NareshIT

# ASP.NET MVC

MVC 5 & CORE

NareshIT

# Software Design Principles

Design patterns are solutions to software design problems you find again and again in real-world application development. Patterns are about reusable designs and interactions of objects.

A design pattern isn't a finished design that can be transformed directly into code. It is a description or template for how to solve a problem that can be used in many different situations.

Design patterns can speed up the development process by providing tested, proven development paradigms. Effective software design requires considering issues that may not become visible until later in the implementation. Reusing design patterns helps to prevent subtle issues that can cause major problems and improves code readability for coders and architects familiar with the patterns.

The 23 Gang of Four (GoF) patterns are generally considered the foundation for all other patterns. They are categorized in three groups:

- ☐ Creational,
- ☐ Structural, and
- ☐ Behavioral

## Creational Design Patterns

In software engineering, creational design patterns are design patterns that deal with object creation mechanisms, trying to create objects in a manner suitable to the situation. The basic form of object creation could result in design problems or added complexity to the design.

| | |
|---|---|
| **Abstract Factory** | Creates an instance of several families of classes |
| **Builder** | Separates object construction from its representation |
| **Factory Method** | Creates an instance of several derived classes |
| **Prototype** | A fully initialized instance to be copied or cloned |
| **Singleton** | A class of which only a single instance can exist |

## Structural Patterns

In Software Engineering, Structural Design Patterns are Design Patterns that ease the design by identifying a simple way to realize relationships between entities.

| | |
|---|---|
| **Adapter** | Match interfaces of different classes |
| **Bridge** | Separates an object's interface from its implementation |
| **Composite** | A tree structure of simple and composite objects |
| **Decorator** | Add responsibilities to objects dynamically |
| **Façade** | A single class that represents an entire subsystem |
| **Flyweight** | A fine-grained instance used for efficient sharing |
| **Proxy** | An object representing another object |

## Behavioral Patterns

In software engineering, behavioral design patterns are design patterns that identify common communication patterns between objects and realize these patterns. By doing so, these patterns increase flexibility in carrying out this communication.

| | |
|---|---|
| **Chain of Resp.** | A way of passing a request between a chain of objects |
| **Command** | Encapsulate a command request as an object |
| **Interpreter** | A way to include language elements in a program |
| **Iterator** | Sequentially access the elements of a collection |

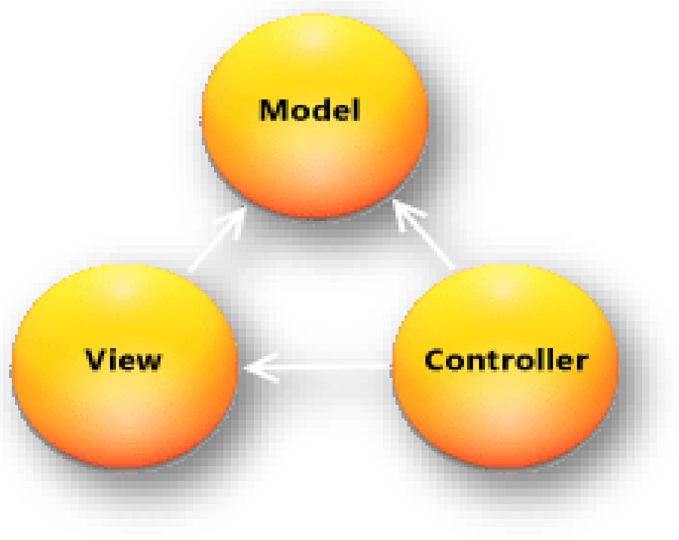
| | |
|---|---|
| **Mediator** | Defines simplified communication between classes |
| **Memento** | Capture and restore an object's internal state |
| **Observer** | A way of notifying change to a number of classes |
| **State** | Alter an object's behavior when its state changes |
| **Strategy** | Encapsulates an algorithm inside a class |
| **Template Method** | Defer the exact steps of an algorithm to a subclass |
| **Visitor** | Defines a new operation to a class without change |

## Architectural Patterns

An architectural pattern is a general, reusable solution to a commonly occurring problem in software architecture within a given context. Architectural patterns are similar to software design pattern but have a broader scope.

