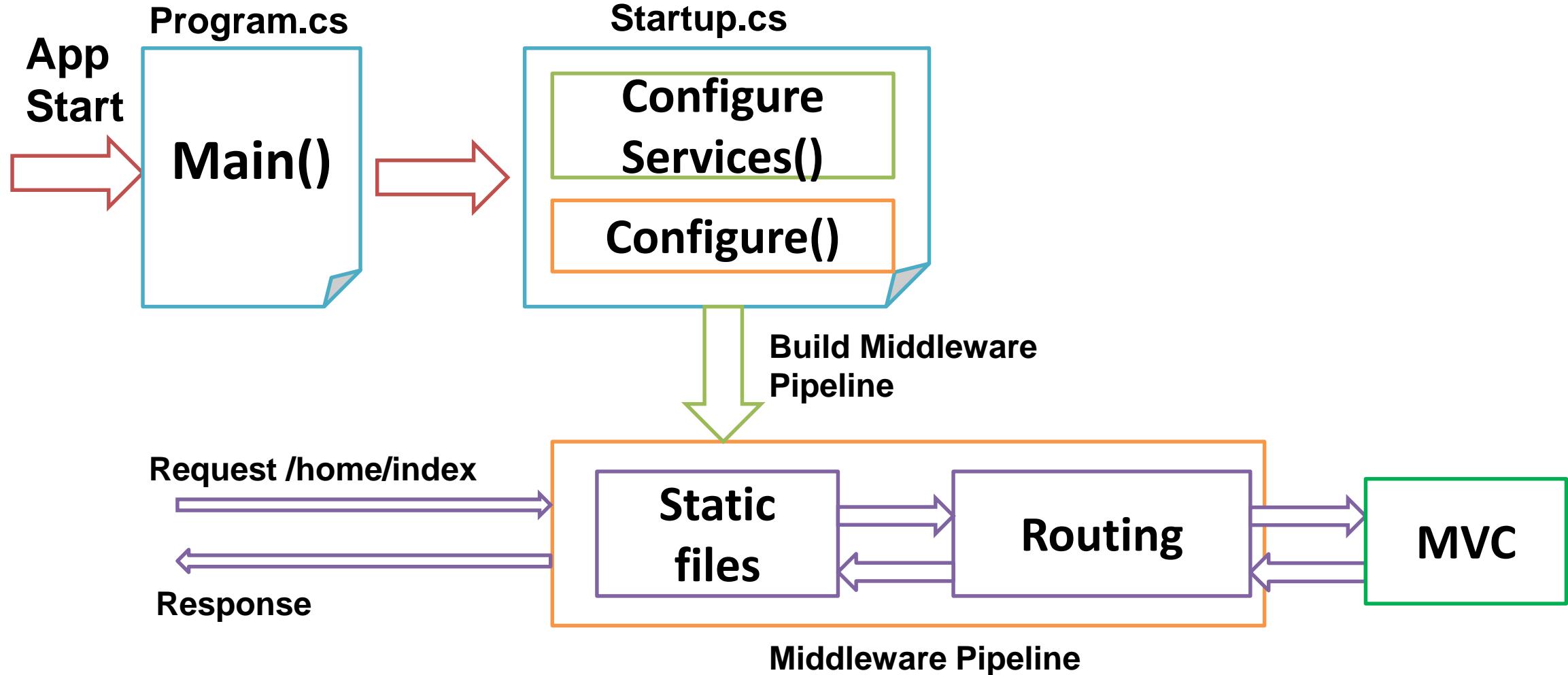
SECTION -XVIII

ASP.NET CORE 2.0 MVC

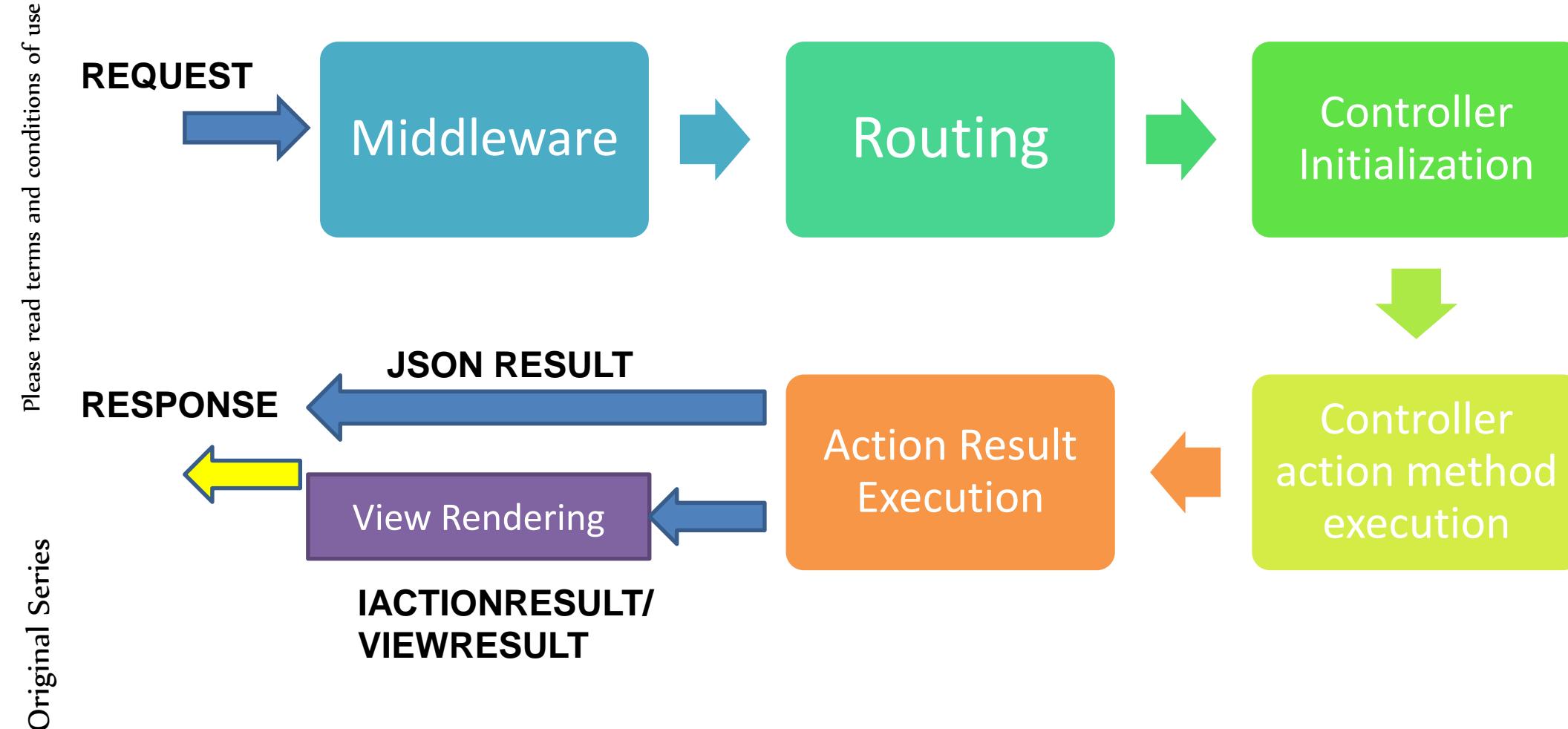
MVC Middleware Pipeline

Please read terms and conditions of use

Original Series

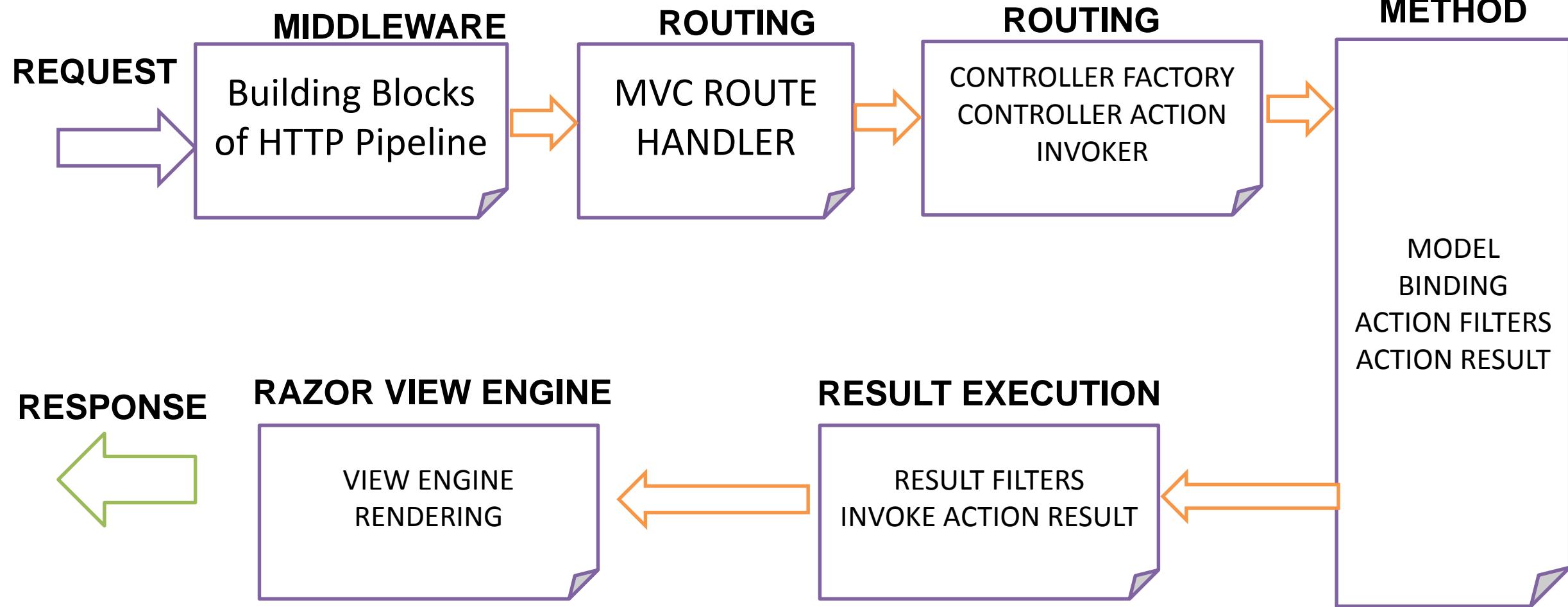


MVC Request Life Cycle



MVC Request Life Cycle (Building Blocks)

Please read terms and conditions of use



Convention Based Routing Using Mvc Middleware function

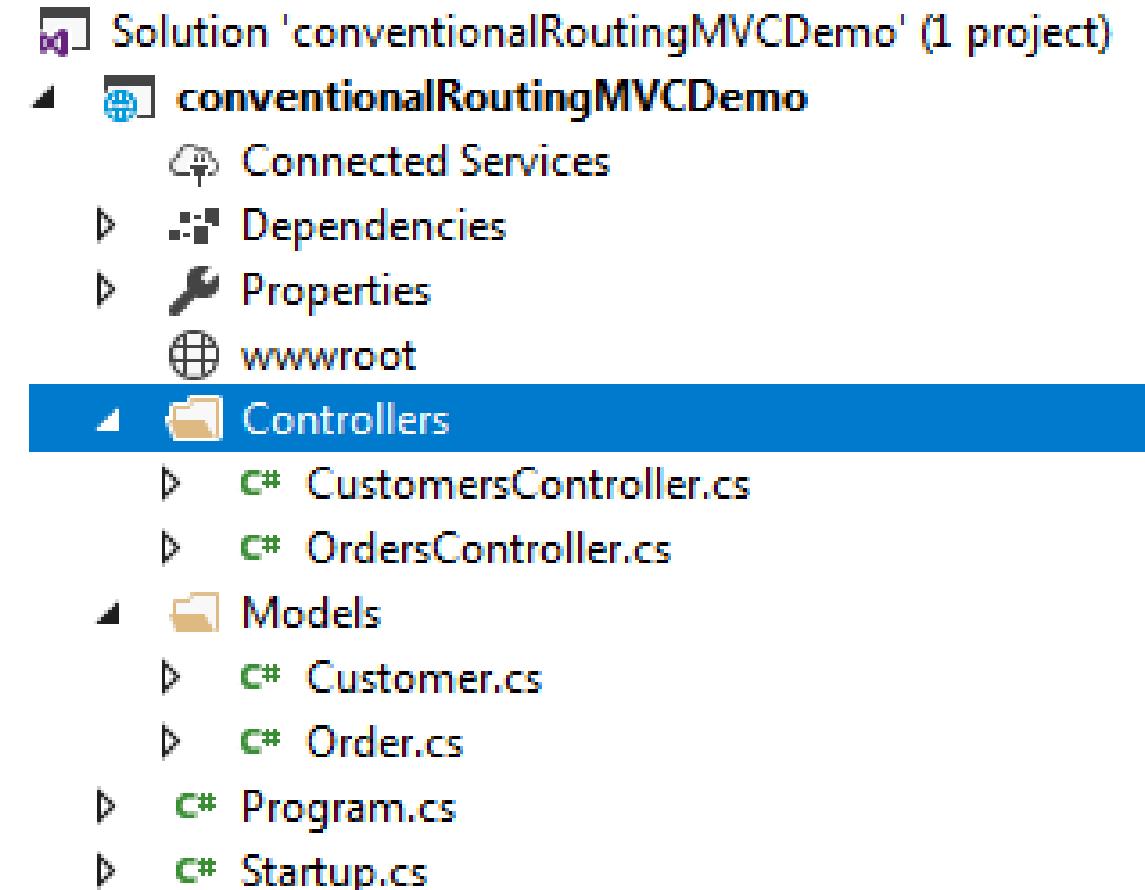
EXERCISE DEMO: 3.1

LEARNING OUTCOMES

- Create an empty asp.net core application
- Add two folders (Controllers,Models)
- Create Models and Controllers.
- Convention based routing using MVC using IRouteBuilder

Create an Empty ASP.NET Core Application.

- Create an empty ASP.NET Core application
- Add Folders “Controllers” and “Models”
- Create an empty Controller
 - CustomersController
 - OrdersController



Setting up ASP.NET MVC Middleware

Please read terms and conditions of use

Original Series

1 Add the package dependency

Microsoft.AspNetCore.All (meta package)

2 Add the MVC Service to Configuration Services(startup.cs)

3 Add the MVC Middleware

4 Define Converntion based Route

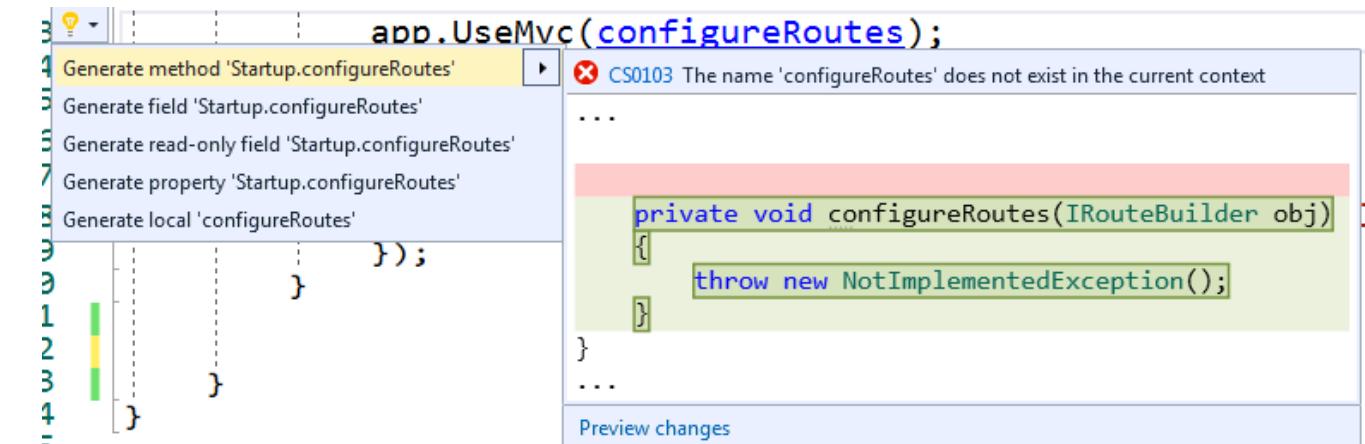
2

```
public void ConfigureServices(IServiceCollection services)
{
    //step 1: addMvc to services(Registering)
    services.AddMvc();
}
```

Add this to startup.cs => configure method

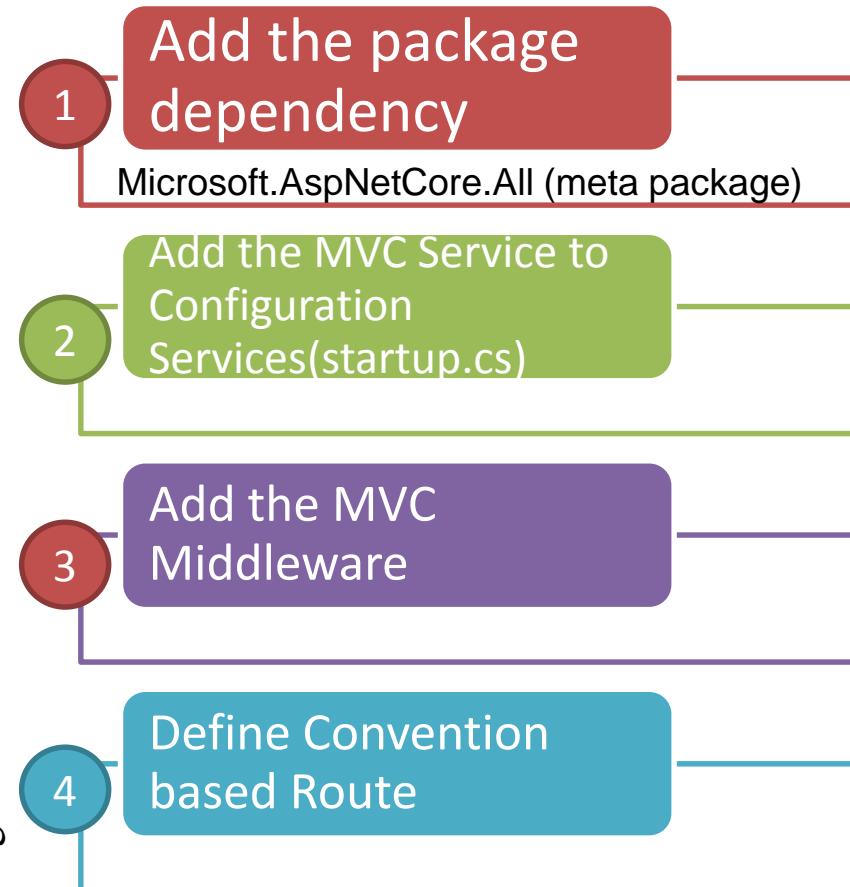
3

```
//step 2: add MVC Middleware without default Route
app.UseMvc(configureRoutes);
```



