# Session-13

# Introduction to ASP.NET

# Contents

- What is ASP.NET?
- ASP.NET and earlier Web Development platforms
- Seven important facts about ASP.NET
- Web Architecture
- ASP.NET page life cycle
- Validation controls

# What is ASP.NET?

ASP.NET is a server-side technology for building powerful, dynamic Web applications and is part of the .NET Framework

# ASP.NET and earlier web development platforms

- ✓ASP.NET features a completely object-oriented programming model, which includes an event-driven, control-based architecture that encourages code encapsulation and code reuse.

- ✓ ASP.NET gives you the ability to code in any supported .NET language (including Visual Basic, C#, J#, and many other languages that have third-party compilers)

# ASP.NET and earlier web development platforms

- ✓ASP.NET is also a platform for building web services, which are reusable units of code that other applications can call across platform and computer boundaries.
- ✓ASP.NET is dedicated to high performance. ASP.NET pages and components are compiled on demand instead of being interpreted every time they’re used. ASP.NET also includes a finetuned data access model and flexible data caching to further boost performance.

# Seven Important Facts About ASP.NET

Fact 1: ASP.NET Is Integrated with the .NET Framework

Fact 2: ASP.NET Is Compiled, Not Interpreted

Fact 3: ASP.NET is Multilanguage

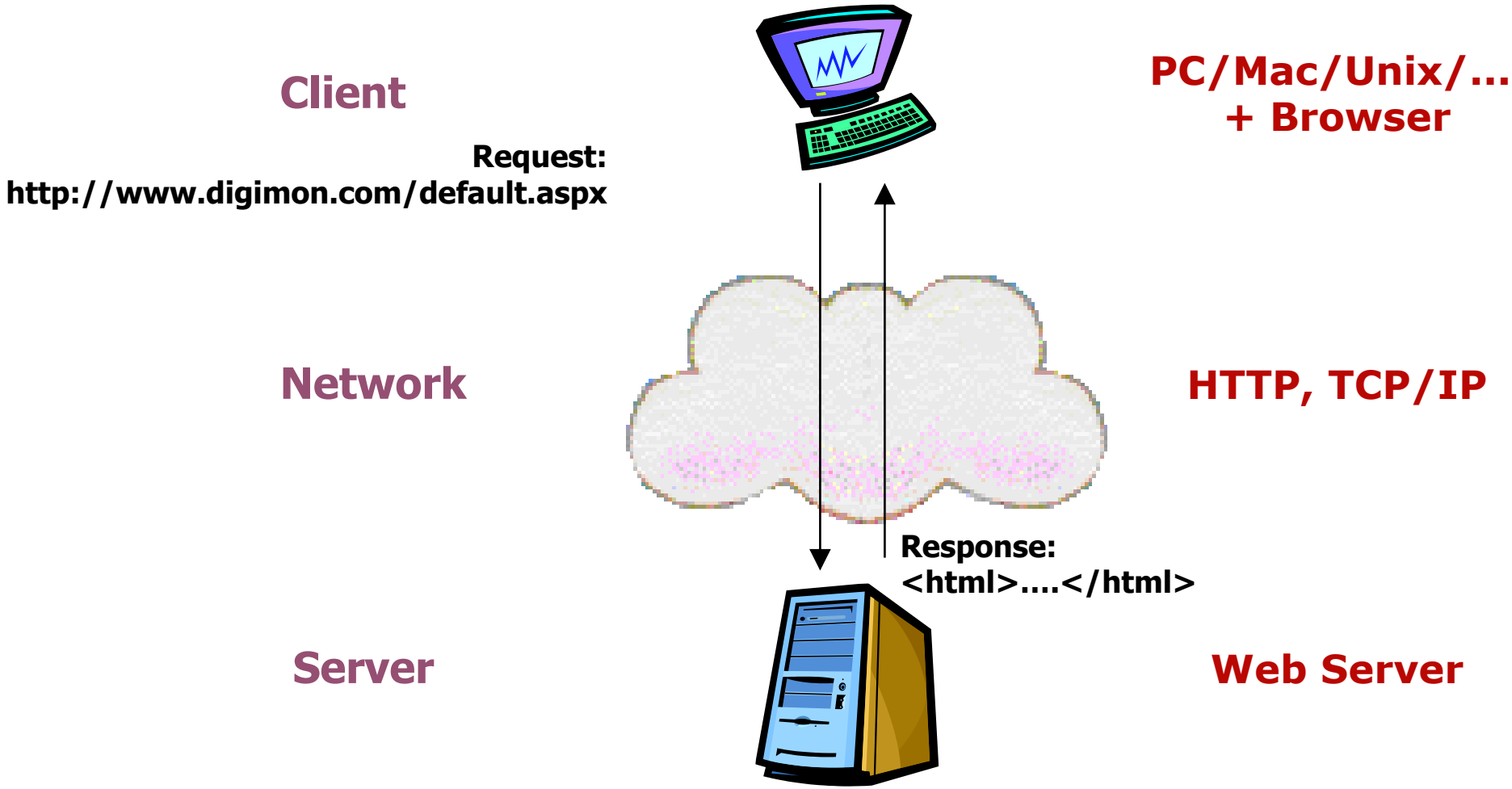
Fact 4: ASP.NET runs inside the CLR

Fact 5: ASP.NET is Object Oriented

Fact 6: ASP.NET is Multidevice and Multibrowser

Fact 7: ASP.NET is Easy to Deploy and configure

# Web Architecture



# ASP.NET Page Life Cycle

| Srno | Stages | Description |
|---|---|---|
| 1. | **Page request** | When the page is requested by a user, ASP.NET determines whether the page needs to be parsed and compiled (therefore beginning the life of a page) |
| 2. | **Start** | Page properties such as Request and Response are set. At this stage, the page also determines whether the request is a postback or a new request and sets the IsPostBack property |
| 3. | **Initialization** | During page initialization, controls on the page are available and each control's UniqueID property is set. A master page and themes are also applied to the page if applicable. |
| 4 | **Load** | During load, if the current request is a postback, control properties are loaded with information recovered from view state and control state. |

# ASP.NET Page Life Cycle

| Srno | Stages | Description |
|---|---|---|
| 5. | **Postback event handling** | If the request is a postback, control event handlers are called. After that, the Validate method of all validator controls is called, which sets the IsValid property of individual validator controls and of the page |
| 6. | **Rendering** | Before rendering, view state is saved for the page and all controls. During the rendering stage, the page calls the Render method for each control, providing a text writer that writes its output to the OutputStream object of the page's Response property. |
| 7. | **Unload** | The Unload event is raised after the page has been fully rendered, sent to the client, and is ready to be discarded. At this point, page properties such as Response and Request are unloaded and cleanup is performed. |

# Validation Controls

- A Validation server control is used to validate the data of an input control.
- If the data does not pass validation, it will display an error message to the user.
- Syntax:

```
<asp:control_name id="some_id" runat="server" />
```

# Validation Controls

| Validation Server Control | Description |
|---|---|
| RequiredFieldValidator | Makes an input control a required field |
| RangeValidator | Checks that the user enters a value that falls between two values |
| CompareValidator | Compares the value of one input control to the value of another input control or to a fixed value |
| RegularExpressionValidator | Ensures that the value of an input control matches a specified pattern |
| CustomValidator | Allows you to write a method to handle the validation of the value entered |
| ValidationSummary | Displays a report of all validation errors occurred in a Web page |

# Validation Controls

**RequiredFieldValidator**: ensures that the required field is not empty. It is generally tied to a text box to force input into the text box.

Syntax:

```
<asp:RequiredFieldValidator
  ID="rfvusername“
   runat="server“
   ControlToValidate =“txtUsername“
    ErrorMessage=“*“ >
</asp:RequiredFieldValidator>
```

# Validation Controls

- The **RangeValidator** control verifies that the input value falls within a predetermined range.
- It has three specific properties:

| Properties | Description |
|---|---|
| **Type** | it defines the type of the data; the available values are: Currency, Date, Double, Integer and String |
| **MinimumValue** | it specifies the minimum value of the range |
| **MaximumValue** | it specifies the maximum value of the range |

```
<asp:RangeValidator  ID="rvclass“
runat="server“
ControlToValidate="txtclass"
ErrorMessage="Enter  your class (6 - 12)"
MaximumValue="12“
MinimumValue="6“
Type="Integer">
 </asp:RangeValidator>
```

# Validation Controls

- The **CompareValidator** control compares a value in one control with a fixed value, or, a value in another control.
- It has specific properties:

| Properties | Description |
|---|---|
| **Type** | it specifies the data type |
| **ControlToCompare** | it specifies the value of the input control to compare with |
| **ValueToCompare** | it specifies the constant value to compare with |
| **Operator** | it specifies the comparison operator, the available values are: Equal, NotEqual, GreaterThan, GreaterThanEqual, LessThan, LessThanEqual and DataTypeCheck |

# Validation Controls

**CompareValidator** control :

Syntax:

```
<asp:CompareValidator
        ID="CompareValidator1"
        runat="server"
        ErrorMessage="Password Must Be Same“
        ControlToValidate="txtConfirmPwd"
        ControlToCompare="txtChoosePwd">
</asp:CompareValidator>
```

# Validation Controls

- The **RegularExpressionValidator** allows validating the input text by matching against a pattern against a regular expression. The regular expression is set in the ValidationExpression property.
- Syntax:

```
<asp:RegularExpressionValidator
  ID="RegularExpressionValidator1“
        runat="server“
        ErrorMessage="Please Enter Valid Email“
        ControlToValidate="txtEmail"
        ValidationExpression="\w+([-+.']\w+)*@\w+([- .]\w+)*\.\w+([-
  .]\w+)*">
</asp:RegularExpressionValidator>
```

# Validation Controls

- The CustomValidator control allows writing application specific custom validation routines for both the client side and the server side validation.
- The client side validation is accomplished through the *ClientValidationFunction* property. The client side validation routine should be written in a scripting language, like **JavaScript** or **VBScript**, which the browser can understand.
- The server side validation routine must be called from the controls *ServerValidate* event handler. The server side validation routine should be written in any .Net language, like **C#** or **VB.Net**.

# Validation Controls

- Syntax:

```
<asp:CustomValidator
        ID="CustomValidator1"
        runat="server"
        ErrorMessage="Password Should not be less  than 5          character"
        ControlToValidate="txtChoosePwd"
        ClientValidationFunction="validateText">
</asp:CustomValidator>
```

# Validation Controls

- The **ValidationSummary** control does not perform any validation but shows a summary of all errors in the page.
- The summary displays the values of the ErrorMessage property of all validation controls that failed validation.
- Syntax:

```
<asp:ValidationSummary
    ID="ValidationSummary1"
    runat="server"
    DisplayMode = "BulletList" />
```

# References


1. C .Net Web Developers Guide by Syngress
2. ASP.NET 4.5, Covers C# and VB Codes, Black Book , Kogent Learning Solutions Inc.

# Session-14

# Introduction to ASP.NET MVC

# Contents

- ASP.NET MVC Architecture
- ASP.NET MVC folder structure
- Creating controllers
- Action Methods
- Different Types of Action Results

# ASP.NET MVC Architecture

- MVC stands for **Model**, **View**, and **Controller**.
- MVC is a standard design pattern
- The ASP.NET MVC framework is a lightweight, highly testable presentation framework that (as with Web Forms-based applications) is integrated with existing ASP.NET features, such as master pages
- The MVC framework is defined in the **System.Web.Mvc** namespace and is a fundamental, supported part of the **System.Web** namespace.

# ASP.NET MVC Architecture

**Models**.

- Model objects are the parts of the application that implement the logic for the applications data domain.
- Often, model objects retrieve and store model state in a database.
- For example, a **Product** object might retrieve information from a database, operate on it, and then write updated information back to a Products table in SQL Server.

# ASP.NET MVC Architecture

**View**

- View in MVC is a user interface.
- View display model data to the user and also enables them to modify them.
- View in ASP.NET MVC is HTML, CSS, and some special syntax (Razor syntax) that makes it easy to communicate with the model and the controller.

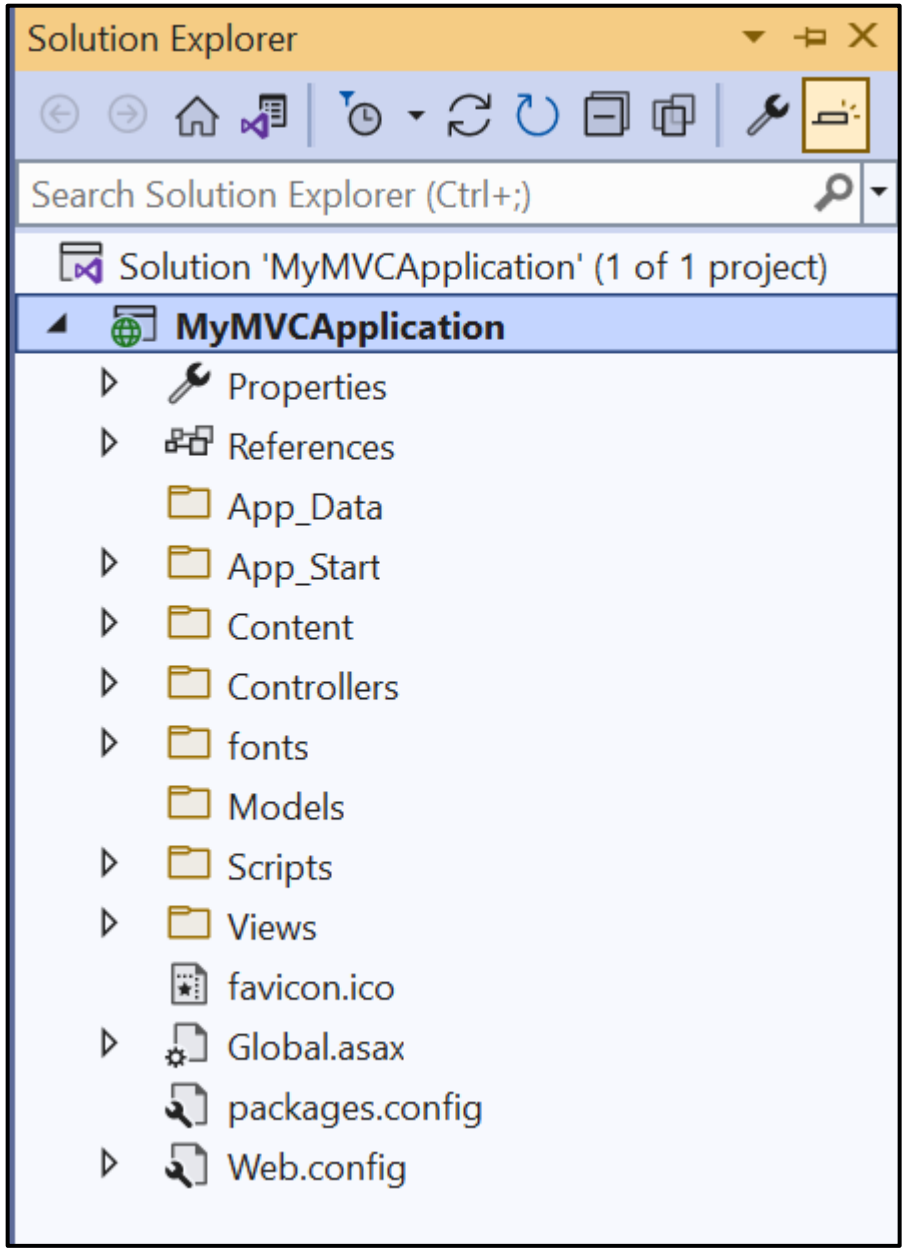
# ASP.NET MVC Architecture

**Controller**:

- The controller handles the user request.
- Typically, the user uses the view and raises an HTTP request, which will be handled by the controller.
- The controller processes the request and returns the appropriate view as a response.
- Controller is the request handler

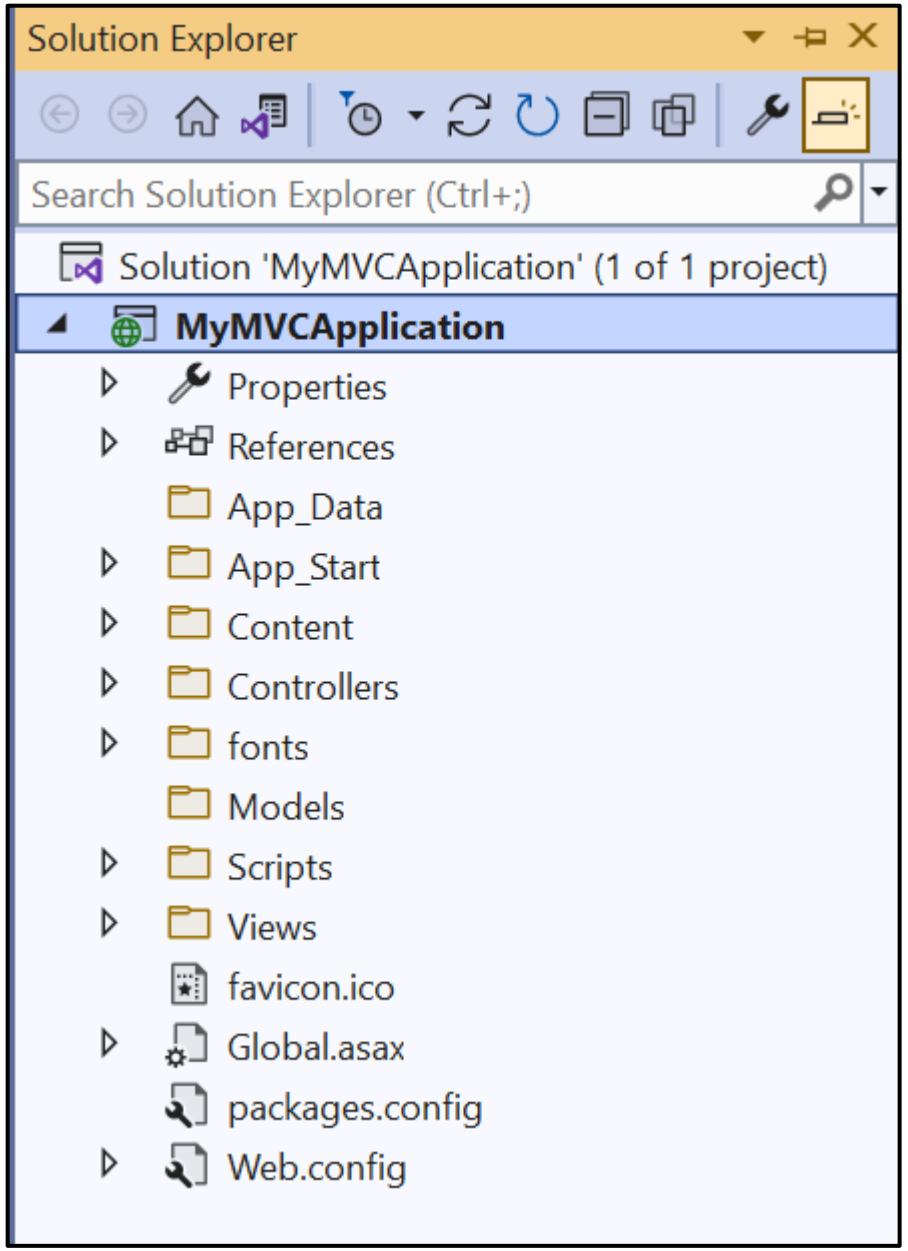
# ASP.NET MVC folder structure

- **App_Data** : folder can contain application data files like LocalDB, .mdf files, XML files, and other data related files.
- **App_Start** : folder can contain class files that will be executed when the application starts
- **Content** : folder contains static files like CSS files, images, and icons files. MVC application includes bootstrap.css, bootstrap.min.css, and Site.css by default.