- Blackboard system
- Broker Pattern
- Event-driven architecture
- Implicit invocation
- Layers
- Microservices
- Model-view-controller (MVC)
- Presentation-abstraction-control
- Model-view-presenter (MVP)
- Model-view-viewmodel (MVVM)
- Multitier architecture (often three-tier or n-tier)

## The MVC Pattern



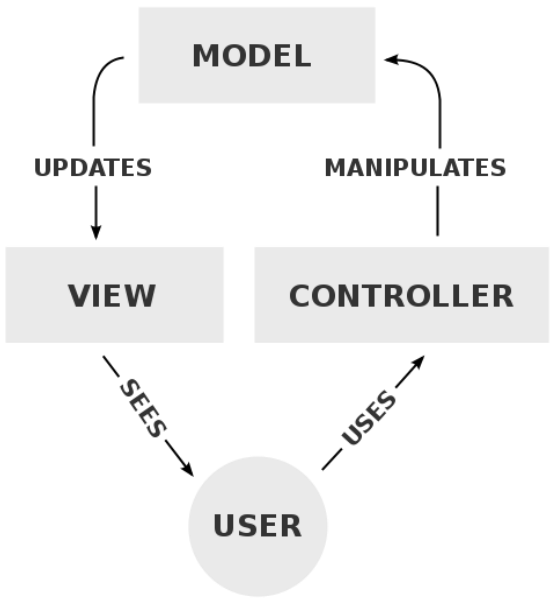
- Model–view–controller (MVC) is a software architecture pattern
- Originally formulated in the late 1970s by Trygve Reenskaug as part of the Smalltalk
- Code reusability and separation of concerns
- Originally developed for desktop, then adapted for internet applications

## Father of MVC

- Trygve Mikkjel Heyerdahl Reenskaug (born 1930 | Age 83) is a Norwegian computer scientist and professor emeritus of the University of Oslo.
- He formulated the model-view-controller (MVC) pattern for Graphic User Interface (GUI) software design in 1979 while visiting the Xerox Palo Alto Research Center (PARC).

## MVC

The Model-View-Controller (MVC) architectural pattern separates an application into three main components: the model, the view, and the controller.



## Model

- Set of classes that describes the data we are working with as well as the business
- Rules for how the data can be changed and manipulated
- May contain data validation rules
- Often encapsulate data stored in a database as well as code used to manipulate the data
- Most likely a Data Access Layer of some kind
- Apart from giving the data objects, it doesn't have significance in the framework

## View

- Defines how the application’s user interface (UI) will be displayed
- May support master views (layouts) and sub-views (partial views or controls)
- Web: Template to dynamically generate HTML

## Controller

- The core MVC component
- Process the requests with the help of views and models
- A set of classes that handles
  - Communication from the user
  - Overall application flow
  - Application-specific logic
- Every controller has one or more "Actions"

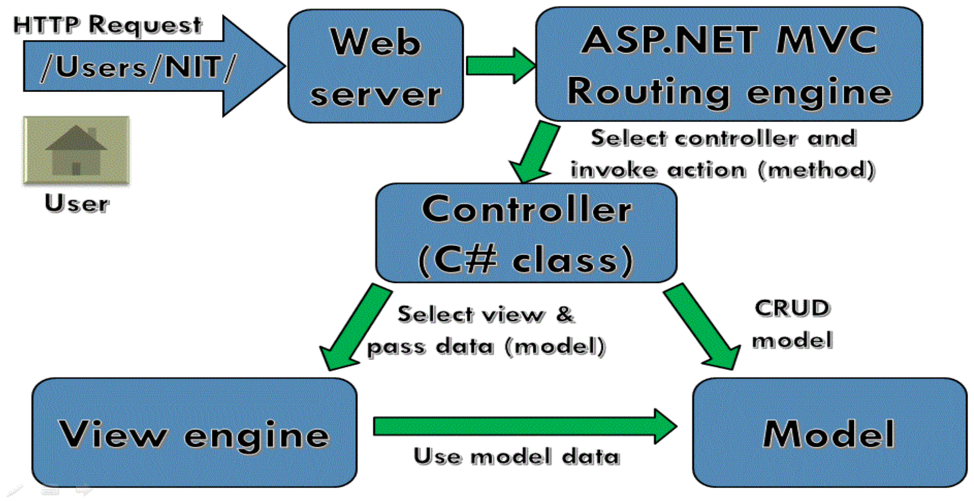
## MVC Frameworks

| | |
|---|---|
| **PHP** | Cake PHP, Code Igniter |
| **Java** | Spring |
| **Perl** | Catalyst, Dancer |
| **Python** | Django, Flask, Grok |
| **Ruby** | Ruby on Rails, Camping, Nitro, Sinatra |
| **Java Script** | Angular JS, JavaScript MVC, Spine |
| **.NET Framework** | ASP.NET MVC |