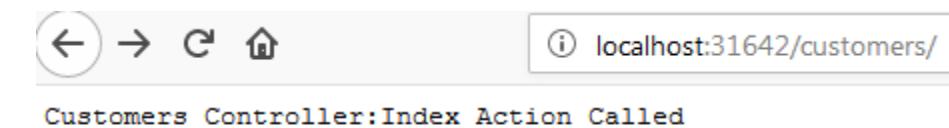
Setting up ASP.NET MVC Middleware

Please read terms and conditions of use



4

```
1 reference | 0 exceptions
private void configureRoutes(IRouteBuilder routeBuilder)
{
    //=> /Controller/action =>/Home/Index
    //=> module/Controller/action => /admin/Home/Index
    //route parameter => /Home/Index/1
    routeBuilder.MapRoute("Default", "{controller=Home}/{action=Index}/{id?}");
}
```



Attribute Based Routing Using Mvc Middleware function

EXERCISE DEMO: 3.2

LEARNING OUTCOMES

- Follow the steps outlined in exercise 3.1
- Attribute based Routing applied at controller level and controller action method level

Define Attribute Routes to Orders Controller

- Attribute Routes can be applied at the Controller Class level or Controller Action method level.
- The current approach uses the convention based route pattern and defines the controller and action accordingly.
- Used for specific action based routes.

```
namespace conventionalRoutingMVCdemo.Controllers
{
    //Alternatively we can use tokens [Route("[controller]")]
    [Route("orders")]
    public class OrdersController : Controller
    {
        [Route("index")]
        public String Index()
        {
            return "Orders Controller called:Index Action";
        }
        //Alternatively we can use tokens [Route("[action]")]
        [Route("list")]
        public String List()
        {
            return "Orders Controller called>List Action ";
        }
    }
}
```



Define Customer and Order Model

- Inside the models folder, let us create a customer and order model classes

```
namespace conventionalRoutingMVCdemo.Models
{
    public class Customer
    {
        public int CustomerId { get; set; }
        public string CustomerName { get; set; }
        public string CustomerGender { get; set; }
        public string CustomerPhone { get; set; }
        public string CustomerEmail { get; set; }
    }
}
```

```
namespace conventionalRoutingMVCdemo.Models
{
    public class Order
    {
        public int OrderId { get; set; }
        public string OrderType { get; set; }
        public DateTime OrderDate { get; set; }
        public string OrderStatus { get; set; }
    }
}
```

Define Controller Action Method

- Define Controller Action method in the OrdersController, with a return type **IActionResult**
- Lets create an instance of order, and pass on the model to new **ObjectResult** which is of type **IActionResult**.
- Run your application and navigate to the route `/{Controller}/{Action}/{id}`

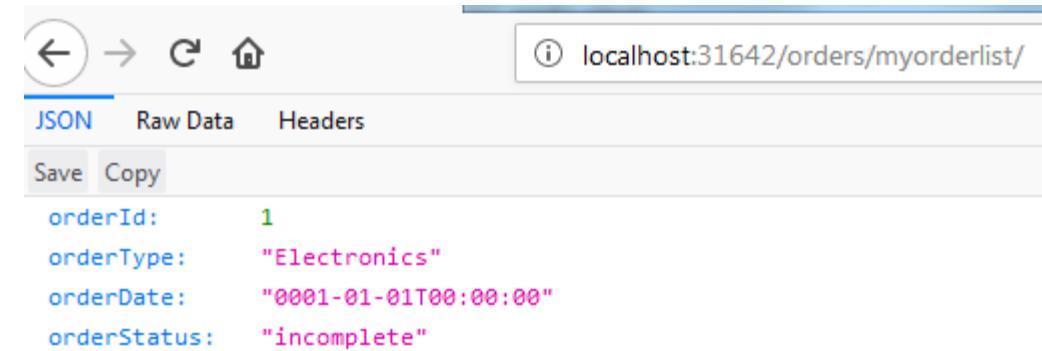
```
namespace conventionalRoutingMVCdemo.Controllers
{
    //Alternatively we can use tokens [Route("[controller]")]
    [Route("orders")]
    public class OrdersController : Controller
    {
        [Route("index")]
        public String Index()
        {
            return "Orders Controller called:Index Action";
        }
        //Alternatively we can use tokens [Route("[action]")]
        [Route("list")]
        public String List()
        {
            return "Orders Contorller called>List Action ";
        }

        [Route("myorderlist")]
        public IActionResult MyOrderList()
        {
            var myModel = new Order{OrderId= 1, OrderType="Electronics",OrderDate= new DateTime
                (),OrderStatus="incomplete" };
            return new ObjectResult(myModel);
        }
    }
}
```

Result

- Navigate to the route defined using attribute routing.
- <http://localhost/orders/myorderlist>
- We successfully binded a model to the controller and rendered it to the view.

<http://localhost/controller/action/parameterid>
<http://localhost/orders/myorderlist/>



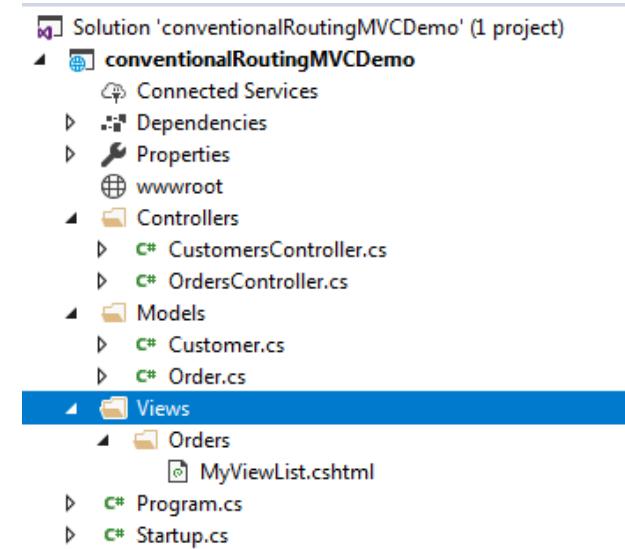
Define Controller Action Method

- Define a Controller Action Method to render a HTML View instead of JSON Object Result.

```
[Route("myviewlist")]
0 references | 0 requests | 0 exceptions
public IActionResult MyViewList()
{
    var myvm= new Order { OrderId = 2, OrderType = "Electronics", OrderDate = new DateTime(), OrderStatus =
        = "delivered" };
    return View(myvm);
}
```

Render a view

- Create a folder “views” and inside that create another folder named “orders”, now lets add MyViewList.cshtml
- Now lets use **@Model.ObjProperty** to bind to the object properties rendered from the controller to the view.



<http://localhost/orders/myviewlist>



Binding the controller to the View

OrderId:2
OrderType:Electronics
OrderDate:1/1/0001 12:00:00 AM
OrderStatus:delivered

ActionResult

- ASP.NET Core has a set of action results which are intended to facilitate the creation and formatting of response data.
- ActionResults are responsible for generating the response.
- When a view is rendered, it scaffolds based on the values and information provided in the controller and binds the properties between the controller and view.
- Categorizing Action Results.
 - Miscellaneous : they exhibit a generic behavior or stand out on their own.
 - Security : they are related to security.
 - Redirect: action results are related to different kinds of redirection.
 - Web API : these action results are likely to be used in API Controllers.
 - Files: Related to files.

IActionResult and ActionResult work as a container for other action results, in that IActionResult is an interface and ActionResult is an abstract class that other action results inherit from.

IActionResult

Please read terms and conditions of use
Original Series

IActionResult Type	
JsonResult	Intended to return JSON-Formatted data, it returns JSON regardless of what format is requested through Accept header
ViewComponentResult	
PartialViewResult	Used to load a part of page through AJAX and return raw rendered HTML.
ViewResult	Intended to render a view to response. Used to render a simple .cshtml view
ChallengeResult	
ContentResult	Default return type of a ContentResult is a “string”, but it’s not limited to string. We can return any type of response by specifying a MIME type
EmptyResult	Returns NO information. According to CQS principle, commands should not return anything.

IActionResult

Please read terms and conditions of use

IActionResult

FileResult

FileContentResult

FileStreamResult

PhysicalFileResult

VirtualFileResult

Original Series

IActionResult

Please read terms and conditions of use

Original Series	ForbidResult
	LocalRedirectResult
	ObjectResult
	CreatedAtActionResult
	CreatedAtRouteResult
	CreatedResult
	BadRequestObjectResult
	NotFoundObjectResult
	OkObjectResult
	AcceptedResult
	AcceptedAtActionResult
	AcceptedAtRouteResult

IActionResult

Please read terms and conditions of use

Original Series

IActionResult Type
RedirectResult
RedirectToActionResult
RedirectToRouteResult
SignInResult
SignOutResult
StatusCodesResult
NoContentResult
NotFoundResult
OkResult
UnauthorizedResult
UnsupportedMediaTypeResult
BadRequestResult

ActionResult using ASP.NET Core 2 MVC Application

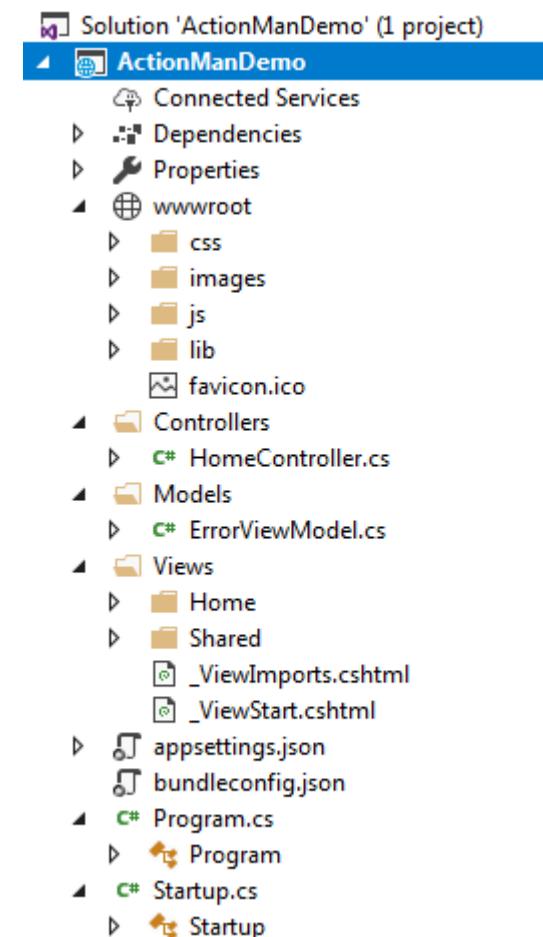
EXERCISE DEMO: 3.3

LEARNING OUTCOMES



Create an ASP.NET Core 2.0 MVC Project

- Create an ASP.NET Core 2.0 MVC Project
- Build your application.



Create Employee and WorkAction Models

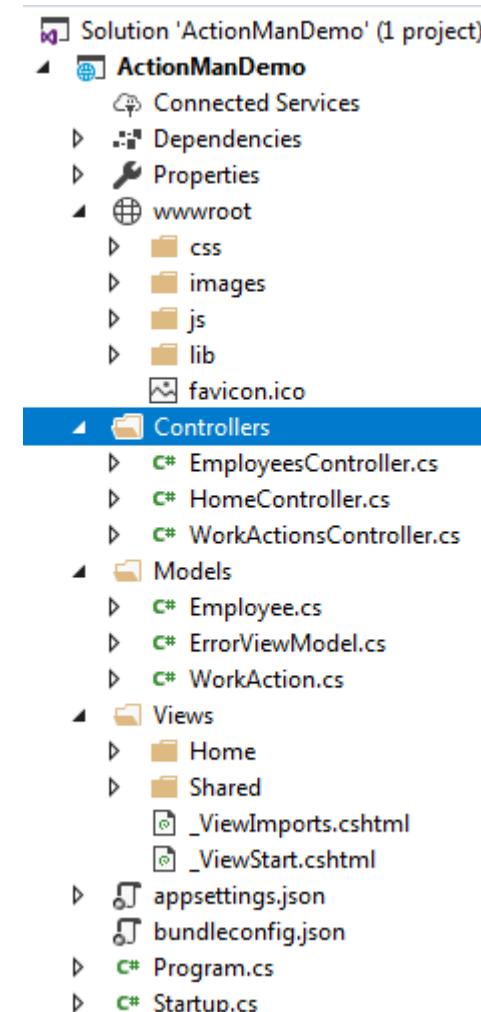
- Create Employee and WorkAction Models

```
namespace ActionManDemo.Models
{
    public class Employee
    {
        public int EmployeeId { get; set; }
        public string EmployeeName { get; set; }
        public string EmployeeRole { get; set; }
        public string EmployeeSalary { get;
            set; }
    }
}
```

```
namespace ActionManDemo.Models
{
    public class WorkAction
    {
        public int TaskId { get; set; }
        public string TaskDesc { get; set; }
        public string Priority { get; set; }
        public string Status { get; set; }
        public DateTime TaskStart { get; set; }
        public DateTime TaskEnd { get; set; }
    }
}
```

Create EmployeesController and WorkActionsController

- Create EmployeesController and WorkActionsController

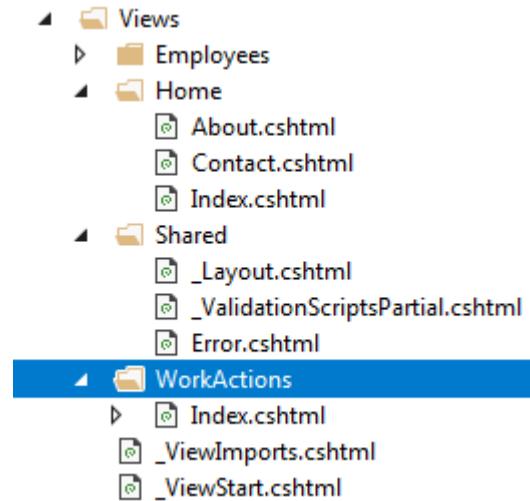


```
namespace ActionManDemo.Controllers
{
    public class EmployeesController : Controller
    {
        public IActionResult Index()
        {
            return View();
        }
    }
}
```

```
namespace ActionManDemo.Controllers
{
    public class WorkActionsController : Controller
    {
        public IActionResult Index()
        {
            return View();
        }
    }
}
```

Create Corresponding views for the Controllers

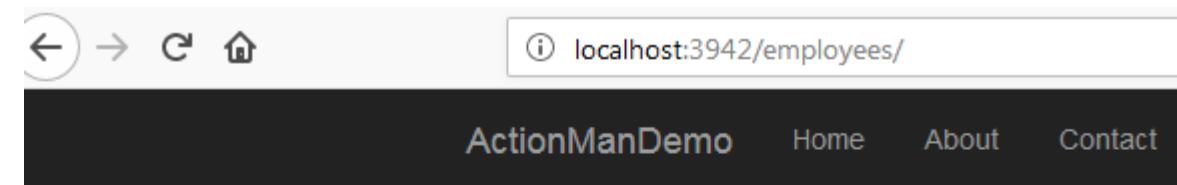
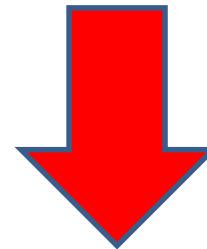
- Inside the Views folder create
 - Employees folder
 - Create empty view with a layout template
“~/Views/Shared/_Layout.cshtml”
 - WorkActions Folder
 - Create empty view with a layout template
“~/Views/Shared/_Layout.cshtml”



Controller Action Methods and Action Result

- We are using `IActionResult` which renders a View, automatically binding it to the controller action method.

```
0 references
public class EmployeesController : Controller
{
    0 references | 0 requests | 0 exceptions
    public IActionResult Index()
    {
        return View();
    }
}
```



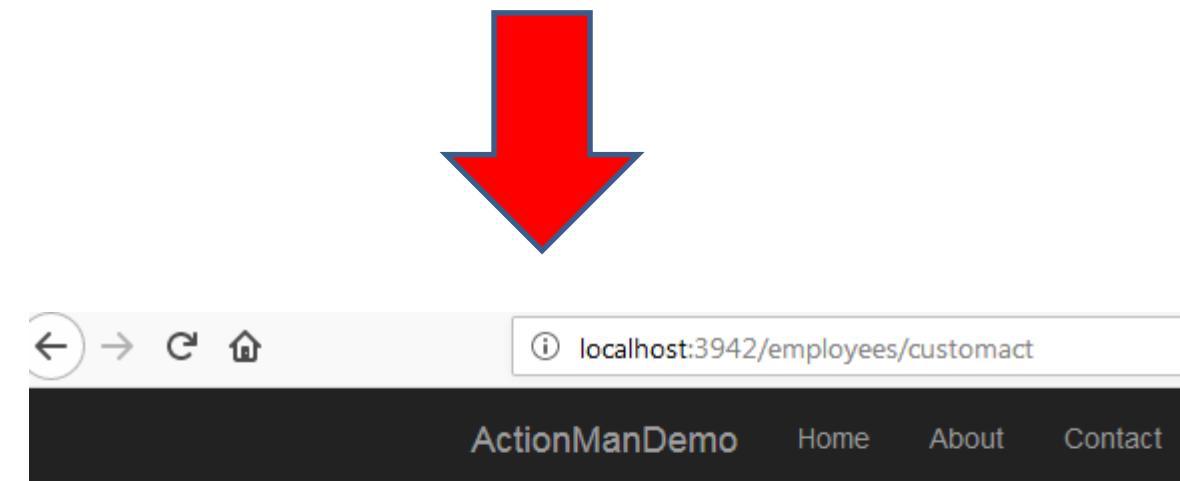
Employee Index

© 2018 - ActionManDemo

Controller Action Methods and Action Result

- Now let us return View with a value passed on to the view.
- Where “Index” refers to the Controller Action Method Name reference.

```
//passing on the Controller Action Method Name to the View.  
0 references | 0 requests | 0 exceptions  
public IActionResult CustomAct()  
{  
    ...  
    return View("CustomAct");  
}
```



CustomAct

© 2018 - ActionManDemo

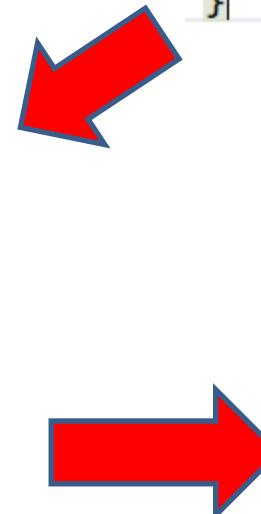
Step 7: Controller Action Methods and Action Result :JsonResult

- Sometimes, we would like to return JSON objects to the view.

