LTI
Let's Solve

Introduction to ASP.NET MVC

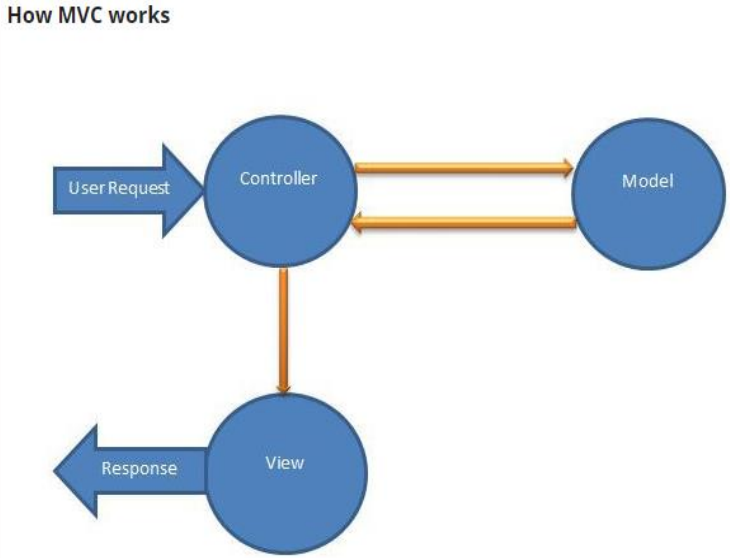
What is ASP.NET MVC

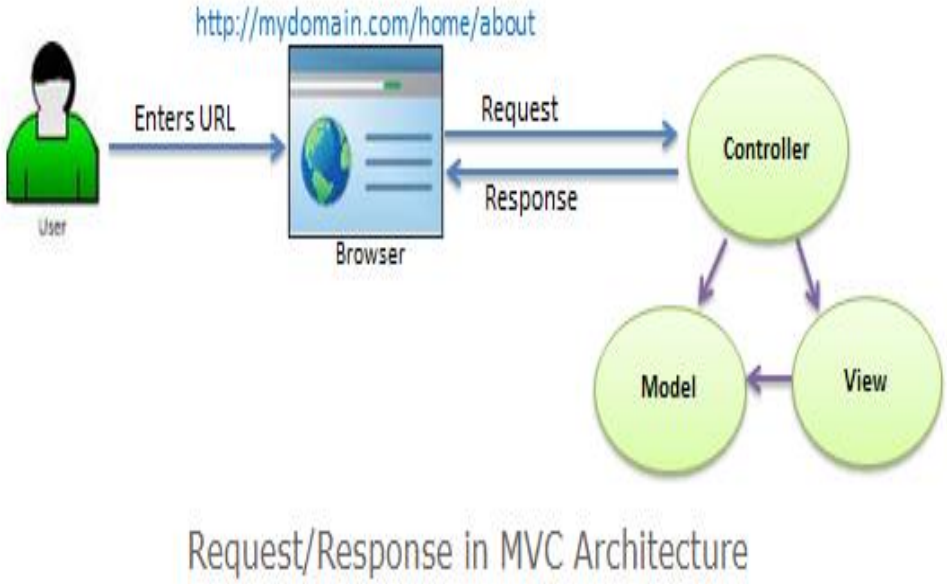
- MVC is a design pattern used to decouple user-interface (view), data (model), and application logic (controller).
- This pattern helps to achieve separation of concerns.
- Using the MVC pattern for websites, requests are routed to a Controller that is responsible for working with the Model to perform actions and/or retrieve data.
- The Controller chooses the View to display, and provides it with the Model.
- The View renders the final page, based on the data in the Model.
- ASP.NET gives you a powerful, patterns-based way to build dynamic websites using the MVC pattern that enables a clean separation of concerns.

Role of Model, View, and Controller

- **Model:**
  - The Model in an MVC application represents the state of the application and any business logic or operations that should be performed by it.
- **View:**
  - Views are responsible for presenting content through the user interface.
  - They use the Razor view engine to embed .NET code in HTML markup.
- **Controller:**
  - Controllers are the components that handle user interaction, work with the model, and ultimately select a view to render.
  - In the MVC pattern, the controller is the initial entry point, and is responsible for selecting which model types to work with and which view to render

How ASP.NET MVC Works



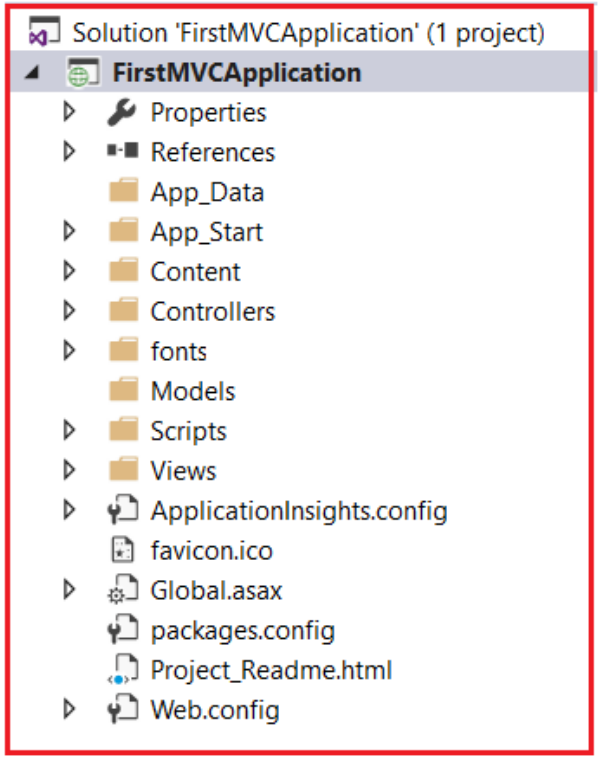


Key Benefits of ASP.NET MVC

- Faster development process:
- Ability to provide multiple views:
- Support for asynchronous technique:
- The modification does not affect the entire model:
- MVC model returns the data without formatting:

Understanding the structure of an ASP.NET MVC project

- App_Data
- App_Start
- Content
- Controllers
- Models
- Scripts
- Views
- Global.asax
- Packages.config
- Web.config



Understanding the structure of an ASP.NET MVC project

- **App_Data:** The App_Data folder of an MVC application is used to contain the application related data files like .mdf files, LocalDB, and XML files, etc.
- **App_Start:** The App_Start folder of an MVC application is used to contain the class files which are needed to be executed at the time of application starts. The classes like BundleConfig, FilterConfig, RouteConfig, IdentityConfig, etc are stored within this folder.
- **Content:** The Content Folder of an MVC application is used to store the static files such as the image files, CSS files, and icons files.
- **Controllers:** The Controllers Folder in MVC application is used to contains all the controllers of your application.
- **Models:** The Models folder of an MVC application is used to store the class files which are used to store the domain data as well as business logic to manage the data.
- **Scripts:** The Scripts Folder of an MVC application is used to contains all the JavaScript files that are required for the application.

Understanding the structure of an ASP.NET MVC project

- **Views:** The Views Folder of an MVC application is used to contains the ".cshtml" files for the application. In MVC, the .cshtml file is a file where we need to write the HTML code along with the C# code.
- **Global.asax:** The Global.asax file in an MVC application allows us to write code that we want to run at the application level, such as Application_BeginRequest, application_error, application_start, session_start, session_end, etc.
- **Packages.config:** The Packages.config file in MVC application is managed by the NuGet which will keep track of what packages and versions have installed within your application.
- **Web.config:** The Web.config file of an MVC application is one of the most useful and important files which contains the application-level configurations.

