

<epam>

# Module "Web"

## Submodule "MVC"

Part 1

# AGENDA

---

**1** ASP.NET: web application types

**2** Configuration

**3** Life cycle

**4** Client state

**5** Server state

# ASP.NET: web application types

# ASP.NET: Overview

---

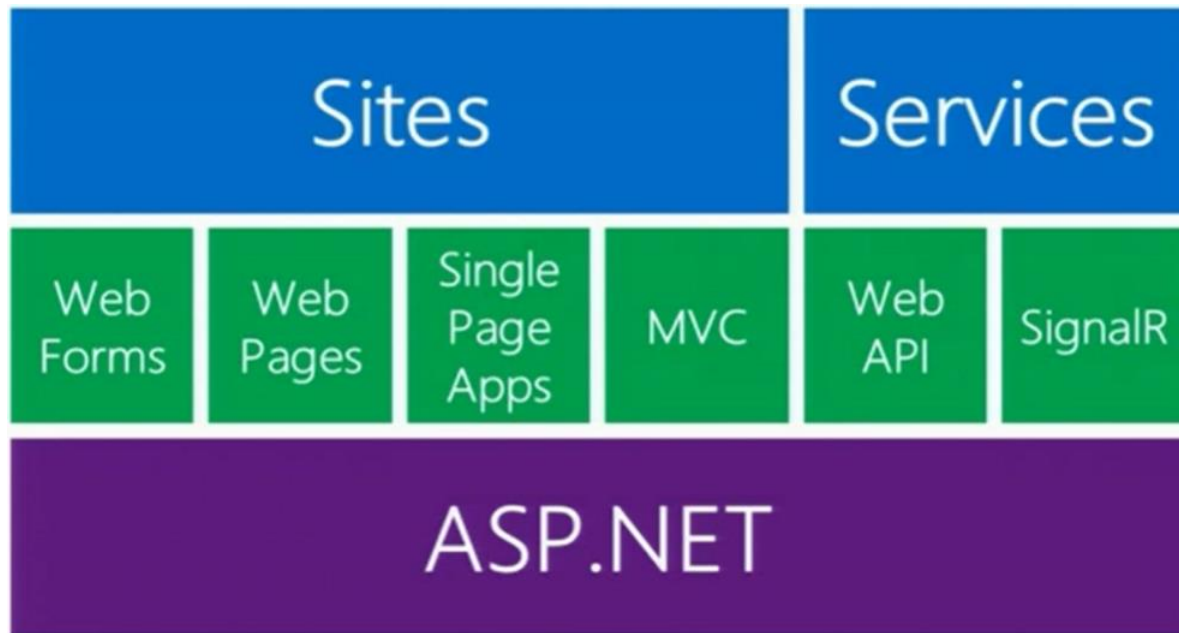
**ASP.NET** is a free web framework for building great websites and web applications using HTML, CSS, and JavaScript. You can also create Web APIs and use real-time technologies like Web Sockets.

**ASP.NET Core** is an alternative to ASP.NET.

# ASP.NET: Websites and web applications

ASP.NET offers three frameworks for creating web applications:

- Web Forms
- ASP.NET MVC
- ASP.NET Web Pages.



# ASP.NET: Websites and web applications

|           | If you have experience in | Development style   | Expertise               |
|-----------|---------------------------|---|-------------------------|
| Web Forms | Win Forms, WPF, .NET      | Rapid development using a rich library of controls that encapsulate HTML markup   | Mid-Level, Advanced RAD |
| MVC       | Ruby on Rails, .NET       | Full control over HTML markup, code and markup separated, and easy to write tests. The best choice for mobile and single-page applications (SPA). | Mid-Level, Advanced     |
| Web Pages | Classic ASP, PHP          | HTML markup and your code together in the same file   | New, Mid-Level          |

# ASP.NET: Web application types - Web Forms

---

With ASP.NET Web Forms, you can build dynamic websites using a familiar drag-and-drop, event-driven model. A design surface and hundreds of controls and components let you rapidly build sophisticated, powerful UI-driven sites with data access

<https://docs.microsoft.com/en-us/aspnet/web-forms/>

# ASP.NET: Web application types - MVC

---

ASP.NET MVC gives you a powerful, patterns-based way to build dynamic websites that enables a clean separation of concerns and that gives you full control over markup for enjoyable, agile development. ASP.NET MVC includes many features that enable fast, TDD-friendly development for creating sophisticated applications that use the latest web standards.

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



# ASP.NET: Web application types - ASP.NET Web Pages

---

ASP.NET Web Pages and the Razor syntax provide a fast, approachable, and lightweight way to combine server code with HTML to create dynamic web content.

<https://docs.microsoft.com/en-us/aspnet/web-pages/>

# ASP.NET: Web application types - Web APIs

---

ASP.NET Web API is a framework that makes it easy to build HTTP services that reach a broad range of clients, including browsers and mobile devices.

ASP.NET Web API is an ideal platform for building RESTful applications on the .NET Framework.

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

# ASP.NET: Web application types - Real-time technologies

---

ASP.NET SignalR is a library for ASP.NET developers that makes developing real-time web functionality easier. SignalR allows bi-directional communication between server and client. Servers can push content to connected clients instantly as it becomes available. SignalR supports Web Sockets, and falls back to other compatible techniques for older browsers. SignalR includes APIs for connection management (for instance, connect and disconnect events), grouping connections, and authorization.

<https://docs.microsoft.com/en-us/aspnet/signalr/>

# ASP.NET: Web application types - Mobile apps and sites

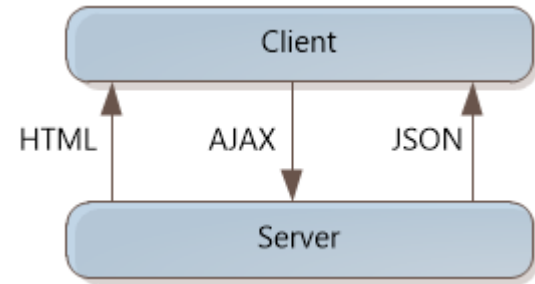
---

ASP.NET can power native mobile apps with a Web API back end, as well as mobile web sites using responsive design frameworks like Twitter Bootstrap. If you are building a native mobile app, it's easy to create a JSON-based Web API to handle data access, authentication, and push notifications for your app. If you are building a responsive mobile site, you can use any CSS framework or open grid system you prefer.

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

# ASP.NET: Web application types - Single-page applications (SPA)

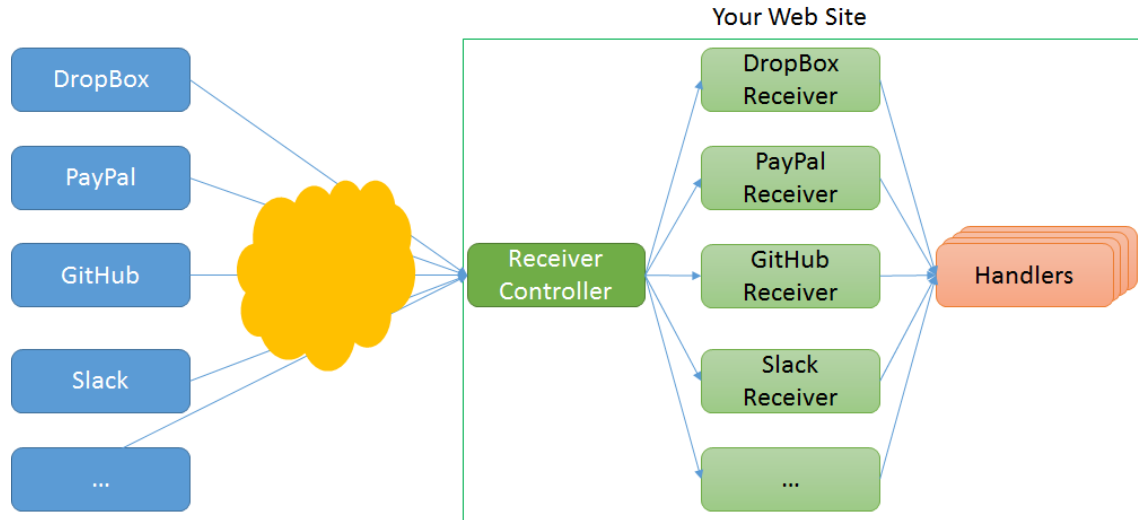
ASP.NET Single Page Application (SPA) helps you build applications that include significant client-side interactions using HTML 5, CSS 3 and JavaScript. Visual Studio includes a template for building single page applications using knockout.js and ASP.NET Web API. In addition to the built-in SPA template, community-created SPA templates are also available for download.



<https://docs.microsoft.com/en-us/aspnet/single-page-application/>

# ASP.NET: Web application types - WebHooks

WebHooks is a lightweight HTTP pattern providing a simple pub/sub model for wiring together Web APIs and SaaS services. When an event happens in a service, a notification is sent in the form of an HTTP POST request to registered subscribers. The POST request contains information about the event which makes it possible for the receiver to act accordingly.



<https://docs.microsoft.com/en-us/aspnet/webhooks/>

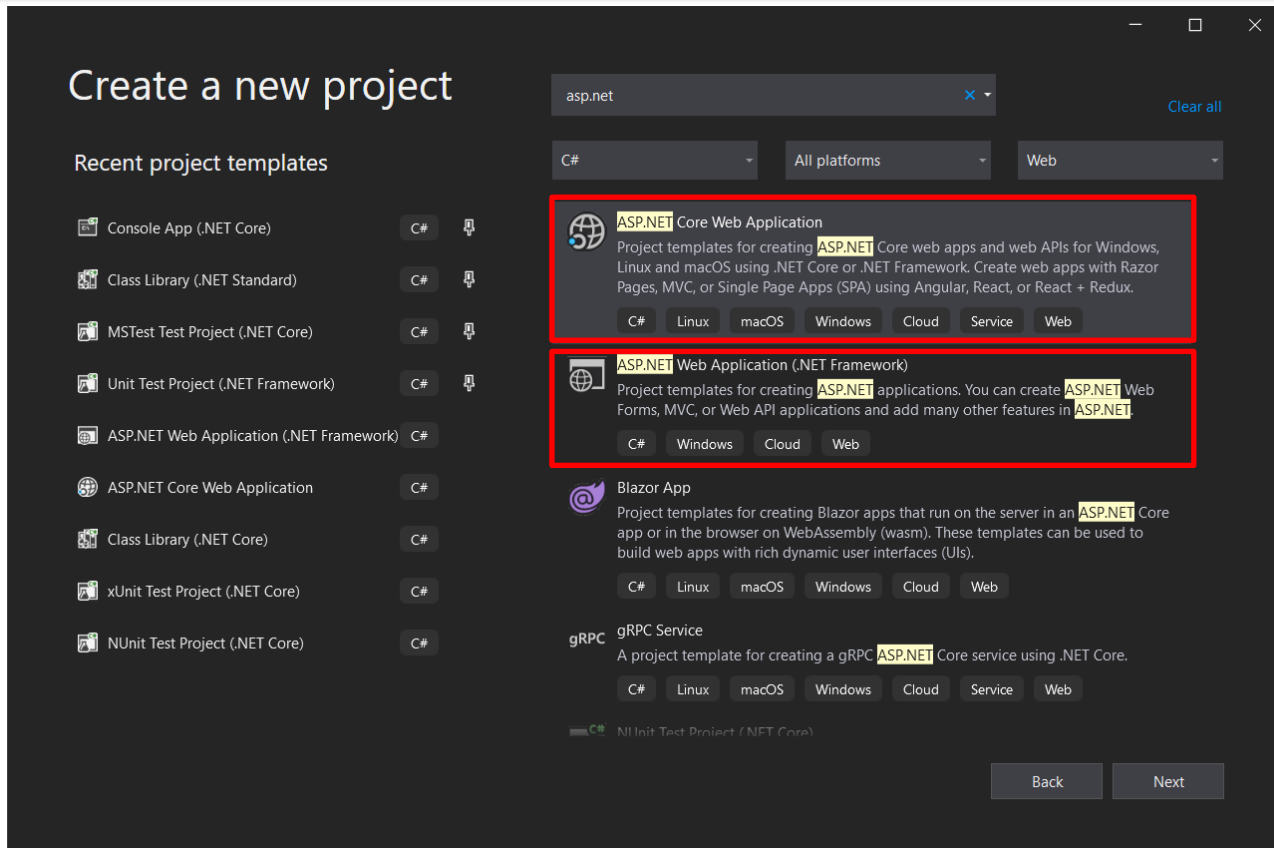
# ASP.NET: Key Benefits of using ASP.NET Technologies