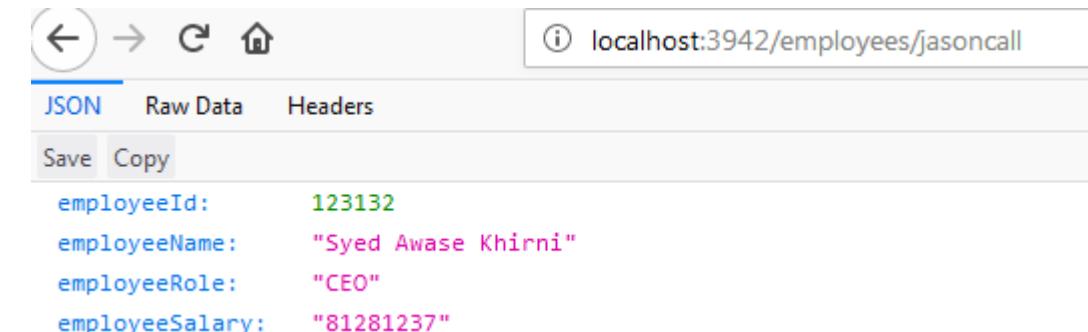
Please read terms and conditions of use

Original Series

```
@{  
    ViewData["Title"] = "JasonCall";  
    Layout = "~/Views/Shared/_Layout.cshtml";  
}  
  
<h2>JasonCall</h2>  
  
<h3>Employee Details </h3>  
<div>  
    EmployeeId: @Model.EmployeeId  
</div>  
<div>  
    EmployeeName: @Model.EmployeeName  
</div>  
<div>  
    EmployeeRole: @Model.EmployeeRole  
</div>  
<div>  
    EmployeeSalary: @Model.EmployeeSalary  
</div>
```



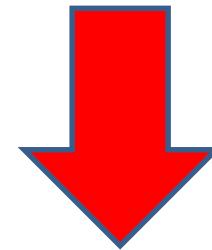
```
//render a json to the view  
0 references | 0 requests | 0 exceptions  
public JsonResult JasonCall()  
{  
    var empList = new Employee  
    {  
        EmployeeId = 123132,  
        EmployeeName = "Syed Awase Khirni",  
        EmployeeRole = "CEO",  
        EmployeeSalary = "81281237"  
    };  
    return Json(empList);
```



Step 7: Controller Action Methods and Action Result : ContentResult

- The default return type of a “ContentResult” is string, but it's not limited to String. We can return any type of response by specifying a MIME Type.

```
//returns 200 with the content and specified media type for the content
0 references | 0 requests | 0 exceptions
public ContentResult SycliqContent()
{
    return Content("[{'Application': 'SycliQPrime'}, {'Application': 'SycliQOne'}]", new MediaTypeHeaderValue("application/json"));
}
```



localhost:3942/employees/sycliqcontent

JSON Raw Data Headers

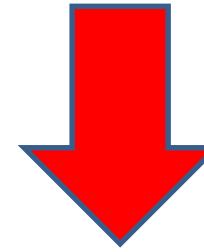
Save Copy

```
[{'Application': 'SycliQPrime'}, {'Application': 'SycliQOne'}]
```

Step 8: Controller Action Methods and Action Result : **EmptyResult**

- The default return type of a “ContentResult” is string, but it's not limited to String. We can return any type of response by specifying a MIME Type.

```
//EmptyResult Action Method  
0 references | 0 requests | 0 exceptions  
public EmptyResult EmptyCall()  
{  
    return new EmptyResult();  
}
```

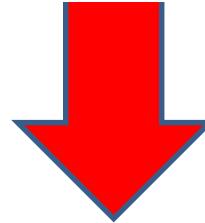


localhost:3942/employees/emptycall

Step 9: Controller Action Methods and Action Result : PocoObject

- The action result returns a POCO Object

```
0 references | 0 requests | 0 exceptions
public Employee EmployeePoco()
{
    return new Employee
    {
        EmployeeId = 123131221,
        EmployeeName = "Syed Ameese Sadath",
        EmployeeRole = "CTO",
        EmployeeSalary = "61281237"
    };
}
```



localhost:3942/employees/employeepoco

JSON Raw Data Headers

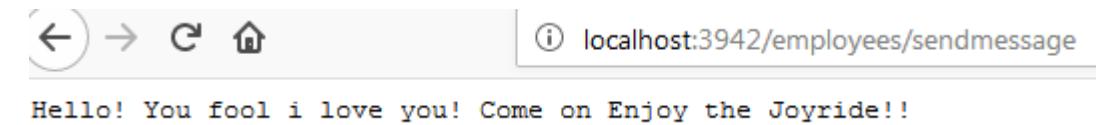
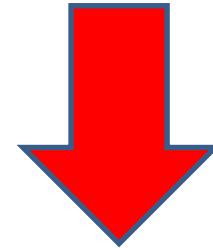
Save Copy

employeeId:	123131221
employeeName:	"Syed Ameese Sadath"
employeeRole:	"CTO"
employeeSalary:	"61281237"

Controller Action Methods and Action Result : String

- The action result returns a string data.

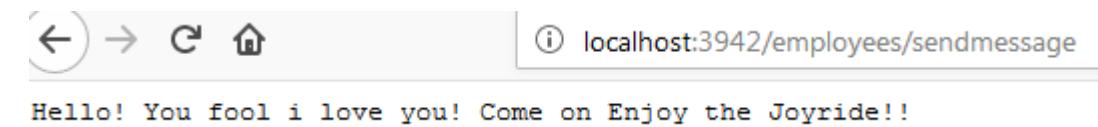
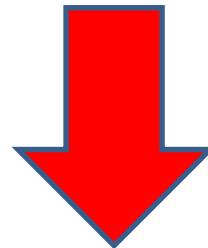
```
0 references | 0 requests | 0 exceptions
public String SendMessage()
{
    return "Hello! You fool i love you! Come on Enjoy the Joyride!!";
}
```



Controller Action Methods and Action Result : String

- The action result returns a string data.

```
0 references | 0 requests | 0 exceptions
public String SendMessage()
{
    return "Hello! You fool i love you! Come on Enjoy the Joyride!!";
}
```



Redirect ActionResult

- There are four types of action results that are related to redirect.
 - RedirectResult
 - RedirectToActionResult
 - RedirectToRouteResult
 - LocalRedirectResult
- RedirectResult: used to redirect to the provided URL, it does not matter if the URL is relative or absolute.
- RedirectToActionResult: used to redirect to an action. It takes in action name, controllername and route value.
 - RedirectToAction
 - RedirectToActionPermanent

Redirect ActionResult

- `RedirectToRouteResult`: used when we want to redirect to a route, it takes a route name, route value and redirects us to that route with the route values provided.
 - It can redirect us permanently or temporarily by setting the `Permanent` Property to true or false.
- `LocalRedirectResult`: to ensure that redirects happens in some context are local to the site.
 - To keep ourselves immune from redirect attacks.
 - Action result type takes a string for URL needed for redirect.

Step 12: Controller Action Methods and Action Result : **RedirectResult**

- Redirection actions

```
0 references | 0 requests | 0 exceptions
public RedirectResult TakemetoEmp()
{
    return Redirect("http://localhost:3942/employees/Index");
}
```

```
0 references | 0 requests | 0 exceptions
public RedirectToActionResult TakemetoAnother()
{
    return RedirectToAction("TakemetoEmp");
}
```

```
0 references | 0 requests | 0 exceptions
public RedirectResult DragmeToHell()
{
    return RedirectPermanent("http://www.territorialprescience.com");
}
```

Step 13: Controller Action Methods and Action Result : **RedirectResult**

- Redirection actions.

```
0 references | 0 requests | 0 exceptions
public RedirectToActionResult TakeMeToStart()
{
    return RedirectToAction("Index");
}
```

```
0 references | 0 requests | 0 exceptions
public RedirectToRouteResult NewRouting()
{
    var routeValue = new RouteValueDictionary(new { action = "Index", controller = "Home", area = "" });

    return RedirectToRoutePermanent(routeValue);
}
```

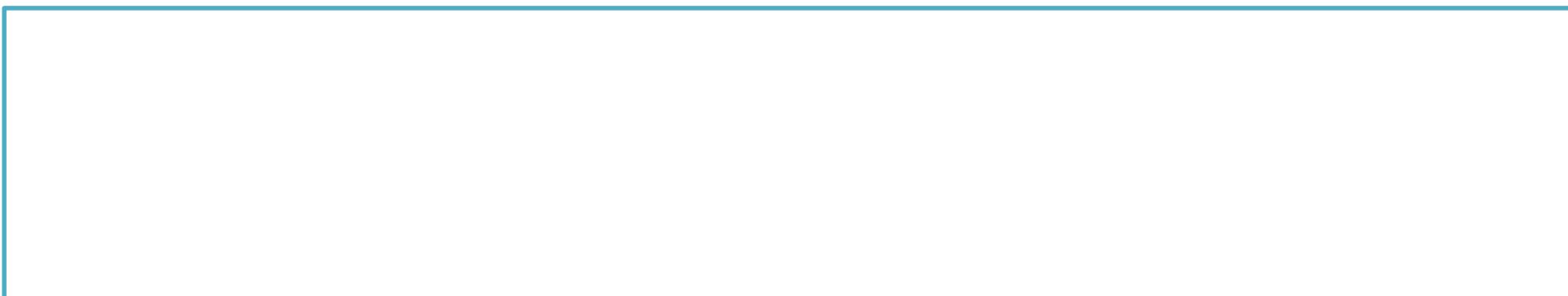
```
0 references | 0 requests | 0 exceptions
public LocalRedirectResult Localized()
{
    var myLocalUrlPath = Url.IsLocalUrl("/Home/Index");

    return LocalRedirect("/Home/Index");
    //return LocalRedirectPermanent("/Home/Index");
}
```

ASP.NET Core 2.0 MVC Application with InMemoryData:HealthWally

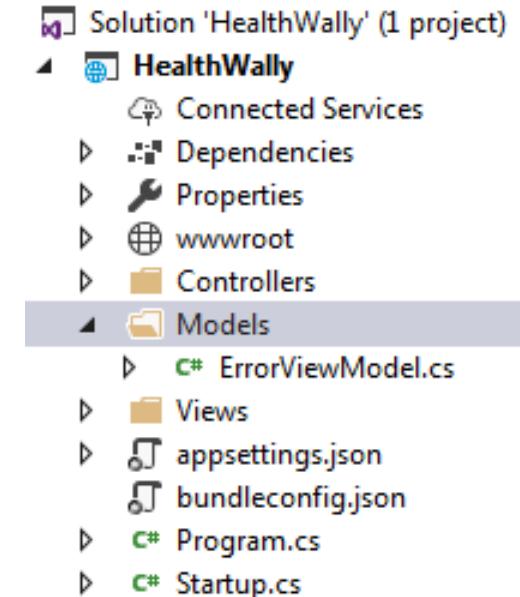
EXERCISE DEMO: 3.4

LEARNING OUTCOMES



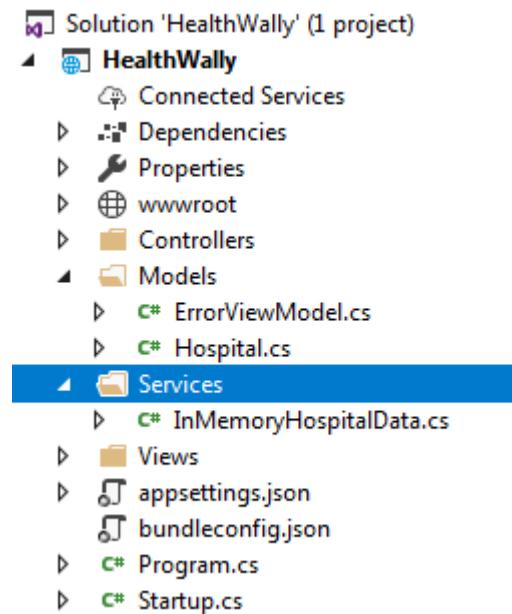
Create an ASP.NET Core 2.0 MVC Project

- Create an ASP.NET Core 2.0 MVC Project
- Build your application.



Create Models for Hospital

- Create a model for Hospital
- Create a folder for services.



```
namespace HealthWally.Models
{
    public class Hospital
    {
        public int HospitalId { get; set; }
        public string HospitalName { get; set; }
        public string HospitalAddress { get; set; }
        public string HospitalType { get; set; }
        public int HospitalRank { get; set; }
    }
}
```

Step 3: Create an

InmemoryDataService for rendering hospital data

- Lets first create an interface IHospitalData that would have abstract methods to be implemented by our service.
- Subsequently, lets create an InmemoryDataService that implements IHospitalData.



The screenshot shows two code files in a code editor:

IHospitalData.cs

```
namespace HealthWally.Services
{
    public interface IHospitalData
    {
        //fetching all the hospitals
        I Enumerable<Hospital> GetAll();
    }
}
```

InMemoryHospitalData.cs

```
namespace HealthWally.Services
{
    public class InMemoryHospitalData : IHospitalData
    {
        List<Hospital> _hospitals;
        public InMemoryHospitalData()
        {
            //lists are not threadsafe
            _hospitals = new List<Hospital>
            {
                new Hospital{HospitalId=1231131,HospitalName="Manipal Hospitals", HospitalAddress="IndiraNagar Bangalore",HospitalType="SuperSpeciality",HospitalRank=7},
                new Hospital{HospitalId=1231132,HospitalName="Sakra World", HospitalAddress="Marthahalli Bangalore",HospitalType="SuperSpeciality",HospitalRank=8},
                new Hospital{HospitalId=1231133,HospitalName="Mallya Hospitals", HospitalAddress="Cubbon Park Bangalore",HospitalType="SuperSpeciality",HospitalRank=8},
                new Hospital{HospitalId=1231134,HospitalName="Specialist Hospitals", HospitalAddress="Kalyan Nagar Bangalore",HospitalType="SuperSpeciality",HospitalRank=6}
            };
        }

        public I Enumerable<Hospital> GetAll()
        {
            return _hospitals.OrderBy(h => h.HospitalId);
        }
    }
}
```

Step 4: Register the InMemoryHospitalData Service with Startup.cs

- AddScoped is a way of telling ASP.NET Core, to create a new instance for every request and perform garbage collection at the end of the request.

Add this to startup.cs => ConfigureServices method

```
// 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)  
{  
    services.AddScoped<IHospitalData, InMemoryHospitalData>();  
    services.AddMvc();  
}
```

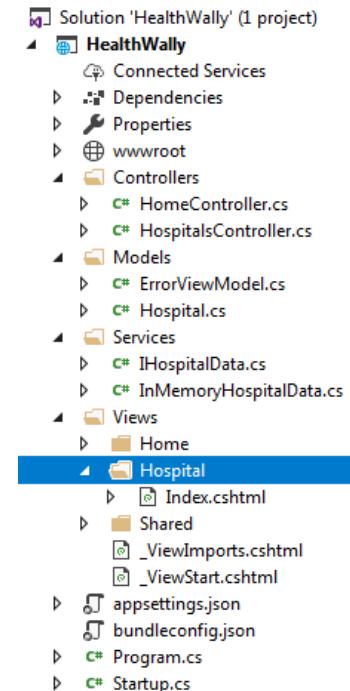
Create HospitalsController and invoke the service

- Create a HospitalsController, inject IHospitalData Service and retrieve a collection of data.

```
namespace HealthWally.Controllers
{
    public class HospitalsController : Controller
    {
        private IHospitalData _hospitalData;
        public HospitalsController(IHospitalData hospitalData)
        {
            _hospitalData = hospitalData;
        }
        public IActionResult Index()
        {
            var hmodel = _hospitalData.GetAll();
            return View(hmodel);
        }
    }
}
```

Step 6: Create a corresponding view to match the Controller Action Method.

- Create Hospital Folder inside Views folder
- Now generate a view, by selecting the Model Class and applying the standard template layout.
- Visual Studio scaffolds a razor view to render a list of hospitals.



```
@model IEnumerable<HealthWally.Models.Hospital>
@{
    ViewData["Title"] = "Index";
    Layout = "~/Views/Shared/_Layout.cshtml";
}
<h2>Index</h2>
<p>
    <a asp-action="Create">Create New</a>
</p>
<table class="table">
    <thead>
        <tr>
            <th>HospitalId</th>
            <th>HospitalName</th>
            <th>HospitalAddress</th>
            <th>HospitalType</th>
            <th>HospitalRanking</th>
            <th></th>
        </tr>
    </thead>
    <tbody>
@foreach (var item in Model) {
        <tr>
            <td>@item.HospitalId</td>
            <td>@item.HospitalName</td>
            <td>@item.HospitalAddress</td>
            <td>@item.HospitalType</td>
            ...
        </tr>
    }
    ...
}

```

Build and run your application

- Navigate to the route using the conventional route pattern
- <http://localhost/hospitals/index>

HealthWally Home About Contact

Index

Create New

HospitalId	HospitalName	HospitalAddress	HospitalType	HospitalRanking	
1231131	Manipal Hospitals	IndiraNagar Bangalore	SuperSpeciality	7	Edit Details Delete
1231132	Sakra World	Marthahalli Bangalore	SuperSpeciality	8	Edit Details Delete
1231133	Mallya Hospitals	Cubbon Park Bangalore	SuperSpeciality	8	Edit Details Delete
1231134	Specialist Hospitals	Kalyan Nagar Bangalore	SuperSpeciality	6	Edit Details Delete

© 2018 - HealthWally

Models and ViewModels

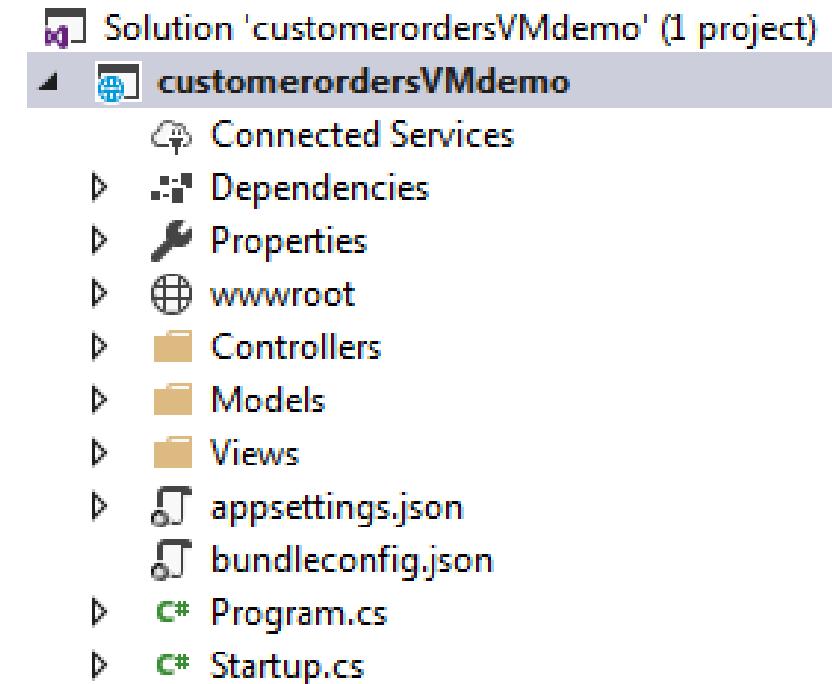
EXERCISE DEMO: 3.5

LEARNING OUTCOMES

- Models and ViewModels demo with ASP.NET Core 2 MVC

Create an ASP.NET Core 2.0 MVC Project

- Create an ASP.NET Core 2.0 MVC Project
- Build your application.