LTI
Let's Solve

Introduction to ASP.NET MVC

Quiz

Q: MVC stands for _____.

- Model, Vision & Control
- Model, View & Controller
- Model, ViewData & Controller
- Model, Data & Controller
- None of the Options

Q: Model is a _____?

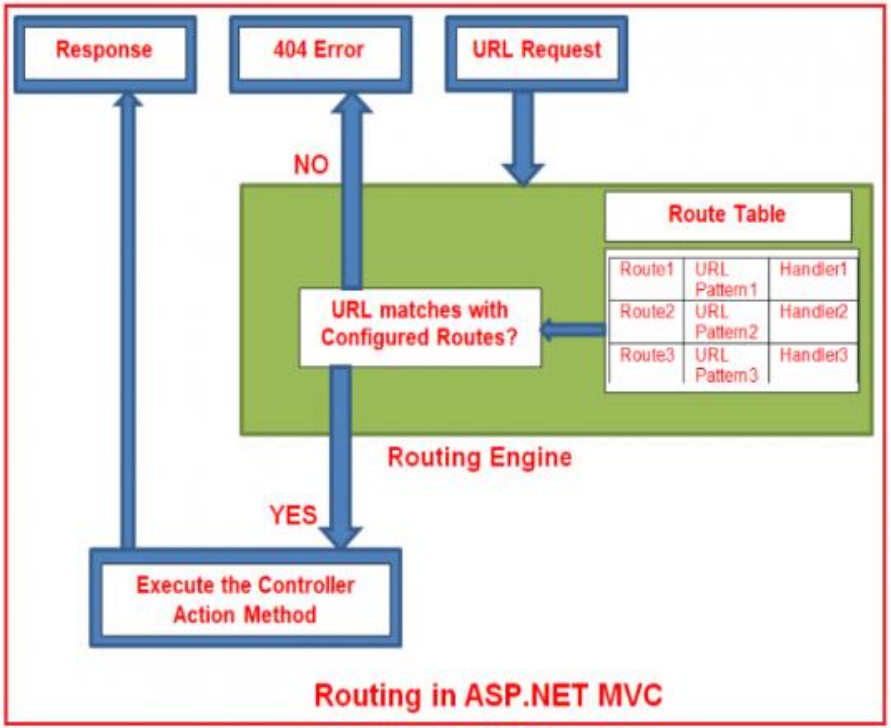
- Shape of data
- Html content
- Collection of data
- Type of data
- None of the Options

LTI
Let's Solve

URL Routing

URL Routing

- The ASP.NET Routing module is responsible for mapping the incoming browser requests (i.e. the incoming URL) to a particular MVC controller action method.
- This mapping is done by the routing rules defined for your application.
- A Route is a URL pattern that is mapped to a handler. The handler can be a physical file.
- All the configured routes of an ASP.NET MVC application stored in the RouteTable and this Route table will be used by the Routing engine to determine the appropriate handler class or file for an incoming request.



URL Routing

- Every MVC application must configure (register) at least one route in the RouteConfig class and by default MVC Framework provide one default route.

- But you can configure as many as routes you want. You can register a route in the RouteConfig class, which is in RouteConfig.cs file under the App_Start folder as shown beside.

```
public class RouteConfig {
public static void
RegisterRoutes(RouteCollection routes) {
routes.MapRoute(
name: "Default", //Route Name
url: "{controller}/{action}/{id}", //Route
Pattern
defaults: new
{
controller = "Home", //Controller Name
action = "Index", //Action method Name
id = UrlParameter.Optional //Default value
for above defined parameter
}
);
}
```

URL Routing

- After configuring the routes in **RouteConfig** class, you need to register it in the **Application_Start()** event in the **Global.asax** file. So that it includes all your routes into the RouteTable.

```
namespace FirstMVCDemo
{
public class MvcApplication : System.Web.HttpApplication
{
protected void Application_Start()
{
AreaRegistration.RegisterAllAreas();
RouteConfig.RegisterRoutes(RouteTable.Routes);
}
}
}
```

URL Routing

- Static URL Segment:

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

URL Routing

- Constraining Routes:

  The **Route Constraint** in ASP.NET MVC Routing allows us to apply a regular expression to a URL segment to restrict whether the route will match the request.

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

URL Routing

- **Using Attribute Routing :**
  - Using the **[Route]** attribute to define routes is called Attribute Routing.
  - It provides you more control over the URIs by defining routes directly on actions and controllers in your ASP.NET MVC application.
- **Enabling Attribute Routing:**

To enable attribute routing, you need to add a call to **routes.MapMvcAttributeRoutes()** method within **RegisterRoutes()** method of **RouteConfig.cs** file.

```
routes.MapMvcAttributeRoutes();
[HttpGet]
[Route("students/{studentID}/courses")]
public ActionResult GetStudentCourses(int studentID)
{ }
```

URL Routing

- **Generating Outgoing URLs in Views:** In order to redirect from one view to another, you can use @Html.ActionLink() helper method:

```
@Html.ActionLink("Click here", // <-- Link text
        "About", // <-- Action Method Name
        "Home" // <-- Controller Name
    )
```

LTI
Let's Solve

URL Routing

Quiz

A Larsen & Toubro Group Company

Q: Which of the following is a default route pattern in MVC?

"/{action}/{controller}/{id}"

"{controller}/{id}“

"{controller}/{action}/{id}"

"{controller}/{action}”

Q: Which of the following default class is used to configure all the routes in MVC?

- FilterConfig
- RegisterRouteConfig
- RouteConfig
- MVCRoutes

LTI
Let's Solve

Razor View Engine

Razor Basics

- Razor is one of the view engine supported in ASP.NET MVC.
- Razor allows you to write mix of HTML and server side code using C# or Visual Basic.
- Razor view with visual basic syntax has .vbhtml file extension and C# syntax has .cshtml file extension.
- In a web page that uses the Razor syntax, there are two kinds of content: client content and server code.
- Client content is the stuff you're used to in web pages: HTML markup (elements), style information such as CSS, maybe some client script such as JavaScript, and plain text.
- Razor syntax lets you add server code to this client content. If there's server code in the page, the server runs that code first, before it sends the page to the browser.
- By running on the server, the code can perform tasks that can be a lot more complex to do using client content alone, like accessing server-based databases.
- Most importantly, server code can dynamically create client content — it can generate HTML markup or other content on the fly and then send it to the browser along with any static HTML that the page might contain.

Implementation of Razor view

"@" symbol is used to implement Razor View in a Asp.Net MVC Views.