## History of ASP (18 Years)

- 1996 – Active Server Pages (ASP)
- 2002 – ASP.NET
- 2008 – ASP.NET MVC
- 2010 – ASP.NET Web Pages
- 2012 – ASP.NET Web API, Signal R
- 2014 – ASP.NET 5 (Renamed as ASP.NET CORE)

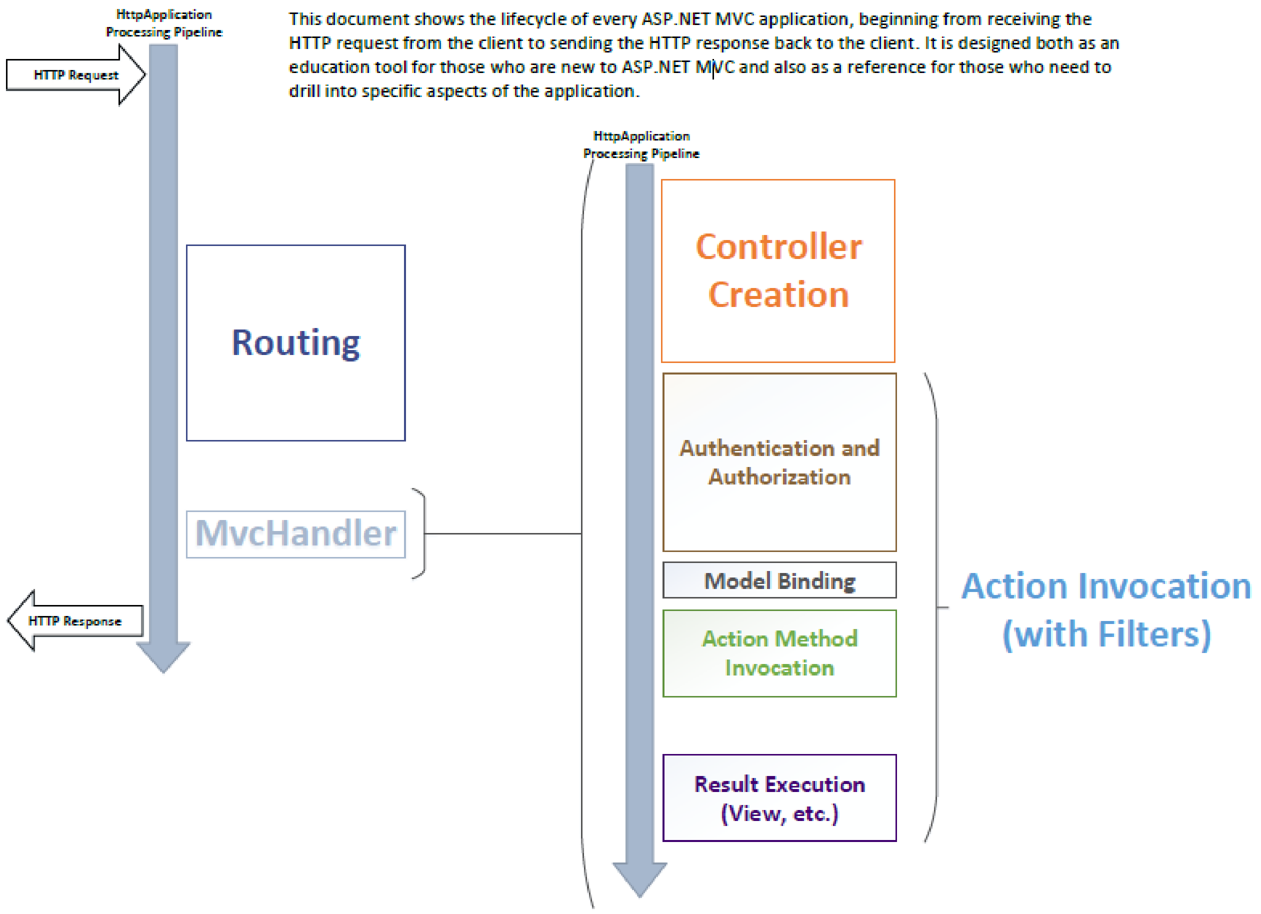
## The MVC Pattern for Web


HTTP Request
/Users/NIT/
User
Web server
ASP.NET MVC Routing engine
Select controller and invoke action (method)
Controller (C# class)
Select view & pass data (model)
CRUD model
View engine
Use model data
Model