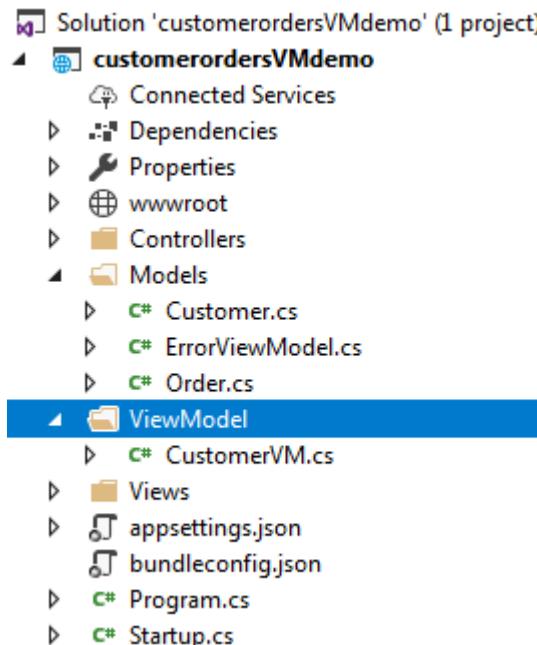


Create models

- Create customer and order models

```
namespace customerordersVMdemo.Models  
{  
    public class Customer  
    {  
        public int CustomerId { get; set; }  
        public string CustomerName { get; set; }  
        public string CustomerAddress { get; set; }  
        public string CustomerPhone { get; set; }  
        public Nullable<System.DateTime> CustomerDob  
        { get; set; }  
        public string CustomerEmail { get; set; }  
    }  
}
```

```
namespace customerordersVMdemo.Models  
{  
    public class Order  
    {  
        public int OrderId { get; set; }  
        public string OrderStatus { get; set; }  
        public Nullable<decimal> OrderPrice { get;  
            set; }  
        public DateTime OrderDate { get; set; }  
    }  
}
```



Create a.viewmodel

- Create a folder.viewmodel .

```
namespace customerordersVMdemo.ViewModel
{
    0 references
    public class CustomerVM
    {
        0 references | 0 exceptions
        public IEnumerable<Customer> Customer { get;
            set; }
        0 references | 0 exceptions
        public IEnumerable<Order> Order { get; set; }
        0 references | 0 exceptions
        public string DeliveredTo { get; set; }
    }
}
```

Create a Service for Customers and Orders

- Create services folder and create services for both customers and orders

```
namespace customerordersVMDemo.Services
{
    public class InMemoryCustomerData : ICustomerData
    {
        List<Customer> _customers;
        public InMemoryCustomerData()
        {
            _customers = new List<Customer>
            {
                new Customer{CustomerId=1231311, CustomerName="Syed Awase", CustomerAddress="Bangalore", CustomerDob=new
                    DateTime(), CustomerEmail="awasekhirni@gmail.com", CustomerPhone="9035433124"},
                new Customer{CustomerId=1231312, CustomerName="Syed Ameese", CustomerAddress="Hyderabad", CustomerDob=new
                    DateTime(), CustomerEmail="s@gmail.com", CustomerPhone="9035433111"},
                new Customer{CustomerId=1231313, CustomerName="Syed Azeez", CustomerAddress="Bangalore", CustomerDob=new
                    DateTime(), CustomerEmail="ssa@gmail.com", CustomerPhone="9035433222"},
                new Customer{CustomerId=1231314, CustomerName="Syed Rayyan", CustomerAddress="Bangalore", CustomerDob=new
                    DateTime(), CustomerEmail="sra@gmail.com", CustomerPhone="9035433333"}
            };
        }
        public IEnumerable<Customer> GetAllCustomers()
        {
            return _customers.OrderBy(c => c.CustomerId);
        }
    }
}
```

```
namespace customerordersVMDemo.Services
{
    public interface ICustomerData
    {
        IEnumerable<Customer> GetAllCustomers();
    }
}
```

```
namespace customerordersVMDemo.Services
{
    public interface IOrderData
    {
        IEnumerable<Order> GetAllOrders();
    }
}
```

```
namespace customerordersVMDemo.Services
{
    public class InMemoryOrderData : IOrderData
    {
        List<Order> _orders;
        public IEnumerable<Order> GetAllOrders()
        {
            return _orders.OrderBy(o => o.OrderId);
        }
        public InMemoryOrderData()
        {
            _orders = new List<Order>
            {
                new Order{OrderId=712361, OrderDate=new DateTime(), OrderPrice=781, OrderStatus="toShip"},
                new Order{OrderId=712362, OrderDate=new DateTime(), OrderPrice=72, OrderStatus="inTransit"},
                new Order{OrderId=712363, OrderDate=new DateTime(), OrderPrice=81, OrderStatus="delivered"},
                new Order{OrderId=712364, OrderDate=new DateTime(), OrderPrice=27, OrderStatus="inTransit"},
                new Order{OrderId=712365, OrderDate=new DateTime(), OrderPrice=78, OrderStatus="atWarehouse"}
            };
        }
    }
}
```

Register the service with startup.cs

- AddScoped is a way of telling ASP.NET Core, to create a new instance for every request and perform garbage collection at the end of the request.

```
// 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)  
{  
    services.AddScoped<ICustomerData, InMemoryCustomerData>();  
    services.AddScoped<IOrderData, InMemoryOrderData>();  
    services.AddMvc();  
}
```

Create CustomerController and invoke the Services.

- Create CustomerController and OrderController

```
namespace customerordersVMdemo.Controllers
{
    1 reference
    public class CustomersController : Controller
    {
        private ICustomerData _customerData;
        0 references | 0 exceptions
        public CustomersController(ICustomerData customerData)
        {
            _customerData = customerData;
        }

        0 references | 0 requests | 0 exceptions
        public IActionResult Index()
        {
            var cmodel = _customerData.GetAllCustomers();
            return View(cmodel);
        }
    }
}

namespace customerordersVMdemo.Controllers
{
    1 reference
    public class OrdersController : Controller
    {
        private IOrderData _orderData;
        0 references | 0 exceptions
        public OrdersController(IOrderData orderData)
        {
            _orderData = orderData;
        }

        0 references | 0 requests | 0 exceptions
        public IActionResult Index()
        {
            var cmodel = _orderData.GetAllOrders();
            return View(cmodel);
        }
    }
}
```

Step 7: Create CustomController by injecting the services required for CustomerVM

- Here we inject `ICustomerData` and `IOrderData` as we would like to render a custom view.
- Next we would like to render a custom view mapping to the `CustomerViewModel` defined earlier `CustomerVM`.

```
namespace customerordersVMDemo.Controllers
{
    public class CustomController : Controller
    {
        private ICustomerData _customerData;
        private IOrderData _orderData;
        public CustomController(ICustomerData customerData, IOrderData orderData)
        {
            _customerData = customerData;
            _orderData = orderData;
        }
        public IActionResult Index()
        {
            var vm = new CustomerVM();
            vm.Customers = _customerData.GetAllCustomers();
            vm.Orders = _orderData.GetAllOrders();
            vm.DeliveredTo = "Governess @ Home name: Rubina";
            return View(vm);
        }
    }
}
```

Create a view mapping to CustomerVM.

- Lets generate a view with CustomerVM as model class and using the layout page view

“~/Views/Shared/_Layout.cshtml”

```
@model customerordersVMdemo.ViewModel.CustomerVM  
  
{@  
    ViewData["Title"] = "Index";  
    Layout = "~/Views/Shared/_Layout.cshtml";  
}  
  
<h2>Index</h2>  
  
@foreach (var customer in Model.Customers)  
{  
    <div>@customer.CustomerId</div>  
    <div>@customer.CustomerName</div>  
    <div>@customer.CustomerAddress</div>  
}  
  
@foreach (var order in Model.Orders)  
{  
    <div>@order.OrderId</div>  
    <div>@order.OrderStatus</div>  
    <div>@order.OrderPrice</div>  
    <div>@order.OrderDate</div>  
}  
  
<div>@Model.DeliveredTo</div>
```

Resulting View

- Now lets run and look at the custom view generated based on the.viewmodel

customerordersVMdemo		Home	About
Syed Awase			
Bangalore			
1231312			
Syed Ameese			
Hyderabad			
1231313			
Syed Azeez			
Bangalore			
1231314			
Syed Rayyan			
Bangalore			
712361			
toShip			
781			
1/1/0001 12:00:00 AM			
712362			
inTransit			
72			
1/1/0001 12:00:00 AM			
712363			
delivered			
81			
1/1/0001 12:00:00 AM			
712364			
inTransit			
27			
1/1/0001 12:00:00 AM			
712365			
atWarehouse			
78			
1/1/0001 12:00:00 AM			
Governess @ Home name: Rubina			

Creating MVC Application with EF Core 2.0 and SQLServer

EXERCISE DEMO: 3.5

LEARNING OUTCOMES

- Installing EntityFramework Core 2.0 CLI
- Creating DbContext File, DbContext Methods and DbContext properties.
- Binding the DbContext with a service
- Exploring dotnet ef cli commands and dbcontext
- Binding the controller to the service to connect to the SQLServer database

Model Validation with Data Annotations

Please read terms and conditions of use

Original Series

- Using `System.ComponentModel.DataAnnotations` namespace
- We can add attributes to the model properties.
- A specific model property can have multiple annotations.
`[Required,MaxLength(80)]`
- `[ValidateAntiForgeryToken]`

Name	Purpose
MinLength/MaxLength	Enforce length of strings
Range	Enforce min and max numbers
RegularExpression	Make a string match a pattern
Display	Set the name and formatting string
DataType	Render as password or email input
Required	Model value is mandatory
Compare	Commonly used for password validation.

Entity Framework core

Please read terms and conditions of use

Original Series

- Open-source, lightweight, extensible and cross-platform version of entity framework data access technology.
- It is an Object/Relational Mapping ORM framework. It is an enhancement to ADO.NET that gives developers an automated mechanism for accessing and storing the data in the database.
- EF Core is intended to be used with .NET Core applications.

ORM

LINQ SUPPORT

LIGHTWEIGHT
& CROSS
PLATFORM

OPENSOURCE

SQL Server and
non-relational
DB support

Code-First Only

EF Core 2.0

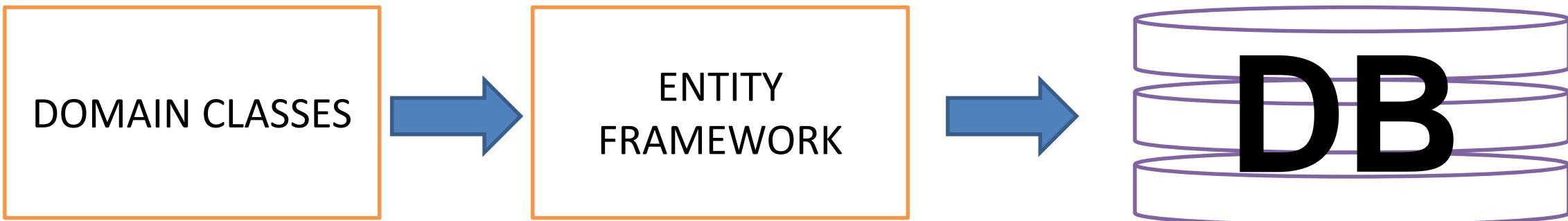
Please read terms and conditions of use
Original Series

Application Types	ASP.NET Core Web,API,Console	.NET 4.5+ Console, WinForm,WPF, ASP.NET	Devices +IOT,Mobile,PC,Xb ox,Surface Hub	Mobile Application, Android,iOS,Windows
EF Core	EF Core	EF Core	EF Core	EF core
Framework	.NET Core	.NET 4.5+	UWP	Xamarin
Operating System	Windows,Mac ,Linux	Windows	Windows 10	Mobile

Approaches

CODE FIRST APPROACH

Please read terms and conditions of use



Original Series

DbContext Methods

Please read terms and conditions of use

Original Series

Method	Usage
Add	Adds a new entity to DbContext with Added state and starts tracking it. This new entity data will be inserted into the database when SaveChanges() is called.
AddAsync	Asynchronous method for adding a new entity to DbContext with Added state and starts tracking it. This new entity data will be inserted into the database when SaveChangesAsync() is called.
AddRange	Adds a collection of new entities to DbContext with Added state and starts tracking it. This new entity data will be inserted into the database when SaveChanges() is called.
AddRangeAsync	Asynchronous method for adding a collection of new entities which will be saved on SaveChangesAsync().
Attach	Attaches a new or existing entity to DbContext with Unchanged state and starts tracking it.
AttachRange	Attaches a collection of new or existing entities to DbContext with Unchanged state and starts tracking it.
Entry	Gets an EntityEntry for the given entity. The entry provides access to change tracking information and operations for the entity.

DbContext Methods

Please read terms and conditions of use

Original Series

Method	Usage
Find	Finds an entity with the given primary key values.
FindAsync	Asynchronous method for finding an entity with the given primary key values.
Remove	Sets Deleted state to the specified entity which will delete the data when SaveChanges() is called.
RemoveRange	Sets Deleted state to a collection of entities which will delete the data in a single DB round trip when SaveChanges() is called.
SaveChanges	Execute INSERT, UPDATE or DELETE command to the database for the entities with Added, Modified or Deleted state.
SaveChangesAsync	Asynchronous method of SaveChanges()
Set	Creates a DbSet< TEntity > that can be used to query and save instances of TEntity.
Update	Attaches disconnected entity with Modified state and start tracking it. The data will be saved when SaveChagnes() is called.

DbContext Methods

Please read terms and conditions of use

Original Series

Method	Usage
UpdateRange	Attaches a collection of disconnected entities with Modified state and start tracking it. The data will be saved when SaveChanges() is called.
OnConfiguring	Override this method to configure the database (and other options) to be used for this context. This method is called for each instance of the context that is created.
OnModelCreating	Override this method to further configure the model that was discovered by convention from the entity types exposed in DbSet< TEntity > properties on your derived context.

DbContext Properties

Please read terms and conditions of use

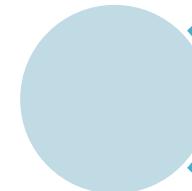
Original Series

Method	Usage
ChangeTracker	Provides access to information and operations for entity instances this context is tracking.
Database	Provides access to database related information and operations for this context.
Model	Returns the metadata about the shape of entities, the relationships between them, and how they map to the database.

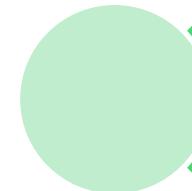
Steps to add EF Core the Application

Please read terms and conditions of use

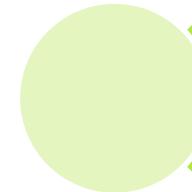
Original Series



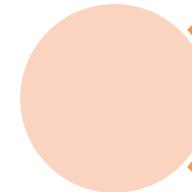
Add domain classes



Define database context



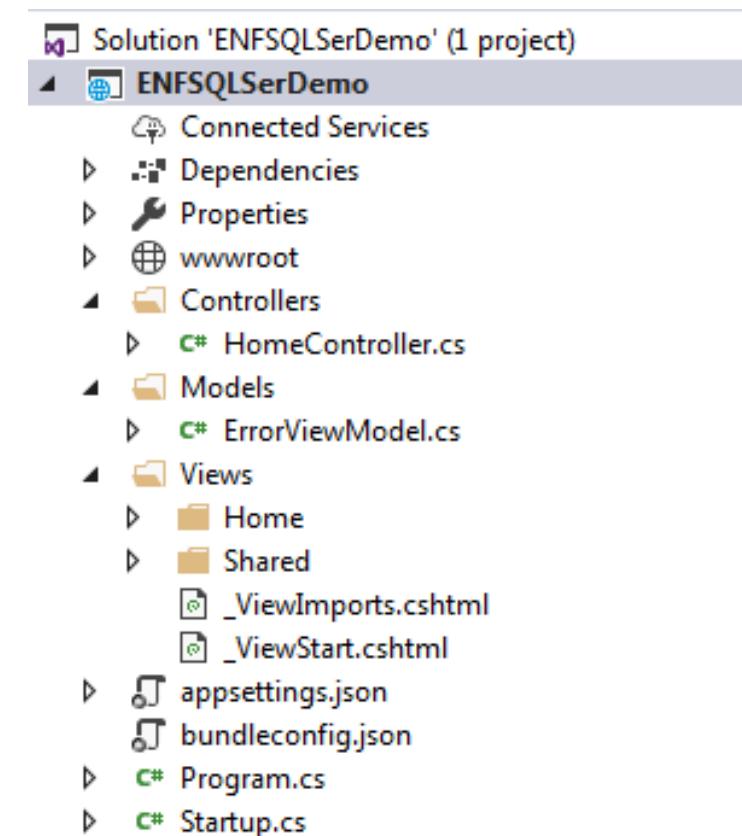
Define the connection string in appsettings.json



Application Configuration in Startup.cs

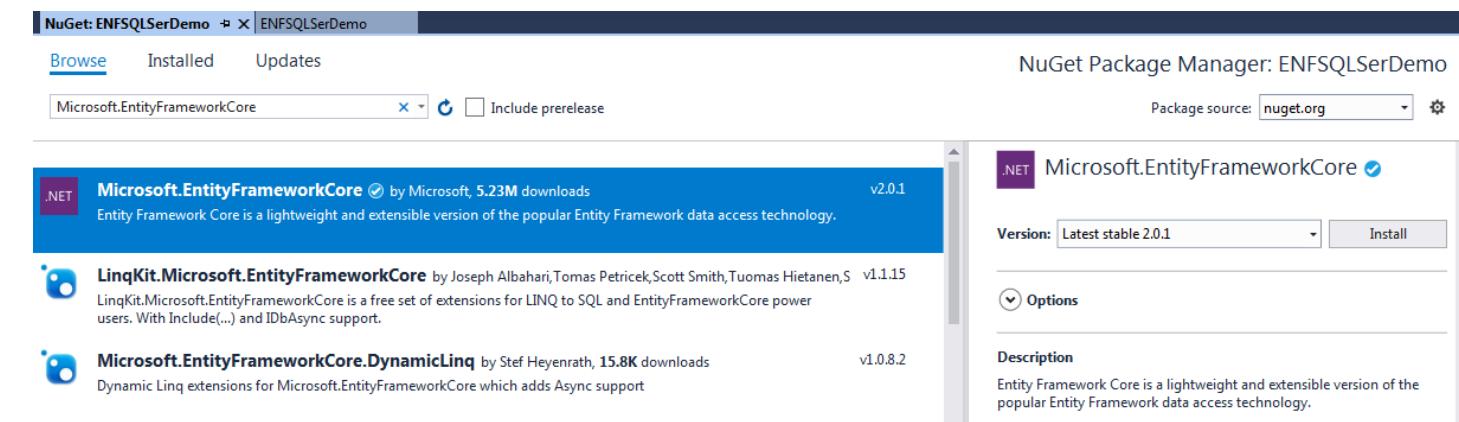
Create ASP.NET Core 2.0 MVC Application

1. Create an ASP.NET Core 2.0 MVC Application
2. Visual Studio would scaffold your application and render it as shown here.