Inline Expression:

```
<h2>@DateTime.Now.ToShortDateString()</h2>
```

Combining Text, Markup, and Code in Code Blocks

```
@if(IsPost) {
<p>Hello, the time is @DateTime.Now </p>
} else {
<p>Hello <em>stranger</em>, today is: <br /> </p> @DateTime.Now
}
MultiStatement:
@{
var greeting = "Welcome!";
var theCount = 3;
var monthlyTotal = theCount + 5;
}
```

Implementation of Razor view

- Conditional and Logical Loops:

```
@{
var showToday = true;
if(showToday)
{
@DateTime.Today;
}
}
Looping Code:
@for(var i = 10; i < 21; i++)
{
<p>Line #: @i</p>
}
```

- Collection Objects:

```
@{
<h3>Team Members</h3>
string[] teamMembers = {"Matt",
"Joanne", "Robert", "Nancy"};
foreach (var person in teamMembers)
{
<p>@person</p>
}
}
```

- Use Model in Razor:

```
@model Student
<p>Student Id: @Model.StudentId</p>
<p>Student Name: @Model.StudentName</p>
<p>Age: @Model.Age</p>
```

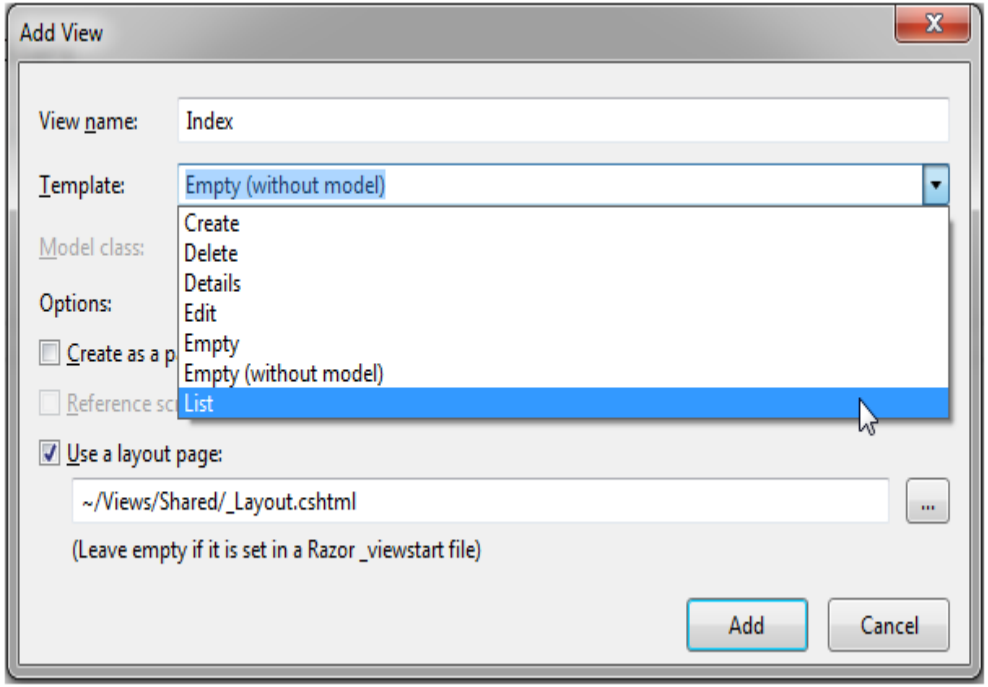
LTI
Let's Solve

Views

A Larsen & Toubro Group Company

Templates and Scaffolding

- **Scaffolding:**
  - ASP.NET Scaffolding is a code generation framework for ASP.NET Web applications.
  - You add scaffolding to your project when you want to quickly add code that interacts with data models.
  - Using scaffolding can reduce the amount of time to develop standard data operations in your project.
- **Templates:**
  - While adding Views, Select the scaffolding template. Template dropdown will show default templates available for Create, Delete, Details, Edit, List or Empty view.



ViewData and ViewBag

- **ViewData:**
  - The ViewData in ASP.NET MVC is a mechanism to pass the data from a controller action method to a view.
  - ViewData is a dictionary which can contain key-value pairs where each key must be string.
  - Passing and retrieve data from ViewData in ASP.NET MVC:

In Controller:

```
ViewData["Employee"] =
employee;
ViewData["Header"] =
"Employee Details";
```

In View:

```
@{
var employee = ViewData["Employee"]
as FirstMVCDemo.Models.Employee;
}
<h2>@ViewData["Header"]</h2>
<table style="font-family:Arial">
<tr>
<td>Employee ID:</td>
<td>@employee.EmployeeId </td>
</tr>
<tr>
<td>Name:</td>
<td>@employee.Name</td>
</tr>
</table>
```

ViewData and ViewBag

- ViewBag:
  - The ViewBag in ASP.NET MVC is one of the mechanisms to pass the data from a controller to a view.
  - the ViewBag is a dynamic property (a new feature introduced in C# 4.0) of the Controller base class.
  - The ViewBag is also like ViewData which also transfers the data from a controller action method to a view.

In Controller:

```
ViewBag.Employee = employee;
ViewBag.Header = "Employee Details";
```

InView:

```
@{
var employee = ViewBag.Employee;
}
<h2>@ViewBag.Header</h2>
<table style="font-family:Arial">
<tr>
<td>Employee ID:</td>
<td>@employee.EmployeeId </td>
</tr>
<tr>
<td>Name:</td>
<td>@employee.Name</td>
</tr>
 </table>
```

ViewData and ViewBag

- **Difference between ViewBag and ViewData:**
  - ViewData is a dictionary of objects that is derived from ViewDataDictionary class and is accessible using strings as keys.
  - ViewBag is a dynamic property that takes advantage of the new dynamic features in C# 4.0.
  - ViewData requires typecasting for complex data type and check for null values to avoid error.
  - ViewBag doesn't require typecasting for complex data type.

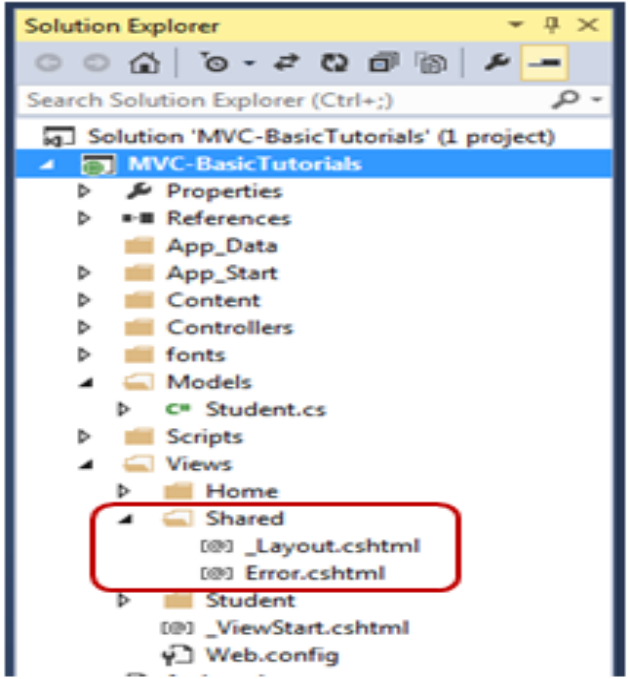
TempData

- TempData is also a dictionary derived from TempDataDictionary class and stored in short lives session and it is a string key and object value.
- TempData keeps the information for the time of an HTTP Request.
- This mean only from one page to another.
- This also works with a 302/303 redirection because it's in the same HTTP Request.
- It helps to maintain data when you move from one controller to other controller or from one action to other action.
- It internally uses session variables.
- Temp data use during the current and subsequent request only means it is used when you are sure that next request will be redirecting to next view.
- It requires typecasting for complex data type and check for null values to avoid error.
- It is generally used to store only one time messages like error messages, validation messages.

```
TempData["ModelName"] = model;
```

Layout Pages

- ASP.NET MVC introduced a Layout view which contains common UI parts such as the logo, header, left navigation bar, right bar or footer section, so that we don't have to write the same code in every page.
- The layout view is same as the master page of the ASP.NET webform application.
- The razor layout view has same extension as other views, .cshtml or .vbhtml.
- Layout views are shared with multiple views, so it must be stored in the Shared folder.
- By default, the name of this layout file is "_Layout.cshtml".