- Incoming request routed to Controller
    - For web: HTTP request
- Controller processes request and creates presentation Model
    - Controller also selects appropriate result (view)
- Model is passed to View
- View transforms Model into appropriate output format (HTML)
- Response is rendered (HTTP Response)

# ASP.NET MVC 5 APPLICATION LIFECYCLE – HIGH-LEVEL VIEW

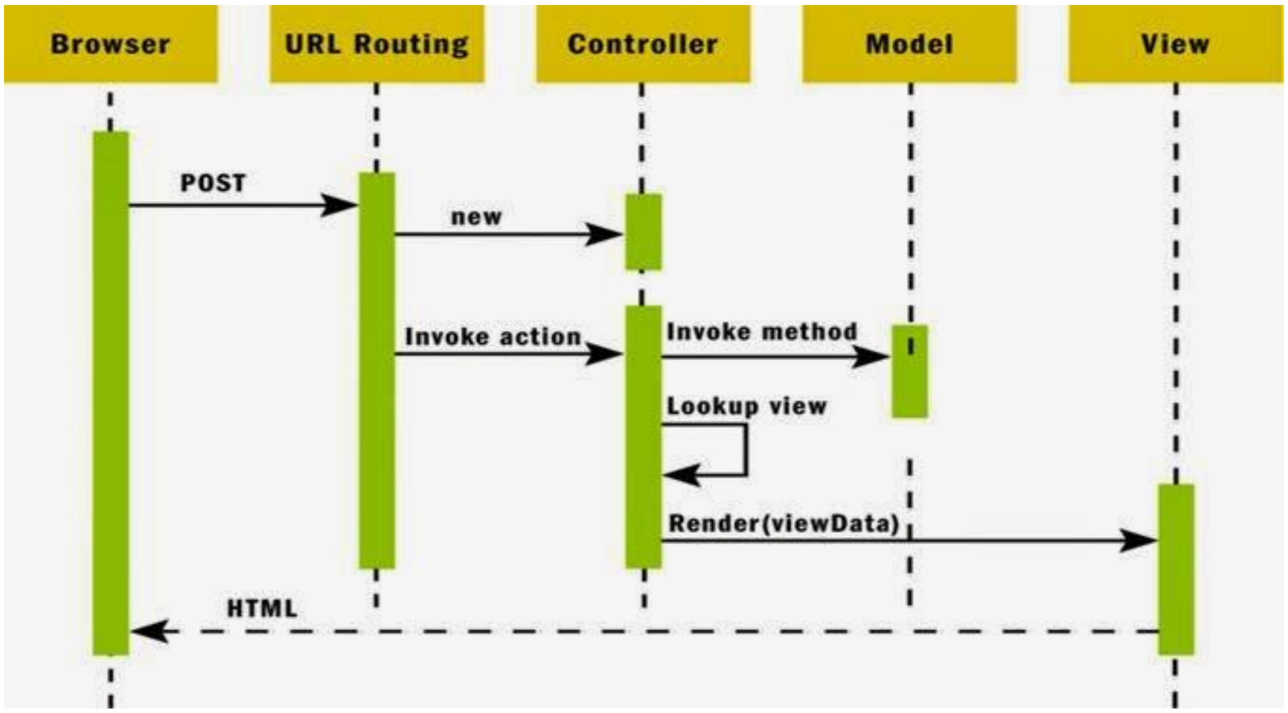
This document shows the lifecycle of every ASP.NET MVC application, beginning from receiving the HTTP request from the client to sending the HTTP response back to the client. It is designed both as an education tool for those who are new to ASP.NET MVC and also as a reference for those who need to drill into specific aspects of the application.


HttpApplication Processing Pipeline
HTTP Request
Routing
MvcHandler
HTTP Response
HttpApplication Processing Pipeline
Controller Creation
Authentication and Authorization
Model Binding
Action Method Invocation
Result Execution (View, etc.)
Action Invocation (with Filters)

| Stage | Details |
|---|---|
| **Receive first request for the application** | In the Global.asax file, Route objects are added to the RouteTable object. |
| **Perform routing** | The UrlRoutingModule module uses the first matching Route object in theRouteTable collection to create the RouteData object, which it then uses to create a RequestContext (IHttpContext) object. |
| **Create MVC request handler** | The MvcRouteHandler object creates an instance of the MvcHandler class and passes it the RequestContext instance. |
| **Create controller** | The MvcHandler object uses the RequestContext instance to identify theIControllerFactory object (typically an instance of the DefaultControllerFactoryclass) to create the controller instance with. |

| | |
|---|---|
| **Execute controller** | The MvcHandler instance calls the controller s Execute method. |
| **Invoke action** | Most controllers inherit from the Controller base class. For controllers that do so, theControllerActionInvoker object that is associated with the controller determines which action method of the controller class to call, and then calls that method. |
| **Execute result** | A typical action method might receive user input, prepare the appropriate response data, and then execute the result by returning a result type. The built-in result types that can be executed include the following: ViewResult (which renders a view and is the most-often used result type), RedirectToRouteResult, RedirectResult,ContentResult, JsonResult, and EmptyResult. |