Installing Entity Framework Core 2.0

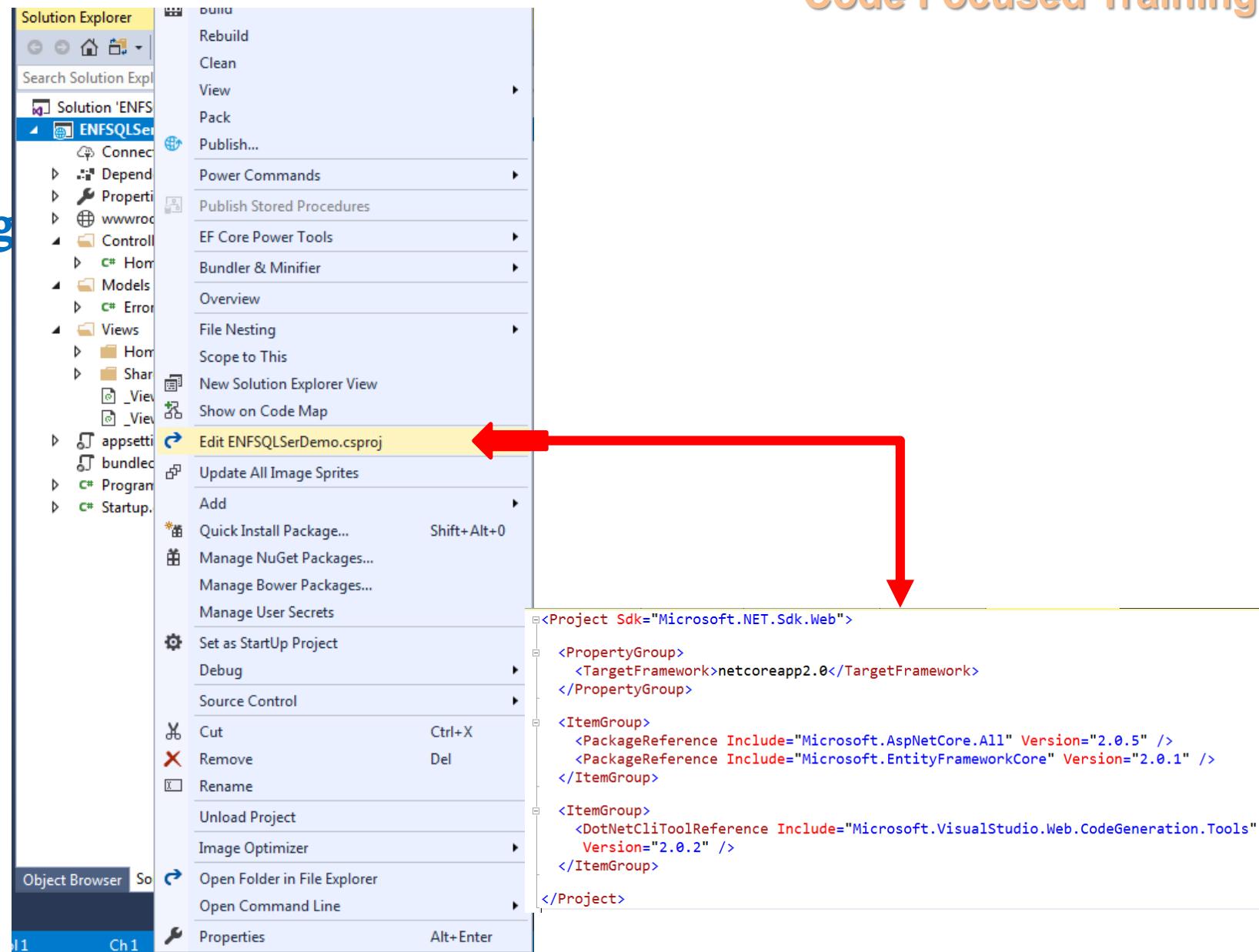
1. Using NuGet package manager, we can install Microsoft.EntityFrameworkCore Core 2.0
2. Alternatively, we can install using command line.



```
PM> dotnet add ENFSQLSerDemo package Microsoft.EntityFrameworkCore
Writing C:\Users\syedawase\AppData\Local\Temp\tmpC0CA.tmp
info : Adding PackageReference for package 'Microsoft.EntityFrameworkCore' into project 'H:\awase-harddisk\CT\C#2017\DOTNETCORE\dotnetcore-appscenarios\apptwentytwo\ENFSQLSerDemo\ENFSQLSerDemo.csproj'.
log  : Restoring packages for H:\awase-harddisk\CT\C#2017\DOTNETCORE\dotnetcore-appscenarios\apptwentytwo\ENFSQLSerDemo\ENFSQLSerDemo.csproj
info :  GET https://api.nuget.org/v3-flatcontainer/microsoft.entityframeworkcore/index.json
info :  OK https://api.nuget.org/v3-flatcontainer/microsoft.entityframeworkcore/index.json 1052ms
info : Package 'Microsoft.EntityFrameworkCore' is compatible with all the specified frameworks in project 'H:\awase-harddisk\CT\C#2017\DOTNETCORE\dotnetcore-appscenarios\apptwentytwo\ENFSQLSerDemo\ENFSQLSerDemo.csproj'.
info : PackageReference for package 'Microsoft.EntityFrameworkCore' version '2.0.1' added to file 'H:\awase-harddisk\CT\C#2017\DOTNETCORE\dotnetcore-appscenarios\apptwentytwo\ENFSQLSerDemo\ENFSQLSerDemo.csproj'.
```

Editing Project.csproj and adding DOTNETCLI package

1. Right click on the project name and select Edit ProjectName.



Add a DotNetCliToolReference

- Now let us add
DotNetCliToolReference for
Microsoft.EntityFrameworkCore
Tools.DotNet

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

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

  <ItemGroup>
    <PackageReference Include="Microsoft.AspNetCore.All" Version="2.0.5" />
    <PackageReference Include="Microsoft.EntityFrameworkCore" Version="2.0.1" />
  </ItemGroup>

  <ItemGroup>
    <DotNetCliToolReference Include="Microsoft.VisualStudio.Web.CodeGeneration.Tools"
      Version="2.0.2" />
    <DotNetCliToolReference Include="Microsoft.EntityFrameworkCore.Tools.DotNet"
      Version="2.0.1" />
  </ItemGroup>

</Project>
```



Check for command line

DOTNET EF

1. Navigate to the project location
2. Run **dotnet ef**

```
PS H:\awase-harddisk\CT\C#\2017\DOTNETCORE\dotnetcore-appscenarios\apptwentytwo\ENFSQLSerDemo\ENFSQLSerDemo> dotnet ef

[Entity Framework logo]

Entity Framework Core .NET Command Line Tools 2.0.1-rtm-125

Usage: dotnet ef [options] [command]

Options:
  --version      Show version information
  -h|--help      Show help information
  -v|--verbose   Show verbose output.
  --no-color     Don't colorize output.
  --prefix-output Prefix output with level.

Commands:
  database    Commands to manage the database.
  dbcontext   Commands to manage DbContext types.
  migrations  Commands to manage migrations.

Use "dotnet ef [command] --help" for more information about a command.
```

Create PolicyAction Model

1. Create a model by name
PolicyAction model

```
namespace ENFSQLSerDemo.Models
{
    public class PolicyAction
    {
        public int PolicyActionId { get; set; }
        [Required, MaxLength(250)]
        public string PolicyActionTitle { get; set; }
        [Required]
        public string PolicyActionProposor { get; set; }
        [Required]
        public string ImpactLevel { get; set; }
        [Required]
        public string PoliticalRepAssignee { get; set; }
        [DisplayFormat(DataFormatString = "{0:MM/dd/yyyy}", ApplyFormatInEditMode = true)]
        [Range(typeof(DateTime), "01/01/2018", "12/12/2018", ErrorMessage = "Check Date")]
        public DateTime Deadline { get; set; }
        public int UpVote { get; set; }
        public string AmendmentNote { get; set; }
    }
}
```

Create ApplicationDbContext

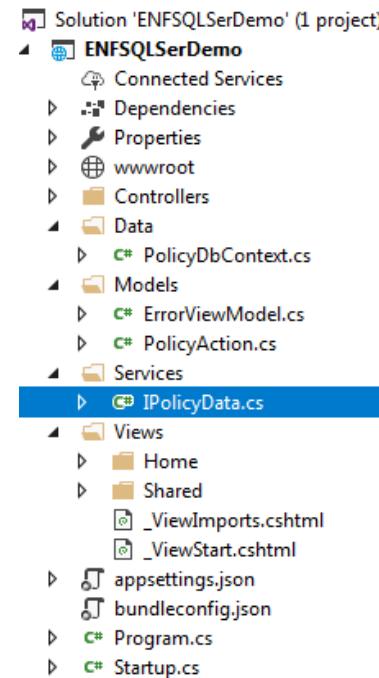
1. Create a folder “Data” and add “PolicyDbContext” which extends from DbContext
2. Import Microsoft.EntityFrameworkCore package.

```
namespace ENFSQLSerDemo.Data
{
    public class PolicyDbContext : DbContext
    {
        public PolicyDbContext(DbContextOptions options) : base(options)
        {
        }

        public DbSet<PolicyAction> PolicyActions { get; set; }
    }
}
```

Create IPolicyData Interface for service

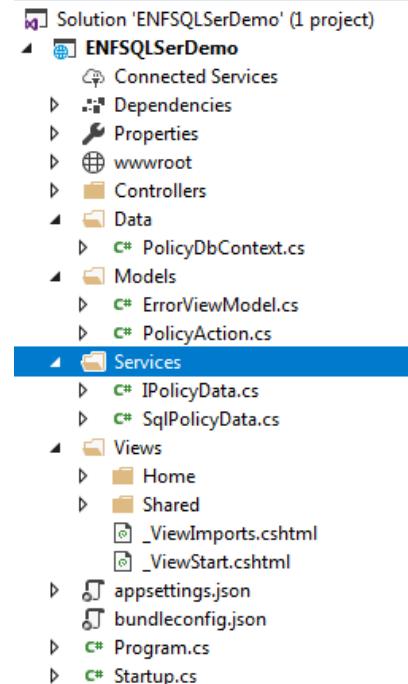
1. Create folder “Services”
2. Create an interface for the service => IPolicyData interface



```
namespace ENFSQLSerDemo.Services
{
    public interface IPolicyData
    {
        IEnumerable<PolicyAction> GetAll();
        PolicyAction GetbyId(int PolicyId);
        PolicyAction Add(PolicyAction policyaction);
    }
}
```

Step 9: Create service implementation for IPolicyData

- Now lets add service that implements the interface IPolicyData



```
namespace ENFSQLSerDemo.Services
{
    public class SqlPolicyData : IPolicyData
    {
        public PolicyDbContext _context;
        public SqlPolicyData(PolicyDbContext context)
        {
            _context = context;
        }
        public PolicyAction Add(PolicyAction policyaction)
        {
            _context.PolicyActions.Add(policyaction);
            _context.SaveChanges();
            return policyaction;
        }
        //for large records IQueryable<T>
        public IEnumerable<PolicyAction> GetAll()
        {
            return _context.PolicyActions.OrderBy(p => p.PolicyActionId);
        }
        public PolicyAction GetbyId(int id)
        {
            return _context.PolicyActions.FirstOrDefault(p => p.PolicyActionId == id);
        }
    }
}
```

Step 10: Configuring the EntityFramework Service to Connect to SQLServer

Please read terms and conditions of use

Original Series

1. We need to configure Entity Framework Service to Connect to SQL Server.
2. Startup.cs => ConfigureServices.
3. Add ConnectionString to AppSettings.json

```
{
  "Logging": {
    "IncludeScopes": false,
    "LogLevel": {
      "Default": "Warning"
    }
  },
  "ConnectionStrings": {
    "DefaultConnection": "Server=(localdb)\\
    \\MSSQLLocalDB;Database=pubpolicydb;Trusted_Connection=True;MultipleActiveResultSets=true"
  }
}
```

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

  2 references | 0 exceptions
  public IConfiguration Configuration { get; set; }

  // 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)
  {
    services.AddDbContext<PolicyDbContext>(options=>options.UseSqlServer(Configuration.GetConnectionString
      ("DefaultConnection")));
    services.AddScoped<IPolicyData, SqlPolicyData>();
    services.AddMvc();
  }
}
```

Dotnet ef commands

Please read terms and conditions of use

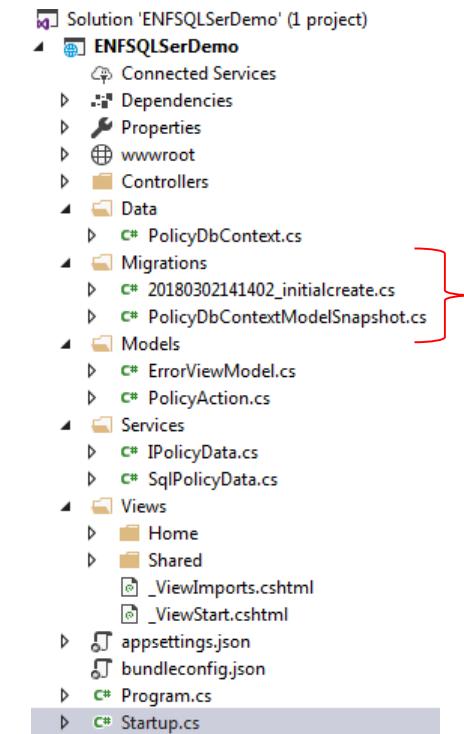
Original Series

Dotnet ef dbcontext list	To list a set of dbcontext
Dotnet ef dbcontext info	To display detailed information about dbcontext providers

Dotnet ef migrations add --help	
Dotnet ef migration add intialcreate	Initial creation migration
Dotnet ef database update -verbose	

Trigger EntityFramework Migrations

1. Lets trigger migrations using the command
2. Dotnet ef migrations add "**<name>**"
3. Dotnet ef migrations update

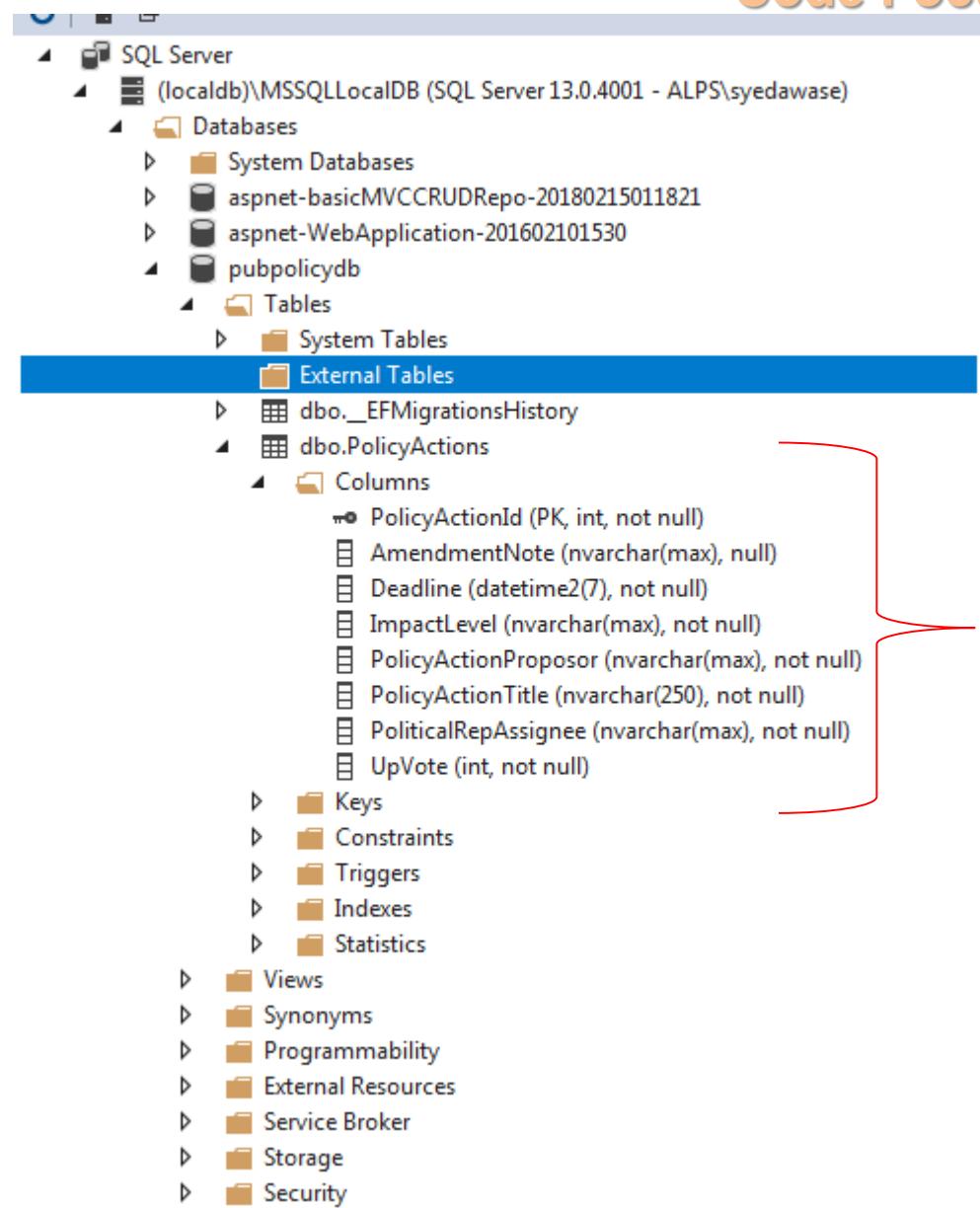


```
PS H:\awase-harddisk\CT\C#201\DOTNETCORE\dotnetcore-appscenarios\apptwentytwo\ENFSQLSerDemo\ENFSQLSerDemo> dotnet ef migrations add initialcreate
info: Microsoft.AspNetCore.DataProtection.KeyManagement.XmlKeyManager[0]
      User profile is available. Using 'C:\Users\syedawase\AppData\Local\ASP.NET\DataProtection-Keys' as key repository and Windows DPAPI to encrypt keys at rest.
info: Microsoft.EntityFrameworkCore.Infrastructure[10403]
      Entity Framework Core 2.0.1-rtm-125 initialized 'PolicyDbContext' using provider 'Microsoft.EntityFrameworkCore.SqlServer' with options: None
Done. To undo this action, use 'ef migrations remove'
```

- dotnet ef migrations add initialcreate
- dotnet ef database update -verbose

Post migrations

1. Post migrations operations run in step 11. we should see that the database is created along with table and columns as shown here.