- ✓ ASP.NET is useful for creating dynamic, robust and scalable web applications
- ✓ ASP.NET is the best choice for building applications that are completely secured owing to its built-in Windows authentication and per-application configuration
- ✓ Minimal coding is required to build large applications using ASP.NET
- ✓ This Microsoft framework comes with toolbox and designer in the Visual Studio integrated development environment. Developer-friendly features of ASP.NET such as automatic deployment, WYSIWYG editing and drag-and-drop server controls.
- ✓ All ASP.NET applications handle requests in run time by closely monitoring and managing processes to replace dead ones with new ones.
- ✓ ASP.NET enables performing common tasks from simple form submission, client authentication to site configuration and deployment.
- ✓ ASP.NET technologies is preferred by developers as it is easy to create and maintain because of the existence of source code and HTML.
- ✓ ASP.NET languages can be built to be language-independent. It allows users to choose the language of their choice or divide the application as per different languages.
- ✓ ASP.NET being a server side technology, the code for such applications gets executed on the server.
- ✓ As all the configuration information is built-in, there is no process of registration of components. This allows for easy deployment of applications.
- ✓ ASP.NET applications executing on server are monitored for all pages, components and applications. This helps in detecting memory leaks, infinite loops and other illegal activities.
- ✓ ASP.NET is the best choice for building heavy, complicated web applications as it is developed with ADO.NET using data-binding and page formatting features.

# Configuration



# ASP.NET: Web application types - DEMO



# ASP.NET: Web application types – DEMO (.NET Framework)

Configure your new project

ASP.NET Web Application (.NET Framework) C# Windows Cloud Web

Project name

WebApplication1

Location

C:\Users\Oleksii\_Leunenko\source\repos

Solution name ⓘ

WebApplication1

☐ Place solution and project in the same directory


Framework

.NET Framework 4.7.2


Back Create

# ASP.NET: Web application types – DEMO (.NET Framework)


## Create a new ASP.NET Web Application

**Empty**


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

**Web Forms**


A project template for creating ASP.NET Web Forms applications. ASP.NET Web Forms lets you build dynamic websites using a familiar drag-and-drop, event-driven model. A design surface and hundreds of controls and components let you rapidly build sophisticated, powerful UI-driven sites with data access.

**MVC**

A project template for creating ASP.NET MVC applications. ASP.NET MVC allows you to build applications using the Model-View-Controller architecture. ASP.NET MVC includes many features that enable fast, test-driven development for creating applications that use the latest standards.

**Web API**

A project template for creating RESTful HTTP services that can reach a broad range of clients including browsers and mobile devices.

**Single Page Application**

A project template for creating rich client side JavaScript driven HTML5 applications using ASP.NET Web API. Single Page Applications provide a rich user experience which includes client-side interactions using HTML5, CSS3, and JavaScript.

### Authentication

No Authentication  
[Change](#)

### Add folders & core references

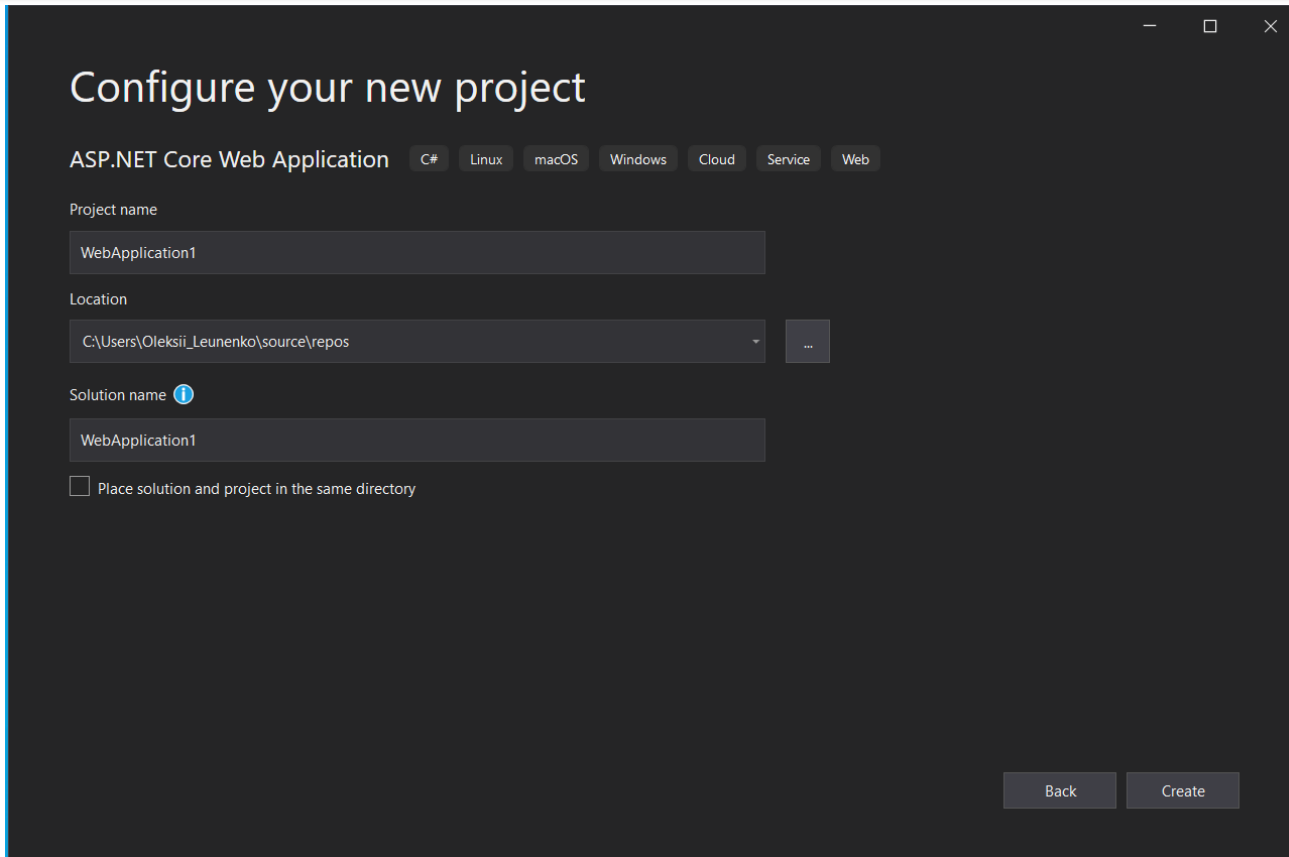
☐ Web Forms  
☒ MVC  
☐ Web API

### Advanced

☒ Configure for HTTPS  
☐ Docker support  
(Requires [Docker Desktop](#))  
☐ Also create a project for unit tests

BackCreate

# ASP.NET: Web application types – DEMO (ASP.NET Core Web Application)



Configure your new project

ASP.NET Core Web Application C# Linux macOS Windows Cloud Service Web

Project name

WebApplication1

Location

C:\Users\Oleksii\_Leunenko\source\repos

Solution name ⓘ

WebApplication1

☐ Place solution and project in the same directory


Back Create

# ASP.NET: Web application types – DEMO (ASP.NET Core Web Application)


## Create a new ASP.NET Core web application

.NET Core


ASP.NET Core 3.1

 **Empty**


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

 **API**


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

 **Web Application**


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

 **Web Application (Model-View-Controller)**

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

 **Angular**

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

 **React.js**

[Get additional project templates](#)

### Authentication

No Authentication

[Change](#)

### Advanced

☒ Configure for HTTPS

☐ Enable Docker Support

(Requires [Docker Desktop](#))