Layout Views in Shared Folder

Layout Pages

- **ViewStart.cshtml:**
  - ViewStart.cshtml is included in the Views folder by default.
  - It sets up the default layout page for all the views in the folder and its subfolders using the Layout property.
  - You can assign a valid path of any Layout page to the Layout property.
  - By default, all the views derived default layout page from _ViewStart.cshtml of Views folder.
  - It sets the layout of all Views by setting the following in the _ViewStart.html:

    Layout= "~/Views/Shared/_Layout.cshtml".

Layout Pages

- **Rendering Methods:**
- ASP.NET MVC layout view renders child views using the following methods.
  - **RenderBody():**

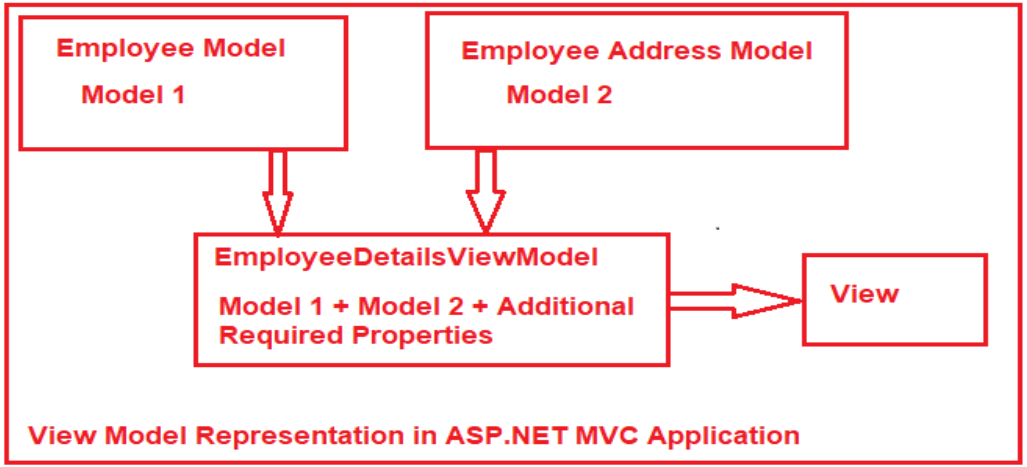
    Renders the portion of the child view that is not within a named section. Layout view must include RenderBody() method.
  - **RenderSection(string name,Boolean required):** Renders a content of named section and specifies whether the section is required. RenderSection() is optional in Layout view.

Layout Pages

- Partial Views:
  - The partial view in MVC is a view that's rendered within another view.
  - The HTML output generated by executing the partial view is rendered into the calling (or parent) view. Like views, partial views use the .cshtml file extension.
  - Partial views are an effective way of breaking up large views into smaller components.
  - They can reduce duplication of view content and allow view elements to be reused.
  - Common layout elements should be specified in _Layout.cshtml.
  - Non-layout reusable content can be encapsulated into partial views.
  - Most common ways to display partial view within a view are:
    - Html.RenderPartial
    - Html.Partial

ViewModel

- Using a ViewModel Object:
  - In ASP.NET MVC application, a single model object may not contain all the necessary data required for a view.
  - For example, a view may require different model data. Then in such situations like this, we need to use the concept ViewModel.



Bundling & Minification

- Bundling and minification are two techniques you can use in ASP.NET 4.5 to improve request load time.
- Bundling and minification improves load time by reducing the number of requests to the server and reducing the size of requested assets (such as CSS and JavaScript.)
- **Bundling:**
  - Bundling is a new feature in ASP.NET 4.5 that makes it easy to combine or bundle multiple files into a single file.
  - You can create CSS, JavaScript and other bundles.
  - Fewer files means fewer HTTP requests and that can improve first page load performance.
- **Minification:**
  - Minification performs a variety of different code optimizations to scripts or css, such as removing unnecessary white space and comments and shortening variable names to one character.

Bundling & Minification

- Bundling and minification is enabled or disabled by setting the value of the debug attribute in the **compilation Element** in the *Web.config* file.

```
<system.web>

<compilation debug="true" />

</system.web>
```

- To enable bundling and minification, set the debug value to "false".

Bundling & Minification

- You can override the Web.config setting with the EnableOptimizations property on the BundleTable class.
- Open the App_Start\BundleConfig.cs file and examine the RegisterBundles method which is used to create, register and configure bundles.

```
public static void RegisterBundles(BundleCollection bundles)
{
bundles.Add(new ScriptBundle("~/bundles/jquery").Include(
"~/Scripts/jquery-{version}.js"));
BundleTable.EnableOptimizations = true;
}
```

LTI
Let's Solve

Views

Quiz

A Larsen & Toubro Group Company

Q: Which of the following view contains common parts of UI?

- Partial view
- Html View
- Layout view
- Razor view

Q: Which of the following methods are used to render partial view?

- Html.Partial()
- Html.RenderPartial()
- Html.RenderAction()
- All of the above

LTI
Let's Solve

HTML Helpers

Basic Helpers

- The ASP.NET MVC Framework provides Html Helper classes which contain different extension methods.
- We can use those extension methods to create HTML controls programmatically within a view.
- All the HtmlHelper methods that are present within the HtmlHelper class generate HTML and return the result as an HTML string.
- An HTML Helper in MVC is an extension method of the HTML Helper class which is used to generate HTML content in a view.

Basic Helpers

- HTML Helper method:
- `@Html.TextBox("firstname", "John")`
- Resulted HTML View:
- `<input id="firstname" name="firstname" type="text" value="John" />`
- Setting addition properties:
- `@Html.TextBox("firstname", "John", new { style = "background-color:Red; color:White; font-weight:bold", title="Please enter your first name" })`

Basic Helpers

- Other HTML Helper methods are:
  - @Html.Label()
  - @Html.Password()
  - @Html.TextArea()
  - @Html.Hidden()
  - @Html.DropDownList()
  - @Html.RadioButton()
  - @Html.CheckBox()
  - @Html.ListBox()
  - And many more ……

Strongly-Typed Helpers

- In ASP.NET MVC, there are two types of HTML Helper methods
  - Simple HTML helper methods (loosely typed) :
  - Strongly type HTML helper:
  - Let's understand it with an example:

    @Html.TextBox() is a loosely typed helper method whereas the **@Html.TextBoxFor()** is a strongly typed helper method.
    - The Html.TextBox() Helper method is not strongly typed and hence they don't require a strongly typed view.
    - On the other hand the Html.TextBoxFor() HTML Helper method is a strongly typed method and hence it requires a strongly typed view and the name should be given using the lambda expression.

- In this, TextBoxFor method binds a specified model object property to input text. So it automatically displays a value of the model property in a textbox and visa-versa.

```
@Html.TextBoxFor(m => m.EmployeeName, new { @class = "form-control" })
```

Templated Helpers

- When we want to customize the data i.e. how the data should be presented to the end-user for the purpose of displaying and editing, then we need to use the display template and editor template helpers.