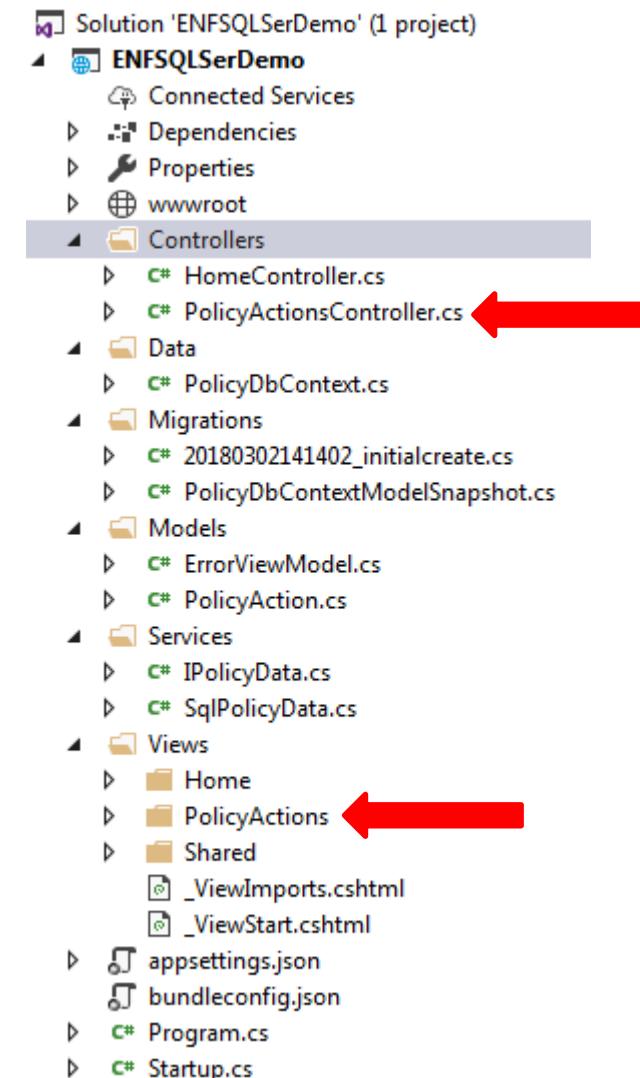


Creating PolicyController and injecting the IPolicyData Interface

Please read terms and conditions of use

Original Series

1. Now lets create the PolicyController and inject IPolicyData Interface.
2. We select from Entity Framework, it would scaffold corresponding views



Build and run your application

1. Prior to that, we shall insert some records into the database.
2. Now lets build and run your application.

PolicyActionId	AmendmentNote	Deadline	ImpactLevel	PolicyActionProposor	PolicyActionTitle	PoliticalRepAs...	UpVote
1	please change the text for the whole district	12/12/2018 12:00:00	district	syed awase	Roadlaying Activity	George	2
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL

localhost:64026/policyactions/

ENFSQLSerDemo Home About Contact

Index

Create New

PolicyActionTitle	PolicyActionProposor	ImpactLevel	PoliticalRepAssignee	Deadline	UpVote	AmendmentNote
Roadlaying Activity	syed awase	district	George	12/12/2018	2	please change the text for the whole district

© 2018 - ENFSQLSerDemo

Loan Management System with ASP

EXERCISE DEMO: 3.6

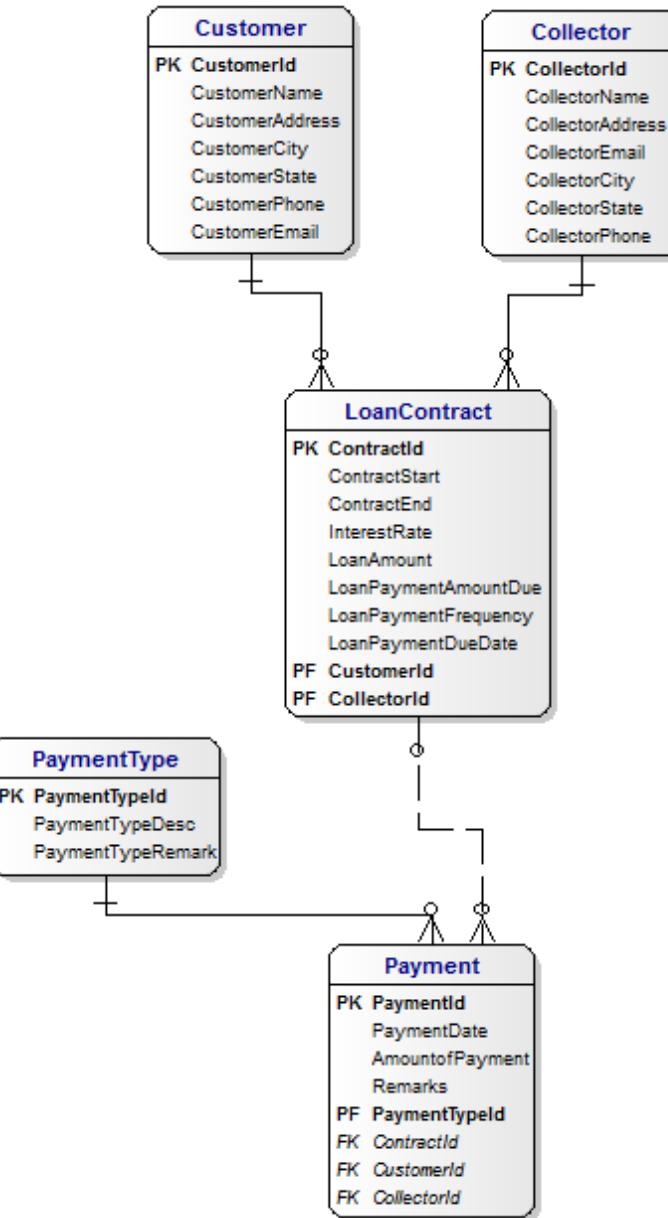
LEARNING OUTCOMES

- Installing EntityFramework Core 2.0 CLI
- Creating DbContext File, DbContext Methods and DbContext properties.
- Binding the DbContext with a service

Loan management system ERD

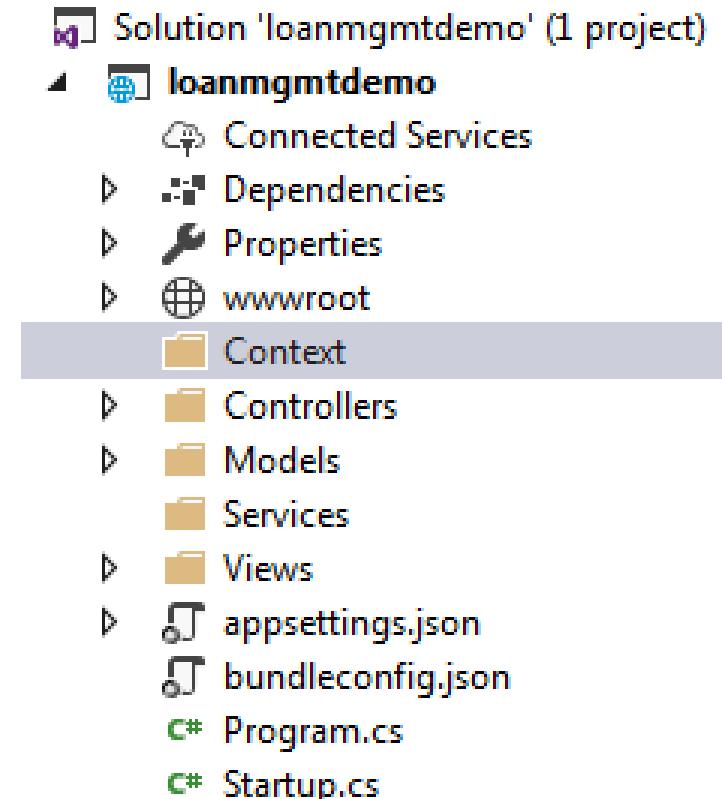
Please read terms and conditions of use

Original Series



Create an ASP.NET Core 2.0 MVC Application

1. Lets create an ASP.NET Core 2.0 MVC Application.
2. We should see a scaffolding project structure as shown
3. Lets add two folders
 1. Services
 2. Context



Create Models based on the Loan Management ERD

1. Lets create domain models corresponding to the loan management ERD diagram
2. Create the following models
 1. Customer
 2. Collector
 3. LoanContract
 4. PaymentType
 5. Payment

```
namespace loanmgmtdemo.Models
{
    1 reference
    public class Customer
    {
        [Key]
        [Required]
        [DatabaseGeneratedAttribute(DatabaseGeneratedOption.Identity)]
        0 references | 0 exceptions
        public int CustomerId { get; set; }

        [Required]
        0 references | 0 exceptions
        public string CustomerName { get; set; }

        [Required]
        0 references | 0 exceptions
        public string CustomerAddress { get; set; }

        [Required]
        0 references | 0 exceptions
        public string CustomerCity { get; set; }

        [Required]
        0 references | 0 exceptions
        public string CustomerState { get; set; }

        [Required,MaxLength(12)]
        0 references | 0 exceptions
        public string CustomerPhone { get; set; }

        [Required]
        0 references | 0 exceptions
        public string CustomerEmail { get; set; }

        // one-to-many reference one customer has many LoanContracts
        0 references | 0 exceptions
        public virtual ICollection<LoanContract> LoanContracts { get; set; }
    }
}
```

Collector model

```
namespace loanmgmtdemo.Models
{
    1 reference
    public class Collector
    {
        [Key]
        [Required]
        [DatabaseGeneratedAttribute(DatabaseGeneratedOption.Identity)]
        0 references | 0 exceptions
        public int CollectorId { get; set; }
        [Required, MaxLength(100)]
        0 references | 0 exceptions
        public string CollectorName { get; set; }
        [Required]
        0 references | 0 exceptions
        public string CollectorAddress { get; set; }
        [Required]
        0 references | 0 exceptions
        public string CollectorEmail { get; set; }
        [Required]
        0 references | 0 exceptions
        public string CollectorCity { get; set; }
        [Required]
        0 references | 0 exceptions
        public string CollectorState { get; set; }
        [MaxLength(12)]
        0 references | 0 exceptions
        public string CollectorPhone { get; set; }

        // one-to-many reference one customer has many LoanContracts
        0 references | 0 exceptions
        public virtual ICollection<LoanContract> LoanContracts { get; set; }
    }
}
```

loanContract

Please read terms and conditions of use

Original Series

```
{  
    3 references  
    public class LoanContract  
    {  
        [Key]  
        [Required]  
        [DatabaseGeneratedAttribute(DatabaseGeneratedOption.Identity)]  
        public int ContractId { get; set; }  
        [Required]  
        [Range(typeof(DateTime), "1/2/2004", "1/2/2009", ErrorMessage = "Value for {0} must be between {1} and {2}")]  
        [DisplayFormat(DataFormatString = "{0:yyyy/MM/DD}", ApplyFormatInEditMode = true)]  
        public DateTime ContractStart { get; set; }  
        [Required]  
        [Range(typeof(DateTime), "1/2/2004", "1/2/2009", ErrorMessage = "Value for {0} must be between {1} and {2}")]  
        [DisplayFormat(DataFormatString = "{0:yyyy/MM/DD}", ApplyFormatInEditMode = true)]  
        public DateTime ContractEnd { get; set; }  
        [Required]  
        public int InterestRate { get; set; }  
        [Required]  
        public int LoanAmount { get; set; }  
        [Required]  
        public int LoanPaymentAmountDue { get; set; }  
        [Required]  
        public int LoanPaymentFrequency { get; set; }  
        [Required]  
        public DateTime LoanPaymentDueDate { get; set; }  
        //one loan contract has many payments  
        public virtual ICollection<Payment> Payments { get; set; }  
        // foreign key reference  
        public virtual Collector Collector { get; set; }  
        public virtual Customer Customer { get; set; }  
    }  
}
```

PaymentType

Please read terms and conditions of use

Original Series

```
namespace loanmgtdemo.Models
{
    1 reference
    public class PaymentType
    {
        [Key]
        [Required]
        [DatabaseGeneratedAttribute(DatabaseGeneratedOption.Identity)]
        0 references | 0 exceptions
        public int PaymentTypeId { get; set; }
        [Required, MaxLength(200)]
        0 references | 0 exceptions
        public int PaymentTypeDesc { get; set; }
        0 references | 0 exceptions
        public int PaymentTypeRemarks { get; set; }

        // one-to-many reference one PaymentType has many Payments
        0 references | 0 exceptions
        public virtual ICollection<Payment> Payments { get; set; }
    }
}
```

Payment

```
namespace loanmgtdemo.Models
{
    public class Payment
    {
        [Key]
        [Required]
        [DatabaseGeneratedAttribute(DatabaseGeneratedOption.Identity)]
        public int PaymentId { get; set; }

        [Required]
        [DisplayFormat(DataFormatString = "{0:yyyy/MM/DD}", ApplyFormatInEditMode = true)]
        public DateTime PaymentDate { get; set; }

        [Required]
        public int AmountofPayment { get; set; }

        public string Remarks { get; set; }

        // foreign key reference
        public virtual PaymentType PaymentType { get; set; }

        // foreign key reference
        public virtual LoanContract LoanContract { get; set; }
    }
}
```

Add DotNetCliToolReference

- Now let us add
DotNetCliToolReference for
`Microsoft.EntityFrameworkCore.Tools.DotNet`

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

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

<ItemGroup>
  <PackageReference Include="Microsoft.AspNetCore.All" Version="2.0.5" />
</ItemGroup>

<ItemGroup>
  <DotNetCliToolReference Include="Microsoft.VisualStudio.Web.CodeGeneration.Tools" Version="2.0.2" />
<DotNetCliToolReference Include="Microsoft.EntityFrameworkCore.Tools.DotNet" Version="2.0.1" />
</ItemGroup>

<ItemGroup>
  <Folder Include="Context\" />
  <Folder Include="Services\" />
</ItemGroup>

</Project>
```

Add LoanMgmtDbContext File

- i. Create a folder “Data” and add “PolicyDbContext” which extends from DbContext
- ii. Import Microsoft.EntityFrameworkCore package.

```
namespace loanmgtdemo.Context
{
    public class LoanMgmtDbContext : DbContext
    {
        public LoanMgmtDbContext(DbContextOptions options) : base(options)
        {

        }

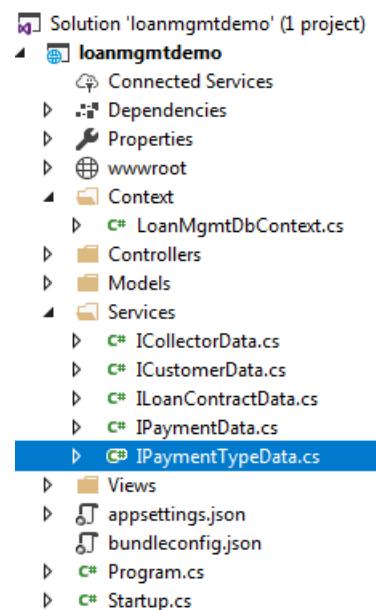
        public DbSet<Customer> Customers { get; set; }
        public DbSet<Collector> Collectors { get; set; }
        public DbSet<LoanContract> LoanContracts { get; set; }
        public DbSet<PaymentType> PaymentTypes { get; set; }
        public DbSet<Payment> Payments { get; set; }
    }
}
```

Create Interfaces for implementing the Services.

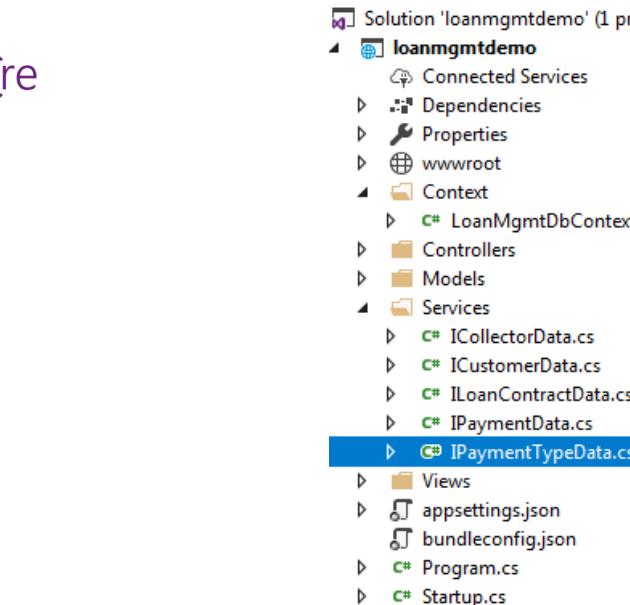
- Create Interfaces for implementing the services.

- ICustomerData
- ICollectorData
- ILoanContractData
- IPaymentTypeData
- IPaymentData

```
namespace loanmgtdemo.Services
{
    0 references
    public interface ICustomerData
    {
        0 references | 0 exceptions
        IEnumerable<Customer> GetAllCustomers();
        0 references | 0 exceptions
        Customer GetCustomerById(int CustomerId);
        0 references | 0 exceptions
        Customer Add(Customer customer);
    }
}
```



```
namespace loanmgtdemo.Services
{
    0 references
    public interface ILoanContractData
    {
        0 references | 0 exceptions
        IEnumerable<LoanContract> GetAllContracts();
        0 references | 0 exceptions
        LoanContract GetContractById(int ContractId);
        0 references | 0 exceptions
        LoanContract Add(LoanContract contract);
    }
}
```



```
namespace loanmgmtdemo.Services
{
    public interface ICollectorData
    {
        IEnumerable<Collector> GetAllCollectors();
        Collector GetbyCollectorId(int CollectorId);
        Collector Add(Collector collector);
    }
}
```

```
namespace loanmgmtdemo.Services
{
    public interface IPaymentData
    {
        IEnumerable<Payment> GetAllPayments();
        Payment GetPaymentById(int PaymentId);
        Payment Add(Payment payment);
    }
}
```

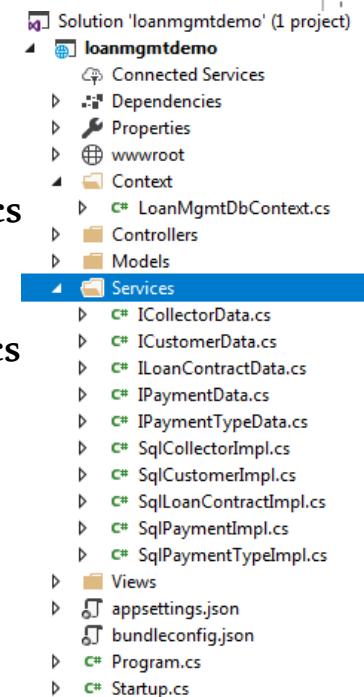
```
namespace loanmgmtdemo.Services
{
    public interface IPaymentTypeData
    {
        IEnumerable<PaymentType> GetAllPaymentTypes();
        PaymentType GetPaymentTypeById(int PaymentTypeId);
        PaymentType Add(PaymentType paymenttype);
    }
}
```

Step 6: Create Service

Implementations for all the interfaces.

i. Create Service Implementations for all the interfaces

- SqlCustomerImpl.cs
- SqlCollectorImpl.cs
- SqlLoanContractImpl.cs
- SqlPaymentImpl.cs
- SqlPaymentTypeImpl.cs



```
namespace loanmgmtdemo.Services

1 reference
public class SqlPaymentImpl : IPaymentData
{
    public LoanMgmtDbContext _context;

0 references | 0 exceptions
public SqlPaymentImpl(LoanMgmtDbContext context)
{
    _context = context;
}

1 reference | 0 exceptions
public Payment Add(Payment payment)
{
    throw new NotImplementedException();
}

1 reference | 0 exceptions
public IEnumerable<Payment> GetAllPayments()
{
    return _context.Payments.OrderBy(p => p.PaymentId);
}

1 reference | 0 exceptions
public Payment GetPaymentById(int PaymentId)
{
    return _context.Payments.FirstOrDefault(p => p.PaymentId == PaymentId);
}
```