Windows


☐ Enable Razor runtime compilation

Author: Microsoft


Source: Templates 3.1.9

Back


Create

 **Angular**

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

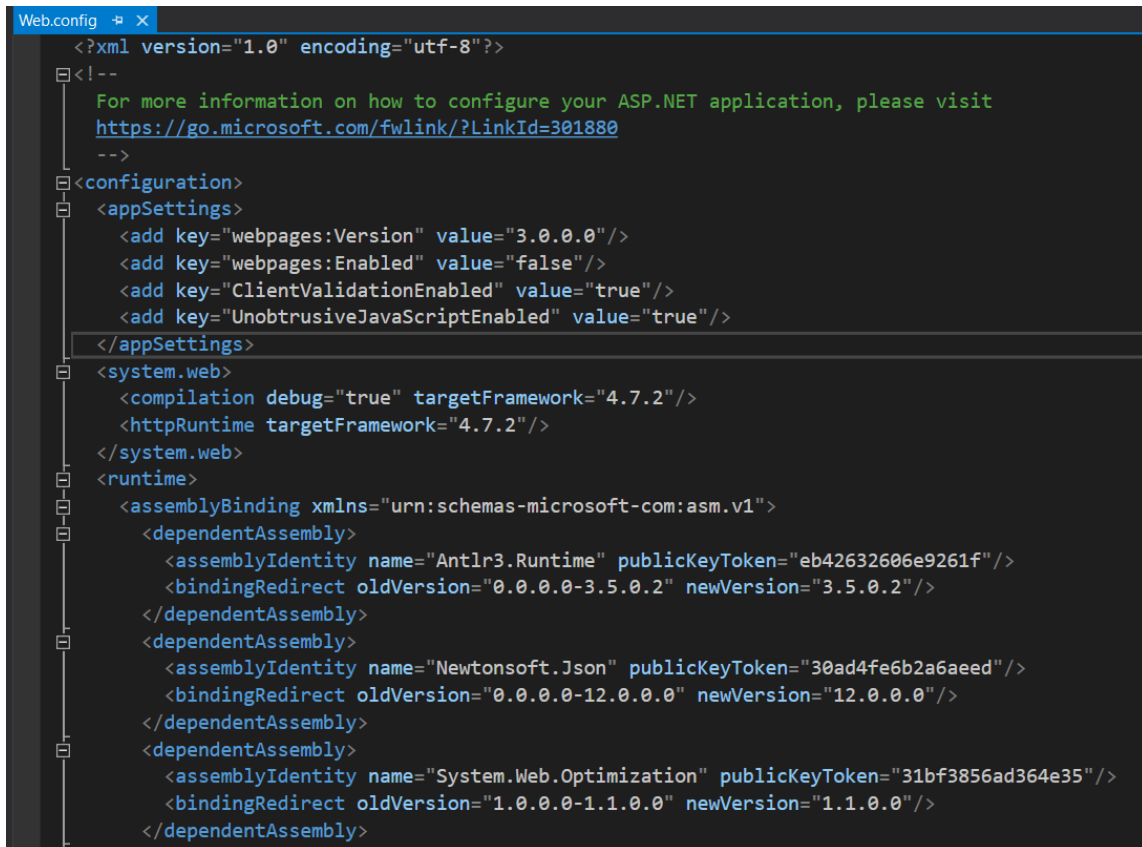
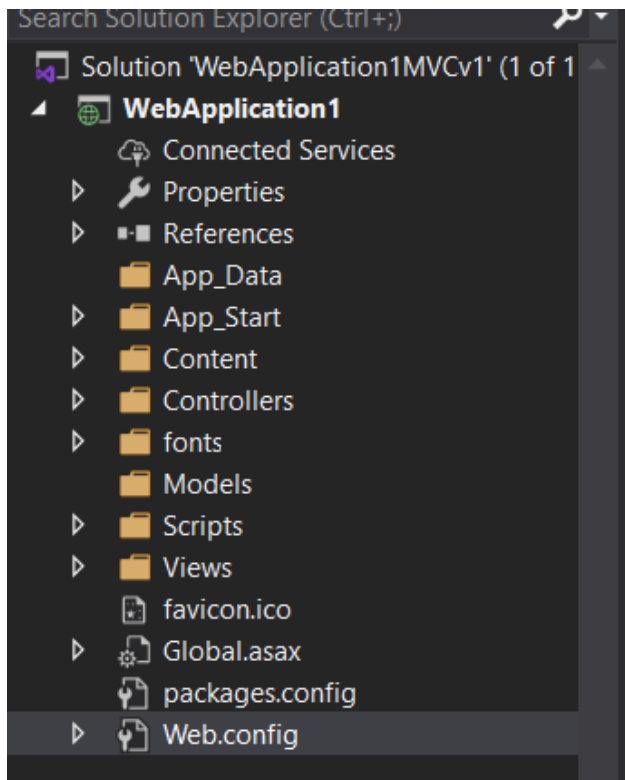
 **React.js**

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

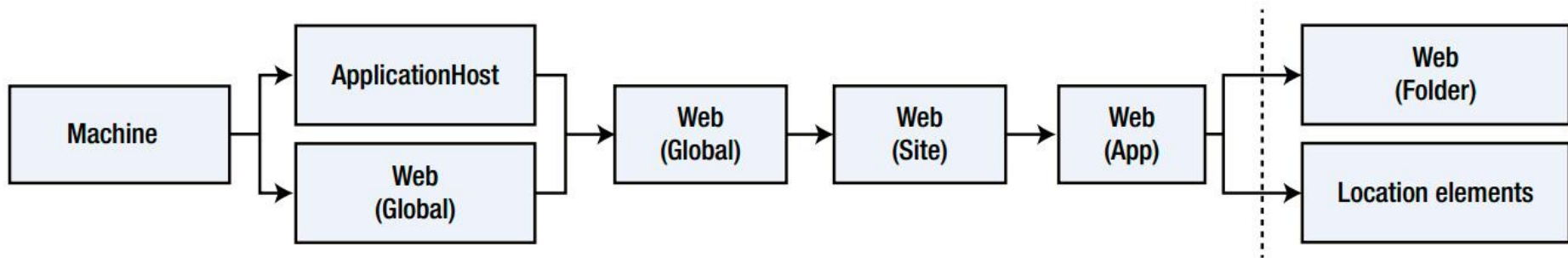
 **React.js and Redux**

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

# Configuration



# Configuration



*Machine.config & Web.config:*

C:\Windows\Microsoft.NET\Framework\<version>\Config\

*ApplicationHost.config:*

C:\Users\<userName>\Documents\IISExpress\config\

# Configuration

```
<?xml version="1.0" encoding="utf-8"?>
<!--
  For more information on how to configure your ASP.NET application, please visit
  https://go.microsoft.com/fwlink/?LinkId=301880
-->
<configuration>
  <appSettings>
    <add key="webpages:Version" value="3.0.0.0"/>
    <add key="webpages:Enabled" value="false"/>
    <add key="ClientValidationEnabled" value="true"/>
    <add key="UnobtrusiveJavaScriptEnabled" value="true"/>
    <add key="city" value="Kyiv"/>
  </appSettings>
  <system.web>
    <compilation debug="true" targetFramework="4.7.2"/>
    <httpRuntime targetFramework="4.7.2"/>
  </system.web>
  <runtime>...</runtime>
  <system.codedom>
    <compilers>
      <compiler language="c#;cs;csharp" extension=".cs"
        type="Microsoft.CodeDom.Providers.DotNetCompilerPlatform.CSharpCodeProvider, Microsoft.CodeDom.Providers.DotNetCompilerPlatform"
        warningLevel="4" compilerOptions="/langversion:default /nowarn:1659;1699;1701"/>
      <compiler language="vb;vbs;visualbasic;vbscript" extension=".vb"
        type="Microsoft.CodeDom.Providers.DotNetCompilerPlatform.VBCodeProvider, Microsoft.CodeDom.Providers.DotNetCompilerPlatform"
        warningLevel="4" compilerOptions="/langversion:default /nowarn:41008 /define:_MYTYPE=\"Web\" /optionInfer+>
    </compilers>
  </system.codedom>
</configuration>
```



# Configuration: WebConfigurationManager class

```
WebApplication1
using System.Linq;
using System.Web;
using System.Web.Configuration;
using System.Web.Mvc;

namespace WebApplication1.Controllers
{
    // 0 references
    public class HomeController : Controller
    {
        // 0 references
        public ActionResult Index()
        {
            WebConfigurationManager
            return View();
        }
    }
}
```

```
System.Web.Configuration.WebConfigurationManager
ConnectionStrings

namespace System.Web.Configuration
{
    public static class WebConfigurationManager
    {
        ... public static ConnectionStringSettingsCollection ConnectionStrings { get; }
        ... public static NameValueCollection.AppSettings { get; }

        ... public static object GetSection(string sectionName);
        ... public static object GetSection(string sectionName, string path);
        ... public static object GetWebApplicationSection(string sectionName);
        ... public static System.Configuration.Configuration OpenMachineConfiguration();
        ... public static System.Configuration.Configuration OpenMachineConfiguration(string locationSubPath);
        ... public static System.Configuration.Configuration OpenMachineConfiguration(string locationSubPath, string server);
        ... public static System.Configuration.Configuration OpenMachineConfiguration(string locationSubPath, string server,
        ... public static System.Configuration.Configuration OpenMachineConfiguration(string locationSubPath, string server,
        ... public static System.Configuration.Configuration OpenMachineConfiguration(string locationSubPath, string server,
        ... public static System.Configuration.Configuration OpenMappedMachineConfiguration(ConfigurationFileMap fileMap);
        ... public static System.Configuration.Configuration OpenMappedMachineConfiguration(ConfigurationFileMap fileMap, str
        ... public static System.Configuration.Configuration OpenMappedWebConfiguration(WebConfigurationFileMap fileMap, str
        ... public static System.Configuration.Configuration OpenMappedWebConfiguration(WebConfigurationFileMap fileMap, str
        ... public static System.Configuration.Configuration OpenWebConfiguration(string path, string site);
        ... public static System.Configuration.Configuration OpenWebConfiguration(string path, string site, string locationS
        ... public static System.Configuration.Configuration OpenWebConfiguration(string path, string site, string locationS
        ... public static System.Configuration.Configuration OpenWebConfiguration(string path, string site, string locationS
        ... public static System.Configuration.Configuration OpenWebConfiguration(string path, string site, string locationS
        ... public static System.Configuration.Configuration OpenWebConfiguration(string path);
    }
}
```

# Configuration: DEMO

```
<configuration>
  <appSettings>
    <add key="webpages:Version" value="3.0.0.0"/>
    <add key="webpages:Enabled" value="false"/>
    <add key="ClientValidationEnabled" value="true"/>
    <add key="UnobtrusiveJavaScriptEnabled" value="true"/>
    <add key="defaultCity" value="Kyiv"/>
    <add key="defaultUtcTimeZoneShift" value="2"/>
  </appSettings>
```

# Configuration: DEMO

```
public class HomeController : Controller
{
    public ActionResult Index()
    {
        var settings = WebConfigurationManager.GetWebApplicationSection("appSettings");
        // or
        NameValueCollection config = WebConfigurationManager.AppSettings;
        return View(config);
    }
}
```

@model System.Collections.Specialized.NameValueCollection

<div>

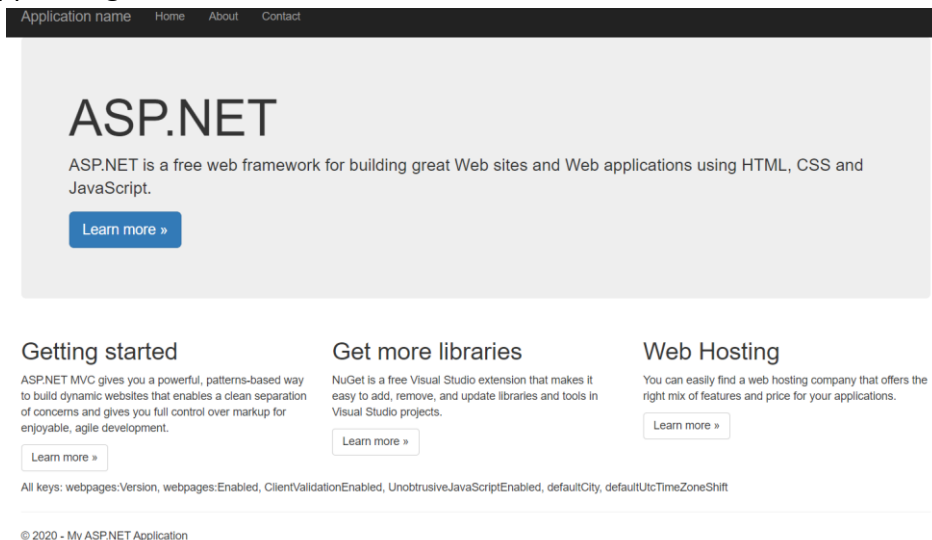
All keys: @string.Join(" ", Model.AllKeys)

</div>

<br />

<div>Total items: @Model.Count</div>

<br />



# Configuration: DEMO

```
public class HomeController : Controller
{
    public ActionResult Index()
    {
        var settings = WebConfigurationManager.GetWebApplicationSection("appSettings");
        // or
        NameValueCollection config = WebConfigurationManager.AppSettings;

        foreach (string key in config)
        {
            System.Diagnostics.Debug.WriteLine(string.Format("{0} - {1}", key, config[key]));
        }

        return View(config);
    }
}
```

```
<div>
    @foreach (string key in Model)
    {
        <div>
            @string.Format("{0} - {1}", key, Model[key])
        </div>
    }
</div>
```