## Request Flow



| ASP.NET Web Forms | ASP.NET MVC |
|---|---|
| Asp.Net Web Form follow a traditional event driven development model. | Asp.Net MVC is a lightweight and follow MVC (Model, View, Controller) pattern based development model. |
| Asp.Net Web Form has server controls. | Asp.Net MVC has html helpers. |
| Asp.Net Web Form supports view state for state management at client side. | Asp.Net MVC does not support view state. |
| Asp.Net Web Form has file-based URLs means file name exist in the URLs must have its physically existence. | Asp.Net MVC has route-based URLs means URLs are divided into controllers and actions and moreover it is based on controller not on physical file. |
| Asp.Net Web Form follows Web Forms Syntax | Asp.Net MVC follow customizable syntax (Razor as default) |
| In Asp.Net Web Form, Web Forms(ASPX) i.e. views are tightly coupled to Code behind(ASPX.CS) i.e. logic. | In Asp.Net MVC, Views and logic are kept separately. |
| Asp.Net Web Form has Master Pages for consistent look and feels. | Asp.Net MVC has Layouts for consistent look and feels. |
| Asp.Net Web Form has User Controls for code re-usability. | Asp.Net MVC has Partial Views for code re-usability. |
| Asp.Net Web Form has built-in data controls and best for rapid development with powerful data access. | Asp.Net MVC is lightweight, provide full control over markup and support many features that allow fast & agile development. Hence it is best for developing interactive web application with latest web standards. |
| Asp.Net Web Form is not Open Source. | Asp.Net Web MVC is an Open Source. |

## When to Create an MVC Application

You must consider carefully whether to implement a Web application by using either the ASP.NET MVC framework or the ASP.NET Web Forms model. The MVC framework does not replace the Web Forms model; you can use either framework for Web applications. (If you have existing Web Forms-based applications, these continue to work exactly as they always have.)

Before you decide to use the MVC framework or the Web Forms model for a specific Web site, weigh the advantages of each approach.

## Advantages of an MVC-Based Web Application

The ASP.NET MVC framework offers the following advantages:

- It makes it easier to manage complexity by dividing an application into the model, the view, and the controller.
- It does not use view state or server-based forms. This makes the MVC framework ideal for developers who want full control over the behavior of an application.
- It uses a Front Controller pattern that processes Web application requests through a single controller. This enables you to design an application that supports a rich routing infrastructure. For more information
- It provides better support for test-driven development (TDD).
- It works well for Web applications that are supported by large teams of developers and for Web designers who need a high degree of control over the application behavior.

## Advantages of a Web Forms-Based Web Application

The Web Forms-based framework offers the following advantages:

- It supports an event model that preserves state over HTTP, which benefits line-of-business Web application development. The Web Forms-based application provides dozens of events that are supported in hundreds of server controls.
- It uses a Page Controller pattern that adds functionality to individual pages. For more information
- It uses view state on server-based forms, which can make managing state information easier.
- It works well for small teams of Web developers and designers who want to take advantage of the large number of components available for rapid application development.
- In general, it is less complex for application development, because the components (the **Page** class, controls, and so on) are tightly integrated and usually require less code than the MVC model.

## ASP.NET MVC Features

- Runs on top of ASP.NET
    - Not a replacement for WebForms
    - Leverage the benefits of ASP.NET
- Embrace the web
    - User/SEO friendly URLs, HTML 5, SPA
    - Adopt REST concepts
- Uses MVC pattern
    - Conventions and Guidance
    - Separation of concerns
- Tight control over markup

- Testable

  Loosely coupled and extensible
- Convention over configuration
- Razor view engine
- One of the greatest view engines
- With intellisense, integrated in Visual Studio
- Reuse of current skills (C#, LINQ, HTML, etc.)
- Application-based (not scripts like PHP)

## Separation of Concerns

- Each component has one responsibility
  - SRP – Single Responsibility Principle
  - DRY – Don’t Repeat Yourself
- More easily testable
  - TDD – Test-driven development
- Helps with concurrent development
  - Performing tasks concurrently
- One developer works on views
- Another works on controllers

## Extensible

- Replace any component of the system
  - Interface-based architecture
- Almost anything can be replaced or extended
- Model binders (request data to CLR objects)
- Action/result filters (e.g. OnActionExecuting)
- Custom action result types
- View engine (Razor, WebForms, NHaml, Spark)
- View helpers (HTML, AJAX, URL, etc.)
- Custom data providers (ADO.NET), etc.