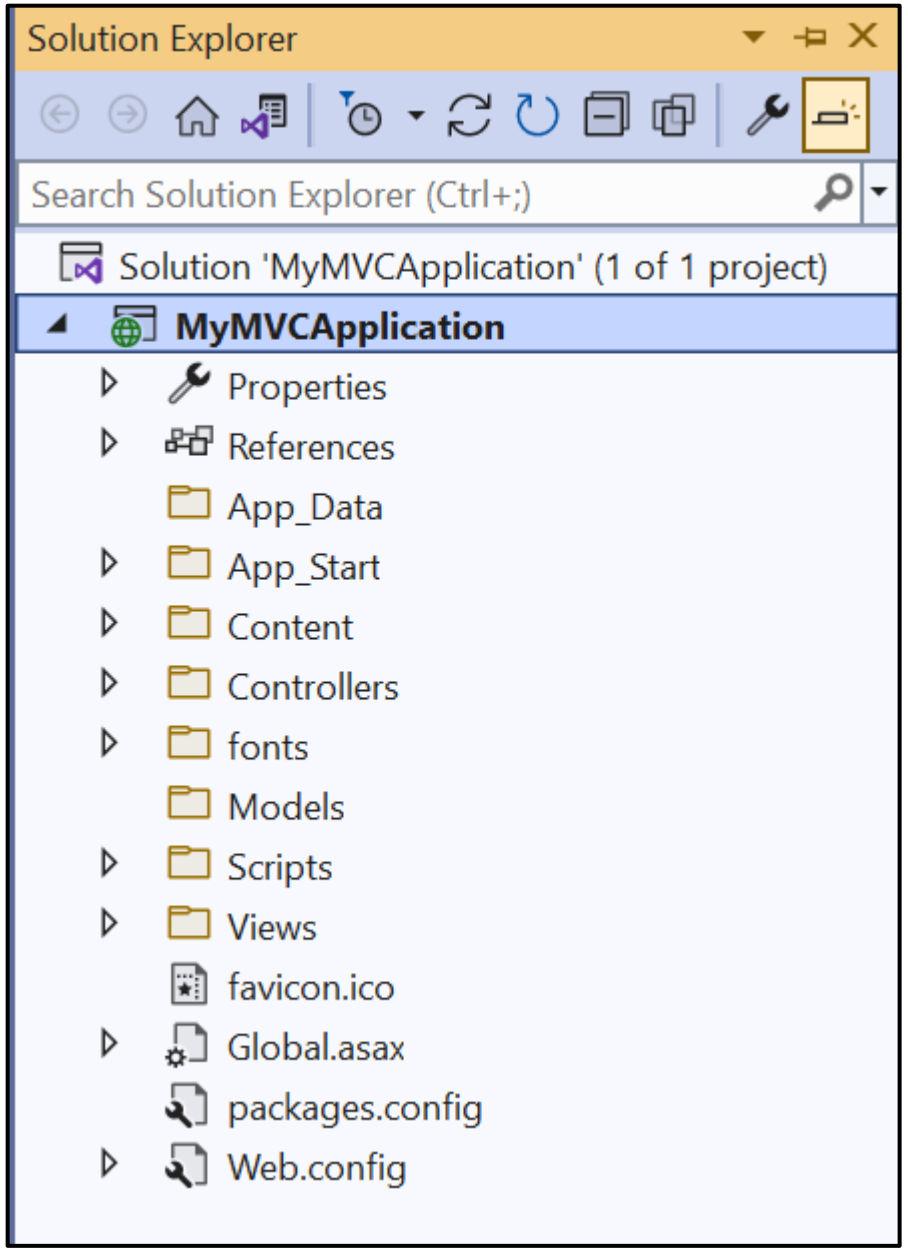
# ASP.NET MVC folder structure

- **Controllers**: folder contains class files for the controllers. A controller handles users' request and returns a response. MVC requires the name of all controller files to end with "Controller". Eg. *HomeController*
- **Fonts :** folder contains custom font files for your application.
- **Models :** folder contains model class files. Typically model class includes public properties, which will be used by the application to hold and manipulate application data.



# ASP.NET MVC folder structure

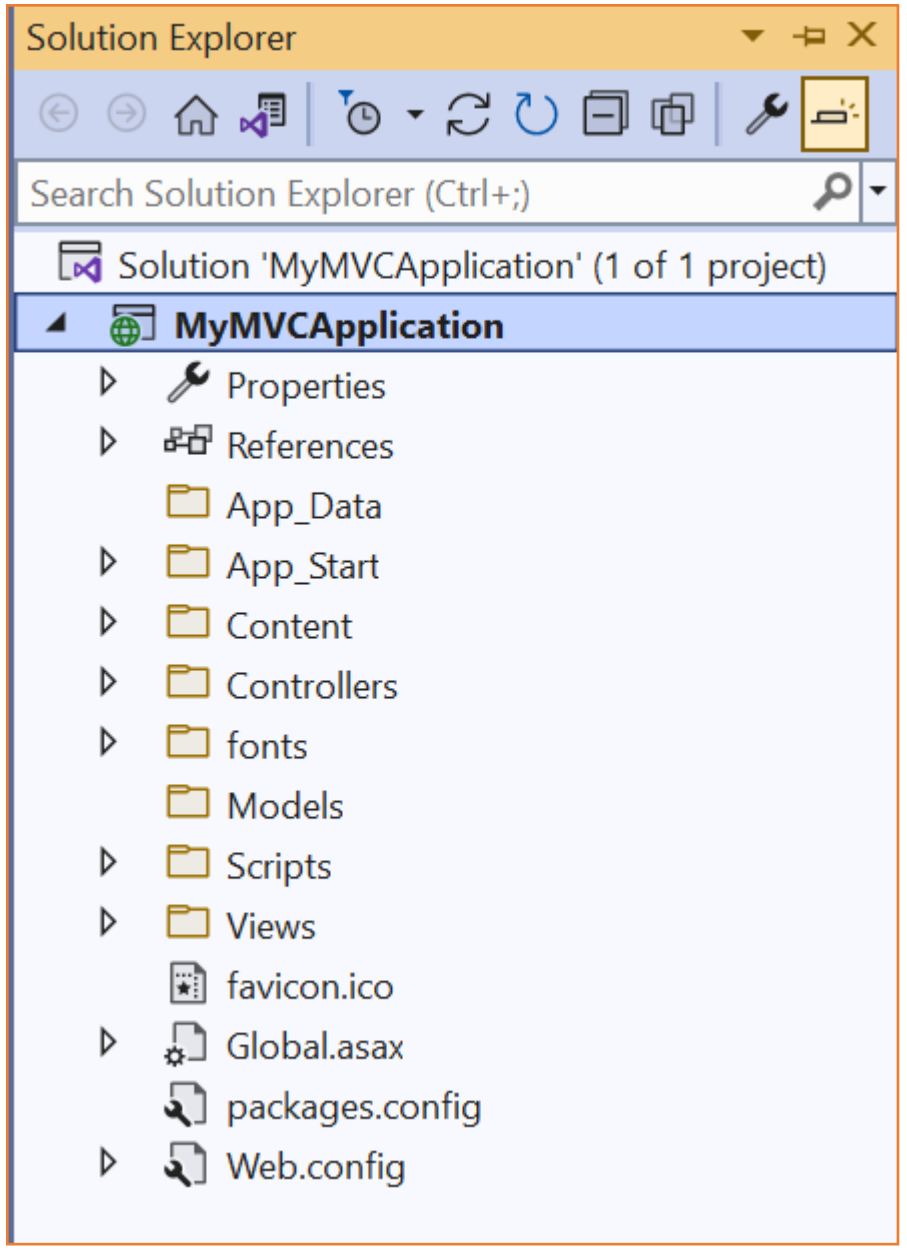
- **Scripts** folder contains JavaScript or VBScript files for the application.
- **Views** folder contains HTML files for the application. Typically view file is a .cshtml file where you write HTML and C# code. The Views folder includes a separate folder for each controller.
- For example, all the .cshtml files, which will be rendered by HomeController will be in View -> Home folder. The Shared folder under the View folder contains all the views shared among different controllers e.g., layout files.



# ASP.NET MVC folder structure

## Configuration files :

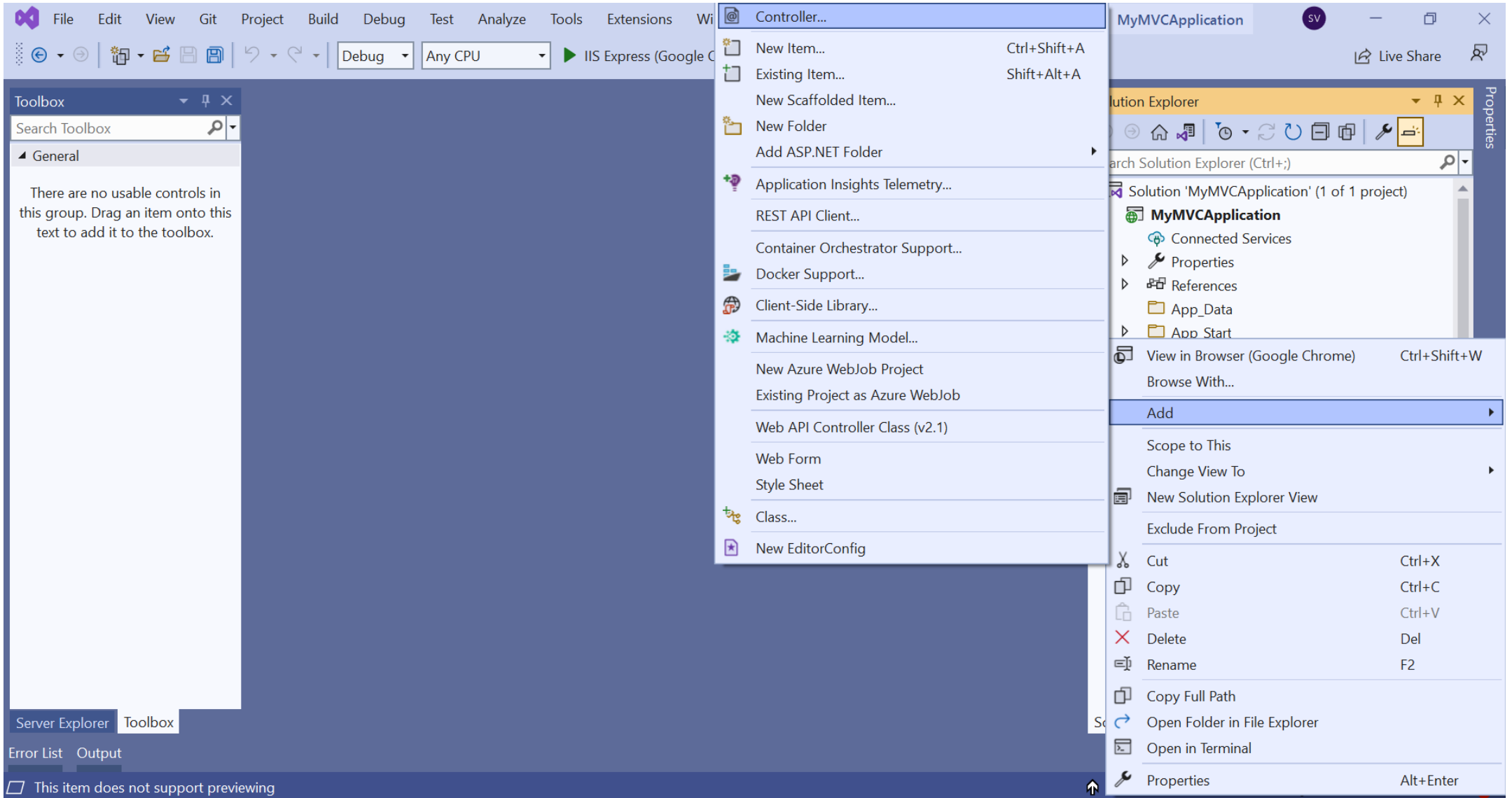
- **Global.asax :** file allows you to write code that runs in response to application-level events, such as *Application_BeginRequest*, *application_start*, *application_error*, *session_start*, *session_end*, etc.
- **Packages.config :** file is managed by NuGet to track what packages and versions you have installed in the application.
- **Web.config :** file contains application-level configurations.



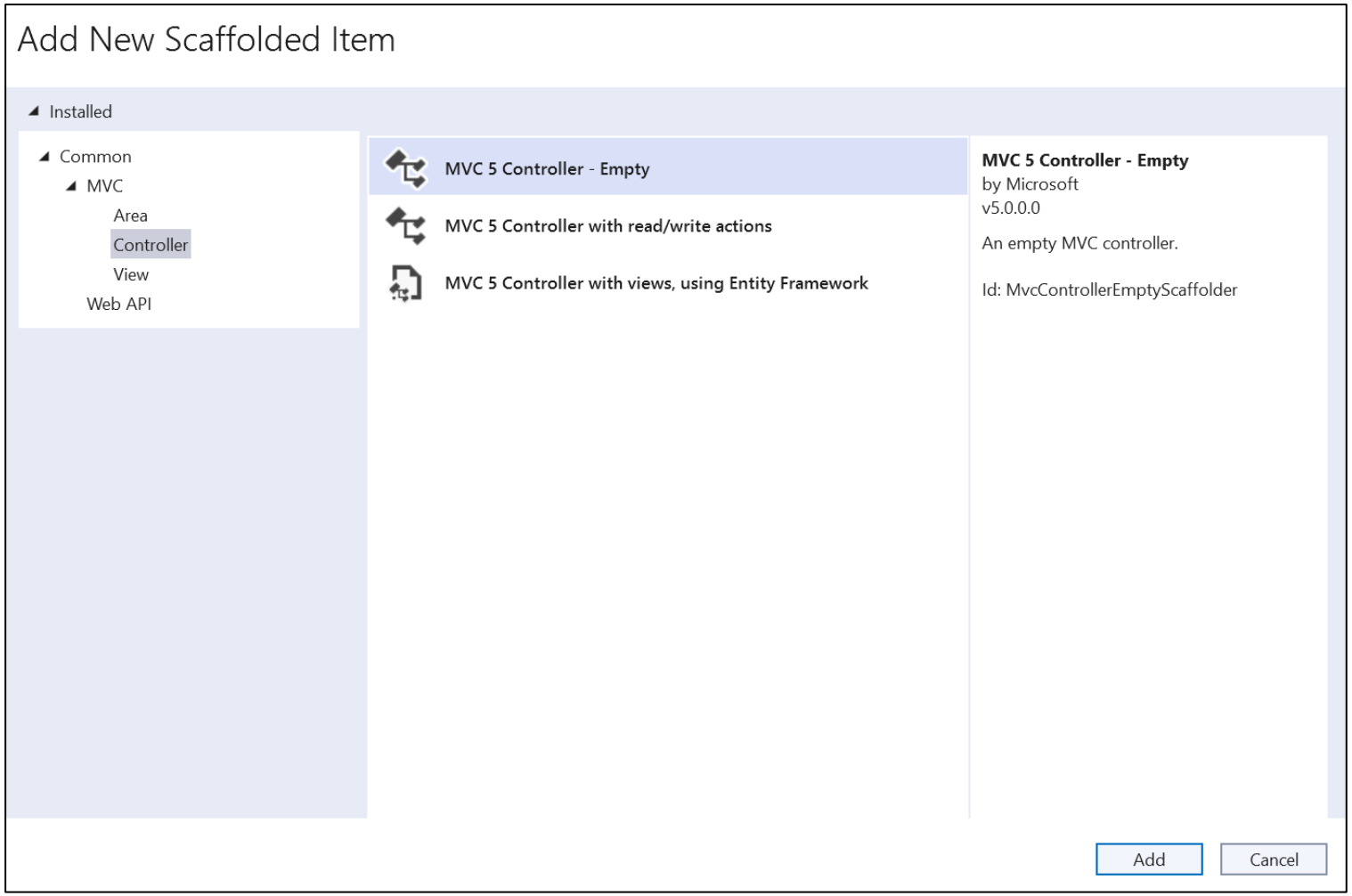
# Creating Controllers

- The Controller in MVC architecture handles any incoming URL request
- Controller is a class derived from the base class System.Web.Mvc.Controller
- Controller class contains public methods called **Action** methods.
- Controller and its action method handles incoming browser requests, retrieves necessary model data and returns appropriate responses.