# Configuration: DEMO

```
<configuration>
  <appSettings>
    <add key="webpages:Version" value="3.0.0.0"/>
    <add key="webpages:Enabled" value="false"/>
    <add key="ClientValidationEnabled" value="true"/>
    <add key="UnobtrusiveJavaScriptEnabled" value="true"/>
    <add key="defaultCity" value="Kyiv"/>
    <add key="defaultUtcTimeZoneShift" value="2"/>
  </appSettings>
```

```
<userCustomSettings>
  <defaultValues email="admin@admin.com" city="Kyiv" timeZoneShift="32" />
</userCustomSettings>
```

# Configuration: DEMO

```
namespace WebApplication1.ServiceCode
{
    public class DefaultValuesConfigSection : ConfigurationSection
    {
        [ConfigurationProperty("email")]
        public string Email
        {
            get { return (string)this["email"]; }
            set { this["email"] = value; }
        }

        [ConfigurationProperty("city")]
        public string City
        {
            get { return (string)this["city"]; }
            set { this["city"] = value; }
        }

        [ConfigurationProperty("timeZoneShift")]
        public int TimeZoneShift
        {
            get { return (int)this["timeZoneShift"]; }
            set { this["timeZoneShift"] = value; }
        }
    }
}
```

```
namespace WebApplication1.ServiceCode
{
    public class UserCustomSettingsConfigGroup : ConfigurationSectionGroup
    {
        public DefaultValuesConfigSection DefaultValues
        {
            get { return (DefaultValuesConfigSection)Sections["defaultValues"]; }
        }
    }
}
```

```
<configSections>
  <sectionGroup name="userCustomSettings"
    type="WebConfigPractice.ServiceCode.UserCustomSettingsConfigGroup">
    <section name="defaultValues"
      type="WebConfigPractice.ServiceCode.DefaultValuesConfigSection"/>
  </sectionGroup>
</configSections>
```

# Configuration: DEMO

```
public ActionResult CustomConfig()
{
    WebConfigurationManager.AppSettings["defaultCity"] = "New York";
    UserCustomSettingsConfigGroup model =
        (UserCustomSettingsConfigGroup)WebConfigurationManager.OpenWebCo
    return View(model);
}
```

```
namespace WebApplication1.ServiceCode
{
    public class DefaultValuesConfigSection : ConfigurationSection
    {
        [ConfigurationProperty("email")]
        public string Email
        //...

        [ConfigurationProperty("city", IsRequired = true)]
        //...

        [ConfigurationProperty("timeZoneShift", DefaultValue = 5)]
        public int TimeZoneShift
        //...
    }
}
```

```
@model WebConfigPractice.ServiceCode.UserCustomSettingsConfigGroup
@{
    Layout = null;
}
```

```
<!DOCTYPE html>").GetSectionGroup("userCustomSettings");
```

```
<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>CustomConfig</title>
    <style>
        html {
            font-size: 2em;
        }
    </style>
</head>
<body>
    <div>
        Email: @Model.DefaultValues.Email
    </div>
    <div>
        City: @Model.DefaultValues.City
    </div>
    <div>
        Time zona: @Model.DefaultValues.TimeZoneShift
    </div>
</body>
</html>
```

# Life cycle



# Life cycle

---

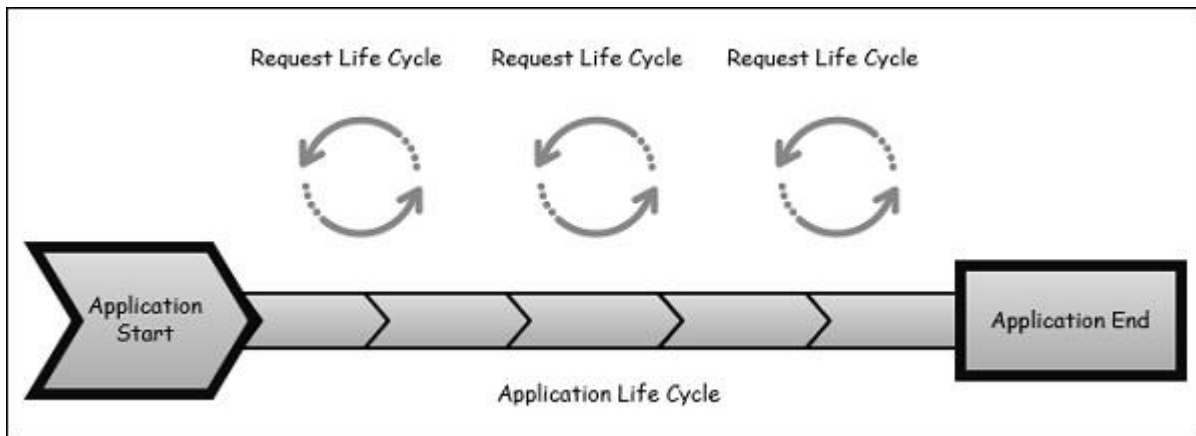
**Life cycle** is simply a series of steps or events used to handle some type of request or to change an application state.

**MVC has two life cycles:**

- The application life cycle
- The request life cycle

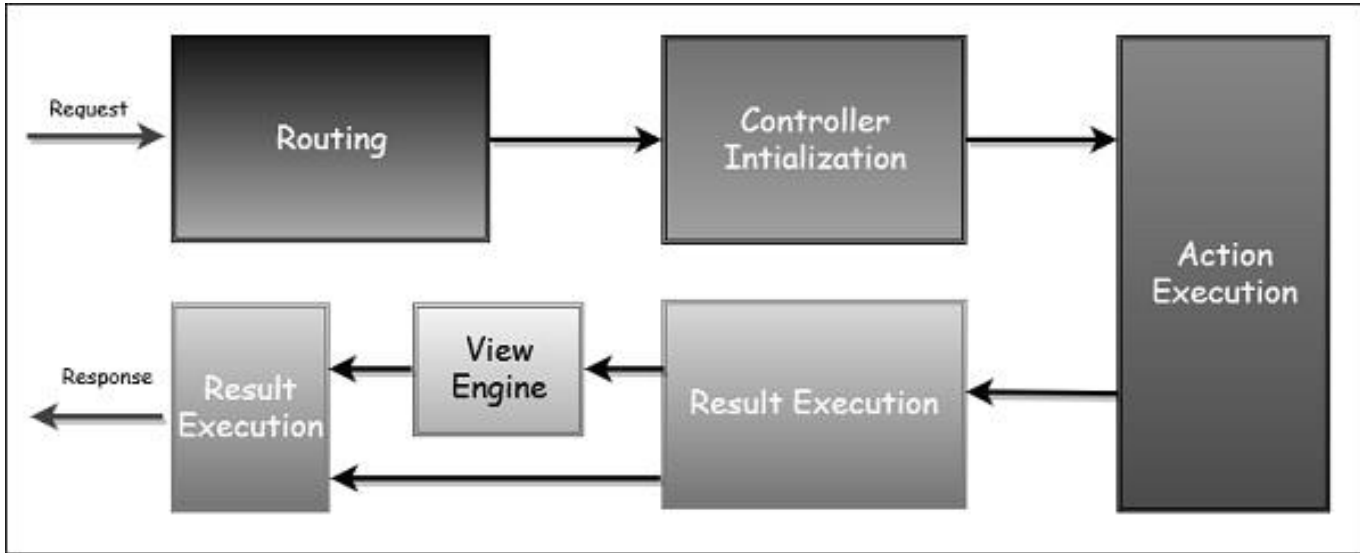
# Life cycle: The Application Life Cycle

The application life cycle refers to the time at which the application process actually begins running IIS until the time it stops. This is marked by the application start and end events in the startup file of your application.



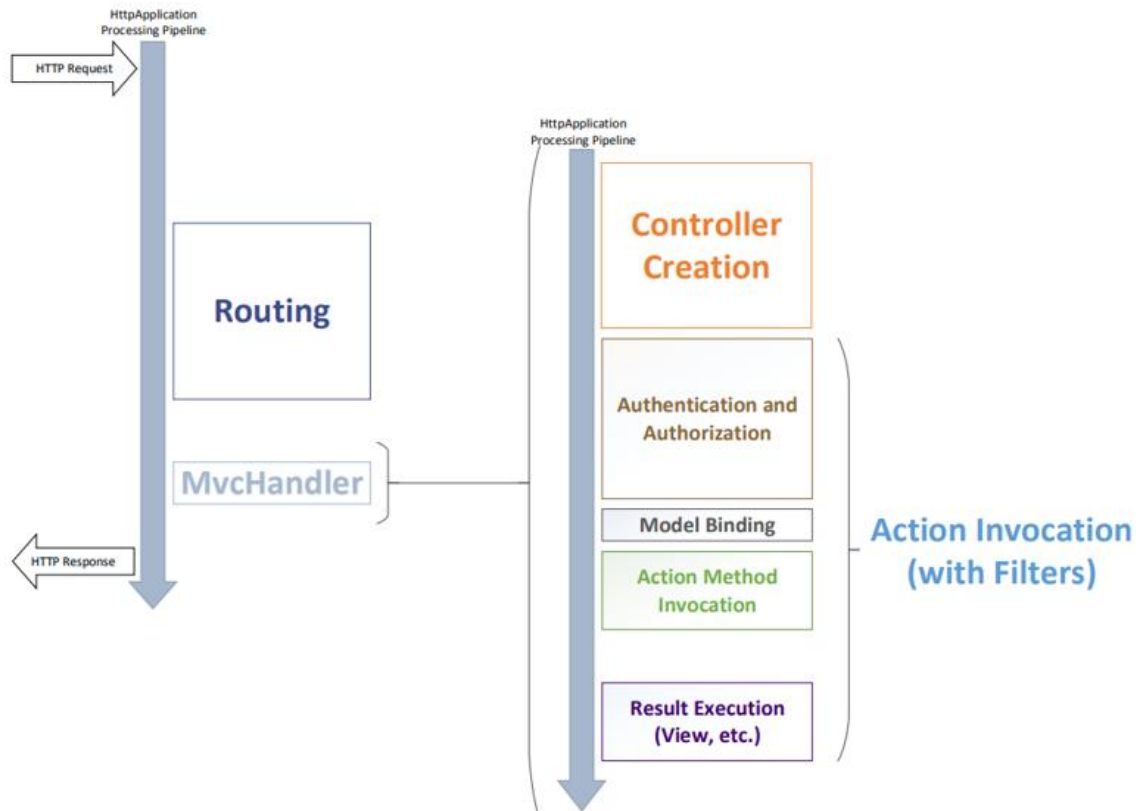
# Life cycle: The Request Life Cycle

The **Request Life Cycle** is the sequence of events that happen every time an HTTP request is handled by our application.