- In ASP.NET MVC, there are two types of HTML Templated Helper methods:
  - Display Templates:
    - @Html.Display("EmployeeData") – Used with a view that is not strongly typed.
    - @Html.DisplayFor(model => model) – Used with strongly typed views. If our model has properties that return complex objects, then this templated helper is very useful.
    - @Html.DisplayForModel() – Used with strongly typed views. It will work through each property of the model to display the object.
  - Editor Templates:
    - @Html.Editor("EmployeeData")
    - @Html.EditorFor(model => model)
    - @Html.EditorForModel()

Custom Helpers

- The MVC framework also provides the facility to create **Custom HTML Helpers in ASP.NET MVC** application. Once you create your custom HTML helper method then you can reuse it many times.

```
public static class CustomHelpers
{
public static IHtmlString Image(this HtmlHelper helper, string src, string
alt)
{
TagBuilder tb = new TagBuilder("img");
tb.Attributes.Add("src", VirtualPathUtility.ToAbsolute(src));
tb.Attributes.Add("alt", alt);
return new MvcHtmlString(tb.ToString(TagRenderMode.SelfClosing));
}
}
```

Custom Helpers (contd.)

- <add namespace="CustomHTMLHelper.CustomHelpers" /> in web.config under “<system.web.webPages.razor>” element.

- @Html.Image(Model.Photo, Model.AlternateText) use in this way in View.

LTI
Let's Solve

HTML Helpers

Quiz

Q: HtmlHelper class _____.

- Generates html elements
- Generates html view
- Generates html help file
- Generates model data

Q: Which of the following methods are used to render partial view?

- Html.Partial()
- Html.RenderPartial()
- Html.RenderAction()
- All of the above

LTI
Let's Solve

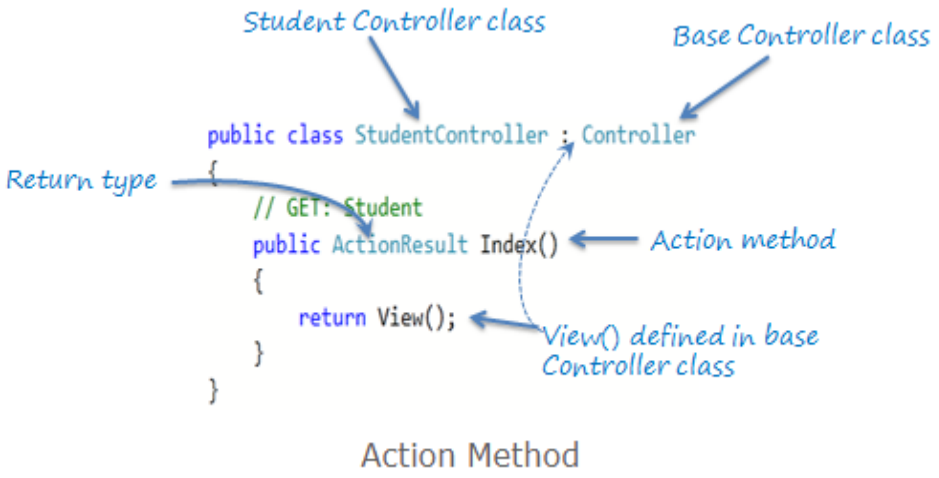
Controllers and Actions

Controllers and Actions

- **IController, ControllerBase, and Controller**
  - 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.
  - **ControllerBase Class:** Represents the base class for all MVC controllers.
  - **IController interface:** Defines the methods that are required for a controller.
  - **Controller Class:** Provides methods that respond to HTTP requests that are made to an ASP.NET MVC Web site.

Controllers and Actions

- **Defining Actions:**
  - All the public methods of a 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.



Action Method

Controllers and Actions

- Action Selectors
  - Action selector is the attribute that can be applied to the action methods. It helps routing engine to select the correct action method to handle a particular request. MVC 5 includes the following action selector attributes:
    - ActionName:

```
[ActionName("find")]
public ActionResult GetById(int id)
{
  return View();
}
```

Controllers and Actions

- NonAction:

```
[NonAction]
public Student GetStudent(int id)
{
return studentList.Where(s => s.StudentId ==
id).FirstOrDefault();
}
```

- ActionVerbs: The ActionVerbs selector is used when you want to control the selection of an action method based on a Http request method. For example, Get, Post, Put, Delete etc.

Controllers and Actions

- Action Filters:
  - Action filter executes before and after an action method executes.
  - Action filter attributes can be applied to an individual action method or to a controller.
  - When action filter applied to controller then it will be applied to all the action methods in that controller.

```
[OutputCache(Duration=100)]
public ActionResult Index()
{
return View();
}
```

Controllers and Actions

- **Returning Data with ActionResult :**
  - ASP.NET MVC has different types of Action Results.
  - Each action result returns a different format of output.
  - There are different Types of action results in ASP.NET MVC.
  - ActionResult is the base class of all the result type action method.
  - Following are the Result types that an action method can return in MVC.
    - ViewResult – Represents HTML and markup.
    - EmptyResult – Represents no result.
    - RedirectResult – Represents a redirection to a new URL.

Controllers and Actions

Following are the Result types that an action method can return in MVC.(contd.)

- JsonResult – Represents a JavaScript Object Notation result that can be used in an AJAX application.
- JavaScriptResult – Represents a JavaScript script.
- ContentResult – Represents a text result.
- FileContentResult – Represents a downloadable file (with the binary content).
- FilePathResult – Represents a downloadable file (with a path).
- FileStreamResult – Represents a downloadable file (with a file stream).

Controllers and Actions

- **Returning Data with ActionResult :**

- These results are categorized into three sections:
  - **Content-returning:**
    - These ActionResults are responsible for returning content to the browser or calling script. The examples are as follows
      - ViewResult
      - PartialViewResult
      - FileResult
      - ContentResult
      - EmptyResult
      - JsonResult
      - JavaScriptResult

Controllers and Actions

- **Returning Data with ActionResult :**
- These results are categorized into three sections:
  - **Redirection:**
    - These ActionResults are responsible for redirecting to other URLs or actions. The examples are as follows
      - RedirectResult
      - RedirectToRouteResult
      - RedirectToActionResult
  - **Status:**
    - These ActionResults are responsible for returning status codes to the browser for it to use. The examples are as follows
      - HttpStatusCodeResult
      - HttpUnauthorizedResult
      - HttpNotFoundResult

LTI
Let's Solve

Controllers and Actions

Quiz

A Larsen & Toubro Group Company

Q: Which of the following is TRUE?

- Action method can be static method in a controller class.
- Action method can be private method in a controller class.
- Action method can be protected method in a controller class.
- Action method must be public method in a controller class.

Q: Which is the default http method for an action method?

- HttpPost
- HttpGet
- HttpPut
- HttpDelete

LTI
Let's Solve

Entity Framework

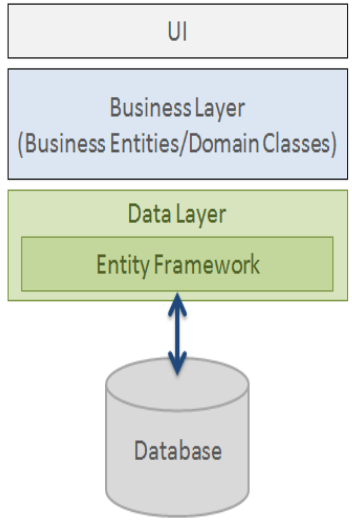
A Larsen & Toubro Group Company

What is Entity Framework?