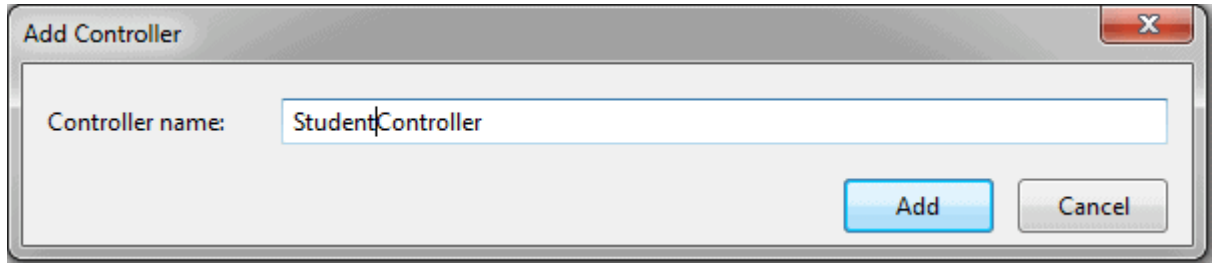
# Creating the Controller



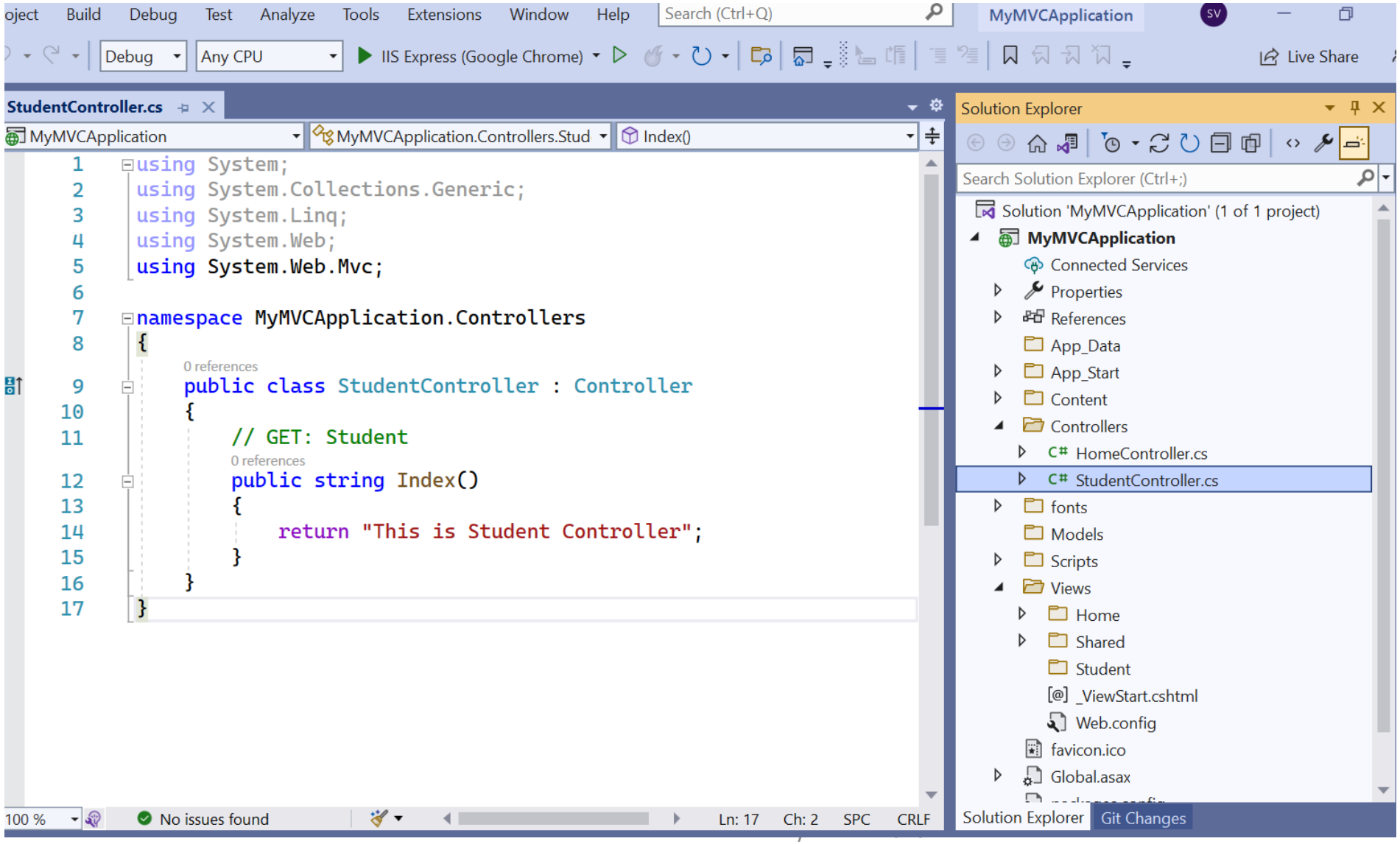
# Creating Controller



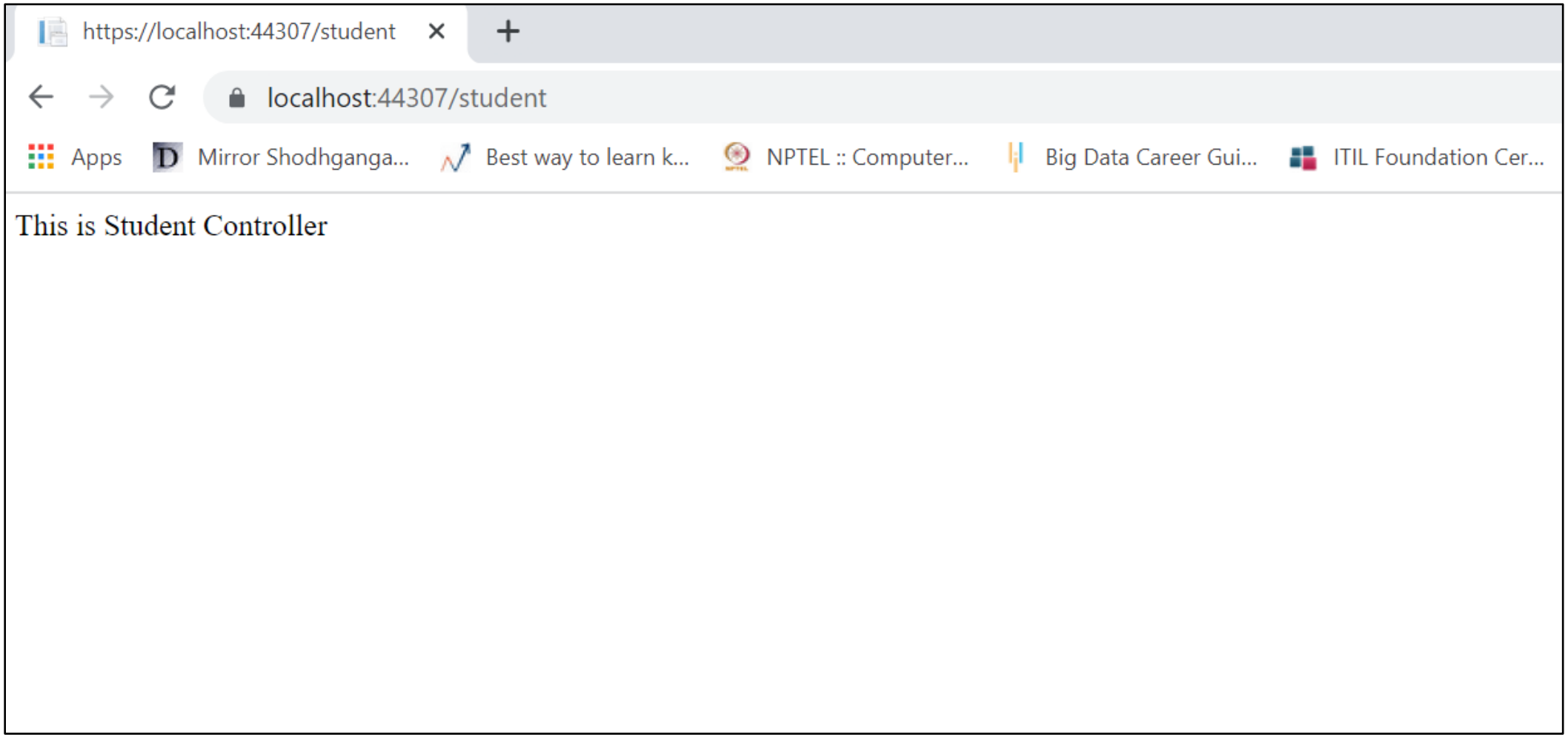
# Creating Controller



# Example : StudentController

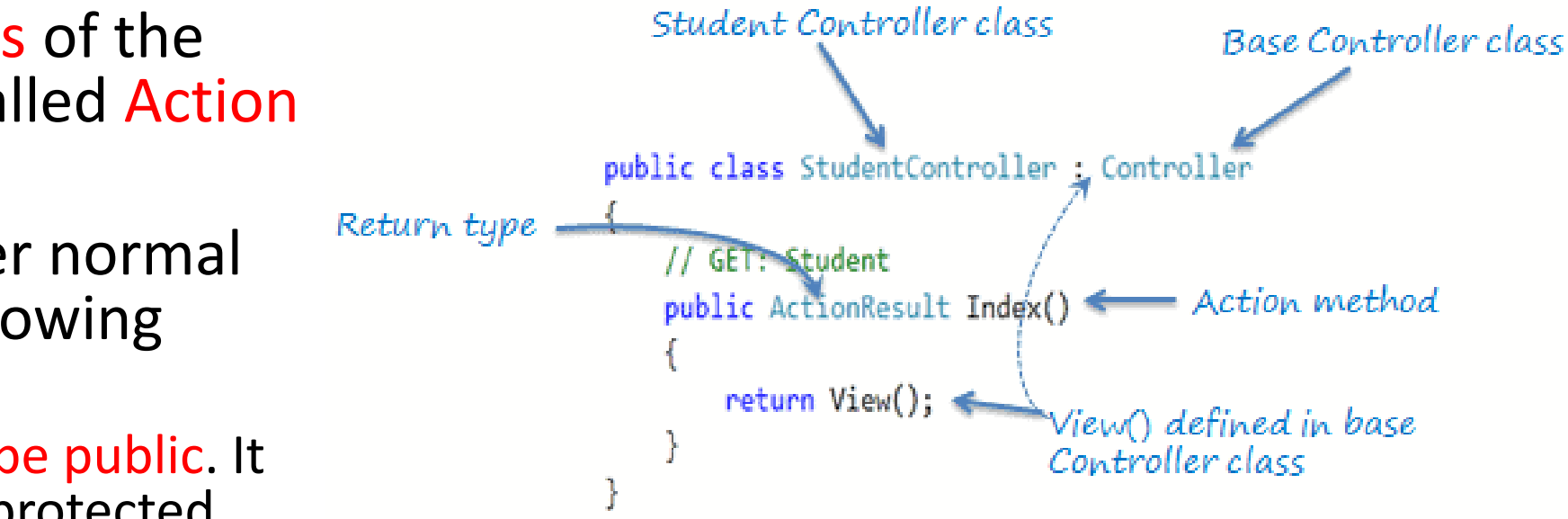


# Example : StudentController

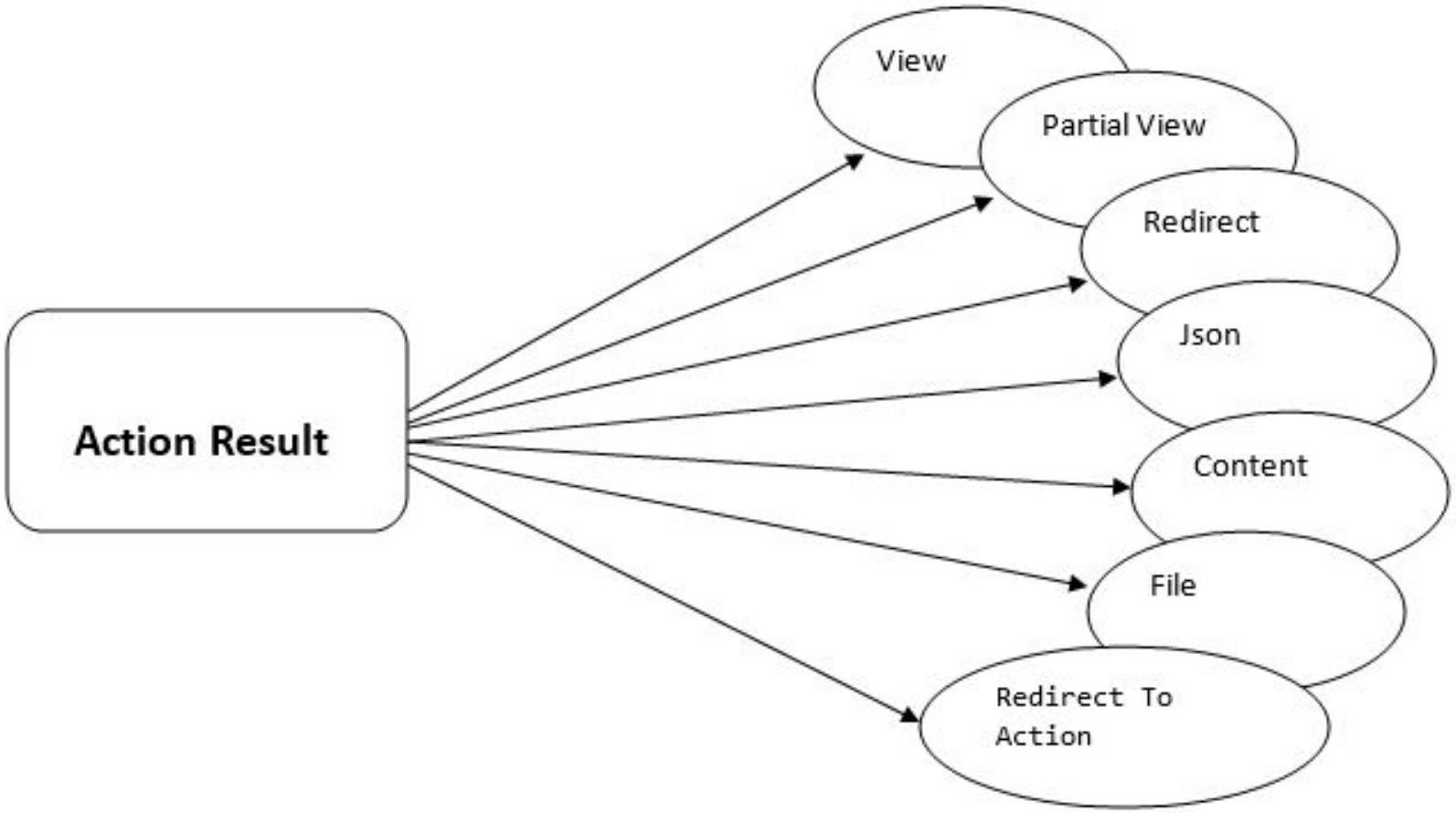


# Action Methods

- All the public methods of the Controller class are called Action methods.
- They are like any other normal methods with the following restrictions:
  - Action method must be public. It cannot be private or protected
  - Action method cannot be overloaded
  - Action method cannot be a static method.



# Different Types Of Action Results In ASP.NET MVC



# View Result

View result is a basic view result. It returns basic results to view page. View result can return data to view page through which class is defined in the model. View page is a simple HTML page.

```
public ViewResult Gallary()
    {
       ViewBag.Message = "This is about gallary page";
       return View();
    }
```

# PartialView Result

Partial View Result is returning the result to Partial view page. Partial view is one of the views that we can call inside Normal view page.

We should create a Partial view inside shared folder, otherwise we cannot access the Partial view.

```
public PartialViewResult Index()
    {
        return PartialView("_PartialView");
    }
```

# Redirect Result

- Redirect result is returning the result to specific URL. It is rendered to the page by URL.

```
public RedirectResult Index()
        {
            return Redirect("Home/Contact");
        }
```

# Redirect to Action Result

Redirect to Action result is returning the result to a specified controller and action method. Controller name is optional in Redirect to Action method. If not mentioned, Controller name redirects to a mentioned action method in current Controller

```
public ActionResult Index()
        {
            return RedirectToAction("GetStudents", "Student");
        }
```

# Json Result

Json (JavaScript Object Notation) result is a significant Action Result in MVC. It will return simple text file format and key value pairs.

```
public JsonResult GetAllStudents()
        {
            List<Student> students = new List<Student>()
            {
                new Student
                {
                    StudentId = 1,
                    StudentName = "Alex"
                },
                new Student
                {
                    StudentId = 2,
                    StudentName = "Smith"
                }
            };

            return Json(students, JsonRequestBehavior.AllowGet);

        }
```

# File Result

File Result returns different file format view page when we implement file download concept in MVC using file result.

```
public ActionResult Index()
        {
            return File("Web.Config", "text");
        }
```

# Content Result

Content result returns different content's format to view. MVC returns different format using content return like HTML format, Java Script format and any other format.

```
public ActionResult Index()
        {
            return Content("<script>alert('Welcome To All');</script>");
        }
```

# References


1. C .Net Web Developers Guide by Syngress
2. ASP.NET 4.5, Covers C# and VB Codes, Black Book , Kogent Learning Solutions Inc.

# ASP.NET MVC

# What is ASP.NET MVC

Asp.Net MVC is a "Web Application Framework", that gives you a powerful pattern based on the way to build dynamic web applications, that enables clean separation of concerns and that gives you full control of over markup

**Main Advantage:**

- Clean separation of concerns
- Fast Performance

**Parts of Asp.Net**

- Web Forms
- Introduced in 2009,current version is “ASP.Net 5.2.5”

# What is MVC?

MVC is an architectural pattern that dictates you to write the application code as composition of three major parts.

**Model**

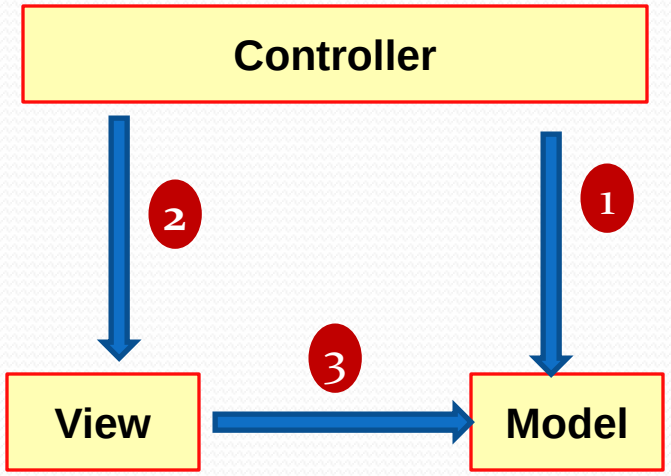
- Data Structure
- Business Logic

**View**

- Presentation Logic
- Reads Data from the model

**Controller**

- Defines execution flow
- Execution starts from controller
- Fills data into model object
- Pass model objects to view

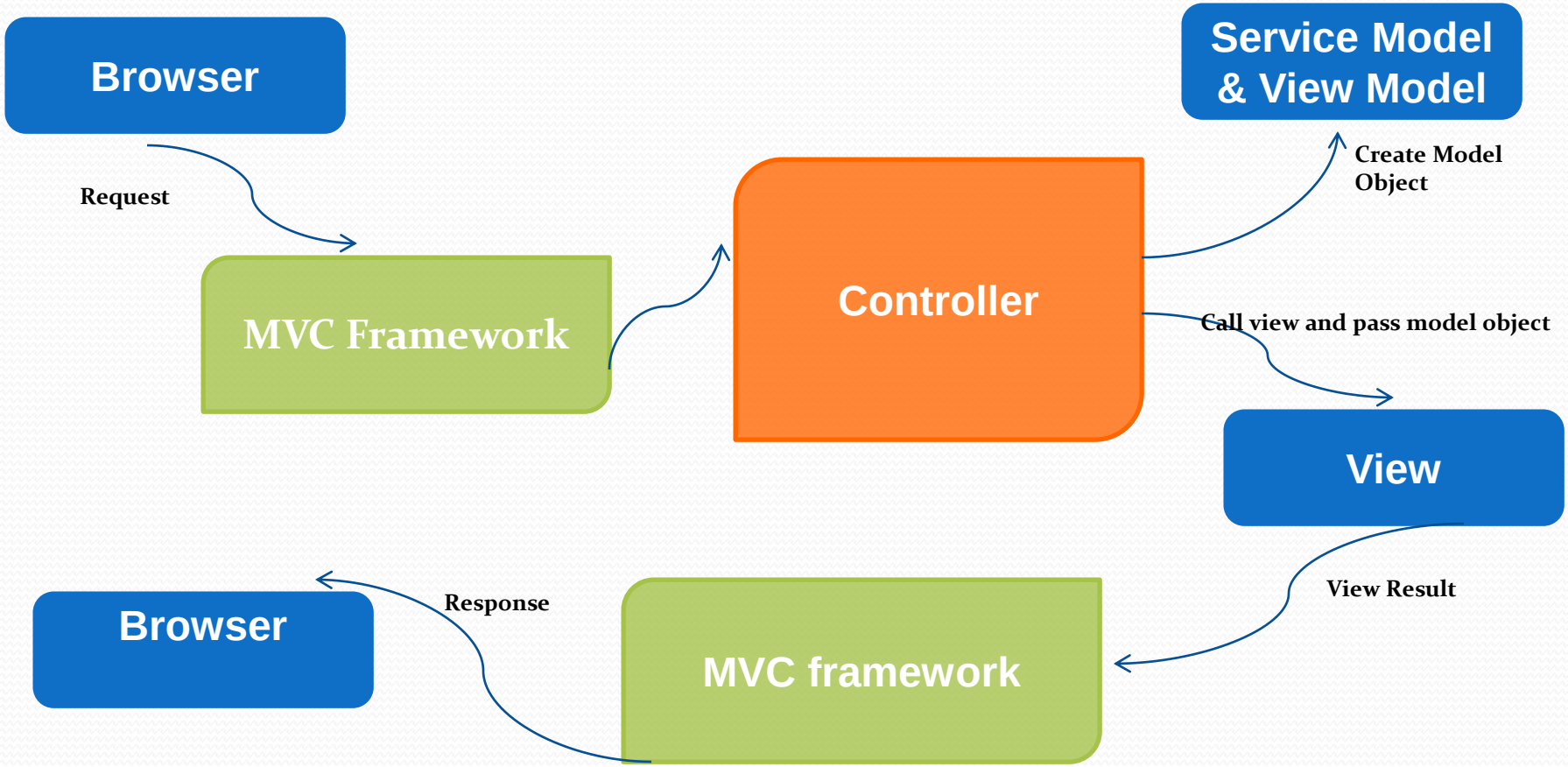


# Why ASP.NET MVC

- Advantages
  - supports "Clean Separation of Concerns".
  - Support Unit Testing
  - Supports Dependency Injection
  - Supports faster performance than ASP.net Web Forms.
  - No Page Life Cycle,controls,Postback and ViewState