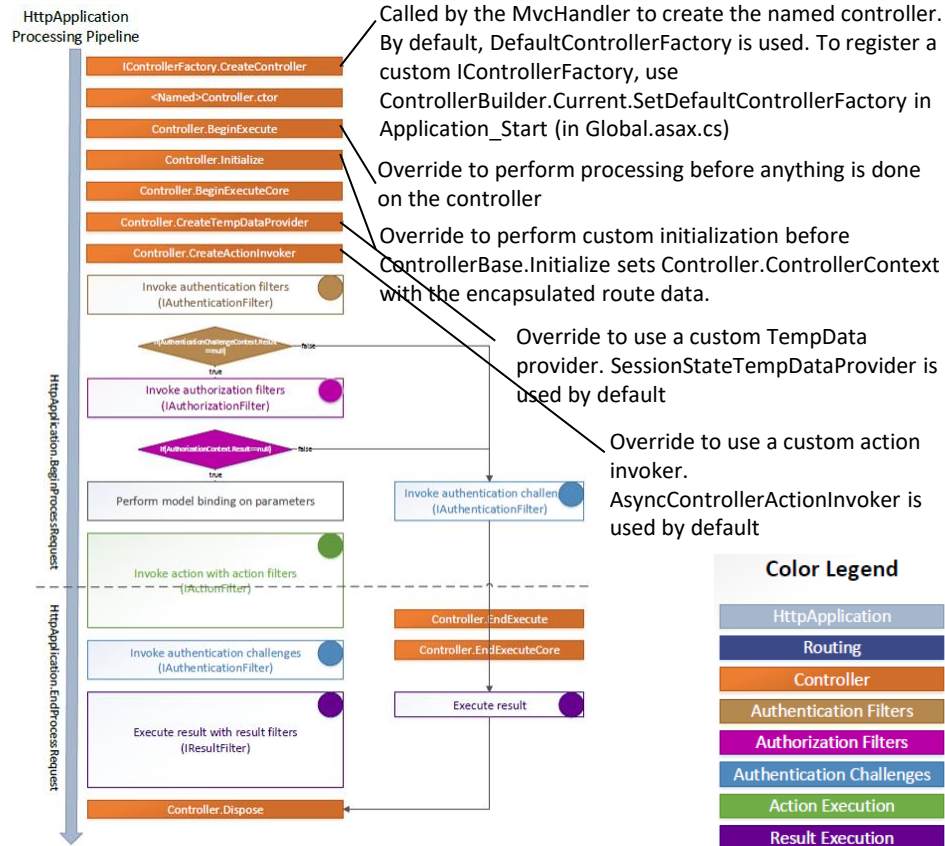
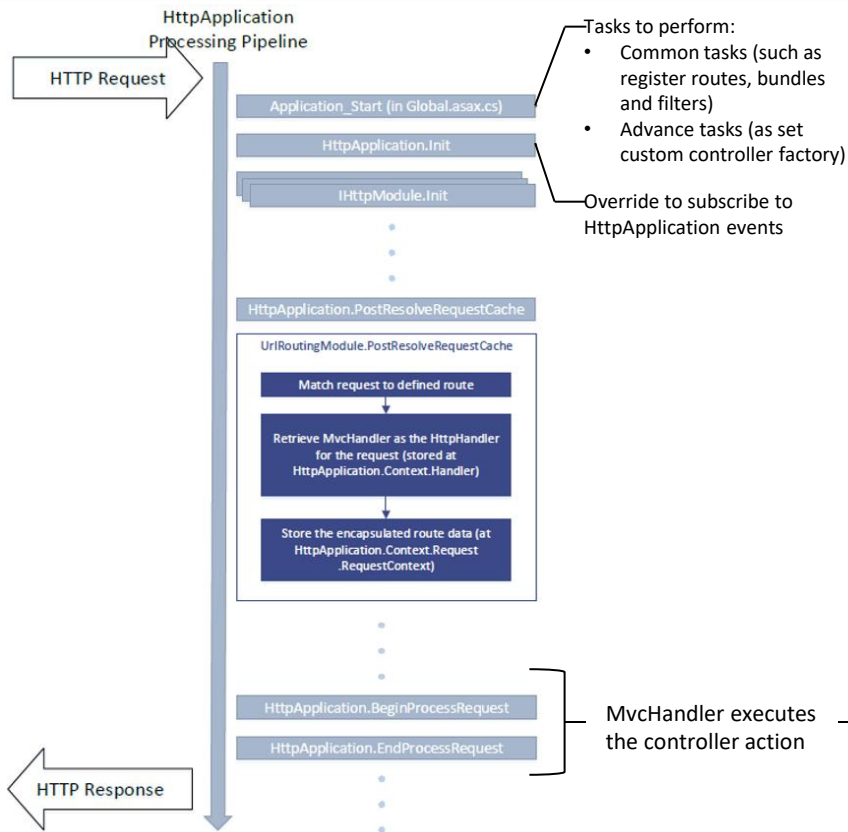
**Modules** are .NET components that can hook into the application life cycle and add functionality.

# Life cycle: – High-level view

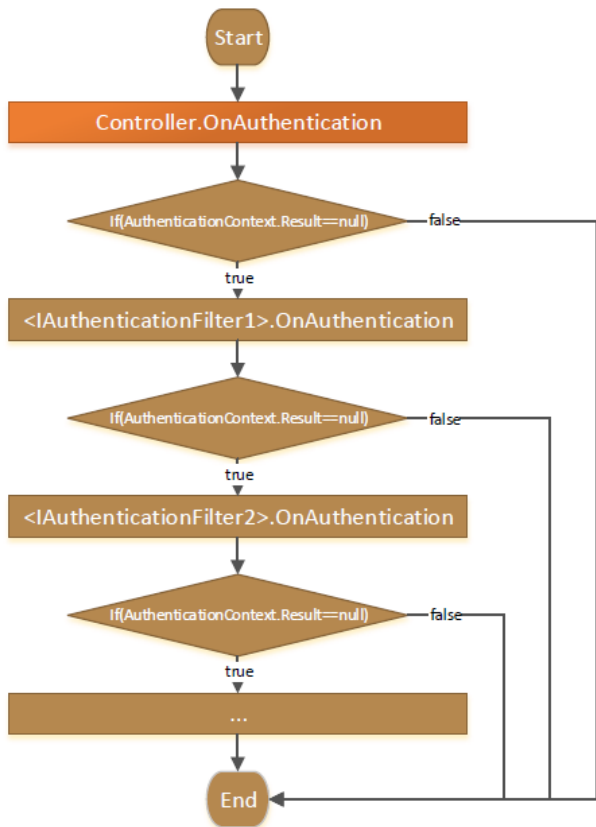


<https://docs.microsoft.com/en-us/aspnet/mvc/overview/getting-started/lifecycle-of-an-aspnet-mvc-5-application>

# Life cycle: Detail view



# Life cycle: Invoke authentication filters (IAuthenticationFilter)

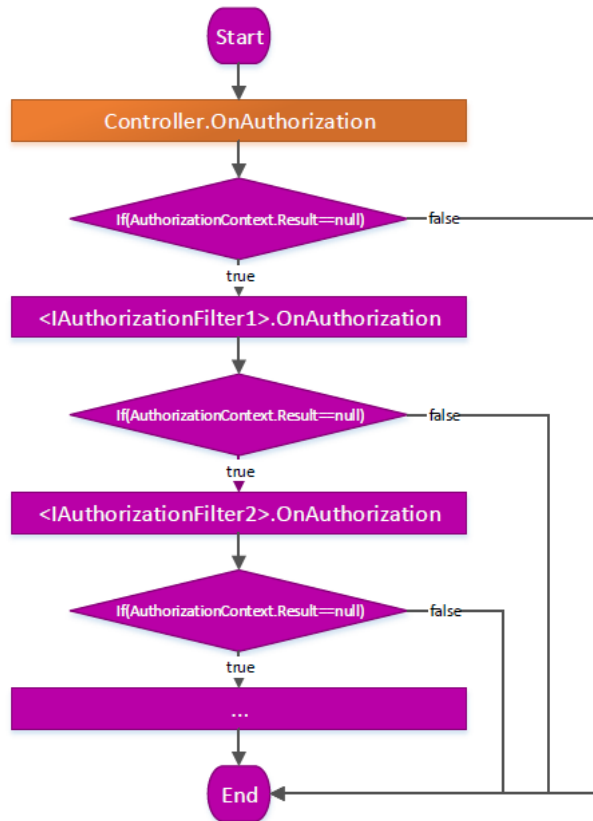


Use [IAuthenticationFilter](#) to authenticate a user action toward the intended resources.

Authentication filters execute in order until one of the filters returns a non-null result.

**Authentication** is where a user provides credentials to access a resource, whereas **authorization** allows access to particular resources based on properties of the user's identity.

# Life cycle: Invoke authorization filters (IAuthorizationFilter)

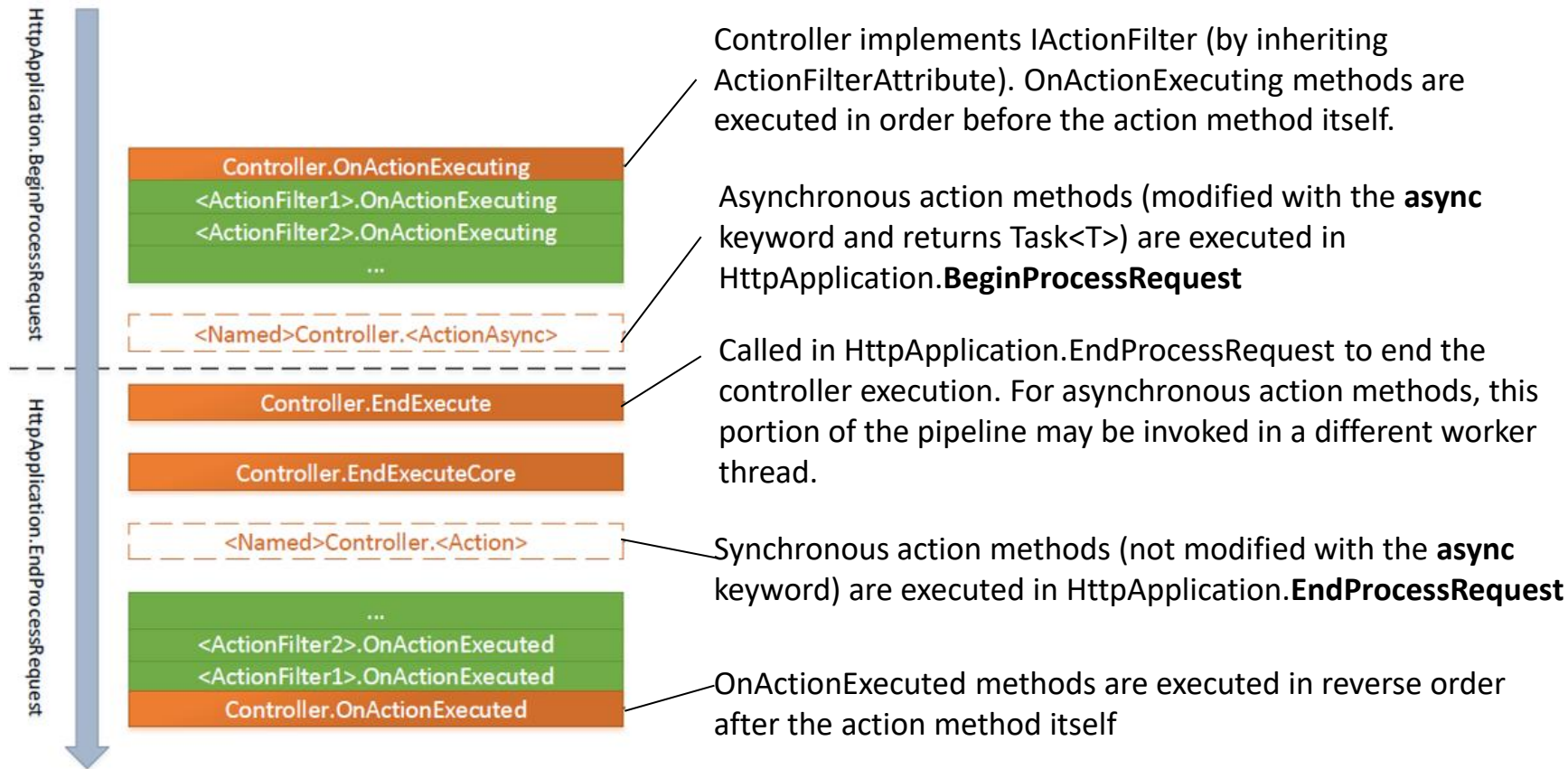


Use [IAuthorizationFilters](#) to authorize a user action toward the intended resources.