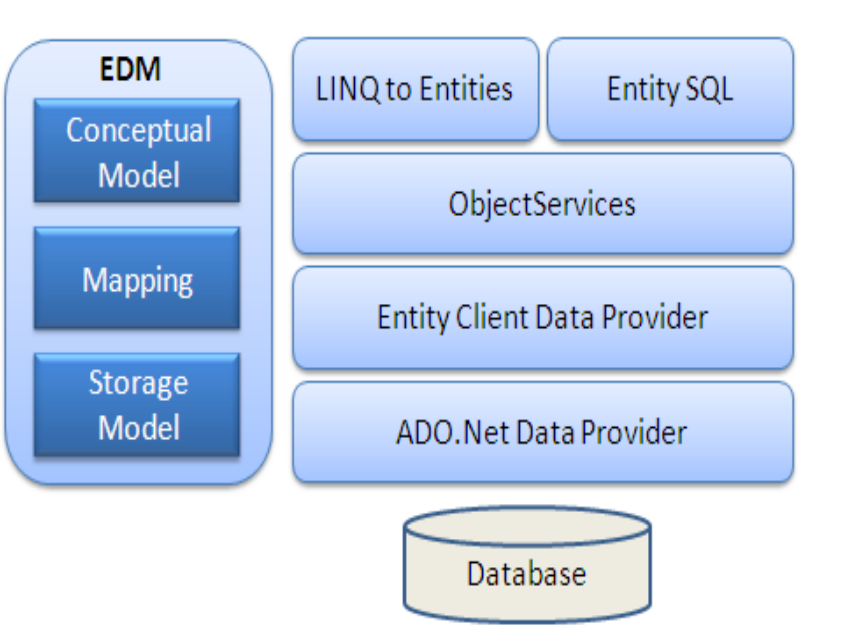
- Entity Framework is an open-source ORM framework for .NET applications supported by Microsoft.
- It enables developers to work with data using objects of domain specific classes without focusing on the underlying database tables and columns wher this data is stored.
- With the Entity Framework, developers can work at a higher level of abstraction when they deal with data, and can create and maintain data-oriented applications with less code compared with traditional applications.

Official Definition: “Entity Framework is an object-relational mapper (O/RM) that enables .NET developers to work with a database using .NET objects. It eliminates the need for most of the data-access code that developers usually need to write.”

- Because the Entity Framework is a component of the .NET Framework, Entity Framework applications can run on any computer on which the .NET Framework starting with version 3.5 SP1 is installed.



Entity Framework Architecture



EDM (Entity Data Model): EDM consists of three main parts - Conceptual model, Mapping and Storage model.

Conceptual Model: The conceptual model contains the model classes and their relationships. This will be independent from your database table design.

Storage Model: The storage model is the database design model which includes tables, views, stored procedures, and their relationships and keys.

Mapping: Mapping consists of information about how the conceptual model is mapped to the storage model.

LINQ to Entities: LINQ-to-Entities (L2E) is a query language used to write queries against the object model. It returns entities, which are defined in the conceptual model. You can use your LINQ skills here.

Entity SQL: Entity SQL is another query language (For EF 6 only) just like LINQ to Entities. However, it is a little more difficult than L2E and the developer will have to learn it separately.

Object Service: Object service is a main entry point for accessing data from the database and returning it back. Object service is responsible for materialization, which is the process of converting data returned from an entity client data provider (next layer) to an entity object structure.

Entity Client Data Provider: The main responsibility of this layer is to convert LINQ-to-Entities or Entity SQL queries into a SQL query which is understood by the underlying database. It communicates with the ADO.Net data provider which in turn sends or retrieves data from the database.

ADO.Net Data Provider: This layer communicates with the database using standard ADO.Net.

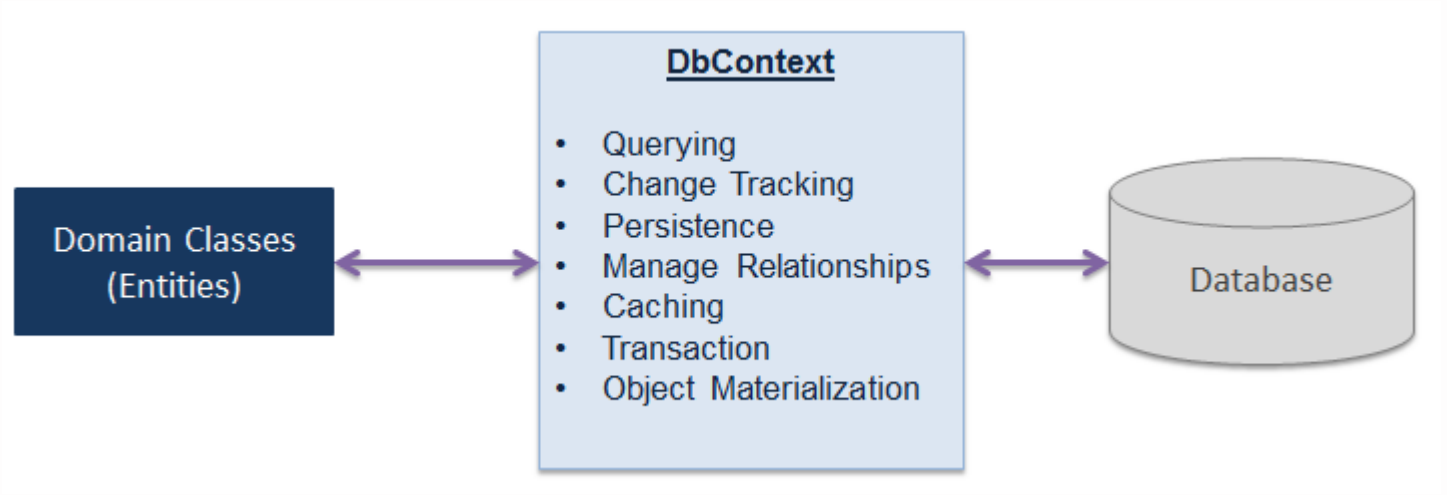
Creating Entity Data Model

- Entity Framework has 3 different approaches:
  - Database-First Approach
  - Code-First Approach
  - Model-First Approach

  - **Database-First Approach:** In this approach, we make use of the existing database to create model class along with ADO.NET Entity Data Model.
  - Entity Framework uses EDM(.edmx file) for all the database-related operations.
  - Entity Data Model is a model that describes entities and the relationships between them.
    - Context & Entity Classes
    - Every Entity Data Model generates a context and an entity class for each database table.
    - It creates two important files, <EDM Name>.Context.tt and <EDM Name>.tt, under which:
    - A DBContext class and and Entity class(also called POCO(Plain old CLR objects) class is created.

Creating Entity Data Model

- **DbContext Class:**
  - DBContext class is derived from *System.Data.Entity.DbContext* class.
  - DbContext is an important class in Entity Framework API.
  - It is a bridge between your domain or entity classes and the database.



Creating Entity Data Model

- DbContext Methods:

Method	Usage
Entry	Gets an DbEntityEntry for the given entity. The entry provides access to change tracking information and operations for the entity.
SaveChanges	Executes INSERT, UPDATE and DELETE commands to the database for the entities with Added, Modified and Deleted state.
SaveChangesAsync	Asynchronous method of SaveChanges()
Set	Creates a DbSet<TEntity> that can be used to query and save instances of TEntity.
OnModelCreating	Override this method to further configure the model that was discovered by convention from the entity types exposed in DbSet<TEntity> properties on your derived context.