# Controllers -Overview

- Controller is a class that defines execution flow in MVC application.
- Controller receives request from the browser, calls model, calls view



# Controller - Development

- Controller is a class
- Optionally, it is a public class
- Controller should be inherited from "**System.Web.Mvc.Controller**" class
- Controller's name should have a suffix called "**Controller**" **Ex:** HomeController

```
Public Class classnameController:System.Web.Mvc.Controller
{


}
```

# Action Methods

**Controller class with Action Methods**

```
Public Class classnameController:System.Web.Mvc.Controller
{
        public  returnType methodName(dataType param1)
        {
            ....
        }

}
```

# Versions of Asp.Net MVC

- Asp.Net  Mvc 1
- Asp.Net  Mvc 2
- Asp.Net  Mvc 3
- Asp.Net  Mvc 4
- Asp.Net  Mvc 5

# Asp. Net MVC 1.0

- March 2009
- Visual studio 2008
- .NET 3.5
- Features
  - MVC patterns with ASPX Engine
  - HTML Helpers
  - Routing
  - Unit Testing

- **ASP.NET MVC 2.0**
  - Visual Stuido 2008,2010
  - .Net 3.5,4.0
  - Features
    - Strongly Typed HTML Helpers
    - Support for data annotations

- **ASP.NET MVC 3.0**
  - January 2011
  - Visual Stuido 2008,2010
  - .Net 4.0
  - Features
    - Razor
    - EF code First
    - Partial Page Output Caching
    - View Bag
    - Global Filters

- **ASP.NET MVC 4.0**
  - August 2012
  - Visual Stuido 2010,2012,2013
  - .Net 4.0,4.5
  - Features
    - Asp.Net Web Api
    - Bundling and Minification
    - Asynchronous Controllers

# ASP.NET MVC 5.0

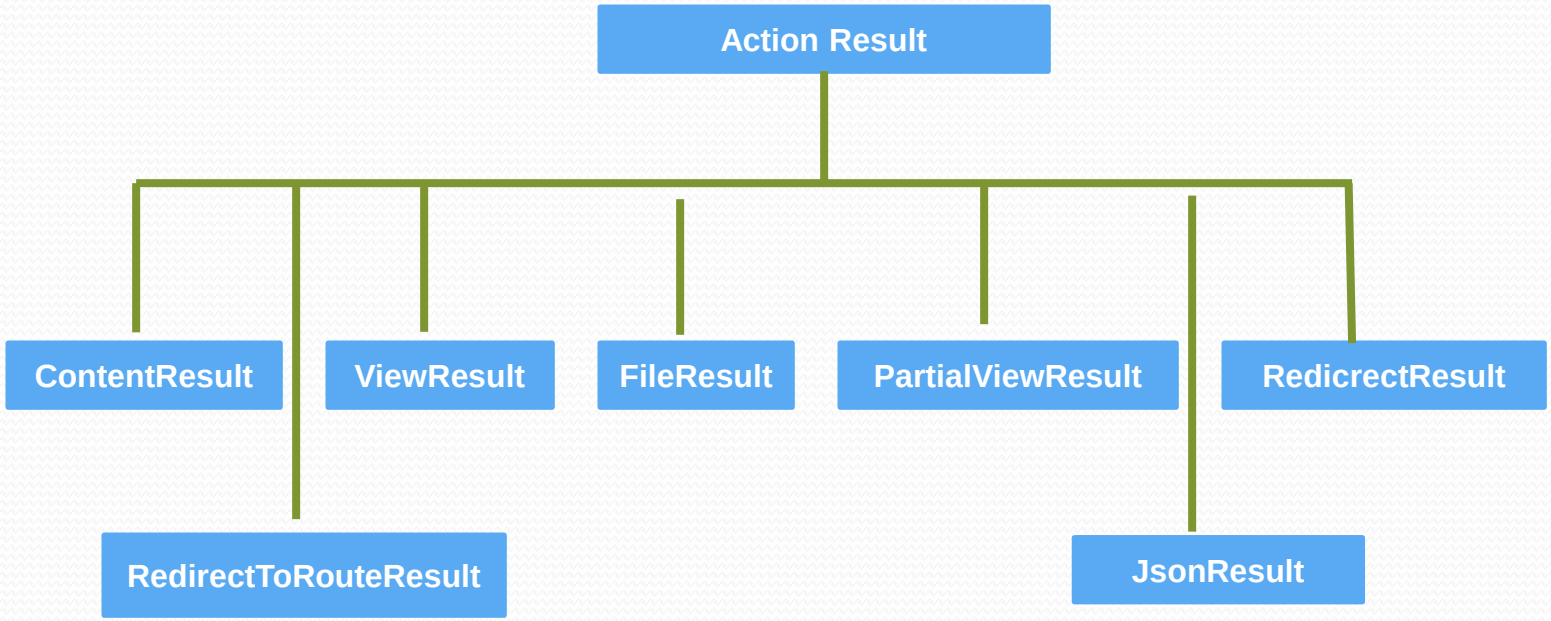
- October 2013
- Visual studio 2012,2013,2015,2017
- .Net 4.5,4.5.1 and up
- Features
  - Asp.Net Web Api
  - Asp.Net Identity
  - Attribute Based Routing
  - Filter Overrides

# Folder structure

| | |
|---|---|
| Project Folder | |
| \App_Start | Contains the files that need to be executed on the first request |
| \App_Data | Contains SQL Server LocalDB Database Files |
| \Controllers | Contains all controllers |
| \Models | Contains all models |
| \Views | Contain all views |
| \Views\web.config | Contain configuration settings for all views |
| \Global.asax | Contains application level and session level events |
| \packages.config | Contains the list of NuGet packages currently installed in the project |

# Action Result

- ActionResult is a class ,that represents “result of an action method”
- Asp.Net Mvc recommend to specify action methods’s return type as “ActionResult”
- ActionResult is an abstract class that has severa child classes,you can return any object of any of the child classes



# Type of Action Result

- ContentResult :Represents any content with specific content-type
- ViewResult :Represents result of a view
- FileResult :Represents content of a file
- JsonResult :Represents json object/json arrray
- RedirectResult :Represents redirection to other website(HTTP 302)
- RedirectToRouteResult :Represents redirection to a specific action method
- PartialViewResult :Represents the result of partial view

# View Engines

- View Engine provides a set of syntaxes to writes C#.net code(server side code) in the view
- View Engine is also responsible to render the view as html
- ASP.NET MVC supports two types of view engines
  - ASPX View Engine
  - Razor View Engine

**ASPX C View Engine**

```
<%
        C# code
%>
```

**Razor View Engine**

```
@{
        C# code
}
```

# ASPX (vs) Razor

**ASPX C View Engine**

```
<%
        C# code
%>
```

- ASPX is older view engine, supported in MVC
- File extension is .aspx
- Syntax is cumbersome, when used in real-world views
- You can't write the html tags in the code blocks

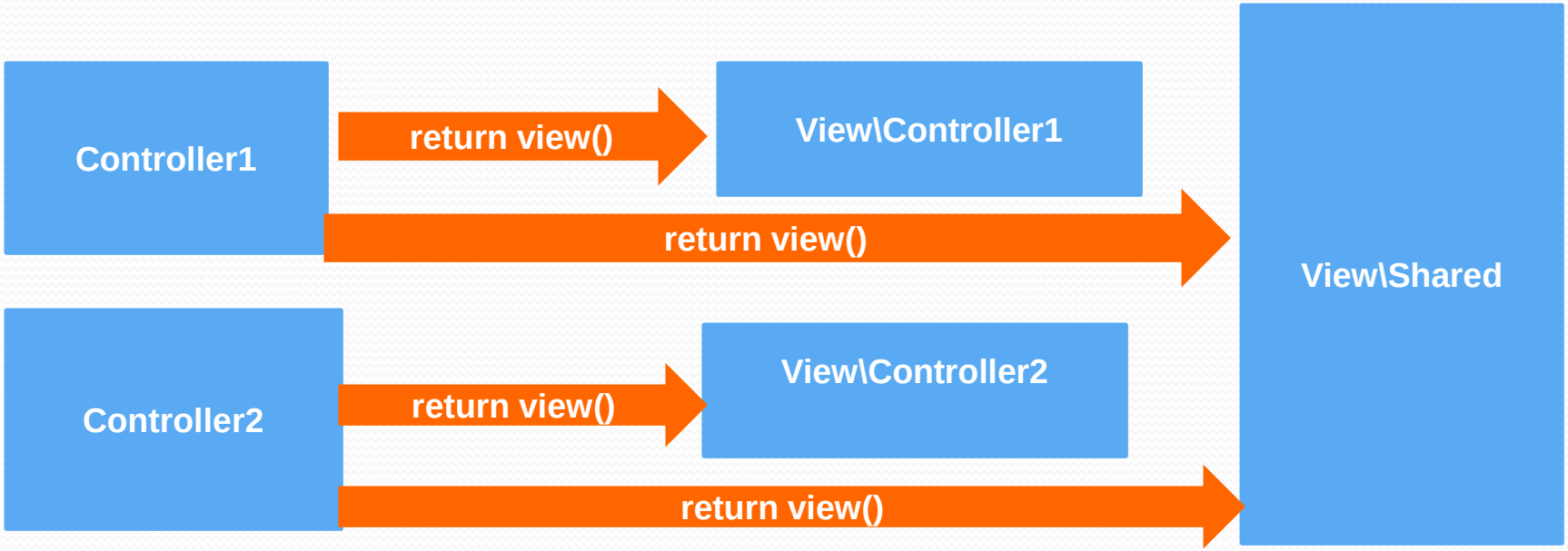
**Razor View Engine**

```
@{
        C# code
}
```

- Razor is latest and advanced view engine in MVC 3,4,5
- File extensions is .cshtml or (.vbhtml)
- Syntax is very clear ,clean and expressive
- You can write the html tags in the code blocks

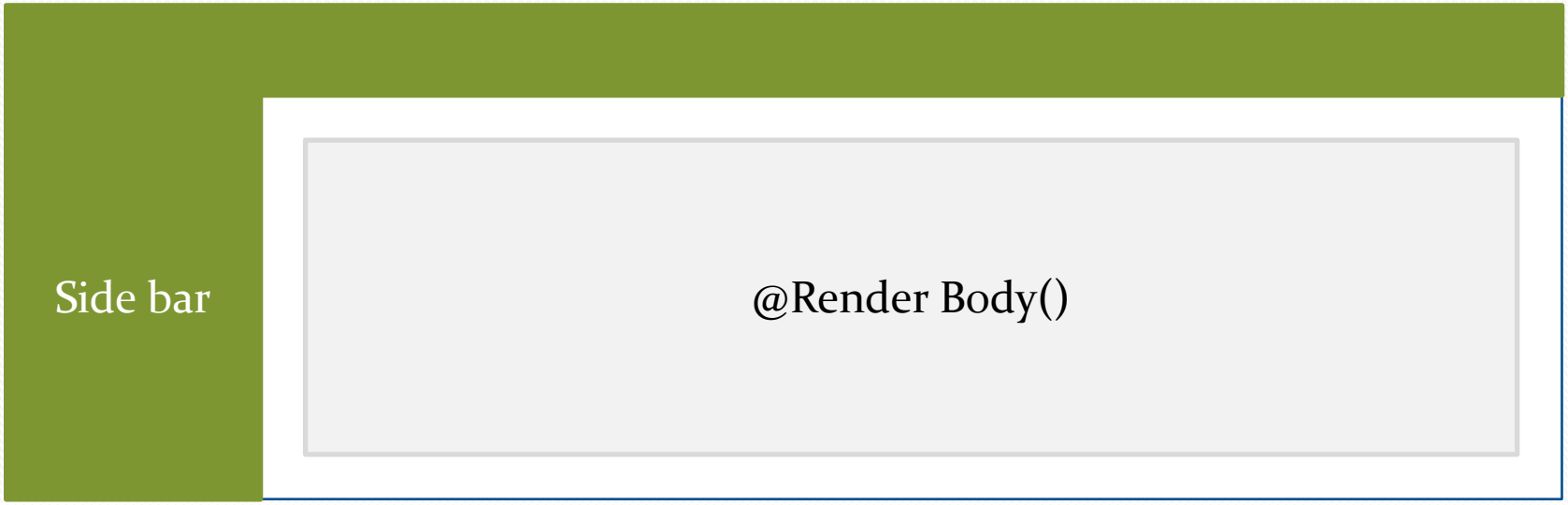
# Shared Views

- Shared views are present in the “View/Shared” folder
- Shared views are the views that can be called from any controller of the entire project
- The views that belongs to all controllers are created as shared views
- When we call a view ,it checks the for the view in the “Views\controllername” folder first ,it is not found ,it will search in the “Views\Shared “ folder



# Layout Views

- Layout views contain “page template", which contains common parts of the UI ,such as logo,header,menubar side bar etc
- @RenderBody() method represents the reserved area for the actual content of view
- Execution Flow: Controller ->view ->Layout view ->Rendered View Result ->Send response to browser



# Section in Layout Views

- Section are used to display view-specific content in the layout view
- Section are defined in the view and rendered in the layout view

**Layout View**

```
@RenderSection("section name")
```

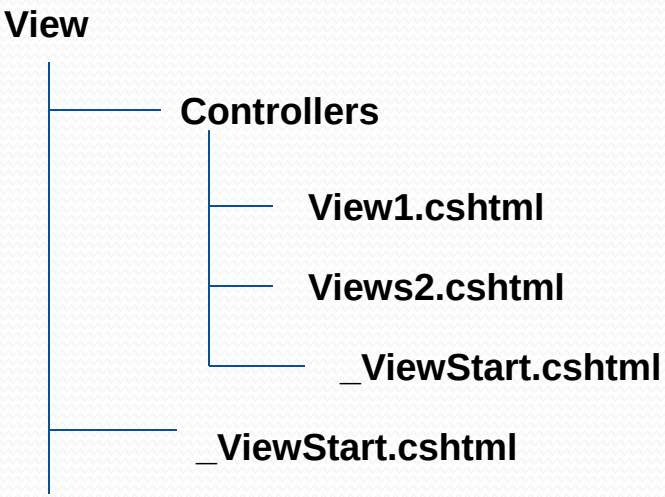
**View**

```
@section name

{
Content here
}
```

# _ViewStart.cshtml

- It defines the default Layout view of all the views of a folder.
- If it is present in “View” folder ,it defines the default Layout view of all views of entire project
- If it is present in “Views\Controllername” folder ,it defines the default layout view of all the views of same controller only
- **Flow of Execution:** Controller->_viewstart.cshtml of "views” folder -> “_viewstart.cshtml of “Controller1”
- Folder-> View->Layout ->Generate View Result ->Response

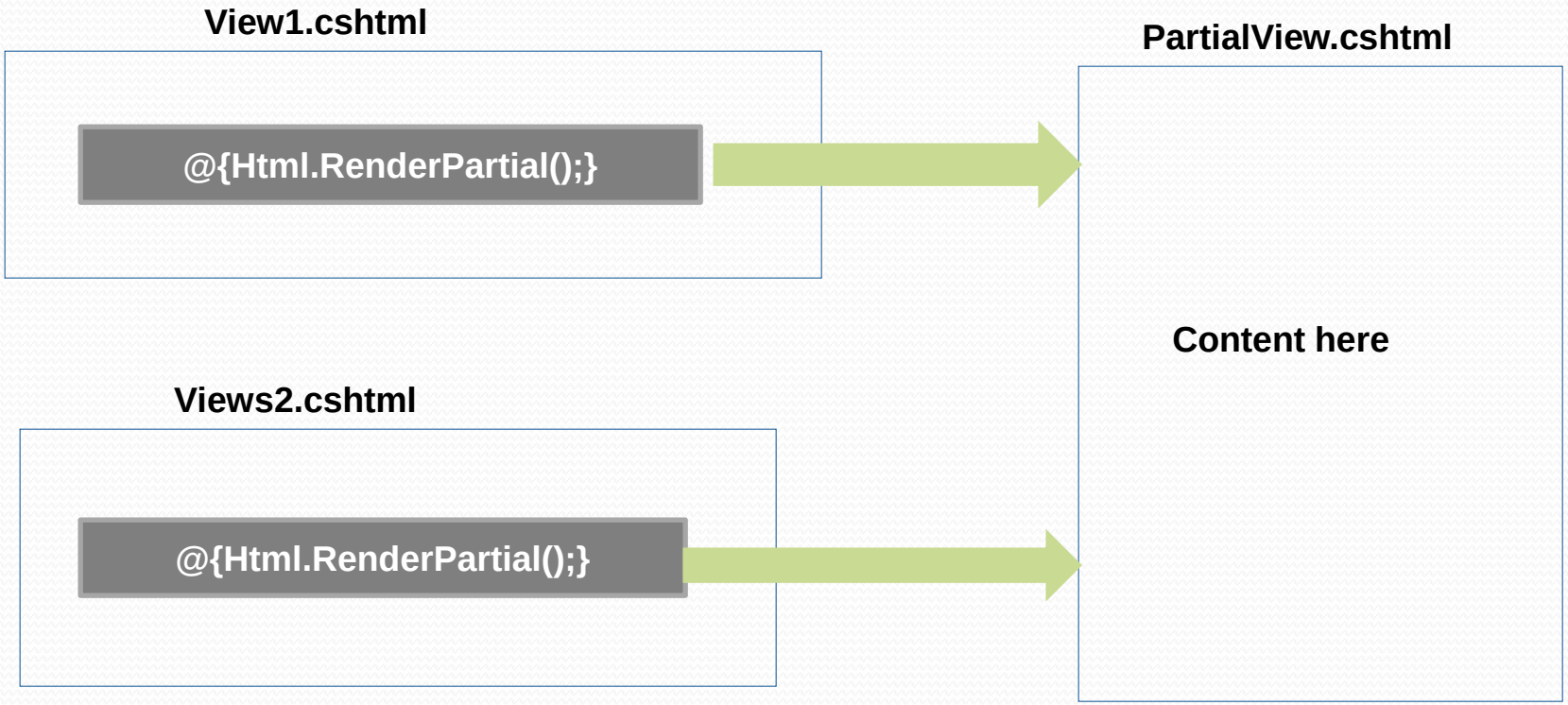



**Layout View**

```
@{
Layout =“Path of Layout  view”
}
```

# Partial View

- Partial  View is a small view that contains content that can be shared among multiple views
- Can be present in “View\Controllername” folder or in “views\Shared” folder