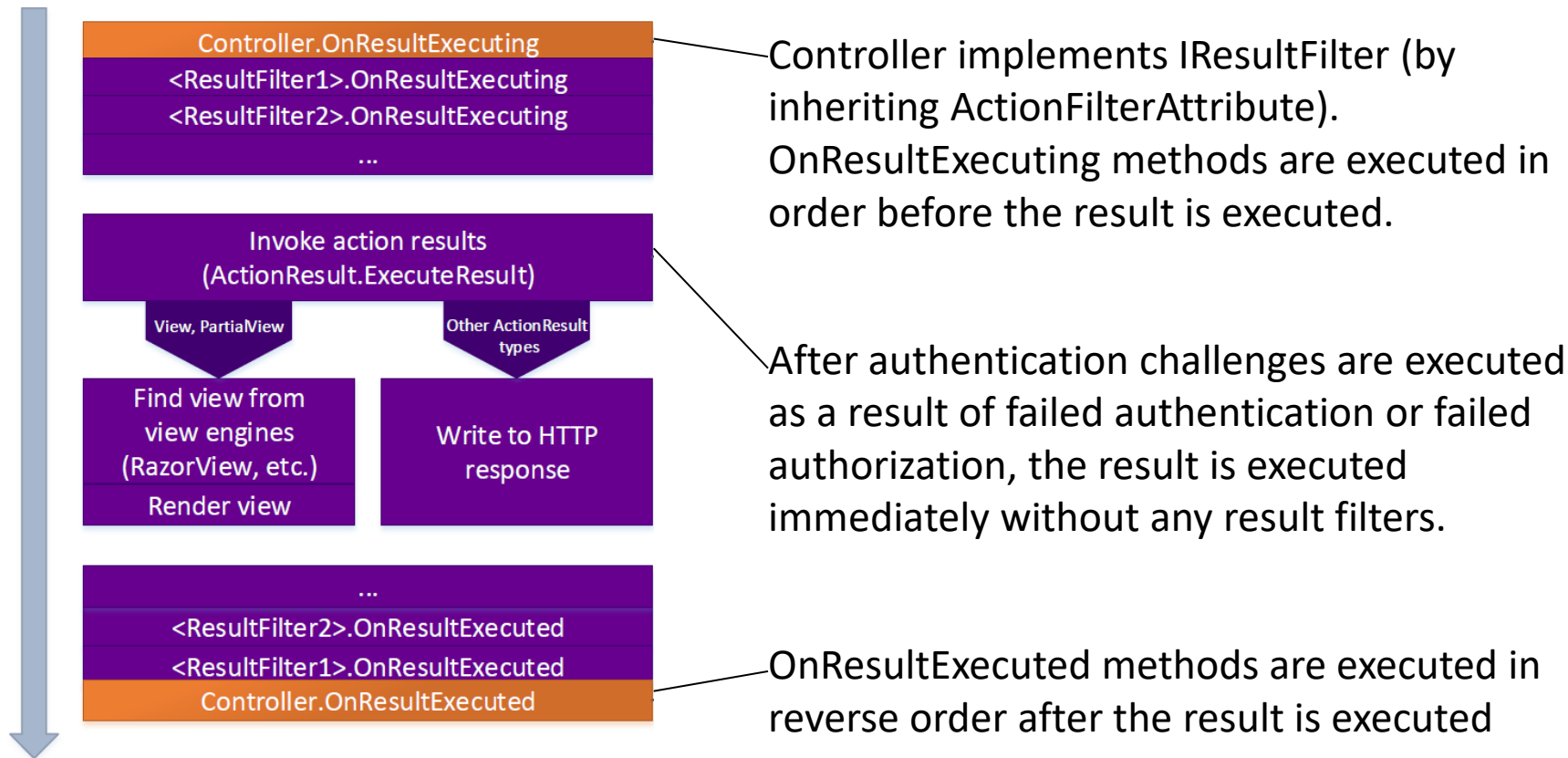
Authorization filters execute in order until one of the filters returns a non-null result



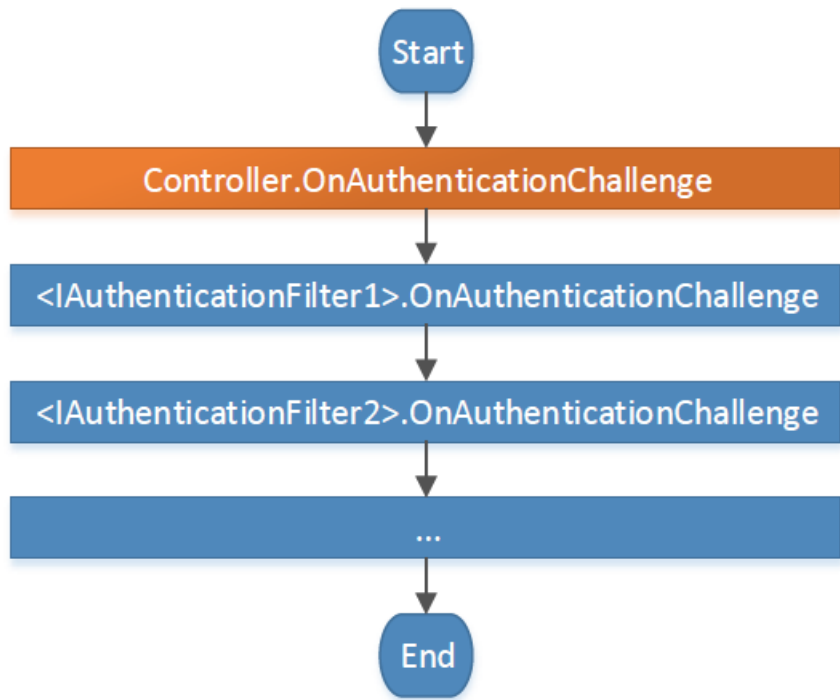
# Life cycle: Execute action method with action filters (IActionFilter)



# Life cycle: Execute results with result filters (IResultFilter)



## Life cycle: Invoke authentication challenges (IAuthenticationFilter)



StartEndAuthentication challenges are invoked whenever authentication or authorization filters return a result to indicate failure. They are also invoked after the action method executes in case the authentication scheme requires it, such as for server authentication to the client. Use `OnAuthenticationChallenge` to create the `ActionResult` you intend to send to the client when authentication challenges occur. When authentication challenges are invoked, all registered `IAuthenticationFilter` contribute to the result.

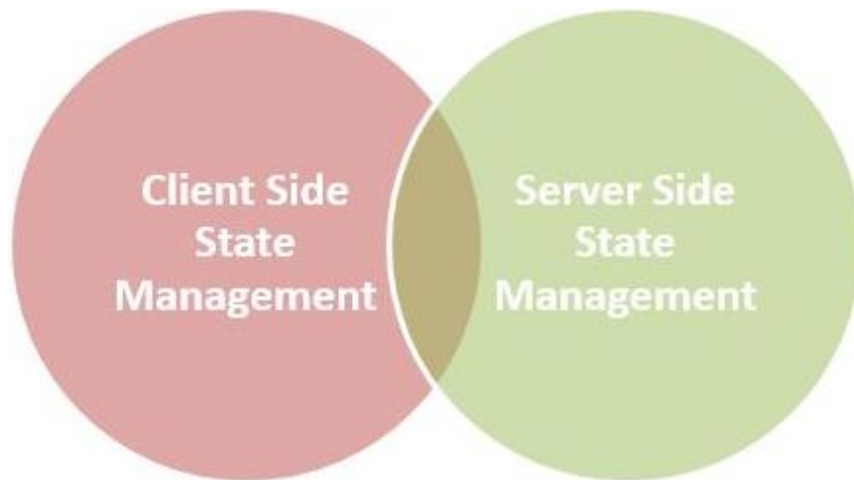
# Client state

# State management

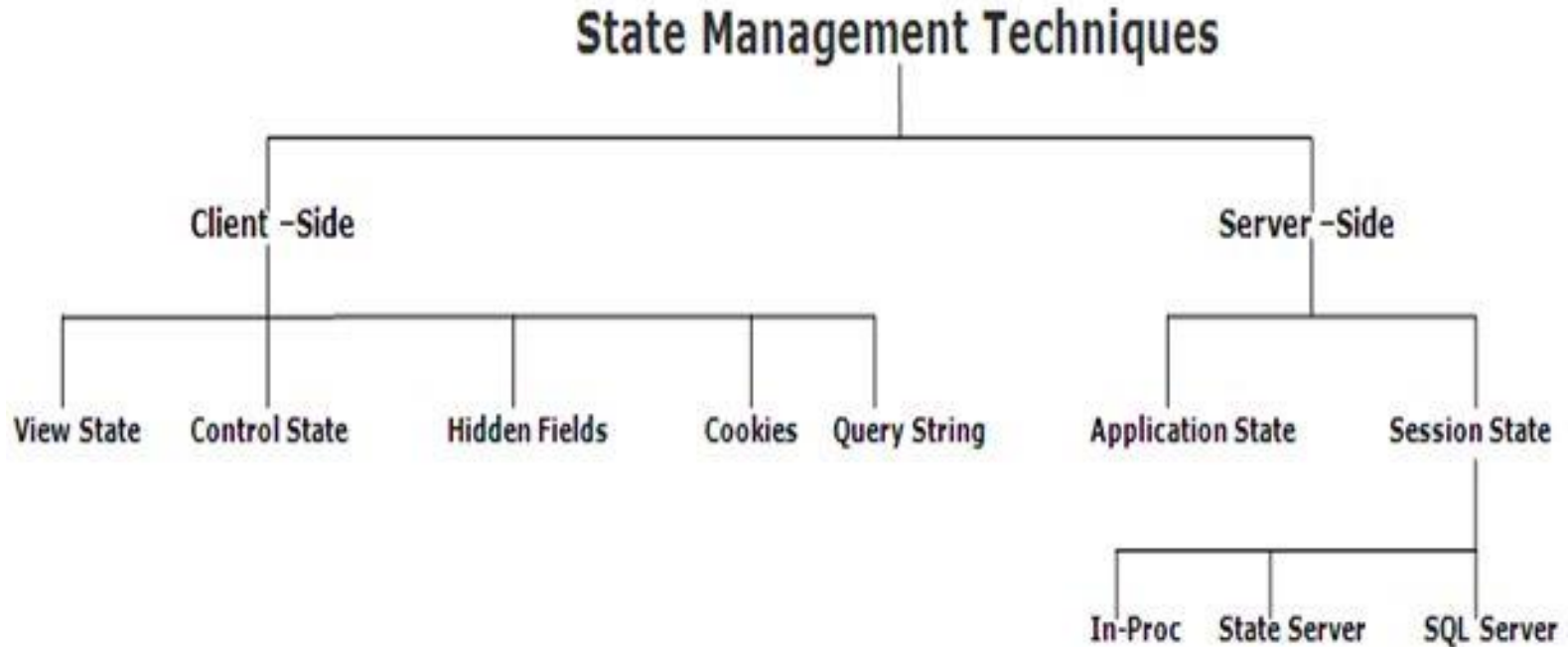
**State management** is the process by which developers can maintain state and page information over multiple request for the same or different pages in web application.