## Clean URL’s

- REST-like
- /products/update
- /blog/posts/2014/11/28/mvc-is-cool
- Friendlier to humans
- /product.aspx?catId=123 or post.php?id=123
- Becomes /products/chocolate/
- Friendlier to web crawlers
- Search engine optimization (SEO)

## ASP.NET MVC Release History

| Date | Version |
|---|---|
| 10 December 2007 | ASP.NET MVC CTP |
| 13 March 2009 | ASP.NET MVC 1.0 |
| 16 December 2009 | ASP.NET MVC 2 RC |
| 4 February 2010 | ASP.NET MVC 2 RC 2 |
| 10 March 2010 | ASP.NET MVC 2 |
| 6 October 2010 | ASP.NET MVC 3 Beta |
| 9 November 2010 | ASP.NET MVC 3 RC |
| 10 December 2010 | ASP.NET MVC 3 RC 2 |
| 13 January 2011 | ASP.NET MVC 3 |
| 20 September 2011 | ASP.NET MVC 4 Developer Preview |
| 15 February 2012 | ASP.NET MVC 4 Beta |
| 31 May 2012 | ASP.NET MVC 4 RC |

| | |
|---|---|
| 15 August 2012 | ASP.NET MVC 4 |
| 30 May 2013 | ASP.NET MVC 4 4.0.30506.0 |
| 26 June 2013 | ASP.NET MVC 5 Preview |
| 23 August 2013 | ASP.NET MVC 5 RC 1 |
| 17 October 2013 | ASP.NET MVC 5 |
| 17 January 2014 | ASP.NET MVC 5.1 |
| 10 February 2014 | ASP.NET MVC 5.1.1 |
| 4 April 2014 | ASP.NET MVC 5.1.2 |
| 22 June 2014 | ASP.NET MVC 5.1.3 |
| 1 July 2014 | ASP.NET MVC 5.2.0 |
| 28 August 2014 | ASP.NET MVC 5.2.2 |
| 9 February 2015 | ASP.NET MVC 5.2.3 |
| 6 November 2014 | ASP.NET MVC 6.0.0-beta1 |
| 18 November 2015 | ASP.NET MVC 6.0.0-rc1 |
| 17 May 2016 | ASP.NET Core MVC 1.0.0-rc2 |
| 12 August 2016 | ASP.NET Core MVC 1.0.0 |

| | |
|---|---|
| 17 August 2016 | ASP.NET Core MVC 1.0.1 |
| 17 November 2016 | ASP.NET Core MVC 1.0.2 |
| 18 November 2016 | ASP.NET Core MVC 1.1.0 |

## What's new in MVC 4?

- ASP.NET Web API
- Refreshed and modernized default project templates
- New mobile project template
- Many new features to support mobile apps
- Enhanced support for asynchronous methods

## What's new in MVC 5?

- Attribute routing improvements
- Bootstrap support for editor templates
- Enum support in views
- Unobtrusive validation for MinLength/MaxLength Attributes
- Supporting the 'this' context in Unobtrusive Ajax