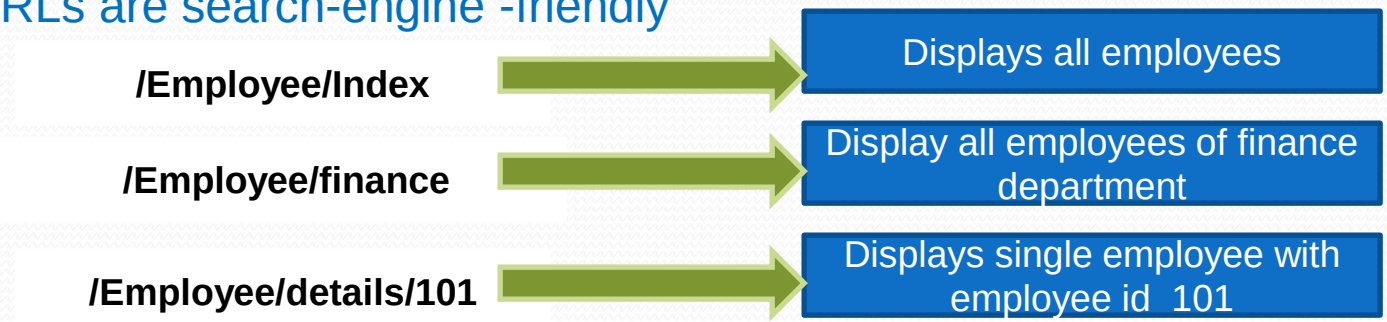


# View (vs) partial View

| View | Partial View |
| --- | --- |
| • View can contain a layout page<br>• _viewStart.cshtml file will be called before the execution of view<br>• View can have html structured elements such as html,head ,body etc | • Partial view doesn’t contain a layout page<br>• _viewStart.cshtml file will not be called before the execution of partial view<br>• Partial view doesn’t contain any html structure elements such as html ,head ,body |

# URL Routing

- URL Routing is a pattern-matching-system that monitors the incoming request url and figure out what to do with that
- It allows you to create meaningful URLs, instead of mapping to physical files on the server
- Advantages:
  - Makes the URL not to map to physical files on the server
    - **Old URL Ex /folder/subfolder/search.aspx**
  - URLs are user-friendly. **New URL Ex: /products/search**
  - URLs are search-engine -friendly



# Route

- Rout is a URL pattern which includes literals/parameters
- Literal is a fixed text that must be present in the URL
- Parameter is a variable ,which value can be entered by the user.

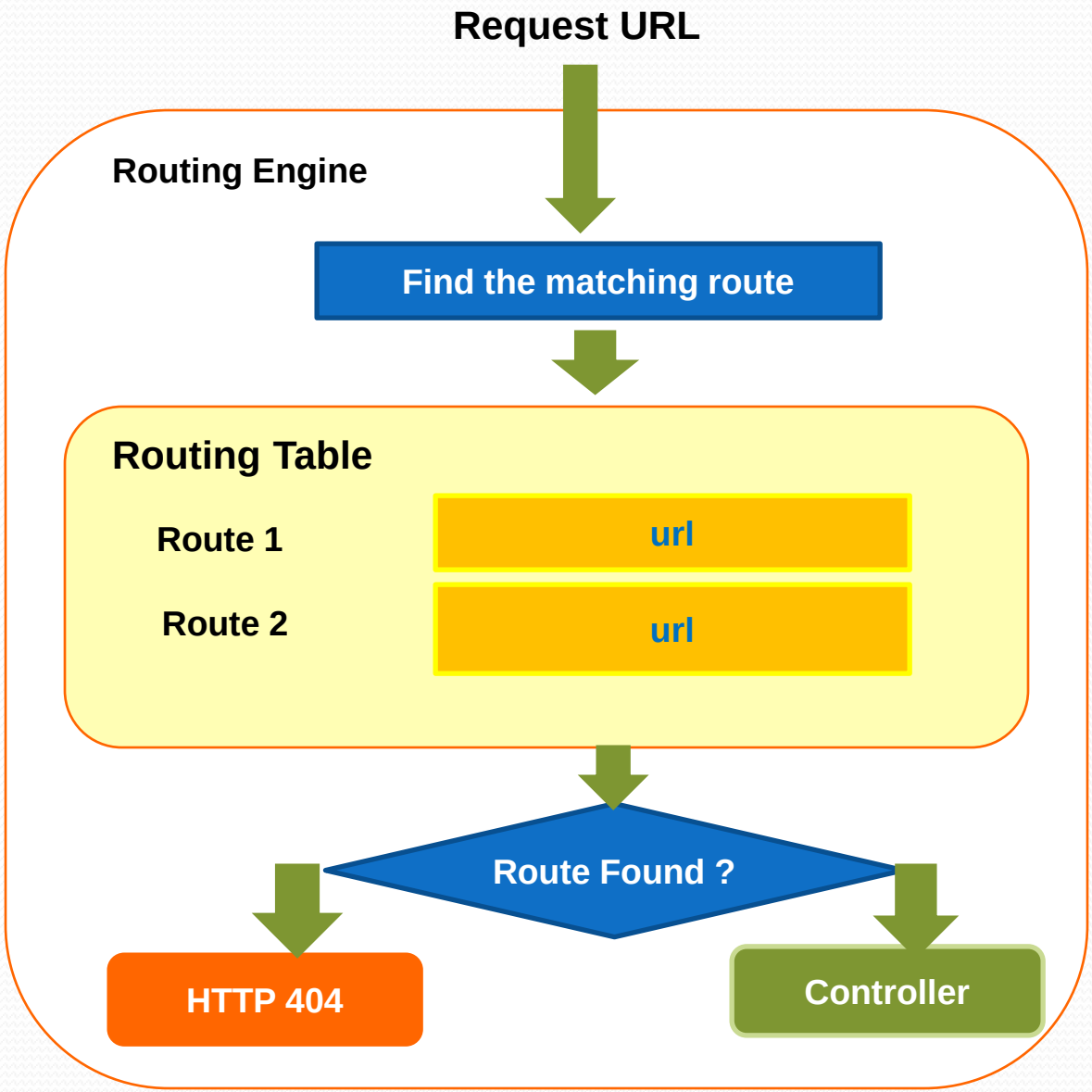
**Employee/details/{empId}**

**Employee/details/101**

**Employee/details/102**

**Employee/details/103**

# Routing table




- Routing table contains the list of routes
- When the request is received from the browser ,the routing Engine (part of Asp.Net MVC framework) searches whether the actual URL is matching with any on of the routes in the RouteTable,IF one matches ,it goes to the corresponding controller

# URL Routing -Development

**Global.asax**

```
protected void Application_Start()
    {
        RouteConfig.RegistrationRoutes(RouteTable.Routes);
}
```

**RouteConfig.cs**

```
public static void RegisterRoutes(RouteCollection routes)
    {

        routes.MapRoute(
        name: “route1",
       url: "{param1}/{param2}”,
     defaults: new {param1:”defaultvalue1” ,param2:”defaultvalue”}    ,
   Constraints: new {param1: “constraint “, param2: “constraint”}     );    }
```

- Application_start() invokes registerRoutes method of RouteConfig class and passes Routes property of RouteTable class
- The Routes property of RouteTable class is  a of “RouteCollection” type which defines the actual routes
- We can add routes into the RouteCollection,using “MapRoute” method
- All the routes tthat are added to the RouteCollection on,will be stored in the RouteTable for the first request

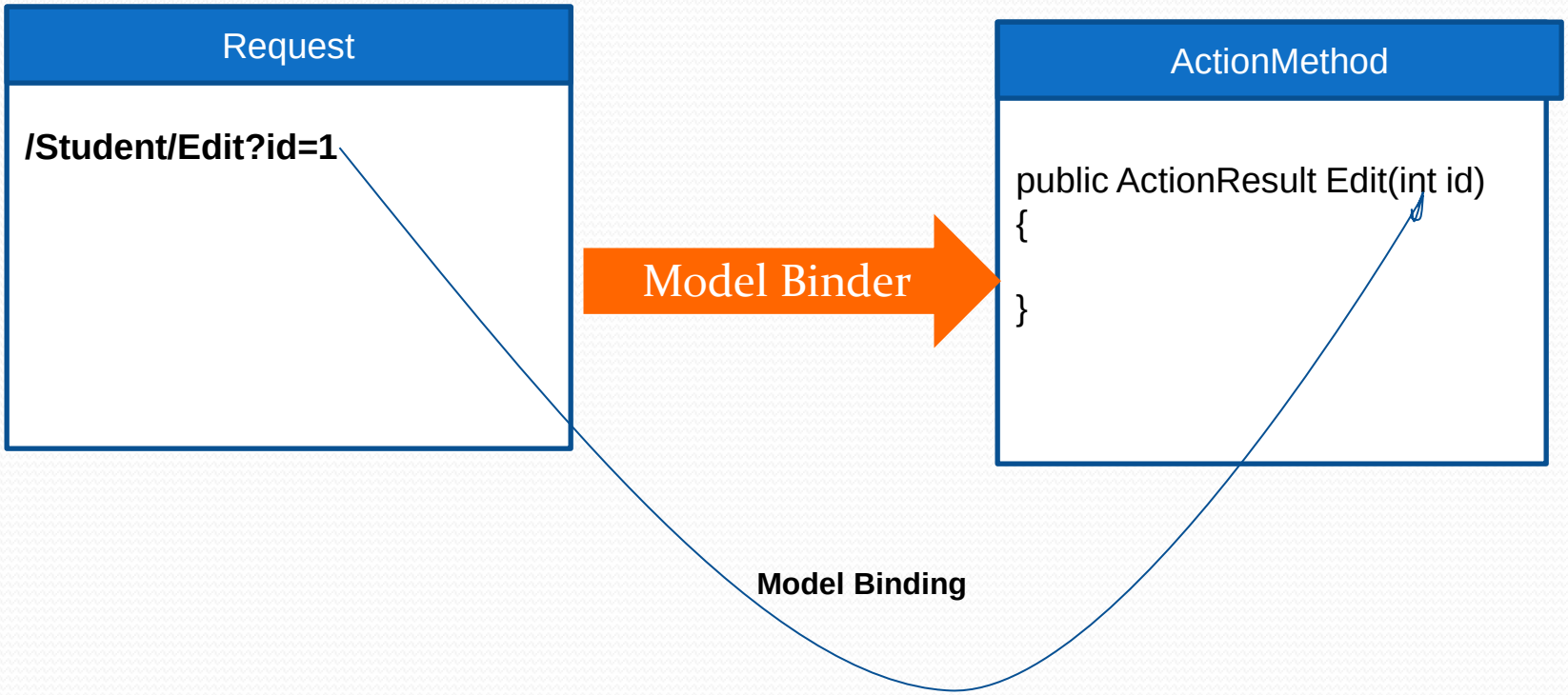
# Model

- Model is a class that defines structure of the data that you want to store /display
- Also contains business logic
- Model will be called by controller and view

| View Model | Domain Model | Server Model |
| --- | --- | --- |
| public class View Model<br>{<br>public datatype<br>propertyname{get;set;}<br>} | public class Domain Model<br>{<br>public datatype<br>propertyname{get;set;}<br>} | public class server Model<br>{<br>public datatype<br>propertyname{get;set;}<br>} |
| **Represents the structure of the data that you want to display to user** | **Represents the structure of the data that you want to store in the database table** | **Represents business logic(code that needs to be executed before inserting /updating etc..** |

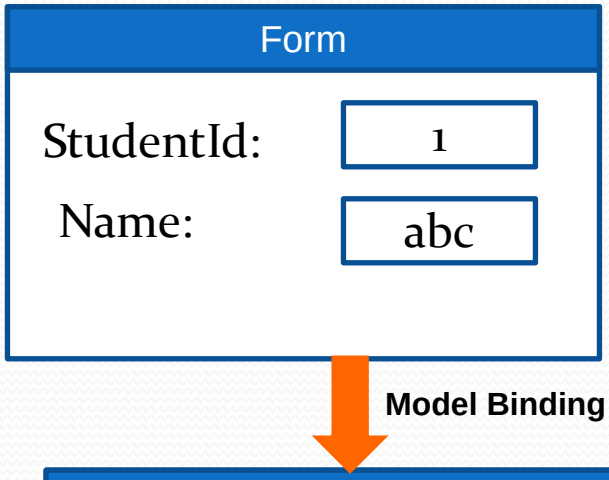
# Model Binding

- Process of “receiving values from different sources of the request and passing them as argument to action method”
- Assigns values to different parameters of the action method automatically!



# Bind Attribute

- The [Bind] Attribute allows you to specify the list of properties that you want to bind into the model object, that is received using “Model Binding”
- It allows you to specify “Include “ and Exclude “ comma-separated list of properties




**Model Binding**

Model class

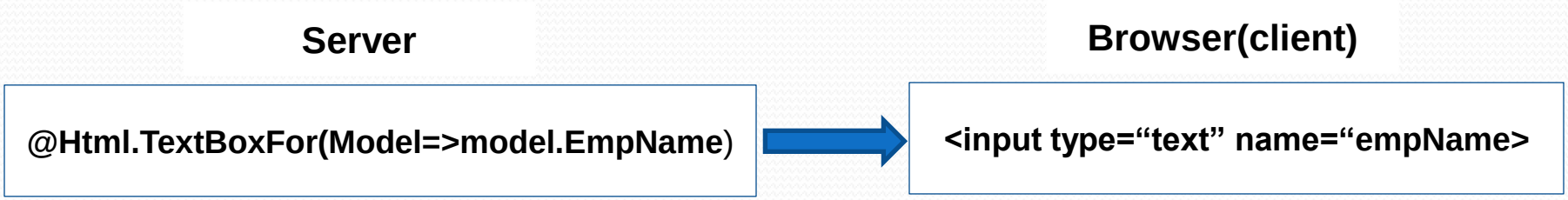
```
public class Student
{
public int Student ID {get;set}
public string Studentname {get;set;}
}
```

Action Method

```
public ActionrEsult Create([Bind (include=“StudentId,StudentName”]Student std)
{
…..
}
```

# HTML Helper

- Binds HTML elements to Model Properties .
- Pre-defined methods that execute on server and generate (render) a specific html tag at run time.
- @Html is an object of HtmlHelper class



# List of HTML Helper

| | |
|---|---|
| HtmlActionLink | Hyperlink |
| HtmlTextBoxFor | Texbox |
| HtmlTextAreaFor | TextArea |
| HtmlCheckBoxFor | Checkbox |
| HtmlradioButtonFor | RadioButton |
| HtmlListBoxFox | Dropdown |
| HtmlHiddenFor | Multi-Select List |
| HtmldisplayFlr | Plain text |
| HtmlLableFor | label |
| HtmlEditorFor | TextBox/TextArea/Numeric TextBox /date textBox |

# Custom Helper

- It is a static method, invoked in the view ,that renders a set of html tags
- Advantage: We can pass model object to the html helper method

```
Public  static class classname
{
    public static MVCHtmlString Methodname(this HtmlHelper helper
,arguemtns)
{
                code here
                return new MvHtmString("html tags");
}
}
```

# Validations

- Defined by using data Annotations
- Data annotations are provided by System.ComponentModelDataAnnotations namespace

```
Public class Employee
{
    [Required]
     public int EmpID{get;set;}

}
```

# Data Annotations for validations

- [Required] Field is mandatory
- [Maxlength] Maximum no of characters
- [Minlength] Minimum no of characters
- [Range] value should be within the min and max
- [Compare] Two fields must be same
- [RegularExpression] Pattern of value
- [EmailAddress] Email address only accepted

# HTML  Helper for Client side Validations

- HTML Helper:
  - ValidationMessageFor :Displays error message
  - ValidationSummary :Displays validations summary

# Custom Validations

- Used to implement user-defined validations
- Create a class that extends validationsAttribute class
- IsValid method executes after submitting the form(after Model Binding)
- Isvalid method receives the input value as argument
- ValidationContext provides details about current model property's details
- Is Valid method return either"Success" or "validationResult with error message"

# Filters

- Filters Execute  at a specific situations ,while executing an action method
- Authentication Filters
  - Executes first ,before any other filter ,to check whether the use is a valid user or not
  - Implemented using IAuthenticationFilter
- Action Filter
  - Executes before and after of action method execution, You can modify the ViewBag,before sending it result
  - Implemented using IActionFilter
- Result Filter
  - Executes ebfore and after of result execution. You can modify the result
  - Implemented using IResultFilter

