What is ASP.NET MVC?

ASP.NET MVC is a Web Application Framework, that gives you a powerful, MVC architecture-based way to build dynamic web applications that enables a clean separation of concerns and that gives you full control over markup.

- Part of Asp.Net.
- Alternative to Asp.Net Web Forms.
- Introduced in 2009; current version is “ASP.NET MVC 5.2.5”.

Main Advantages:

- Clean separation of concerns.
- Faster performance.

What is MVC

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

Model

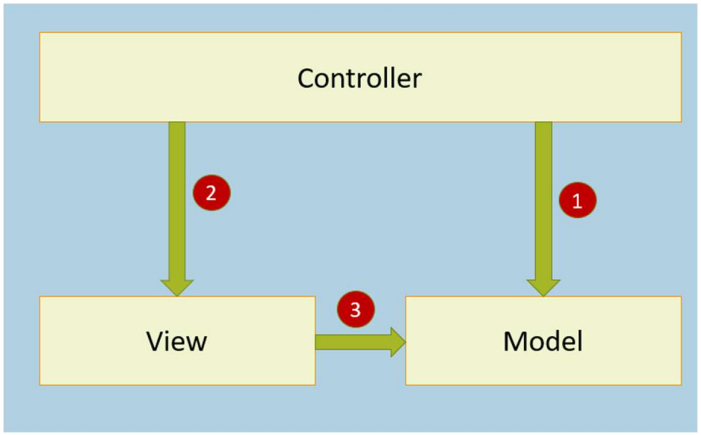
- Data Structure
- Business Logic

View

- Presentation Logic
- Reads data from Model

Controller

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



Advantages:

Supports Clean Separation of Concerns.

Supports Unit Testing.

Supports Dependency Injection.

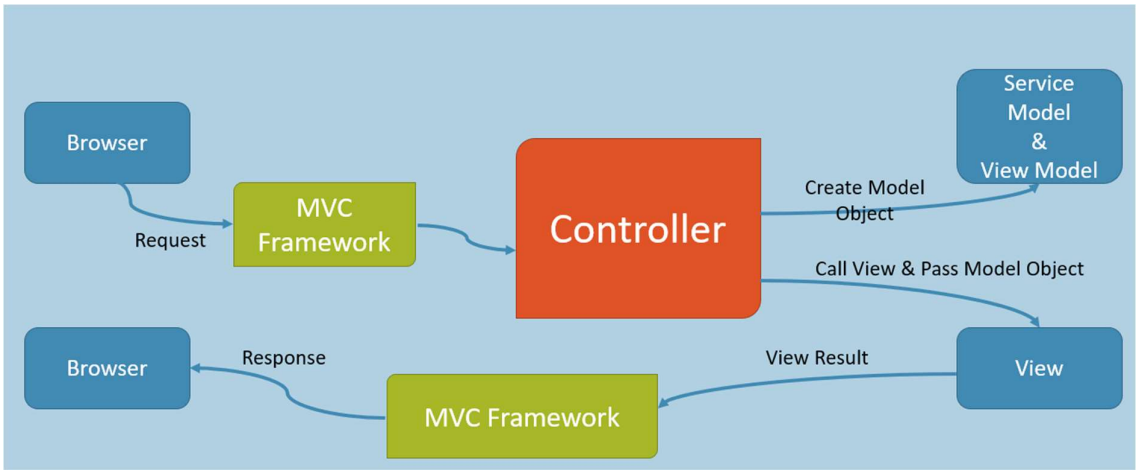
Supports Faster Performance than ASP.NET Web Forms.

No Page Life Cycle, Controls, Postback and ViewState.

Controllers

Controller is a class that defines execution flow in MVC application.

Controller receives request from browser, call the model, call the view



- 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 suffix "Controller". Eg: HomeController

```
1. public class classnameController : System.Web.Mvc.Controller
2. {
3. }
```

Action Methods

Controller's methods are called as "action methods"; each action method represents a specific operation.

Controller with Action Methods

```
1. public class classnameController : System.Web.Mvc.Controller
2. {
3. public returnType methodName(dataType param1, …)
4. {
5. …
6. }
7. }
```

Asp.Net Web Forms (vs) Asp.Net Mvc

Asp.Net Mvc

1. Supports Clean Separation of Concerns.
2. Business logic layer, Controllers are unit testable.
3. Supports Dependency Injection.
4. Faster Performance than ASP.NET Web Forms.
5. No Page Life Cycle, Controls, Postback and ViewState.
6. Runs based on the principle of "Web is stateless".