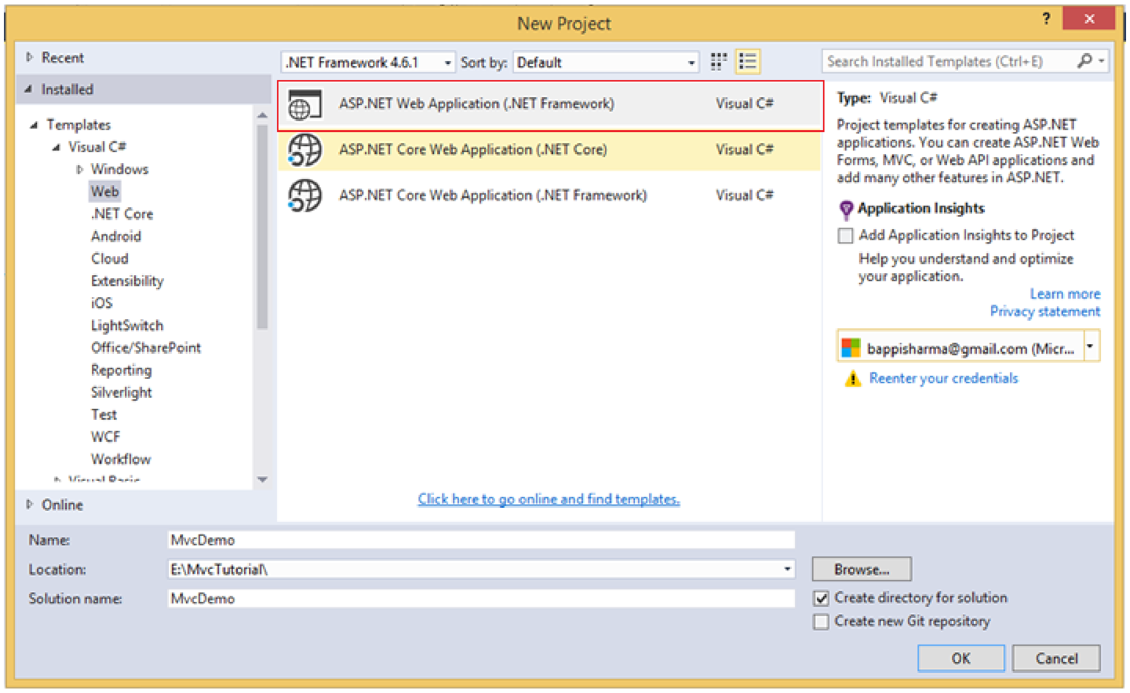
## What's new in MVC 6?

- Rebuilt from the Ground Up
- MVC, Web API, and Web Pages are merged into one framework, called MVC 6. The new framework uses a common set of abstractions for routing, action selection, filters, model binding, and so on.
- Dependency injection is built into the framework. Use your preferred IoC container to register dependencies.
- vNext is host agnostic. You can host your app in IIS, or self-host in a custom process. (Web API 2 and SignalR 2 already support self-hosting; vNext brings this

- +same capability to MVC.)
- vNext is open source and cross platform.
- Leaner, Faster
- MVC 6 has no dependency on System.Web.dll. The result is a leaner framework, with faster startup time and lower memory consumption.
- vNext apps can use a cloud-optimized runtime and subset of the .NET Framework. This subset of the framework is about 11 megabytes in size compared to 200 megabytes for the full framework, and is composed of a collection of NuGet packages.
- Because the cloud-optimized framework is a collection of NuGet packages, your app can include only the packages you actually need. No unnecessary memory, disk space, loading time, etc.
- Microsoft can deliver updates to the framework on a faster cadence, because each part can be updated independently.
- True Side-by-Side Deployment
- The reduced footprint of the cloud-optimized runtime makes it practical to deploy the framework with your app.
- You can run apps side-by-side with different versions of the framework on the same server.
- Your apps are insulated from framework changes on the server.
- You can make framework updates for each app on its own schedule.
- No errors when you deploy to production resulting from a mismatch between the framework patch level on the development machine and the production server.
- New Development Experience
- vNext uses the Roslyn compiler to compile code dynamically.
- You can edit a code file, refresh the browser, and see the changes without rebuilding the project.
- Besides streamlining the development process, dynamic code compilation enables development scenarios that were not possible before, such as editing code on the server using Visual Studio Online (“Monaco”).
- You can choose your own editors and tools.

# Creating a New ASP.NET MVC Project

- Open Visual Studio 2015
- Select **Create New Project**
- In Visual C# Templates Select "Web"
- Select "ASP.NET Web Application"