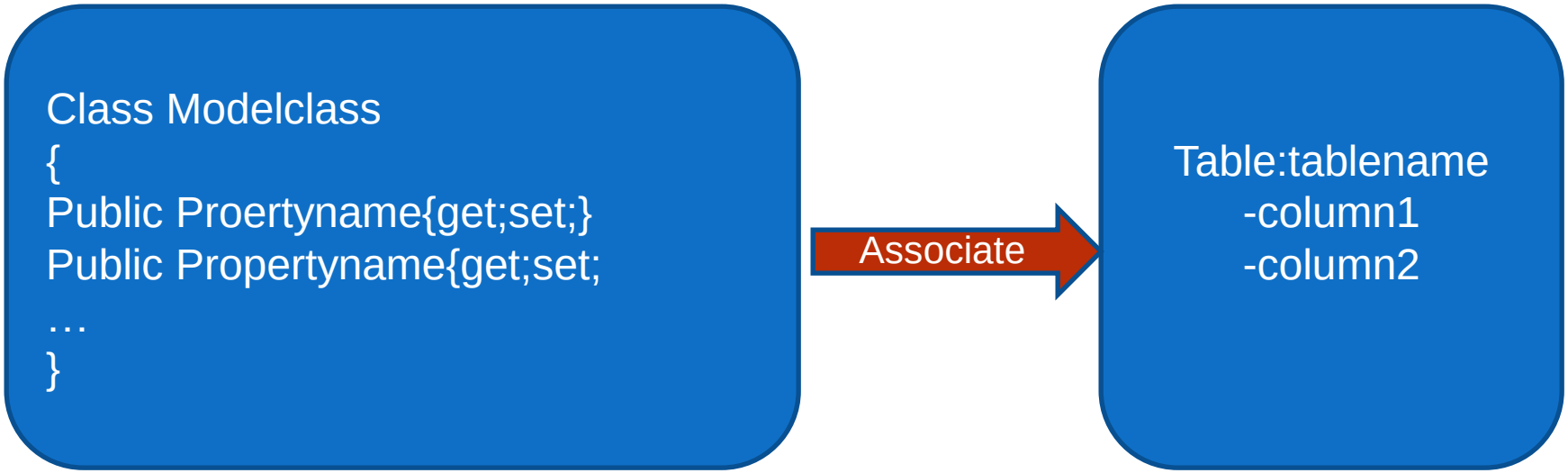
# Built-in Filters

- Authorization Filters
  - [ChilActiononly]
  - [ValidateAnitForgeryToken]
  - [Authorize]
- Action Filters
  - [ActionName]
  - [NonAction]
  - [HttpGet]
  - [OutPutCache]
- Result Filters
  - (No Filters)
- Exception Filters
  - [HandleError]

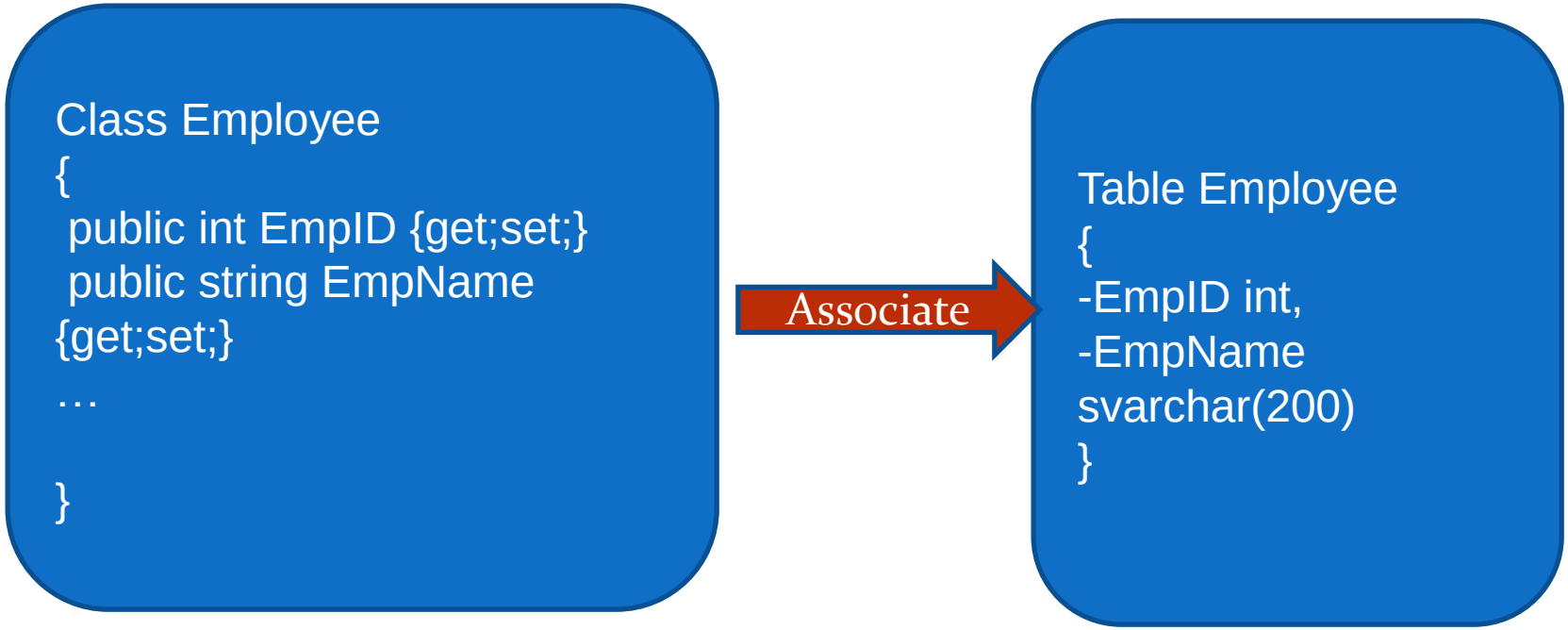
# Entity Framework

# Entity Framework

- Entity Framework (EF) is a database technology ,which is built based on ADO.NET ,that supports ORM(Object-Relational Mapping) pattern
- It bridges between “Object Oriented Programming and ‘relational databases", using Model Classes



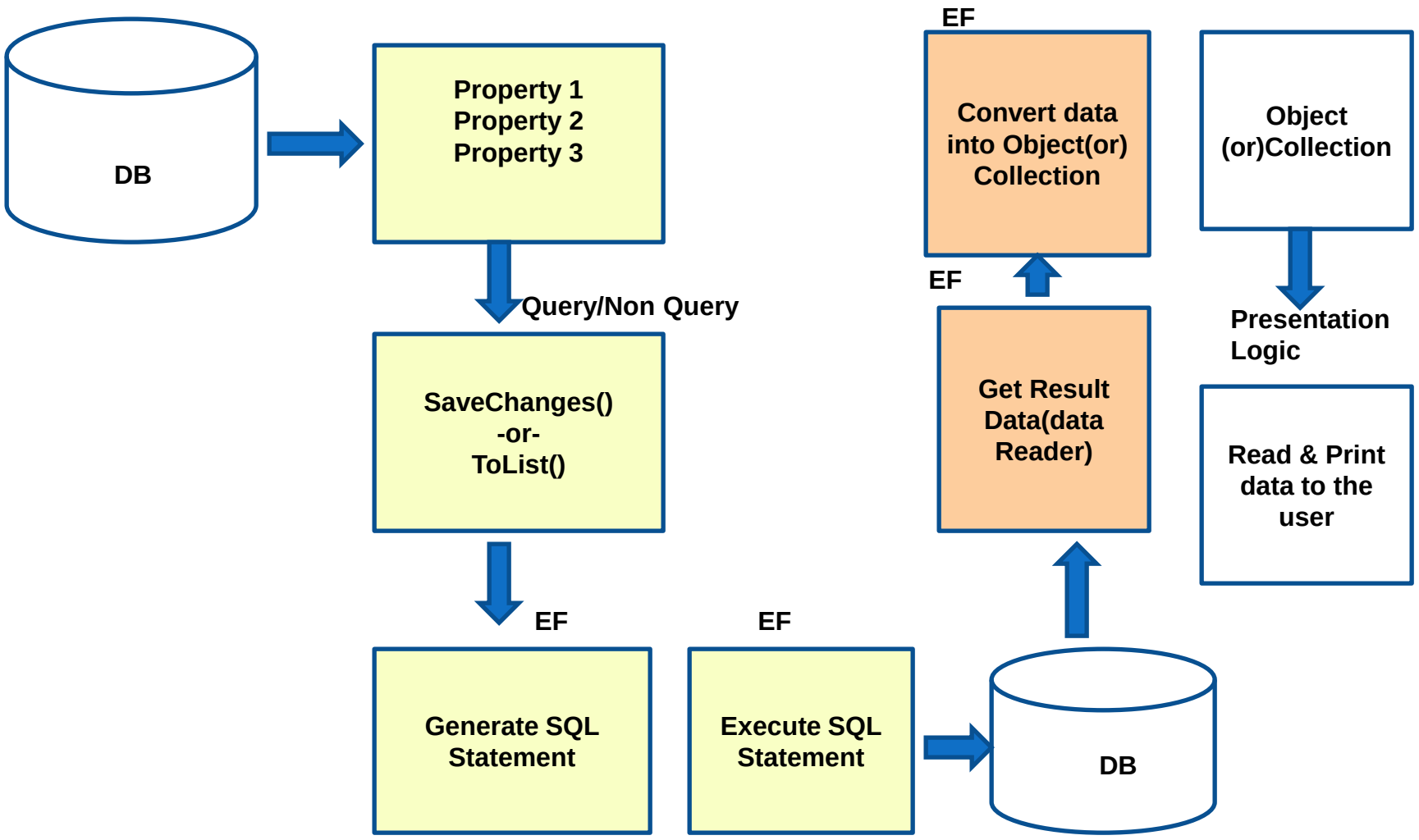
# Object-Relational mapping ORM



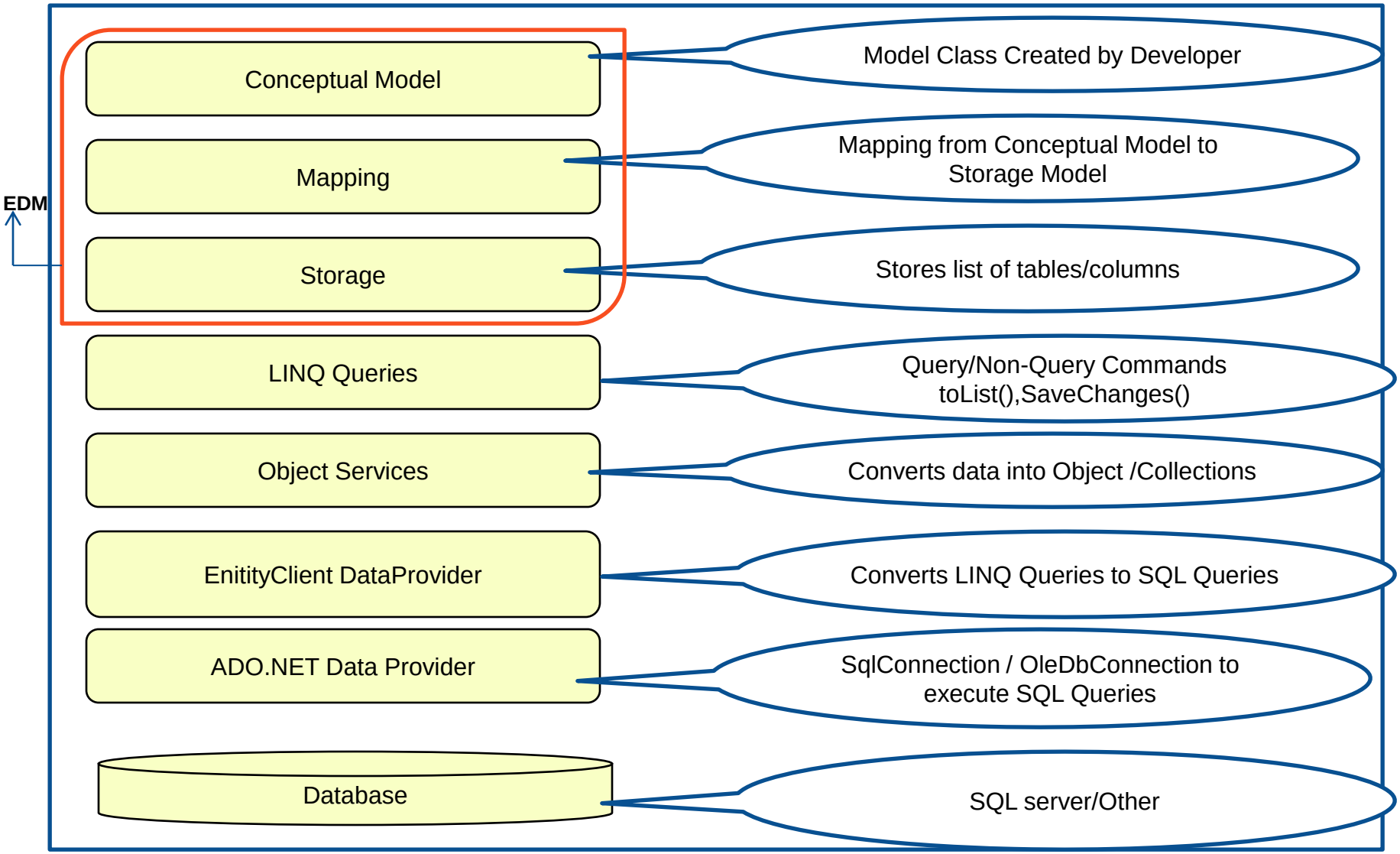
# Features of EF

- Modeling
- Querying
- Change tracking
- Saving
- Concurrency
- Transactions
- Caching

# EF Work Flow

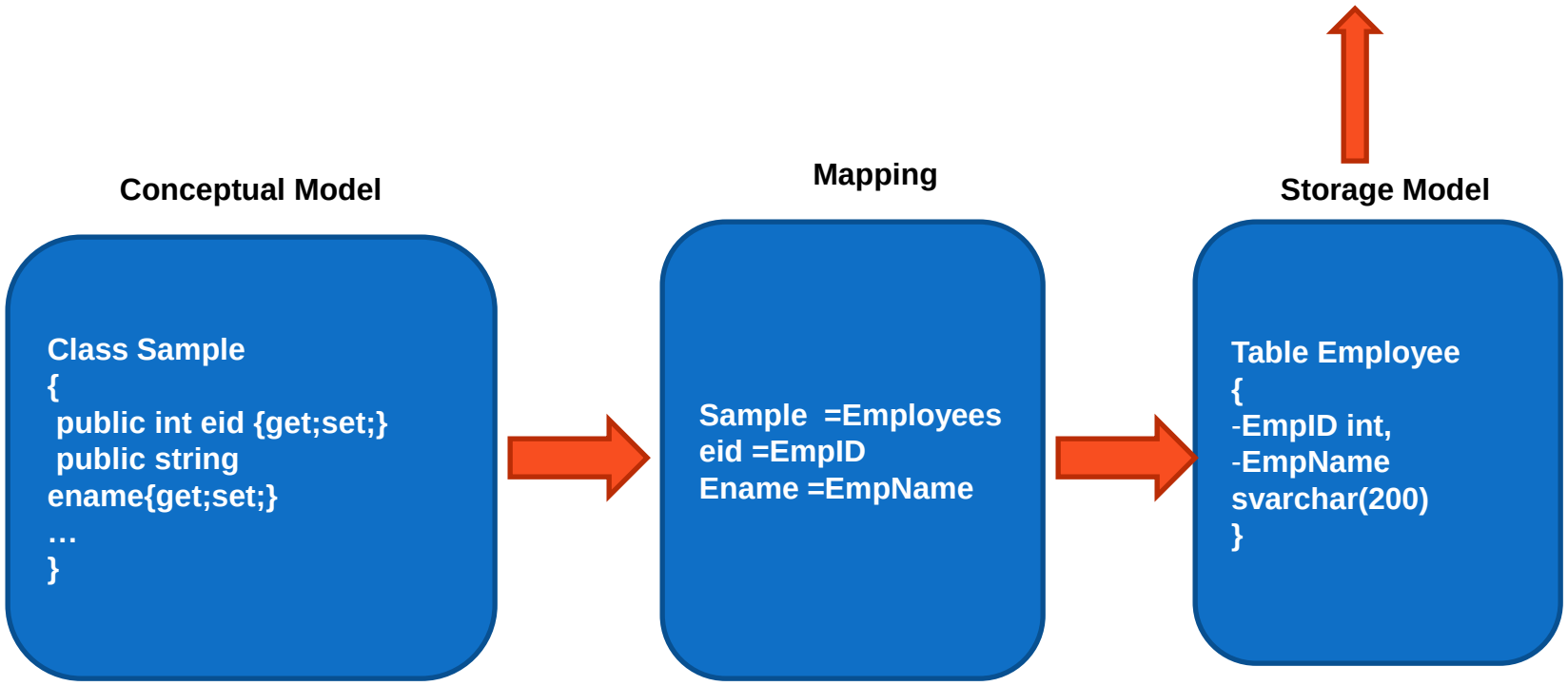

DB
Property 1
Property 2
Property 3
Query/Non Query
SaveChanges()
-or-
ToList()
EF
Generate SQL Statement
EF
Execute SQL Statement
DB
EF
Get Result Data(data Reader)
EF
Convert data into Object(or) Collection
Object (or)Collection
Presentation Logic
Read & Print data to the user

# EF Architecture



# Entity Framework

| EmpID | EmpName |
| --- | --- |
| | |
| | |



# DbContext and DbSet

- DbContext is a class ,based on which you can write LINQ queries to perform CRUD operations on table
- DbContext is a collection of DbSet's
- DbSet object that represents a table

```
Using Modelclassname;
Class Classname:DbContext
{
        public Dbset<Modelclass>class{get;set;}
        public Dbset<Modelclass>class{get;set;}

}
```

# Windows Communication Foundation

# What is WCF?

- WCF stands for Windows Communication Foundation. It is a framework for building, configuring, and deploying network-distributed services.
- Microsoft's unified programming model (the service model) for building service-oriented Applications.
- Communication between two application

# WCF Design Goals

- *"A unified programming model for building service-oriented applications on the Windows platform"*

**Unification**

- Unifies today's distributed technology stacks
- Compos able functionality
- Appropriate for use on-machine ,in the internet and cross the internet

**Productive Service-Oriented Programming**

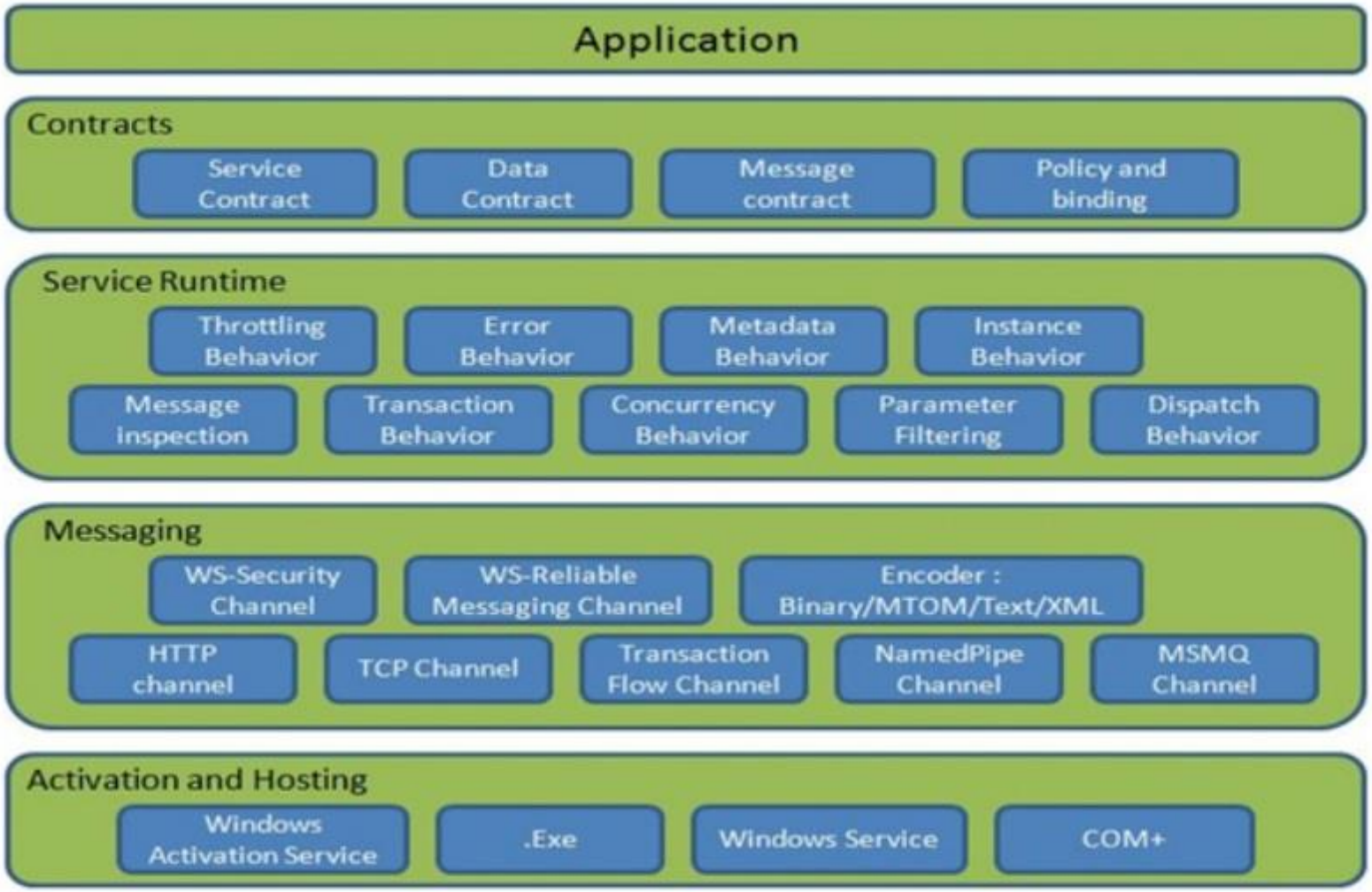
- Service-oriented programming model
- Improve developer productivity