```
namespace loanmgmtdemo.Services
{
    1 reference
    public class SqlCustomerImpl : ICustomerData
    {
        public LoanMgmtDbContext _context;
        0 references | 0 exceptions
        public SqlCustomerImpl(LoanMgmtDbContext context)
        {
            _context = context;
        }
        1 reference | 0 exceptions
        public Customer Add(Customer customer)
        {
            _context.Customers.Add(customer);
            _context.SaveChanges();
            return customer;
        }

        1 reference | 0 exceptions
        public IEnumerable<Customer> GetAllCustomers()
        {
            return _context.Customers.OrderBy(c => c.CustomerId);
        }

        1 reference | 0 exceptions
        public Customer GetCustomerById(int CustomerId)
        {
            return _context.Customers.FirstOrDefault(c => c.CustomerId == CustomerId);
        }
    }
}
```

```
namespace loanmgmtdemo.Services
{
    1 reference
    public class SqlLoanContractImpl : ILoanContractData
    {
        public LoanMgmtDbContext _context;
        0 references | 0 exceptions
        public SqlLoanContractImpl(LoanMgmtDbContext context)
        {
            _context = context;
        }

        1 reference | 0 exceptions
        public LoanContract Add(LoanContract contract)
        {
            _context.LoanContracts.Add(contract);
            _context.SaveChanges();
            return contract;
        }

        1 reference | 0 exceptions
        public IEnumerable<LoanContract> GetAllContracts()
        {
            return _context.LoanContracts.OrderBy(l => l.ContractId);
        }

        1 reference | 0 exceptions
        public LoanContract GetContractById(int ContractId)
        {
            return _context.LoanContracts.FirstOrDefault(l => l.ContractId == ContractId);
        }
    }
}
```

```
namespace loanmgmtdemo.Services
{
    public class SqlCollectorImpl : ICollectorData
    {
        public LoanMgmtDbContext _context;
        public SqlCollectorImpl(LoanMgmtDbContext context)
        {
            _context = context;
        }
        public Collector Add(Collector collector)
        {
            _context.Collectors.Add(collector);
            _context.SaveChanges();
            return collector;
        }

        public IEnumerable<Collector> GetAllCollectors()
        {
            return _context.Collectors.OrderBy(c => c.CollectorId);
        }

        public Collector GetbyCollectorId(int CollectorId)
        {
            return _context.Collectors.FirstOrDefault(c => c.CollectorId == CollectorId);
        }
    }
}
```

```
namespace loanmgmtdemo.Services
{
    public class SqlPaymentTypeImpl : IPaymentTypeData
    {
        public LoanMgmtDbContext _context;
        public SqlPaymentTypeImpl(LoanMgmtDbContext context)
        {
            _context = context;
        }
        public PaymentType Add(PaymentType paymenttype)
        {
            _context.PaymentTypes.Add(paymenttype);
            _context.SaveChanges();
            return paymenttype;
        }

        public IEnumerable<PaymentType> GetAllPaymentTypes()
        {
            return _context.PaymentTypes.OrderBy(p => p.PaymentTypeId);
        }

        public PaymentType GetPaymentTypeById(int PaymentTypeId)
        {
            return _context.PaymentTypes.FirstOrDefault(p => p.PaymentTypeId == PaymentTypeId);
        }
    }
}
```

Register the service with startup.cs

- **AddTransient** is a way of telling ASP.NET Core, to create a new instance for every request and perform garbage collection at the end of the request.

Appsettings.json

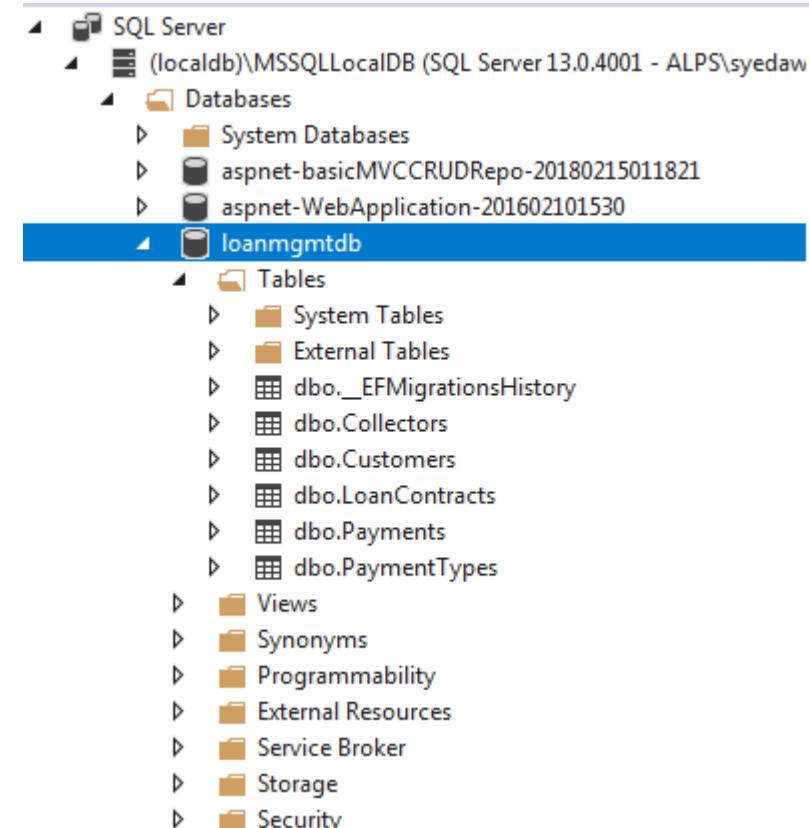
```
{  
    "Logging": {  
        "IncludeScopes": false,  
        "LogLevel": {  
            "Default": "Warning"  
        }  
    },  
    "ConnectionStrings": {  
        "loanDb": "Server=(localdb)\\  
        \\MSSQLLocalDB;Database=loanmgmtdb;Trusted_Connection=True;MultipleActiveResultSets=true"  
    }  
}
```