## Types of state management

- Client side state management
- Server side state management



# State Management Techniques



# Client state

In client side page management the information is stored on client system. This information will travel back and forth with every request and response to and from server.

## Advantage

The major advantages of having this kind of state management is that it saves a lot of server memory. We relieve server from the burden of keeping the state related information.

## Disadvantage

- It takes more bandwidth as considerable amount of data is traveling back and forth. Due to this, web page becomes slow to load.
- The main draw back is it creates security issue for sensitive information like passwords, credit card numbers etc.



# Client state

---

ASP.NET provides following types of client side methods to manage state in web applications.

- Hidden Field
- View State ( Not support in MVC )
- Cookies
- Control State ( Not support in MVC )
- Query Strings
  
- View Data ( only in MVC )
  
- View Bag ( only in MVC )
  
- Temp Data ( only in MVC )

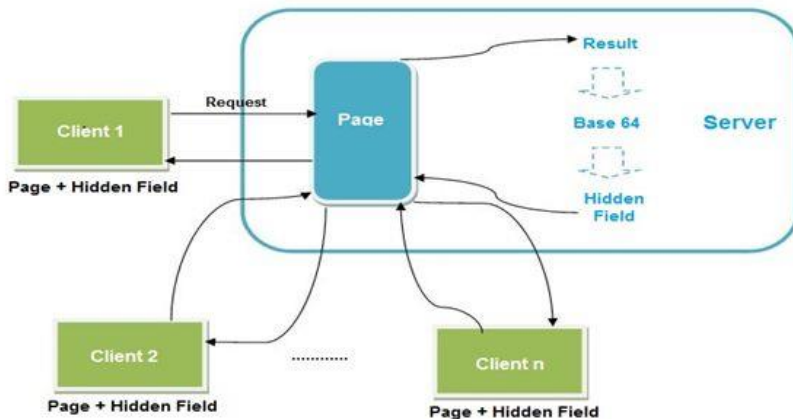


# Client state: Hidden field

Hidden field is used to store small amounts of data on the client system. It is a preferable way when a variable's value is changed frequently. The only constraint on hidden field is that it will keep the information when HTTP post is being done. It will not work with HTTP get.

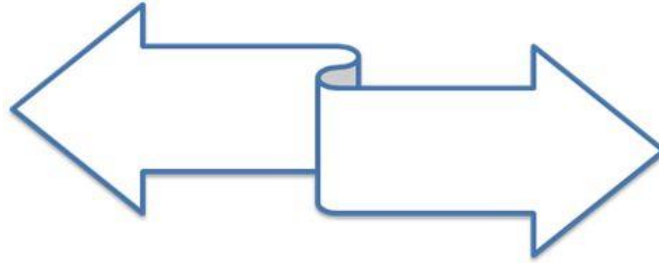
Mvc5DemoProject.Models.User

```
<div class="display-field">  
    @Html.HiddenFor(m => m.Id)  
</div>
```



# Client state: Cookies

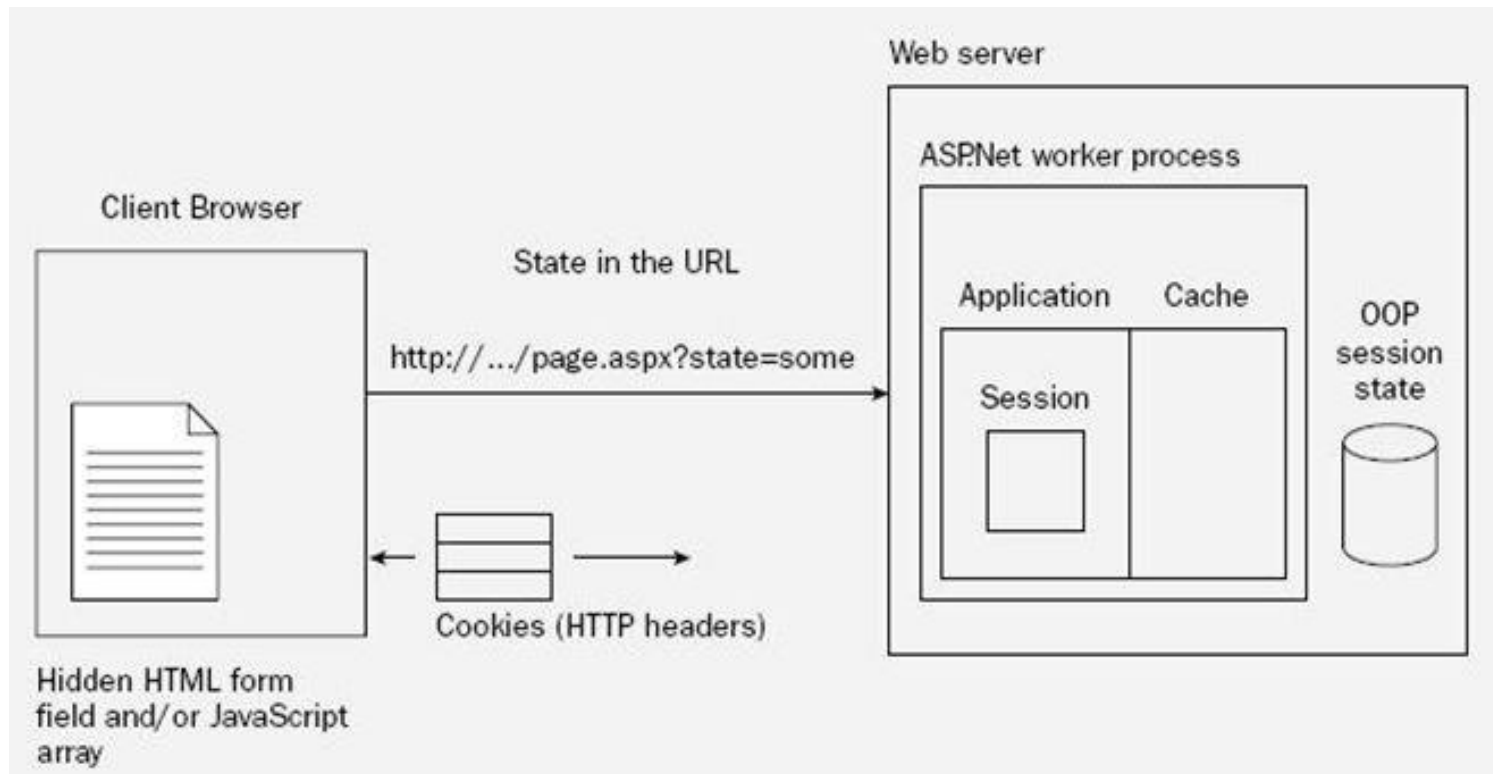
**Cookie** is a small text file which is created by the server and stored on the client hard disk by the browser. It does not use server memory.



Cookies are two types:

- **Persistence:** cookies are permanently stored till the time we set.
- **Non-Persistence:** cookies are not permanently stored on the user's system. When user closes the browser the cookie will be discarded.

# Client state: Cookies



# Client state

```
public ActionResult NonPersistenceCookie()
```

```
{
```

```
    HttpCookie cookie = new HttpCookie("MyCookie");
```

```
    cookie.Value = "Hello Cookie! CreatedOn: " + DateTime.Now.ToShortTimeString();
```

```
    this.ControllerContext.HttpContext.Response.Cookies.Add(cookie);
```

```
    return RedirectToAction("Index", "Home");
```

```
}
```

```
public ActionResult PersistenceCookie()
```

```
{
```

```
    if(this.ControllerContext.HttpContext.Request.Cookies.AllKeys.Contains("MyCookie"))
```

```
    {
```

```
        HttpCookie cookie = this.ControllerContext.HttpContext.Request.Cookies["MyCookie"];
```

```
        cookie.Expires = DateTime.Now.AddSeconds(10);
```

```
        this.ControllerContext.HttpContext.Response.Cookies.Add(cookie);
```

```
    }
```

```
    return RedirectToAction("Index", "Home");
```

```
}
```

size of a cookie is not  
more than 4 KB of data

# Client state: Query Strings

A Query String is a string variable which is appended to the end of the Page URL. It can be used to send data across pages.

It stores information in a key/value pair.