**Interoperability & Integration**

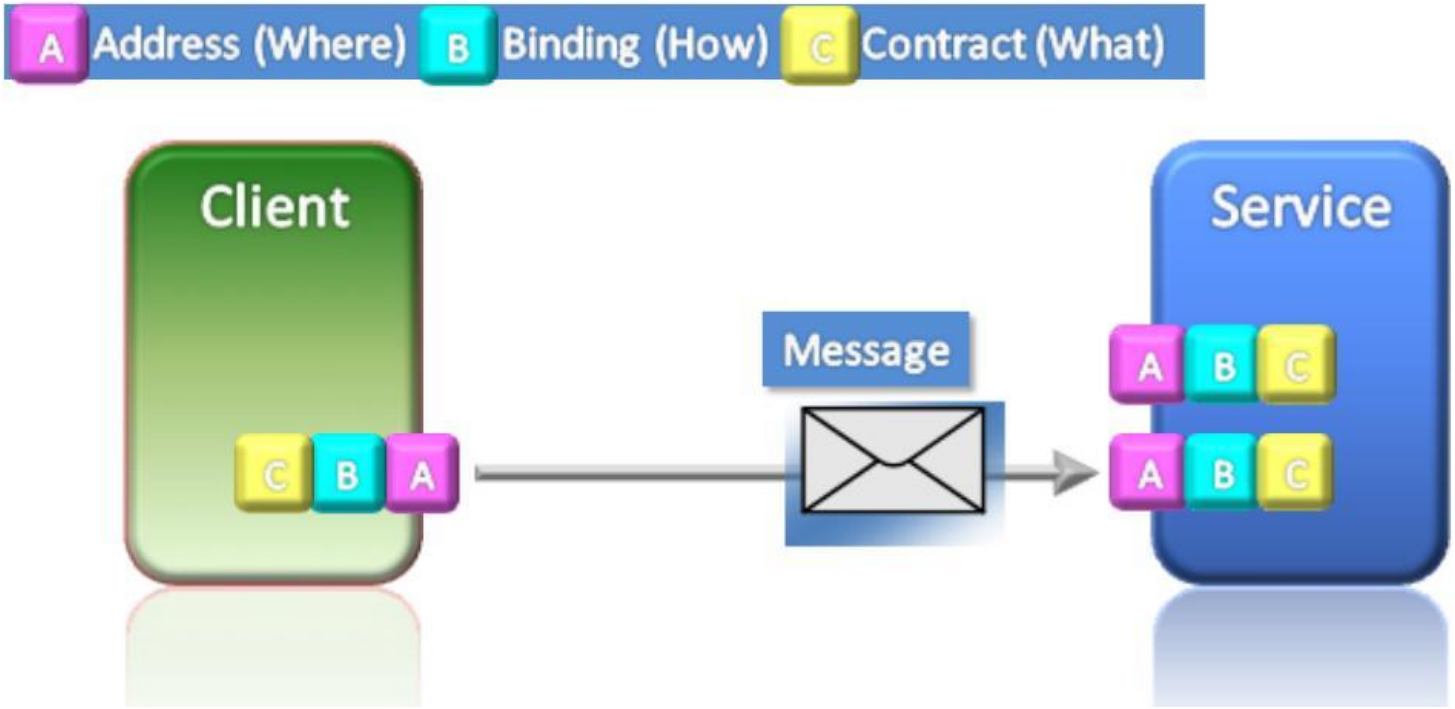
WS-* interoperability with applications running
on other platforms
Interoperability with today's distributed stacks

# Different technology combined to form WCF

# WCF Architecture



# The ABC of WCF



# WCF Endpoints

Every service has

- Address
  - Where the service is
- Binding
  - How to talk to the service
- Contract
  - What the service can do

# Address

- Defines where a service is located
- Specifies a URI where the service is located

  -Relative or absolute
- Address consist of

  -Scheme

  HTTP,TCP,Named pipes,MSMQ

  -Machine

  -Port

  -Path
- Examples

  - http://www.mystore.com/StoreFront

  – net.tcp://mycomputer:9000/StoreFront

# Bindings

- Describes ***how a service communicates***
- Specifies set of binding elements
  - Transport; http, tcp, np, msmq
  - Encoding format; text, binary, MTOM, ...
  - Security requirements
  - Reliable session requirements
  - Transaction requirements
- Set of predefined standard bindings
  - Can be customized
- Custom binding

# Types of Bindings

- BasicHttpBinding
- WSHttpBinding
- WS2007HttpBinding
- WSDualHttpBinding

# Types of Bindings

- WSFederationHttp Binding
- WS2007FederationHttpBinding
- NetTcpBinding
- NetNamePipeBinding
- NETMSMqBinding

# Contracts

- Defines ***what*** *a service communicate*
- Service Contracts
  - Describe the operations a service can perform
  - Map CLR types to WSDL
- Data Contracts
  - Describes a data structure
  - Maps CLR types to XSD
- Message Contracts
  - Defines the (application specific) structure of the message on the wire
  - Maps CLR types to SOAP messages

# Contracts

- Service Contracts
  - Describe which operations the client can perform on the service.

- Data Contracts
  - Define which data types are passed to and from the service. WCF defines implicit contracts for built-in types such as int and string, but you can easily define explicit opt-in data contracts for custom types

# Contracts

- Fault Contracts
  - Define which errors are raised by the service, and how the service handles and propagates errors to its clients
- Message Contracts
  - Allow the service to interact directly with messages. Message contracts can be typed or untyped, and are useful in interoperability cases and when there is an existing message format you have to comply with.

# Service Contracts

- **[ServiceContract] – Defines a 'set' of operations**
- **[OperationContract] – Defines a Single method**

```
[serviceContract]
Public interface Iservice
{
    [OperationContract]
    string GetData(int value);
}
```

```
Public class ConcreteService:Iservice
{
  public string GetData(int value);
  {..}
  public string OtherMethod()
  {...}

}
```

# Data Contracts

- **[DataContract] –Specifies type as a data contract**
- **[datamember] –Members that are part of contract**

```
[DataContract]
Public class Customer
{  [DataMember]
   public bool MyFlag { get; set;}
[DataMember]
   public string mystring {get; set;}}
[DataMember]
   public string mystring {get; set;}}
```

# Behavior

- Modifies or extends service or client runtime behaviour
- Examples
  - Instancing; Singleton, Private Session, shared Session, Per Call
  - Concurrency; Multiple, Reentrant, Single
  - Throttling; connections, threading
  - Metadata customization
  - Transactions; AutoEnlist, Isolation, AutoComplete

# Metadata Exchange

- Service can also expose endpoint for Metadata Exchange(MEX)
- It provides a mechanism for clients to find out about:
  - Address of other end points
  - Bindings that are used
  - Contracts used –Service, Operation, Data,etc

# Advantage

- Its made of a lot of different components, so you can create new components for security, transport, authentication.

- In WCF, there is no need to make much change in code for implementing the security model and changing the binding. Small changes in the configuration will make your requirements

# Advantage

- Its faster than ASMX(Web Service)
- Its interoperability for java and more
- WCF is interoperable with other services when compared to .Net Remoting ,where the client and service have to be .Net
- WCF services provide better reliability and security in compared to ASMX web services.

# Disadvantage

- WCF is Microsoft's implementation of SOA and hence its API are solely controlled by MS which makes interoperability a bit difficult.

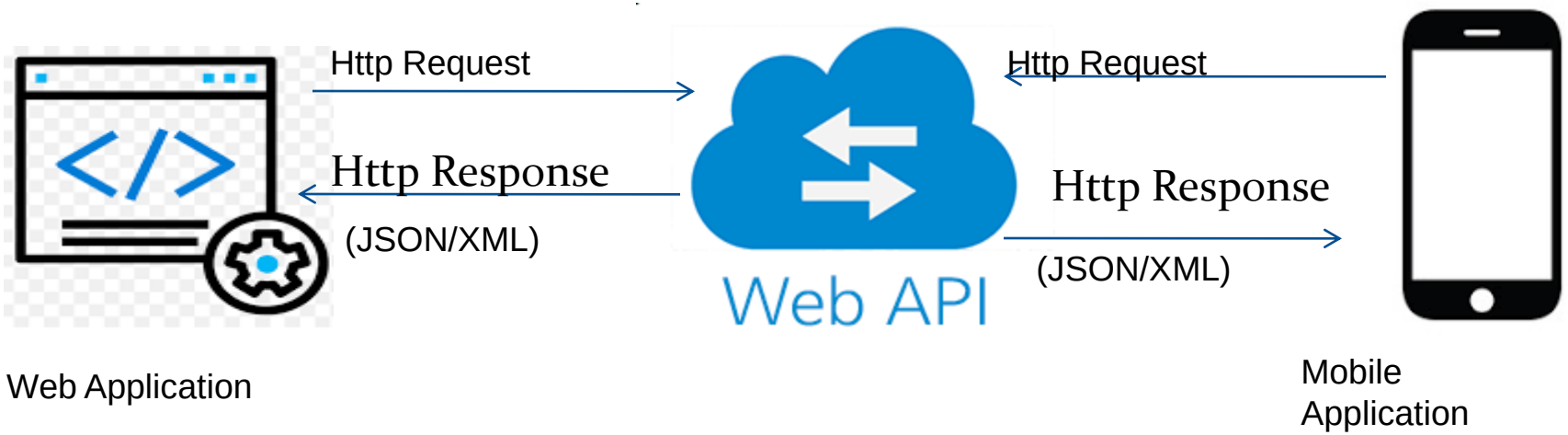
- To deploy WCF apps, need more underlying hardware resources on the platform on which the WCF applications will be running ,since there is an additional layer of abstraction to deal with.

# Thank You

# WEB API

# Introduction

“Web Api” Framework is used to create HTTP services, that can be called from any application
Web Api service/Http Service executes when the browser sends request using AJAX .It performs CRUD operations on database table



# Advantages of Web API

- Can be accessible from any application
- Supports RESTful standards (GET,POST,PUT,DELETE)
- Supports Automatic Model Binding
- Supports JSON Serialization
- Maps HTTP verbs to methods

# HTTP Verbs

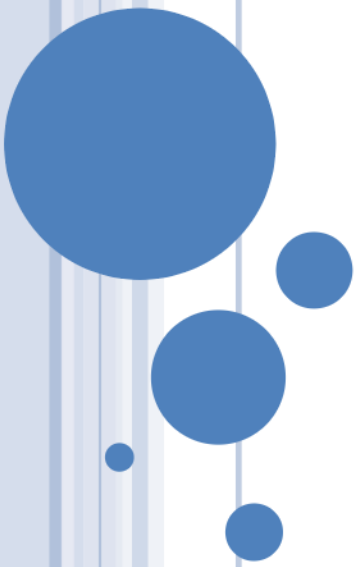
- Get : Used to retrieve /search data from server
- Post : Used to insert data to server
- Put : Used to update data on server
- Delete : Used to delete data from server

# Web Api Controller /Http Service

It receives request from browser and returns response to browser

```
Using System;
Using System.Web.Http

namespace namespacename
{
    public class controllername:ApiController
    {
            public returntype Methodname()
            {
                    return value;
            }
    }
}
```

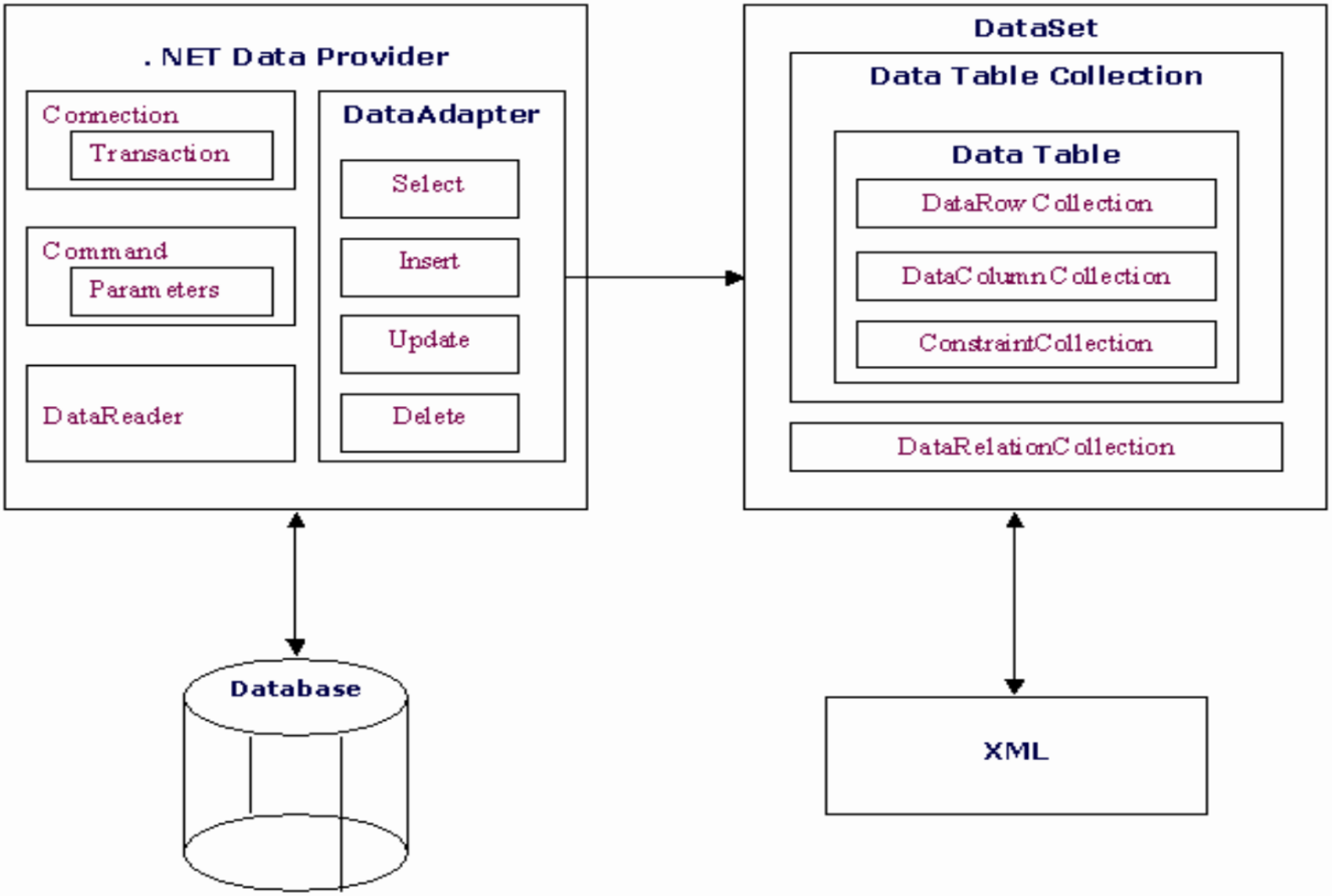


# ADO.NET

# Introduction to ADO.NET

- Microsoft ADO.NET is part of the Microsoft .NET Framework: a set of tools and layers that allows your application to easily manage and communicate with its file-based or server-based data store.

- In the .NET Framework, the ADO.NET libraries appear under the *System.Data* namespace.




ADO .NET Data Architecture

# ADO.NET Architecture

- Data Access in ADO.NET relies on two components: *DataSet* and *Data Provider*.
- The dataset is a disconnected, in-memory representation of data.
- The Data Provider is responsible for providing and maintaining the connection to the database.
- A Data Provider is a set of related components that work together to provide data in an efficient and performance driven manner.