Asp.Net Web Forms

1. Presentation Logic (.aspx) and Event Logic (.aspx.cs) are tightly coupled.
2. Presentation Logic and Application Logic are not unit testable.
3. Doesn't Support Dependency Injection.
4. Slower Performance than ASP.NET MVC.
5. Supports Page Life Cycle, Controls, Postback and ViewState.
6. Hides the "stateless nature of web" and tries to mask the developers as they are in "stateful environment".

Versions of Asp.Net Mvc

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

1. MVC pattern with ASPX Engine
2. HTML Helpers
3. Routing
4. Unit Testing

ASP.NET MVC 2.0

- March 2010
- Visual Studio 2008, 2010
- .NET 3.5, 4.0

Features

1. Strongly Typed HTML Helpers
2. Support for data annotations

ASP.NET MVC 3.0

- January 2011
- Visual Studio 2010, 2012
- .NET 4.0

Features

1. Razor
2. EF Code First
3. Partial Page Output Caching
4. ViewBag
5. Global Filters

ASP.NET MVC 4.0

- August 2012
- Visual Studio 2010, 2012, 2013
- .NET 4.0, 4.5

Features

1. Asp.Net Web Api
2. Bundling and Minification
3. Asynchronous Controllers

ASP.NET MVC 5.0

- October 2013
- Visual Studio 2012, 2013, 2015, 2017
- .NET 4.5, 4.5.1 and up

Features

1. Asp.Net Web Api 2
2. Asp.Net Identity
3. Attribute Based Routing
4. Filter Overrides

Folder Structure of Mvc App

ProjectFolder:

- \App_Start: Contains the files that needs to be executed on the first request.
- \App_Data: Contains SQL Server LocalDb Database Files.
- \Controllers: Contains all controllers.
- \Models: Contains all models.
- \Views: Contains all views.
- \Views\web.config: Contains 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.
- \Web.config: Contains web application configuration settings, that needs to be initialized on each request.

Packages of ASP.NET MVC

1. Microsoft.AspNet.Mvc

Contains the necessary DLL files that are needed to execute ASP.NET MVC applications.

2. Microsoft.AspNet.Razor

Contains the necessary DLL files that are needed to execute Razor views in MVC.

3. Microsoft.AspNet.WebPages

Contains the necessary DLL files that are needed to execute HTML Helpers in Razor Views.

4. Microsoft.CodeDom.Providers.DotNetCompilerPlatform

Contains a new generation .net compiler called Roslyn compiler.

5. Microsoft.Web.Infrastructure

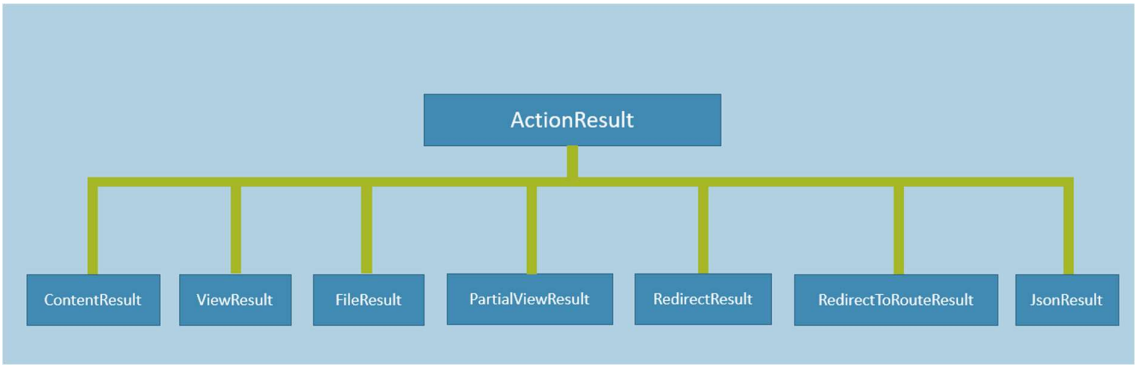
Necessary to dynamically register HTTP modules at run time.

ActionResult

ActionResult is a class, that represents “result of an action method”.

Asp.Net Mvc recommends to specify action method's return type as "ActionResult".

ActionResult is an abstract class that has several child classes, you can return an object of any of the child classes.