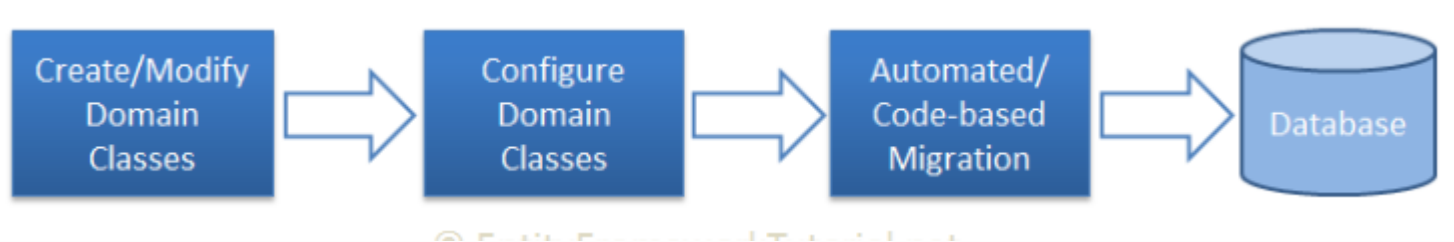
Creating Entity Data Model

- DbSet in Entity Framework:
  - The DbSet class represents an entity set that can be used for create, read, update, and delete operations.
  - The context class (derived from DbContext) must include the DbSet type properties for the entities which map to database tables and views.
  - It has to a property in the DbContext class.
  - Each DbSet Property is mapped to a table in the connected database.

```
public partial class EmployeeDbContext : DbContext
{
public EmployeeDbContext()
: base("name=MVC_DBEntities")
{ }
public virtual DbSet<Employee> Employees { get; set; }
}
```

Creating Entity Data Model

- **Code-First Approach:**
  - Entity Framework introduced the Code-First approach with Entity Framework 4.1.
  - In the Code-First approach, you focus on the domain of your application and start creating classes for your domain entity rather than design your database first and then create the classes which match your database design.
    - code-first development workflow:



Model Binding

- The **ASP.NET MVC Model Binding** allows us to map HTTP request data with a model.
- It is the process of creating .NET objects using the data sent by the browser in an HTTP request.
- Model binding is a well-designed bridge between the HTTP request and the C# action methods.
- It makes it easy for developers to work with data on forms (views) because POST and GET are automatically transferred into a data model we specify. ASP.NET MVC uses default binders to complete this behind the scene.

Model Binding

- Model binding can be done in following ways:

```
public ActionResult Create([Bind(Include =
"EmployeeId,Name,Gender,City,Salary")] Employee employee)
public ActionResult CreateUsingParameter(string Name,string Gender,string
City,decimal Salary)
public ActionResult CreateUsingModel(Employee employee)
public ActionResult CreateUsingFormCollection(FormCollection
formCollection)
```

Eager and Lazy Loading

- **Eager Loading:**
  - Eager loading is the process whereby a query for one type of entity also loads related entities as part of the query.
  - Eager loading is achieved by use of the Include method.

```
public ActionResult Index()
{
var students = db.Students.Include(s => s.Course);
return View(students.ToList());
}
```

  - Eagerly loading multiple levels:

```
var blogs1 = context.Blogs
.Include(b => b.Posts.Select(p => p.Comments))
.ToList();
```

Eager and Lazy Loading

- Lazy Loading:
  - Lazy loading is the process whereby an entity or collection of entities is automatically loaded from the database the first time that a property referring to the entity/entities is accessed.
  - When using POCO entity types, lazy loading is achieved by creating instances of derived proxy types and then overriding virtual properties to add the loading hook.

```
public class Blog
{
   public int BlogId { get; set; }
   public string Name { get; set; }
   public string Url { get; set; }
   public string Tags { get; set; }

   public ICollection<Post> Posts { get; set; }
}
```

  - Turn off Lazy loading:
    - this.Configuration.LazyLoadingEnabled = false;

Eager and Lazy Loading

- Lazy Loading:
  - Lazy loading is the process whereby an entity or collection of entities is automatically loaded from the database the first time that a property referring to the entity/entities is accessed.
  - When using POCO entity types, lazy loading is achieved by creating instances of derived proxy types and then overriding virtual properties to add the loading hook.

```
public class Blog
{
public int BlogId { get; set; }
public string Name { get; set; }
public string Url { get; set; }
public string Tags { get; set; }

public ICollection<Post> Posts { get; set; }
}
```

Eager and Lazy Loading

- Turn off Lazy loading:

```
this.Configuration.LazyLoadingEnabled = false;
```

LTI
Let's Solve

Entity Framework

Quiz

Q: Which of the following is True?

- Entity Framework is an MVC framework.
- Entity Framework is an open source ORM framework.
- Entity Framework is a framework for database management tool.
- Entity Framework is object mapping tool.

Q: Which of the following development approaches are supported in Entity Framework?

- Code-First
- Database-First
- Model-First
- All of the above

LTI
Let's Solve

Model Validation

A Larsen & Toubro Group Company

Data Annotations

- ASP.NET MVC Framework provides a concept called Data Annotation which is used for model validation.
- It's inherited from **System.ComponentModel.DataAnnotations** assembly.
- The Data Annotation Attributes includes built-in validation attributes for different validation rules, which can be applied to the model class properties.
- **The System.ComponentModel.DataAnnotations assembly has many built-in validation attributes, for example:**
  - Required
  - Range
  - RegularExpression,
  - Compare
  - StringLength
  - Data type

Data Annotations

- Along with the above build-in validation attributes, there are also many data types the user can select to validate the input. Using this data type attribute, the user can validate for the exact data type as in the following:
  - Credit Card number
  - Currency
  - Date
  - DateTime
  - Duration
  - Email Address
  - Image URL
  - Multiline text
  - Password
  - Phone number
  - Postal Code
  - Upload

Validation Message and Validation Summary

- There are three HtmlHelper extension method validation message:
  - **Validation Message:**
    - The Html.ValidationMessage() is an extension method, that is a loosely typed method.
    - It displays a validation message if an error exists for the specified field in the ModelStateDictionary object.

      MvcHtmlString ValidateMessage(string modelName, string validationMessage, object htmlAttributes)

      Example,

```
@Html.EditorFor(model => model.FirstName, new { htmlAttributes = new {
@class = “form-control” } })
@Html.ValidationMessage(“FirstName”, “”, new { @class = “text-danger” })
```

Validation Message and Validation Summary

- ValidationMessageFor:
  - The Html.ValidationMessageFor() is a strongly typed extension method.
  - It displays a validation message if an error exists for the specified field in the ModelStateDictionary object.

    Example,

```
@Html.EditorFor(model => model.FirstName, new { htmlAttributes = new {
@class = “form-control” } })
@Html.ValidationMessageFor(model => model.FirstName, “”, new { @class =
“text-danger” })
```

-

Validation Message and Validation Summary

ValidationSummary:

- The ValidationSummary helper method generates an unordered list (ul element) of validation messages that are in the ModelStateDictionary object.
- The ValidationSummary can be used to display all the error messages for all the fields. It can also be used to display custom error messages.

```
@Html.ValidationSummary(false, “Please fix the following errors and then submit the form”)
```

LTI
Let's Solve

Model Validation

Quiz

A Larsen & Toubro Group Company

Q: _____ attributes can be used for data validation in MVC.