# .NET Data Provider

- Constituent objects of Data Provider
  - Connection
  - Command
  - DataReader
  - DataAdapter

- Micrososft .NET ships with four data providers
  - SQL Server (for SQL Server 7.0 or above)
  - OLE DB
  - Oracle
  - ODBC

# .NET Data Providers

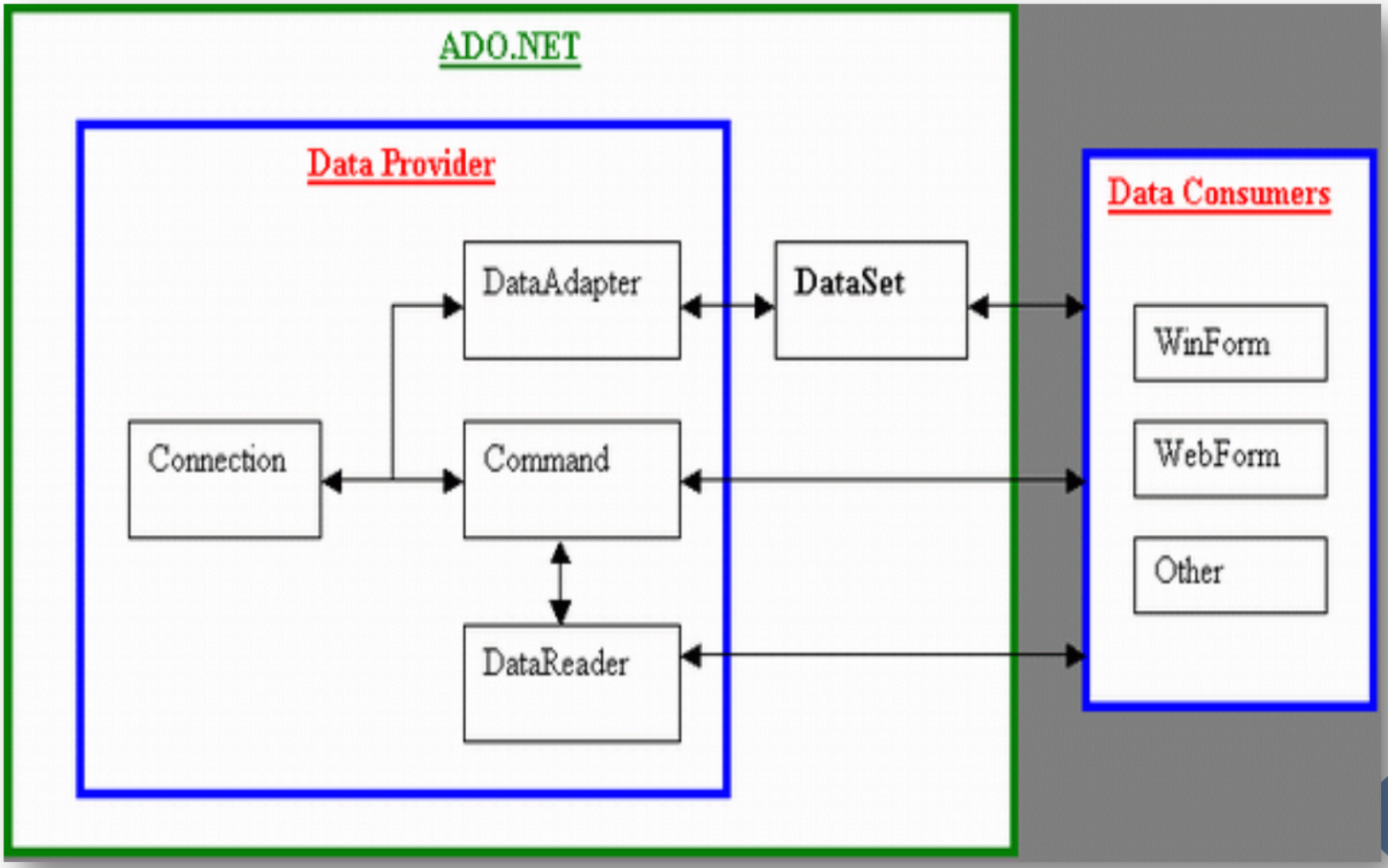
*The ADO.NET Data Provider Objects*

| | SQL Server .NET Provider | OLE DB .NET Provider | Oracle .NET Provider | ODBC .NET Provider |
|---|---|---|---|---|
| Connection | SqlConnection | OleDbConnection | OracleConnection | OdbcConnection |
| Command | SqlCommand | OleDbCommand | OracleCommand | OdbcCommand |
| DataReader | SqlDataReader | OleDbDataReader | OracleDataReader | OdbcDataReader |
| DataAdapter | SqlDataAdapter | OleDbDataAdapter | OracleDataAdapter | OdbcDataAdapter |

# Data Access Modes

- Connected
  - One installs a terminal or client-server application on the user's desk. This connects the user directly to the database.
- Disconnected
  - Disconnected access—dealing with sets, graphs, or trees of data.

# ADO.NET Object Model



# ADO.NET Objects

- Connection
  - Establishes a connection with the data source.
  - Examples of connection objects are *OleDbConnection, SqlConnection etc.*
- Command
  - Command object represents an executable command on the underlying data source.
  - It used to manipulate existing data, query existing data, and update or even delete existing data.
  - e.g. SqlCommand, OleDbCommand etc.

# ADO.NET Objects

- DataReader
  - Read-only, forward-only recordset.
  - e.g. SqlDataReader, OleDbDataReader etc.
- DataAdapter
  - A gateway between the disconnected and connected flavors of ADO.NET.
  - e.g. SqlDataAdapter, OleDbDataAdapter etc.
- Parameter
  - A command needs to be able to accept parameters.
  - e.g. SqlParameter

# ADO.NET Objects

- Transactions
  - Represents a Transaction object.
  - Allows to execute multi-step operation in one go.
  - e.g.SqlTransaction
- DataSet
  - In-memory database.
  - Collection of *DataTables* & *DataRelations*.
  - Provider-independent.

# ADO.NET Objects

- DataTable
  - Represents a table in database.
  - Consists of *DataRows* and *DataColumns*.
- DataRow
  - *Rows* Property of *DataTable* is of *DataRowCollection* type, which represents an enumerable collection of *DataRow* objects.
  - Logical equivalent of a *DataRow* in a database is a row in a table.

# ADO.NET Objects

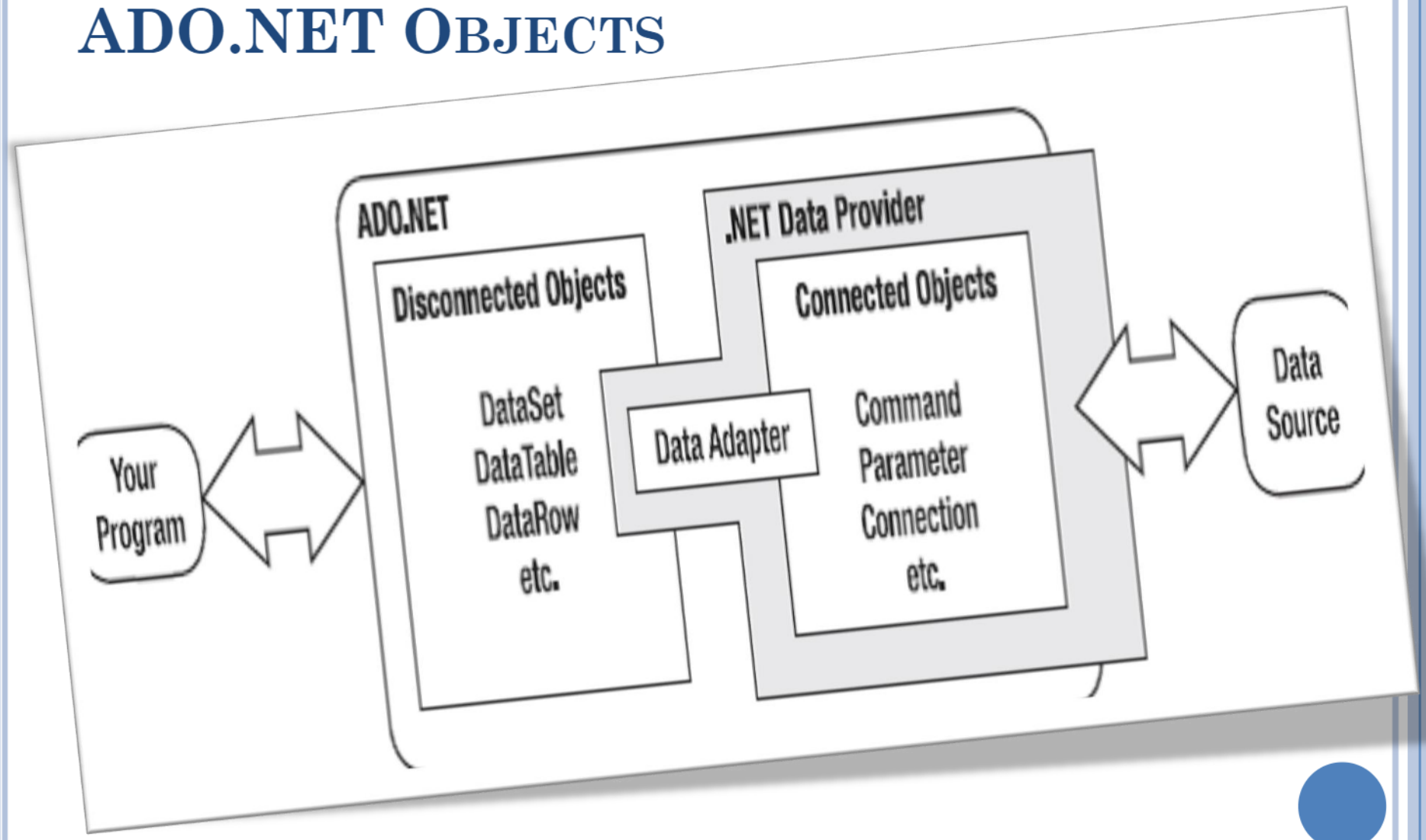
- DataColumn
  - Represents an individual column in a given table in a database.
- DataView
  - A view in a database.
  - One can create a view of *DataTable*.
- Constraint
  - Represents constraints like UNIQUE, PRIMARY KEY.

# ADO.NET Objects

- DataRelation
  - A *DataRelation* object lets you specify relations between various tables that allow one to both validate data across tables and browse parent and child rows in various *DataTable*s.

- DataSet, DataRow, DataColumn, DataView, DataRelation, Constraint are all disconnected objects. Find these in System.Data namespace.

# ADO.NET Objects



# SqlConnection Class

- Belongs to *System.Data.SqlClient* namspace.
- To create a connection

```
SqlConnection con=new SqlConnection();
```

**Or**

```
SqlConnection con=new SqlConnection(conString);
```

**Connection String**

# Connection String

- Data Source *or* Server
  - Specifies the SQL server to connect to.
- Initial Catalogue *or* Database
  - Specifies the database to use.
- User ID *or* uid
  - Specifies login id for connection.
- Password *or* pwd
  - Specifies password for the log in.

# Connection String

```
SqlConnection con=new SqlConnection();
con.ConnectionString=
        "Server=.;Database=AdventureWorks;user id=sa;password=sa;"
```

Or

```
SqlConnection con=new SqlConnection
        ("Server=.;Database=AdventureWorks;user id=sa;password=sa;");
```

To open a connection

```
con.Open();
```

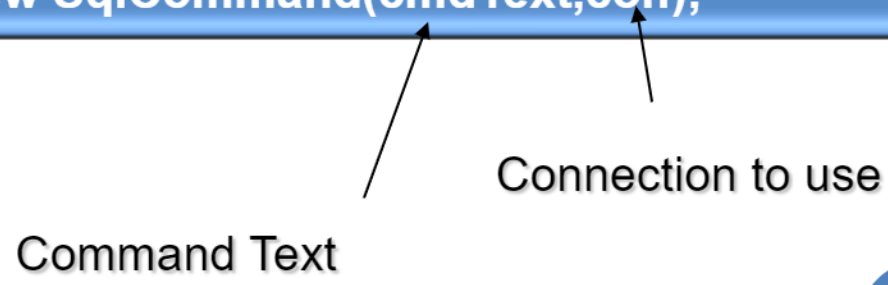
# SqlCommand Class

- Namespace: *System.Data.SqlClient*
- Creating a command object

```
SqlCommand cmd=new SqlCommand();
```

**Or**

```
SqlCommand cmd=new SqlCommand(cmdText,con);
```



# CommandText & Connection Property

- CommandText
  - Accepts string literal representing T-SQL statement or name of stored procedure or name of the table.
- CommandType
  - Specifies the command type.
  - Text, StoredProcedure, or TableDirect.
- Connection
  - Specifies the connection object to use to access database.

# Execute Methods

- ExecuteScalar
  - Returns a single value (e.g. aggregate value) from the database.
  - Mostly used for select queries with aggregate functions.
- ExecuteReader
  - Returns a data reader object with the data queried.
  - Used for SELECT queries.

# Execute Methods

- ExecuteNonQuery
  - Used with DML commands only.
  - Returns integer showing number of rows affected.
  - It is also used to create database objects.

# SqlDataReader Class

- *ExecuteDataReader* method returns a *SqlDataReader* object if invoked on SqlCommand object.

- Read()
  - Returns true or false.
  - If true, data reader has some more record.
  - If false, no records left.
  - This method chooses next record as current one.
  - At least once, one should invoke this to make the first record as current one before accessing the data.

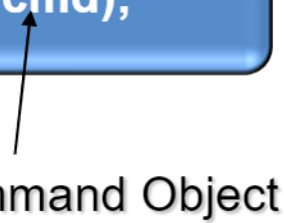
# SqlDataAdapter Class

- Creating a data adapter.

```
SqlDataAdapter da=new SqlDataAdapter();
```

Or

```
SqlDataAdpater da =new SqlDataAdapter(cmd);
```



# SqlDataAdpater Class

- Fill()
  - Fills the specified data table or dataset with the queried data.

- Update()
  - Updates the changes in data table or dataset back to the database.

```
da.Update(ds);
```

```
da.Fill(ds);
```

# Calling Stored Procedure

```
SqlDataAdapter daCategory = new SqlDataAdapter();
daCategory.SelectCommand = new SqlCommand();
daCategory.SelectCommand.Connection = conn;
daCategory.SelectCommand.CommandText = "ProductCategoryList"
daCategory.SelectCommand.CommandType =
        CommandType.StoredProcedure;
```

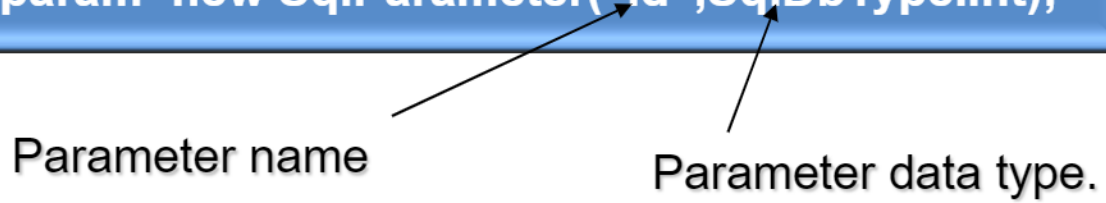
# SqlParameter Class

- Represents a *SqlCommand* parameter.

```
SqlParameter param=new SqlParameter();
```

**Or**

```
SqlParameter param=new SqlParameter("id",SqlDbType.Int);
```



# SqlParameter Object Properties

- ParameterName
  - Gets or sets the name of the parameter.
- SqlDbType
  - Gets or sets the SQL Server database type of the parameter value.
- Direction
  - Sets or gets the direction of the parameter, such as Input, Output, or Both.

# SqlParameter Object Properties

- Size
  - Sets or gets the size of the parameter value.
- Value
  - Sets or gets the value provided to the parameter object.
- SourceColumn
  - Read-write property maps a column from a *DataTable* to the parameter.

# Using Parameters

- Identify the available parameters
  - **Input**
  - **Output**
  - **InputOutput**
  - **ReturnValue**

- Include parameters in the parameters collection

  or

  Include parameter values in the command string

# Passing Input Parameters

- Create parameter, set direction and value, add to the Parameters collection

```
SqlParameter param = new SqlParameter
        ("@Beginning_Date", SqlDbType.DateTime);
param.Direction = ParameterDirection.Input;
param.Value = Convert.ToDateTime
                        (txtStartDate.Text);
da.SelectCommand.Parameters.Add(param);
```

- Run stored procedure and store returned records

```
ds = New DataSet();
da.Fill(ds, "Products");
```

# Using Output Parameters

- Create parameter, set direction, add to the Parameters collection

```
param = new SqlParameter("@ItemCount", SqlDbType.Int);
param.Direction = ParameterDirection.Output;
da.SelectCommand.Parameters.Add(param);
```

- Run stored procedure and store returned records

```
ds = new DataSet();
da.Fill(ds);
```

- Read output parameters

```
iTotal = da.Parameters("@ItemCount").Value;
```

# Creating Relationships

- Identify Parent Column

```
DataColumn pcol=ds.Tables["Customers"].Columns["CustomerID"];
```

- Create a DataRelation.

```
DataColumn ccol=ds.Tables["Orders"].Columns["CustomerID"];
```

```
DataRelation dr=new DataRelation("CustOrder",pcol,ccol);
```

# SqlCommandBuilder Class

- The CommandBuilder examines the DataAdapter object used to create the DataSet, and it adds the additional Command objects for the InsertCommand, DeleteCommand, and UpdateCommand properties.

```
SqlCommandBuilder cmb=new SqlCommandBuilder(da);
da.Update(ds);
```

# Data Binding

- The process of linking a control to a data source is called *data binding*.
- Data binding tells a control where to find data and how one want it displayed, and the control handles the rest of the details.
- Simple Binding: A control that supports single binding typically displays only a single value at once.

# Data Binding Objects

- BindingContext
- ControlBindingCollection
- PropertyManager
- CurrencyManager

# BindingContext

- A *BindingContext* object has a collection of *BindingManagerBase* instances.
- The instances are created and added to the binding manager object when a control is data-bound.
- The *BindingContext* might contain several data sources, wrapped in either a *CurrencyManager* or a *PropertyManager*.
- If the data source contains objects of *IList*, *CurrencyManager* is used.
- If data source returns single value, *PropertyManager* is used.

# PropertyManager & CurrencyManager

- *Binding* class links a property of the control to a member of the data source.
- When a *Binding* object is created, a corresponding *CurrencyManager* or *PropertyManager* object is also created.
- A *CurrencyManager* can maintain the current position within that data source.

# DataGridView Control

- The DataGridView control gives you a variety of views of the same data.

```
SqlConnection con = new SqlConnection(ConfigurationManager.
ConnectionStrings["NorthWindCon"].ConnectionString);
SqlCommand cmd = new SqlCommand("Select * from customers", con);
SqlDataAdapter da = new SqlDataAdapter(cmd);
DataSet ds = new DataSet();
da.Fill(ds,"Customers");
dgv1.AutoGenerateColumns = true;
dgv1.DataSource = ds;
dgv1.DataMember = "Customers";
```

*Add reference to System.Configuration.dll assembly first.*

# Data Sources

- DataSource property can be set to any of the below:
  - An One-Dimensional Array
    - When using an array as the source of data for a *DataGridView* control, the grid looks for the first public property of the object within the array and displays its value.
  - DataTable
  - DataView
  - DataSet or DataViewManager
  - Components implementing *IListSource* interface.
  - Components implementing *IList* interface.
  - Any Generic Collection Classes

# Data Table As a Data Source

- One can display a *DataTable* within a *DataGridView* control in two ways
  - *DataTable* if standalone, set *DataSource* property to it.
  - If *DataTable* is in *DataSet*, set *DataSource* to *DataSet* & *DataMember* property to the name of the *DataTable*.

# DataGridView Properties

- AutoGenerateColumns
  - Determines whether to generate columns based on data source columns automatically.
- AllowUserToAddRows
  - Indicates whether the option to add rows is displayed to the user.
- AllowUserdToDeleteRows
  - Indicates whether the user is allowed to delete rows from the DataGridView.
- AllowUsersToOrderColumns
  - Indicates whether manual column repositioning is enabled.

# DataGridView Properties

- AllowUsersToResizeColumns
  - Indicates whether the user can resize the columns.
- Columns
  - One can specify the columns collection.
- EditMode
  - Determines how the cell editing will be started.
- MultiSelect
  - Indicates whether the user is allowed to select more than one cell, row or columns of the DataGridView at a time.

# DataGridView Properties

- ReadOnly
  - Indicates whether the user can edit the cells of the DataGridView control.

- RowHeaderVisible
  - Specifies whether to display column headers.