```
// 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)  
{  
    services.AddDbContext<LoanMgmtDbContext>(options => options.UseSqlServer  
        (Configuration.GetConnectionString("loanDb")));  
    services.AddTransient<ICustomerData, SqlCustomerImpl>();  
    services.AddTransient<ICollectorData, SqlCollectorImpl>();  
    services.AddTransient<ILoanContractData, SqlLoanContractImpl>();  
    services.AddTransient<IPaymentTypeData, SqlPaymentTypeImpl>();  
    services.AddTransient<IPaymentData, SqlPaymentImpl>();  
    services.AddMvc();  
}
```

Trigger EntityFramework Migrations

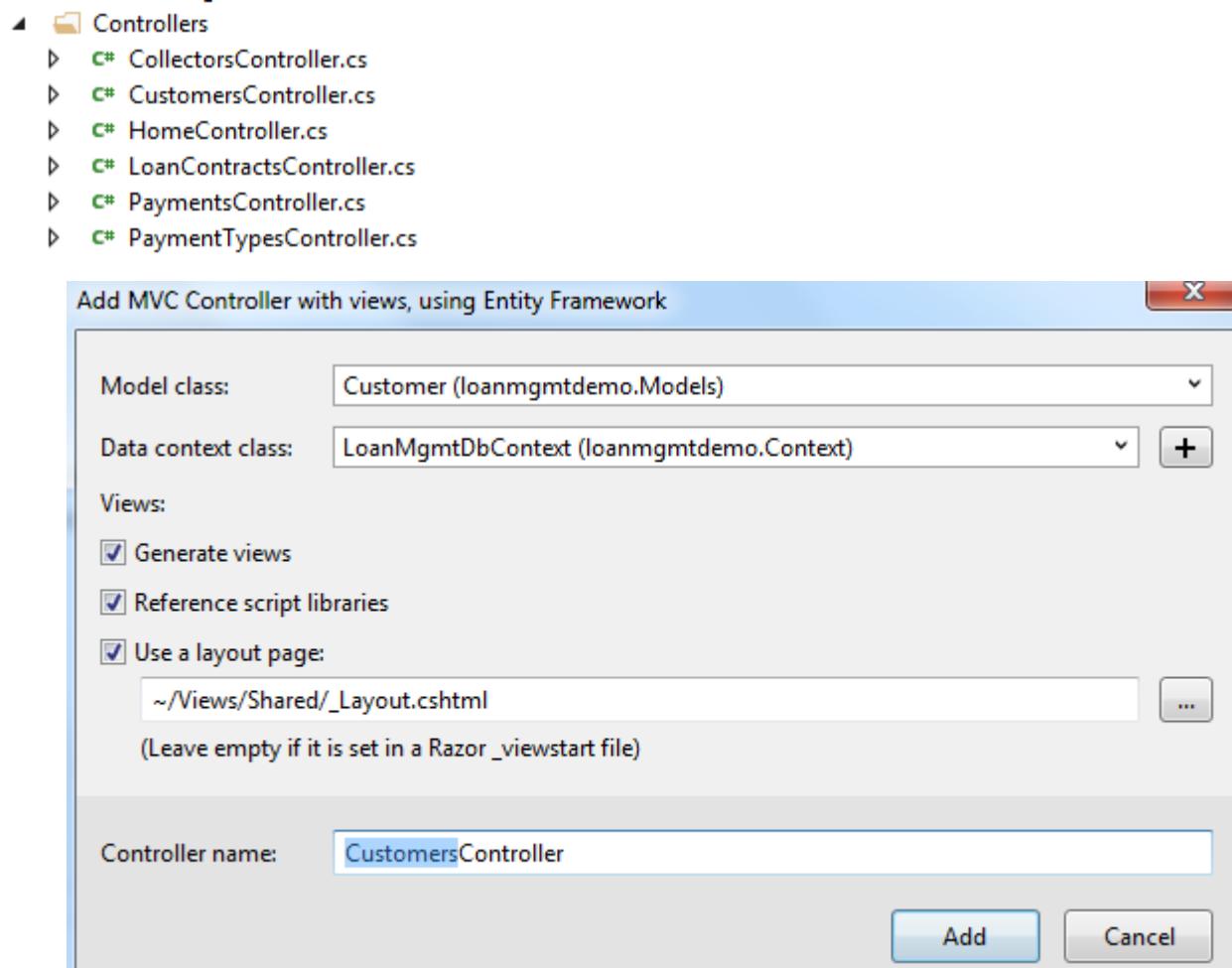
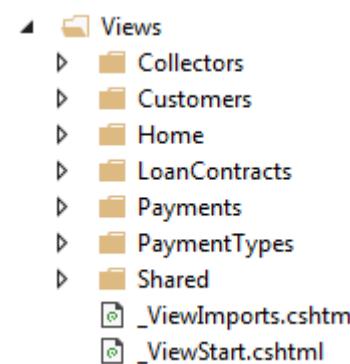
1. Lets trigger migrations using the command
2. Dotnet ef migrations add “<name>”
3. Dotnet ef migrations update

- dotnet ef migrations add initialcreate
- dotnet ef database update -verbose



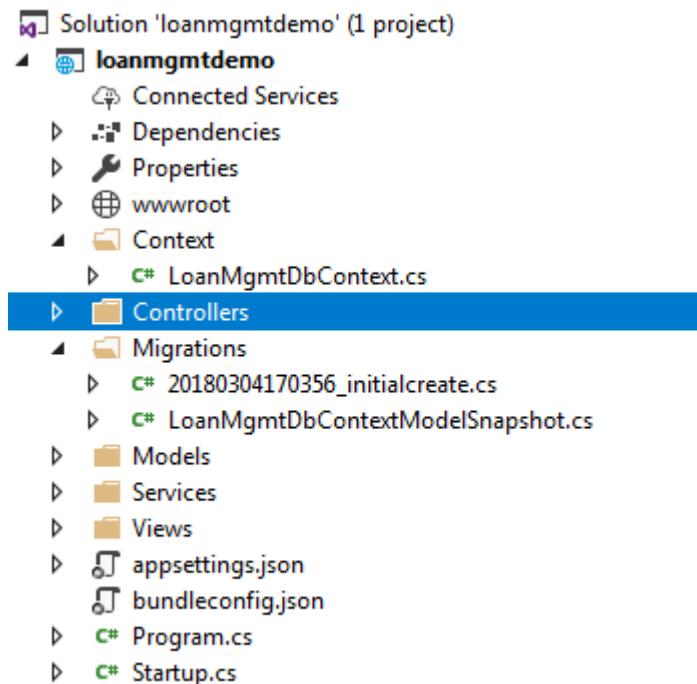
Create Controllers for all the domain models.

- i. Build your solution and then create controller
- ii. Create controllers for all the domain models.
- iii. It would generate corresponding views to be rendered based on the controller.



Build and run your application

i. Build and run your application



localhost:18687/customers

loanmgmtdemo Home About Contact

Index

Create New

CustomerName	CustomerAddress	CustomerCity	CustomerState	CustomerPhone	CustomerEmail

© 2018 - loanmgmtdemo

SECTION -XVIII

SIGNAL R

TPRI-SYCLIQ PROGRAMS OVERVIEW

EMPOWERING YOU

Artificial Intelligence

We also train on AI Stack

Reach out to us sak@sycliq.com

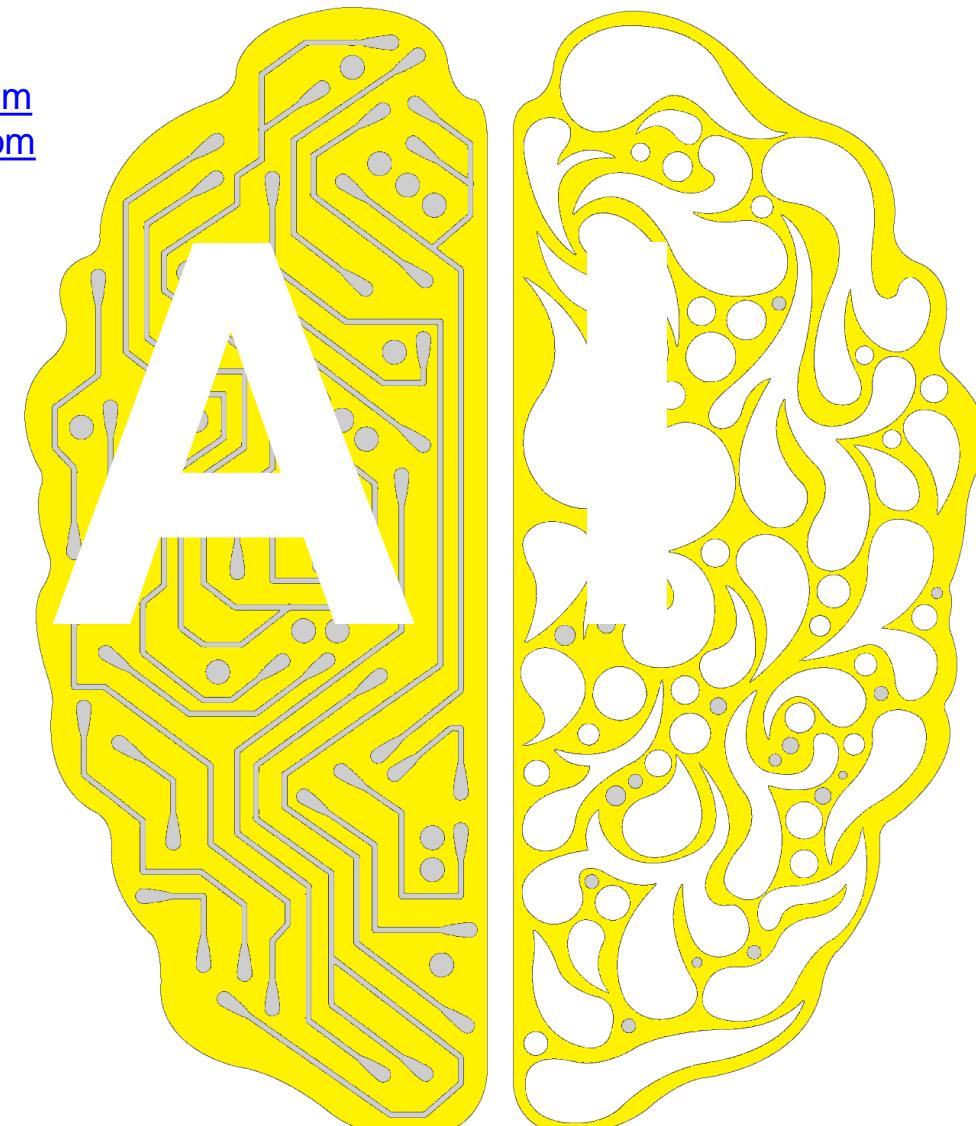
~~Office~~ sak@territorialprescience.com

~~www~~ www.territorialprescience.com

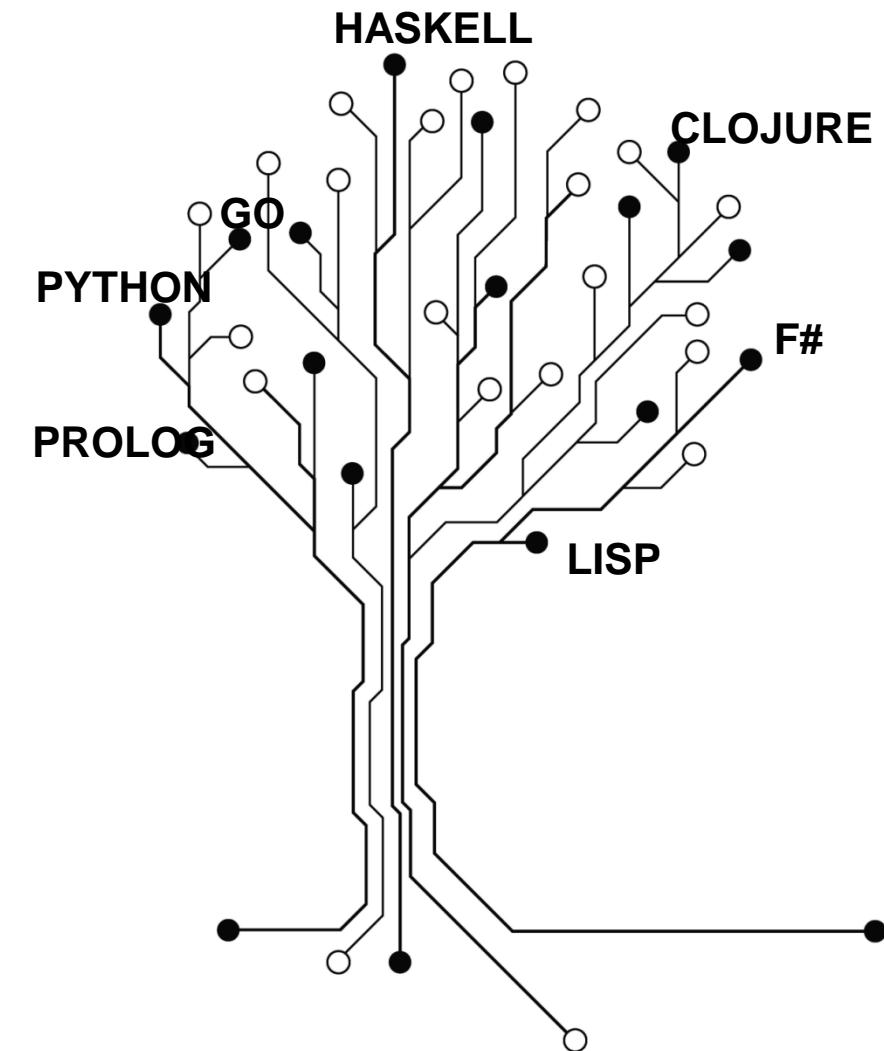
~~www~~ www.sycliq.com

91.9035433124

Please read terms and conditions



© TERRITORIAL™ SAK - Syed Awase 2017 C# GROUND UP
SERIES

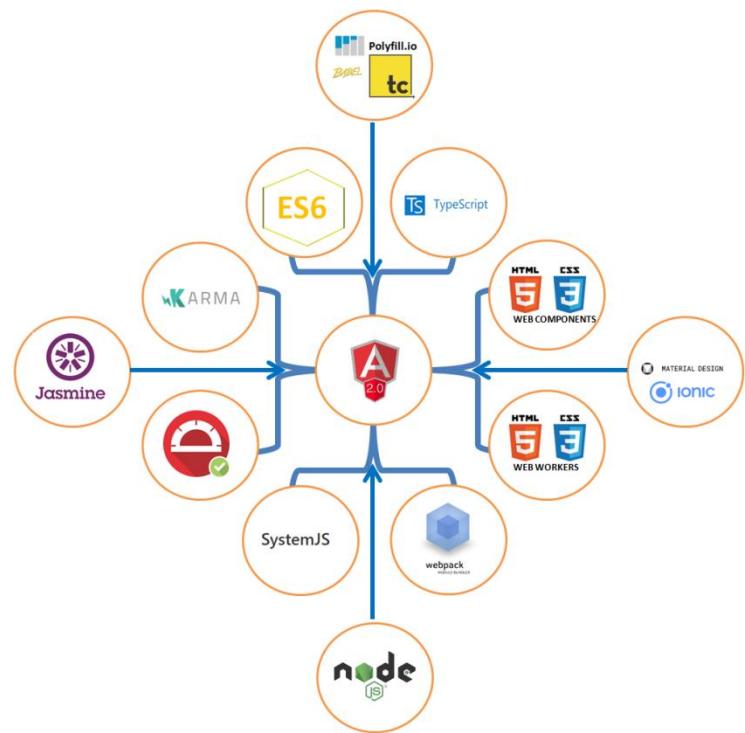


Angular 2.x/Angular JS 1.5.x

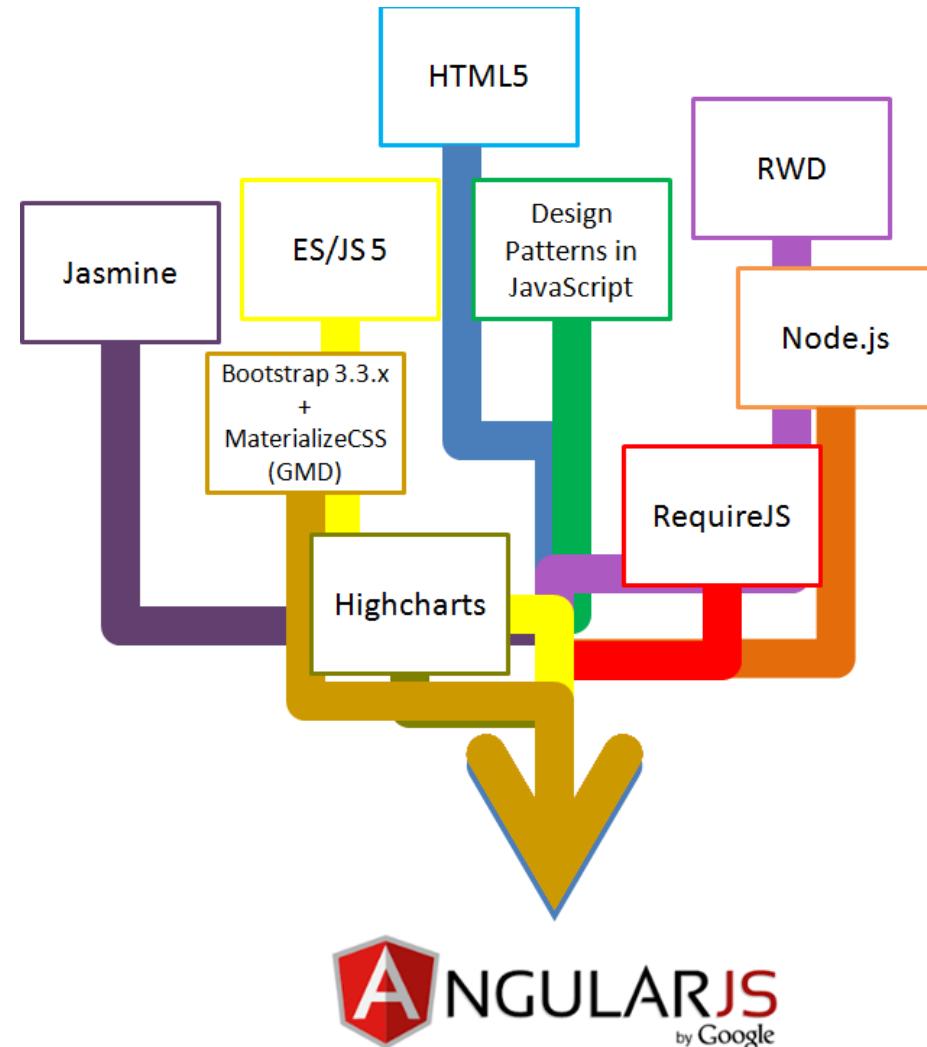
Please read terms and conditions of use

Dr. Syed Awase 2016 Session Feedbacks: <http://bit.ly/2hhNg58>

Reach out to sak@territorialprescience.com/+91.9035433124



© TPRI/SYCLIQ -Syed Awase 2017 C# GROUND UP
SERIES



Dr. Syed Awase also offers Machine learning Stack, R Statistical Stack, .NET Stack, Java Stack, RaspberryPi Stack. Get the pulse of performance from here <http://bit.ly/2hhNg58>

JavaScript Frameworks

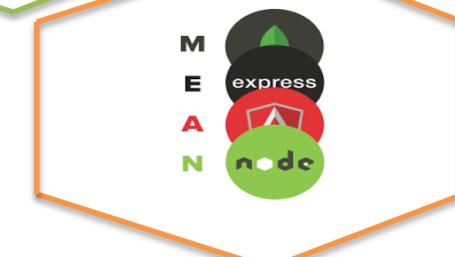
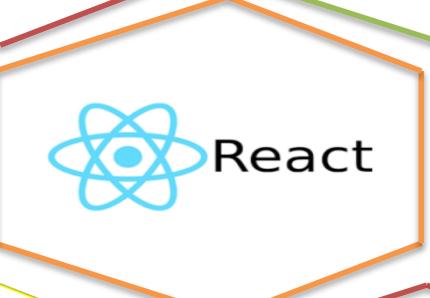
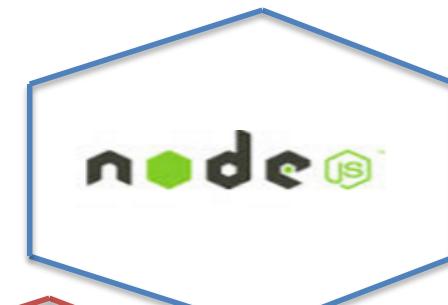
Code DRIVEN CAPACITY BUILDING PROGRAM

Dr. Syed Awase 2016 Session Feedbacks: <http://bit.ly/2hhNg58>

Reach out to sak@territorialprescience.com/+91.9035433124

Please read terms and conditions of use

Original Series



Dr. Syed Awase also offers Machine learning Stack, R Statistical Stack, .NET Stack, Java Stack, RaspberryPi Stack. Get the pulse of performance from here <http://bit.ly/2hhNg58>

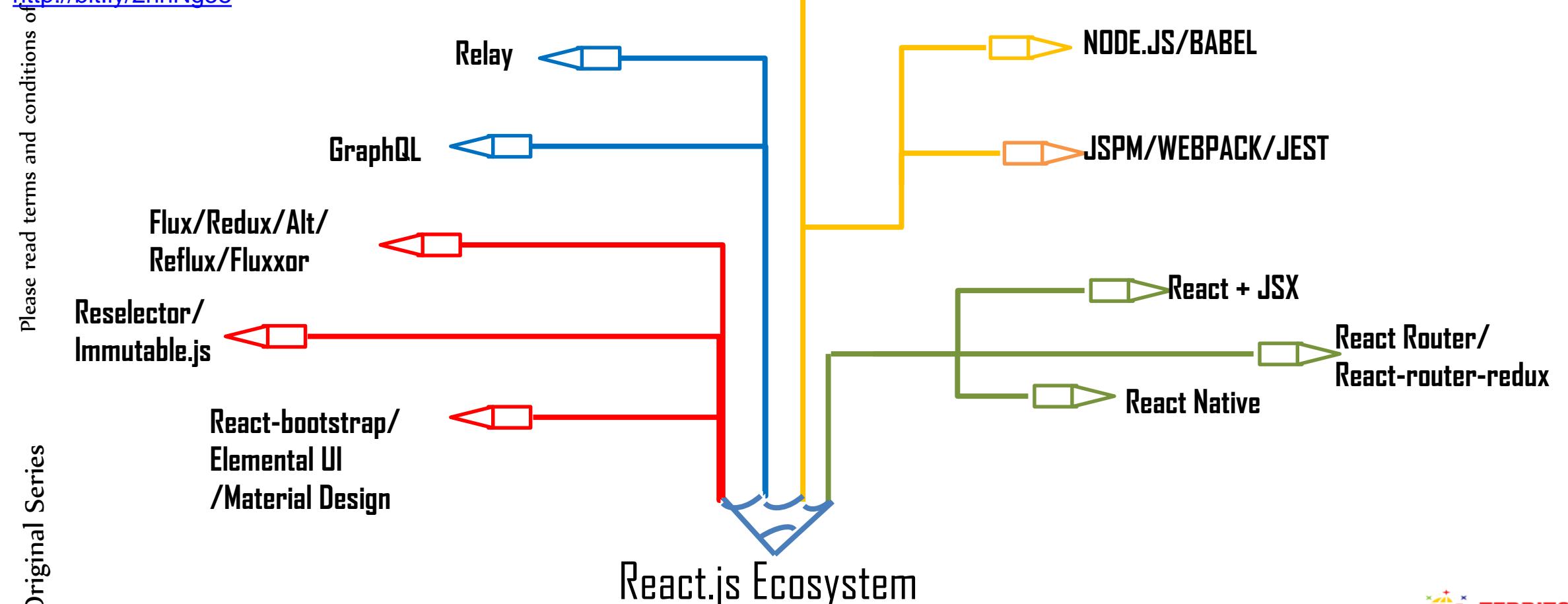
© TPRI/SYCLIQ -Syed Awase 2017 C# GROUND UP SERIES

Dr. Syed Awase also offers Machine learning Stack, R Statistical Stack, .NET Stack, Java Stack, RaspberryPi Stack. Get the pulse of performance from here
<http://bit.ly/2hhNg58>

ES 5/6

REACT.JS LEARNING PATH

Dr. Syed Awase
Now Offering



R-Statistical Programming

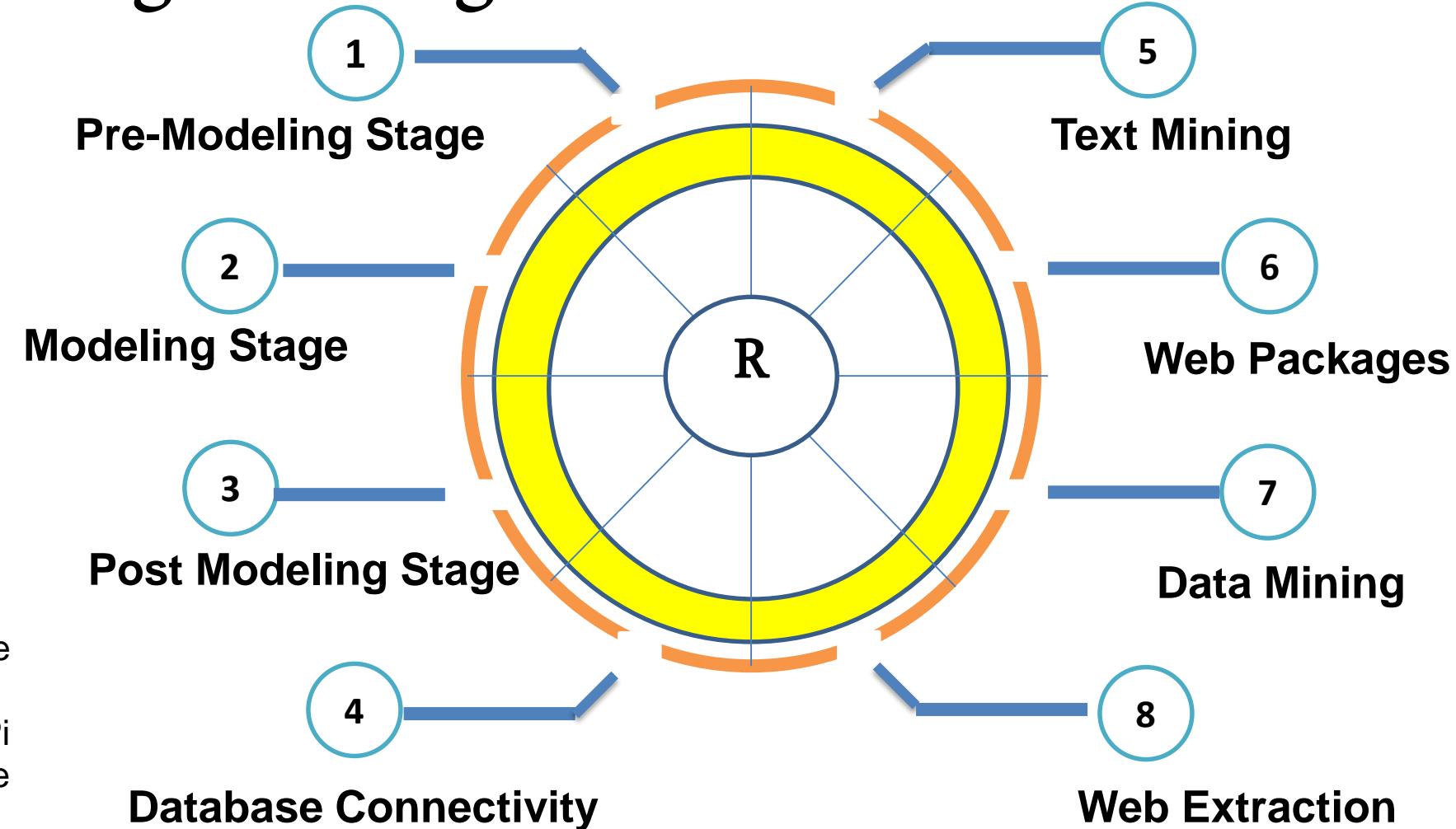
Please read terms and conditions of use

Dr. Syed Awase 2016 Session
Feedbacks: <http://bit.ly/2hhNg58>

Reach out to
sak@territorialprescience.com/
+91.9035433124

Dr. Syed Awase also offers Machine
learning Stack, R Statistical Stack,
.NET Stack, Java Stack, RaspberryPi
Stack. Get the pulse of performance
from here <http://bit.ly/2hhNg58>

Original



Thank You

We also provide Code Driven Open House Trainings : sak@territorialprescience.com or sak@sycliq.com

Please read terms and conditions of use



- Java Technologies
- Core Java
 - Hibernate
 - Spring Framework
 - Play Framework
 - Hadoop
 - Groovy & Grails



- Microsoft Technologies
- C# Core
 - Entity Framework
 - MVC 5/6
 - Web Api
 - OWIN/KATANA
 - WCF
 - WPF



- Python
- Python
 - Django
 - Flask
 - Numpy
 - Scipy
 - Machine Learning



- Data Science
- R Statistical Programming
 - Julia



- SQL and NoSQL
- Oracle
 - PostgreSQL
 - MSSQL
 - MongoDB
 - Neo4j
 - Redis
 - Firebase
 - Apache Cassandra



- Client-Side Frameworks
- Angular JS 1.5.x
 - Angular 2.4.x
 - React JS
 - KnockOut JS
 - VueJS
 - Backbone JS
 - EMBER JS
 - Hapi JS
 - METEORJS
 - MEANJS
 - Coffeescript
 - Dart



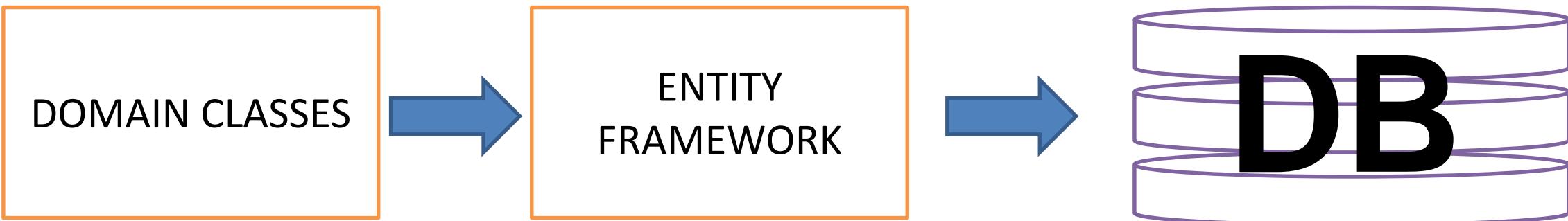
- Others
- LISP
 - CLOJURE
 - RUST
 - GO
 - Raspberry PI
 - Coming Soon
 - PHP
 - RoboticOS

Original Series

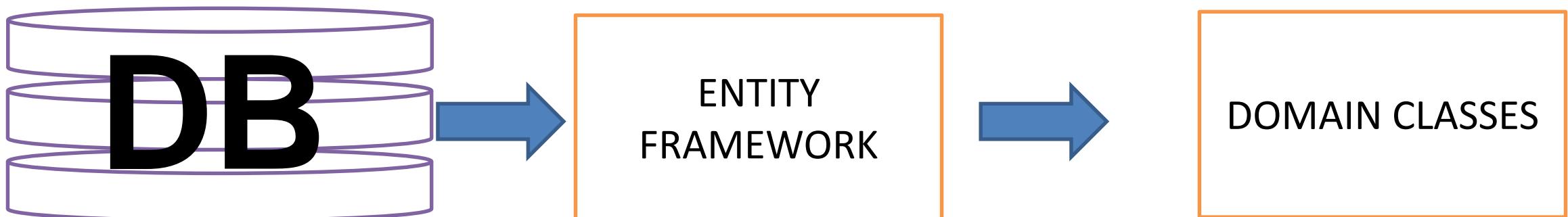
- <http://hamidmosalla.com/2017/03/29/asp-net-core-action-results-explained/>
- <https://www.dustinhorne.com/post/2017/10/27/deeper-dive-into-ef-core-2-part-1>

Approaches

CODE FIRST APPROACH



DATABASE FIRST APPROACH



Please read terms and conditions of use

Original Series