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

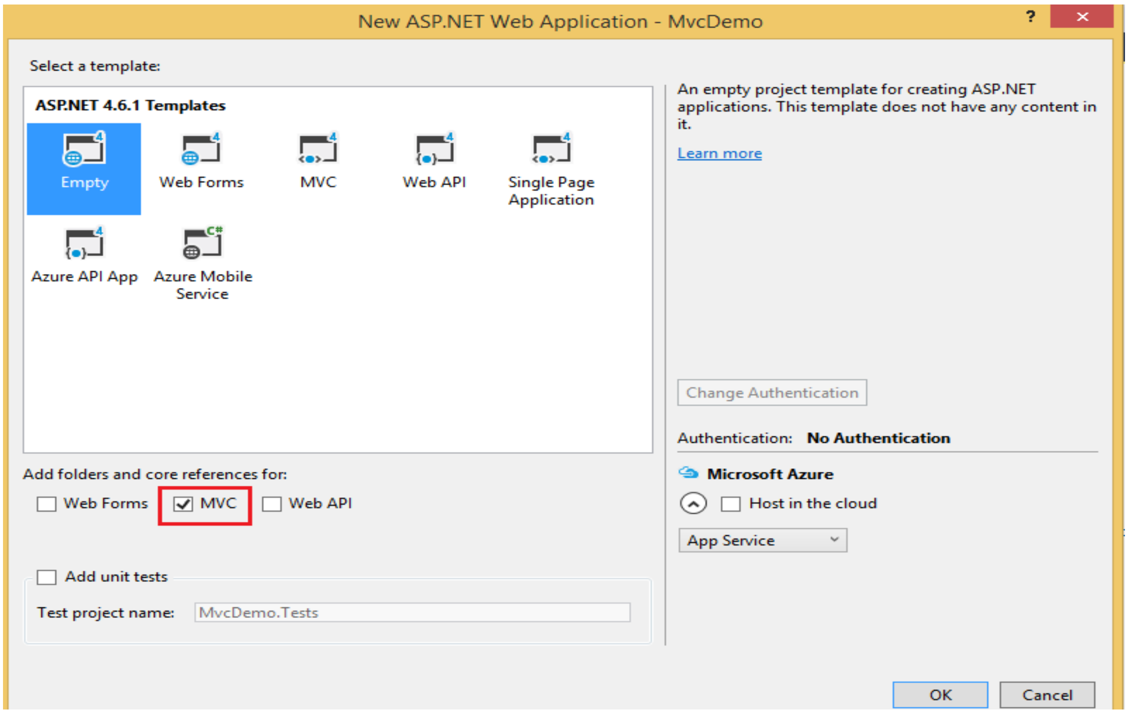
Type: Visual C#
Project templates for creating ASP.NET applications. You can create ASP.NET Web Forms, MVC, or Web API applications and add many other features in ASP.NET.

Application Insights
Add Application Insights to Project
Help you understand and optimize your application.
Learn more
Privacy statement
bappisharma@gmail.com (Micr...
Reenter your credentials

Click here to go online and find templates.

Name: MvcDemo
Location: E:\MvcTutorial\
Solution name: MvcDemo
[✓] Create directory for solution
[ ] Create new Git repository

- Select **Empty** Template
- Select add references for **MVC**



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

Authentication: **No Authentication**

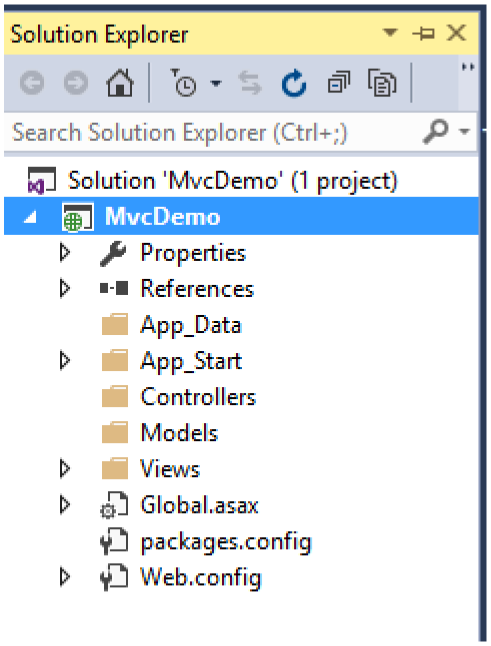
Add folders and core references for:
[ ] Web Forms [✓] MVC [ ] Web API

[ ] Add unit tests
Test project name: MvcDemo.Tests

[ ] Host in the cloud

# Understanding the MVC Application Structure

When you create a new ASP.NET MVC application with Visual Studio, it automatically adds several files and directories to the project, as shown above. ASP.NET MVC projects by default have Few top-level directories.



| Folder / File | Description |
|---|---|
| **App_Data** | Where you store data files you want to read/write |
| **App_Start** | Contains classes that are intended to run on application startup. It includes<br>- RouteConfig.cs<br>- FilterConfig.cs<br>- BundleConfig.cs<br>- StartupAuth.cs |
| **Controllers** | Where you put Controller classes that handle URL requests. |
| **Models** | Where you put classes that represent and manipulate data and business objects |
| **Views** | Where you put UI template files that are responsible for rendering output, such as HTML |