- DataAnnotations
- Fluent API
- DataModel
- HtmlHelper

Q: For which ModelState.IsValid Validate?

- It checks for Entityframework Model state
- It checks for valid Model State using DataAnnotations
- It checks for SQL database state
- None of the Options

LTI
Let's Solve

Security in MVC

Authentication and Authorization

- **Authentication:**
  - Authentication is a process that ensures and confirms a user's identity.
  - It is a process to validate someone against some data source.
- **Authorization:**
  - Authorization is a security mechanism which is used to determine whether the user has access to a particular resource or not.
- **Types of Authentication:**
  - **Forms Authentication:** In this type of authentication the user needs to provide his credentials through a form.
  - **Windows Authentication:** Windows Authentication is used in conjunction with IIS authentication. The Authentication is performed by IIS in one of three ways such as basic, digest, or Integrated Windows Authentication.
  - **Passport Authentication:** It is a centralized authentication service (paid service) provided by Microsoft which offers a single logon and core profile services for member sites.
  - **None:** No Authentication provided. This is default Authentication mode

Authentication and Authorization

- ASP.NET Identity:
  - ASP.NET Identity is the membership system for authentication and authorization of the users by building an ASP.NET application.
  - It allows you to add customized login/logout functionality and customized profile features that make it easy to customize the data about the logged-in user.
  - The Role based authorization also plays an important role in ASP.NET Identity.
  - In order to whether a user is authenticated or not, you can use the Identity property of Iprinciple interface as:

    ```
    @User.Identity.Name
    ```

  - To check the Role of the authenticated user:

    ```
    @User.Identity.IsInRole("Admin")
    ```

Authentication and Authorization

Configuring Forms Authentication :

- In **web.config** file of your application, you can specify the Authentication mode as shown below.

```
<authentication mode="[Windows|Forms|Passport|None]">
</authentication>
```

Authentication and Authorization

- **Enable role-based security:**
  - Roles are the permissions given to a particular user to access some resources.
  - A single user can have multiple roles and Roles plays an important part in providing security to the system.
  - In order to implement Role based Authorization, you need to create a class which inherits the **RoleProvider** class.
  - The **RoleProvider** class belongs to **System.Web.Security** namespace.
  - Add **<roleManager>** element in the web.config file of your application.

Authentication and Authorization

- [Authorize] attribute needs to be applied above Controller, if you want to give apply authorization on all controller's action method, or it can also be applied on specific action method based on business rules.

```
[Authorize]
public class EmployeesController : Controller
{ }
[Authorize(Roles ="Admin,User,Customer")]
public ActionResult Index()
{}
```

LTI
Let's Solve

Security in MVC

Quiz

Q: What is default authentication in Internet Information Services (IIS)?

- Standard User
- Administrator
- Anonymous
- None of the Options

Q: What is AuthConfig.cs in ASP.Net MVC ?

- AuthConfig.cs is used to configure route settings
- AuthConfig.cs is used to configure security settings including sites oAuth Login.
- Both of the Options
- None of the Options

LTI
Let's Solve

Filters

A Larsen & Toubro Group Company

Filters

- The Filters in ASP.NET MVC are the attribute that allows us to inject some logic or code which is going to be executed either before or after an action method is invoked.
- ASP.NET MVC Filters are used to perform the following common functionalities in your application.
  - Caching
  - Logging
  - Error Handling
  - Authentication and Authorization, etc.
- The ASP.NET MVC5 framework provides five different types of Filters. They are as follows
  - Authentication Filter (Introduced in MVC 5)
  - Authorization Filter
  - Action Filter
  - Result Filter
  - Exception Filter
- **The above is also the order of the execution of Filters if more than one filter is applied.**

Filters

- Some of the filters are already built by the ASP.NET MVC framework and they are ready to be used. For example
  - Authorize
  - ValidateInput
  - HandleError
  - RequireHttps
  - OutputCache, etc
- If the built-in filters do not serve your purpose then you can create your own custom filter as per your business requirements.
- You can configure the filters at three different levels of our application. They are as follows
  - Global Level: Register the Filter within the **Application_Start()** method of **Global.asax.cs.**
  - Controller Level: Apply the filter at the top of the controller name.
  - Action Level: Apply the filter on the top of the action method name.

Filters

- **Authentication Filter:**
  - The Authentication filter is the first filter that is going to be executed before executing any other filter or action method.
  - This filter checks that the user from where the request is coming is a valid user or not.
  - The Authentication filters in MVC implements the **IAuthenticationFilter** interface.

```
public interface IAuthenticationFilter
{
void OnAuthentication(AuthenticationContext filterContext);
void OnAuthenticationChallenge(AuthenticationChallenge filterContext)
}
```

- As of now, there is no in-built Authentication Filter provided MVC Framework.
- If you want to create Custom Authentication Filter then you need to implement he **IAuthenticationFilter** interface.

Filters

- **Authorization Filter:**
- The Authorization Filters executed after the Authentication Filter.
- This filter is used to check whether the user has the rights to access the particular resource or page.
- The built-in **AuthorizeAttribute** and **RequireHttpsAttribute** are examples of Authorization Filters.
- The Authorization Filters in MVC implements the **IAuthorizationFilter** interface.

```
public interface IAuthorizationFilter
{
void OnAuthorization(AuthorizationContext filterContext)
}
```

- To create a **Custom Authorization Filter** then you need to implement the **IAuthorizationFilter** interface.

Filters

- **Action Filter:**
  - The Action Filter will be executed before the action method starts executing or after the action has executed.
  - You can implement some custom logic that should be get executed before and after an action method executes.
  - The Action filters implement the **IActionFilter** interface that has two methods **OnActionExecuting** and **OnActionExecuted.**

```
public interface IActionFilter
{
void OnActionExecuted(ActionExecutedContext filterContext);
void OnActionExecuting(ActionExecutingContext filterContext);
}
```

Filters

- **Result Filters:**
  - The Result filters in MVC application are executed before or after generating the result for an action.
  - Action Result type can be ViewResult, PartialViewResult, RedirectToRouteResult, RedirectResult, ContentResult, JsonResult, FileResult and EmptyResult which derives from the ActionResult abstract class.
  - Result filters are called after the Action filters.
  - The Result Filters in MVC implements the **IResultFilter** interface.

```
public interface IResultFilter
{
void OnResultExecuted(ResultExecutedContext filterContext);
void OnResultExecuting(ResultExecutingContext filterContext);
}
```

Filters

- **Exception Filter:**
  - The Exception filters are executed when there is an unhandled exception occurs during either the execution of actions or filters.
  - The in-built **HandleErrorAttribute** class is an example of Exception Filters.
  - The **IExceptionFilter** interface is used to create **Custom Exception Filter** which provides the **OnException** method which will be executed when there is an unhandled exception occurs during the actions or filters execution.

```
public interface IExceptionFilter
{
void OnException(ExceptionContext filterContext);
}
```

LTI
Let's Solve

Filters

Quiz

A Larsen & Toubro Group Company

Q: Which filter will be executed at first using ASP.Net MVC?

- Action filters
- Authorization filters
- Response filters
- Exception filters

Q: Which filter will be executed at last using ASP.Net MVC?

- Action filters
- Authorization filters
- Response filters
- Exception filters

Let's Solve