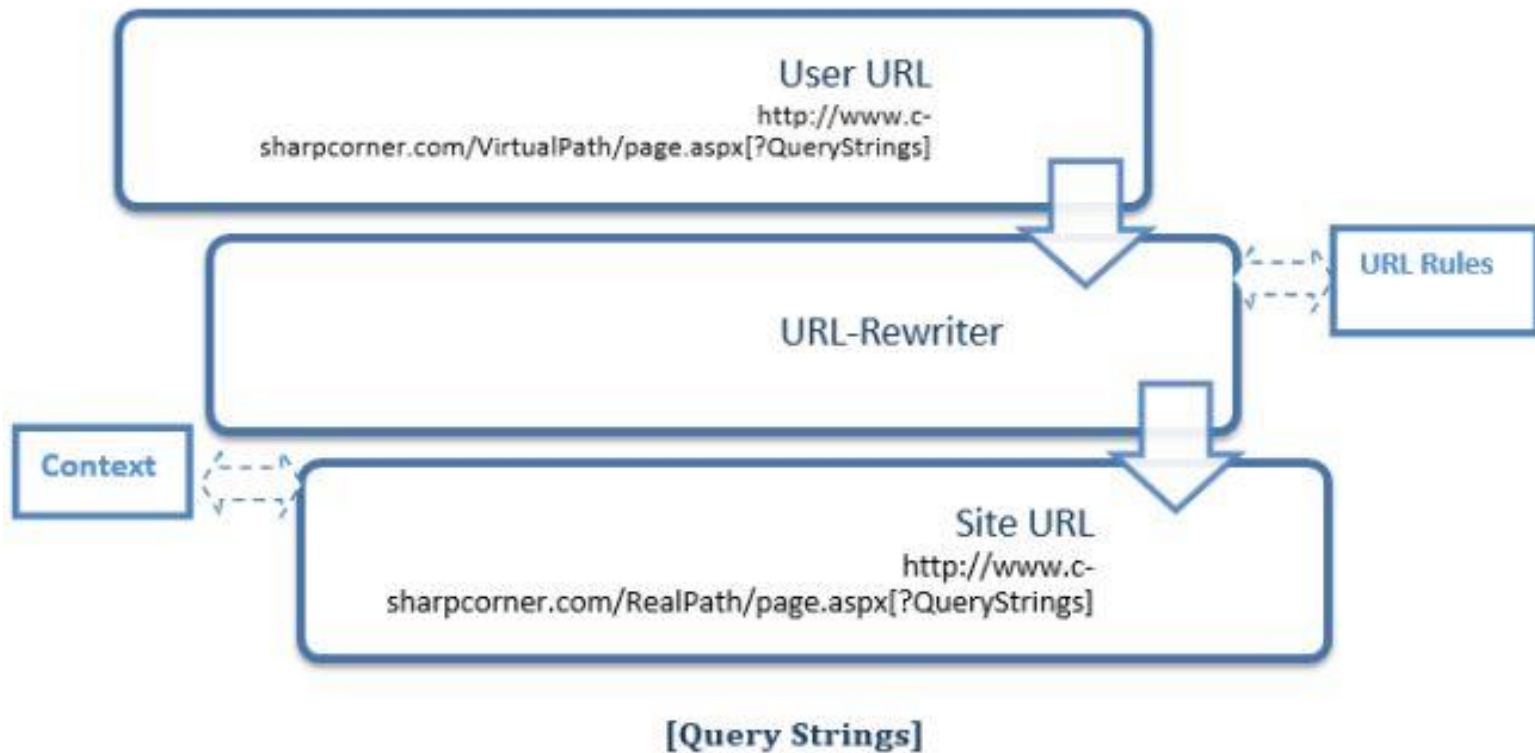
A “?” signature is used to append the key and value to the page URL.

<http://MyDomain/product/Edit/1?name=Mobile>

```
public ActionResult Edit(int id, string name)
{
    //To Do
    return View();
}
```

**Note:** Most browsers impose a limit of 255 characters on URL length. We should encrypt query values. Most browsers impose a limit of 255 characters on URL length. We should encrypt query values.

# Client state: Query Strings



# Client state: ViewData

**ViewData** is a dictionary object which is derived from **ViewDataDictionary** class. It will be accessible using strings as keys like

```
public ActionResult Index()
{
    ViewData["hasPermission"] = true;
    return View();
}
```

ViewData lies only during the current request from controller to respective view . If redirection occurs then it's value becomes null. It required typecasting for getting data on view.

```
@{
    bool hasPermission = (bool)ViewData["hasPermission"];
}
```

# Client state: ViewBag

**ViewBag** object is a dynamic property which is wrapped around the ViewData object. **Dynamic property** is a feature in ASP.NET Dynamic Language. We can simply set properties on the dynamic ViewBag property within controller.

```
public ActionResult Index()
{
    ViewBag.HasPermission = true;
    return View();
}
```

It also lies only during the current request from controller to respective view as ViewData. If redirection occurs it's value becomes null. It does not required typecasting for getting data on view.

```
@if(ViewBag.HasPermission)
{
    //do somethings....
}
```



# Client state: TempData

**TempData** is a dictionary object which stores data as key/value pair and derived from TempDataDictionary class. TempData helps to maintain data when we move from one controller to other controller or from one action to other action.

RedirectToAction has no impact over the TempData until TempData is read. Once TempData is read its values will be lost.

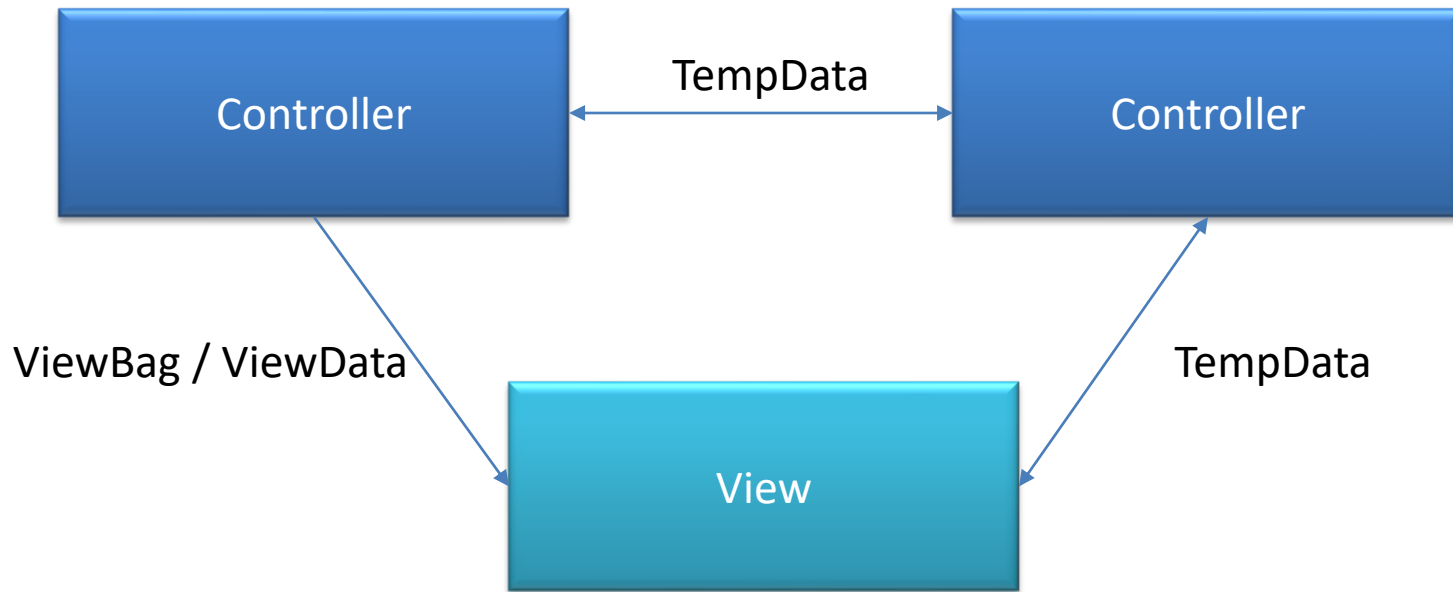
```
public class ViewController : Controller
{
    public ActionResult Index()
    {
        TempData["MyData"] = "Hello! We come";
        return RedirectToAction("About");
    }

    public ActionResult About()
    {
        return RedirectToAction("Test1");
    }
}
```

```
public ActionResult Test1()
{
    string Str = Convert.ToString(TempData["MyData"]);
    TempData.Keep(); // Keep TempData
    return RedirectToAction("Test2");
}

public ActionResult Test2()
{
    string Str = Convert.ToString(TempData["MyData"]); //OutPut
    return View();
}
}
```

# Client state: Flow Diagram



# Server state

# Server state

In server side state management we keep all the information in server memory.

## Advantage

The major advantages of having this kind of state management is that it secure user's confidential and sensitive information.

## Disadvantage

The downside of this is it usage more server memory.



# Server state

---

ASP.NET provides following types of Server side methods to manage state in web applications.

- Session state
- Application state ( Not support in MVC )
- Profile Properties
  
- Cache

# Server state: Session

In ASP.NET MVC Session manages to store and retrieve values for a user when user navigates between views.

The server maintains the state of user information by using a session ID.

```
// Model
namespace MvcSessionApp.Models
{
    public class UserAccount
    {
        public string UserName { get; set; }
        public string UserType { get; set; }
        public string Email { get; set; }
    }
}

//Controller
namespace MvcSessionApp.Controllers
{
    public class UserAccountController : Controller
    {
        //
        // GET: /UserAccount/
        public ActionResult Index()
        {
            UserAccount ucObj = new UserAccount();
            ucObj.UserName = "Biswa";
            ucObj.UserType = "Admin";
            ucObj.Email = "biswa@session.com";
            Session["AdminUser"] = ucObj;
            return View();
        }
    }
}

//View
@model MvcSessionApp.Models.UserAccount
@{
    ViewBag.Title = "Index";
}
@{
    var user = (MvcSessionApp.Models.UserAccount)Session["AdminUser"];
}
<h2>Index</h2>
<fieldset>
    <legend>UserAccount</legend>
    <div class="display-label">
        @Html.DisplayNameFor(model => model.UserName) : @user.UserName
    </div>
    <div class="display-label">
        @Html.DisplayNameFor(model => model.UserType) : @user.UserType
    </div>
    <div class="display-label">
        @Html.DisplayNameFor(model => model.Email) : @user.Email
    </div>
</fieldset>
```

# Server state: Profile properties

ASP.NET provides profile properties, which allows us to store user customized data as per user's convenient appearance. It is similar to session state, except profile data is not lost when a user's session expires. The profile properties feature uses to store in a persistent format and associated with an individual user.

```
<profile defaultProvider="DefaultProfileProvider">
  <providers>
    <add name="DefaultProfileProvider" type="System.Web.Providers.DefaultProfileProvider, System.Web.Providers, Version=1.0.0.0,
Culture=neutral, PublicKeyToken=31bf3856ad364e35" connectionStringName="DefaultConnection" applicationName="/" />
  </providers>
</profile>
```

# Server state: Caching

---

Caching provides a way of storing frequently accessed data and reusing that data. The output cache enables us to cache the content returned by a controller action.

The ***OutputCache*** filter allow to cache the data that is output of an action method. By default, this attribute filter cache the data till 60 seconds.

```
[OutputCache(Duration=10, VaryByParam="none")]  
public ActionResult Index()  
{  
    return View();  
}
```



# Server state: OutputCache Filter Parameters

| Parameter             | Description  |
|-----------------------|--|
| CacheProfile          | Specify the name of the output cache policy which is defined with in <outputCacheSettings> tag of Web.config.  |
| Duration              | Specify the time in sec to cache the content.  |
| Location              | Specify the location of the output to be cached. Values: <b>Any, Client,Downstream, Server, None, or ServerAndClient</b> . By default location is Any.             |
| NoStore               | This is used only to protect very sensitive data. Enable/Disable where to use HTTP Cache-Control.  |
| SqlDependency         | Specify the database and table name pairs on which the cache content depends on. The cached data will expire automatically when the data changes in the database.  |
| VaryByCustom          | Specify the list of custom strings that the output cache uses to vary the cache content.   |
| VaryByHeader          | Specify the semicolon separated list of HTTP header names that are used to vary the cache content.   |
| VaryByParam           | Specify the semicolon separated list of form POST or query string parameters that are used to vary the cache content. If not specified, the default value is none. |
| VaryByContentEncoding | Specify the semicolon separated list of character set that the output cache uses to vary the cache content.  |

# .NET Online UA Training Course Feedback

---

I hope that you will find this material useful.

If you find errors or inaccuracies in this material or know how to improve it, please report on to the electronic address:

Oleksii\_Leunenko@epam.com

With the note [.NET Online UA Training Course Feedback]

Thank you.

# Q&A



DRIVEN



CANDID



CREATIVE



ORIGINAL



INTELLIGENT



EXPERT

UA .NET Online LAB