Types of ActionResult

1. ContentResult: Represents any content with specific content-type.

Ex: receives emp id as i/p argument and sends emp name as o/p

2. ViewResult: Represents result of a view. Sends rendered html o/p to the view.

3. FileResult: Represents content of a file.

Ex: receives emp id as i/p argument and sends emp payslip as o/p

4. JsonResult: Represents json object / json array.

5. RedirectResult: Represents redirection to other website (HTTP 302).

6. RedirectToRouteResult: Represents redirection to a specific action method (HTTP 302).

7. PartialViewResult: Represents the result of partial view.

Methods to return different types of ActionResult

1. ContentResult

```
ContentResult Content(string Content, string ContentType)
```

2. ViewResult :

```
ViewResult View(string ViewName)
```

3. FileResult

```
FileResult File(string FilePath, string ContentType)
```

4. JsonResult

```
JsonResult Json(object data, JsonRequestBehavior behavior)
```

5. RedirectResult

```
RdirectReesult Redirect(string url)
```

6. RedirectToRouteResult

```
RedirectToRouteResult RedirectToAction(string ActionName,
string ControllerName)
```

7. PartialViewResult

PartialViewResult PartialView(string ViewName)

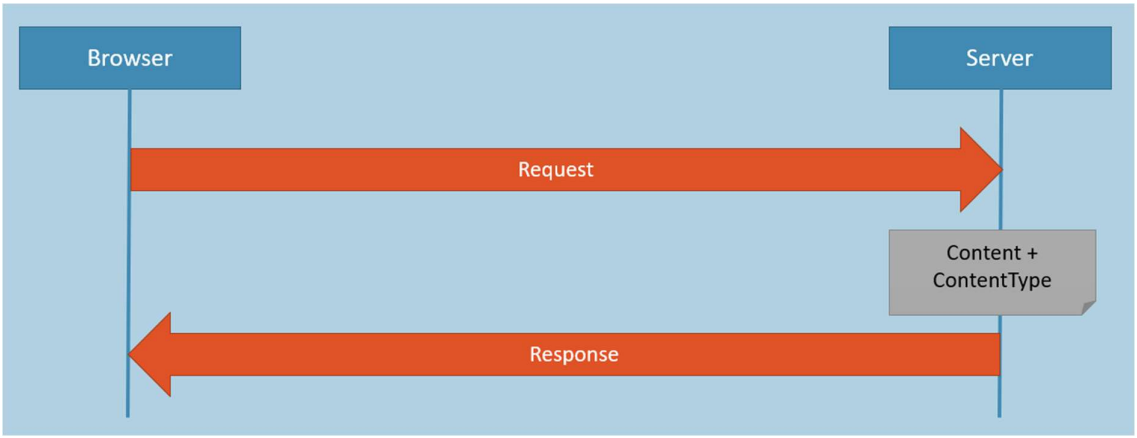
ContentResult

This class's object represents some content (plain text / html / css / javascript) along with its content type.

The default content type is "text/html".

The Content( ) method creates and returns an object of "ContentResult" class.

Syntax: `return Content("content", "content type");`

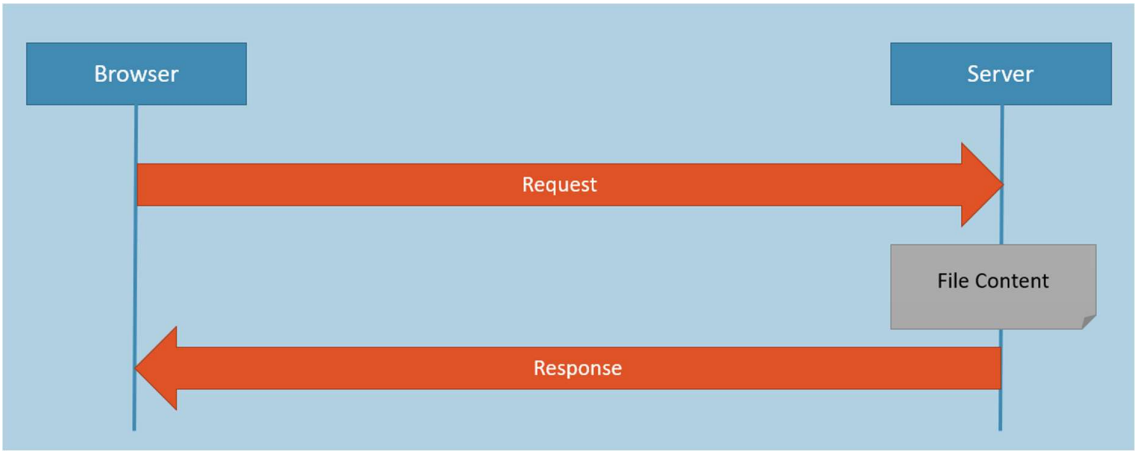


FileResult

This class's object represents content of a file.

The File( ) method creates and returns an object of "FileResult" class.

Syntax: `return File("~/filepath");`



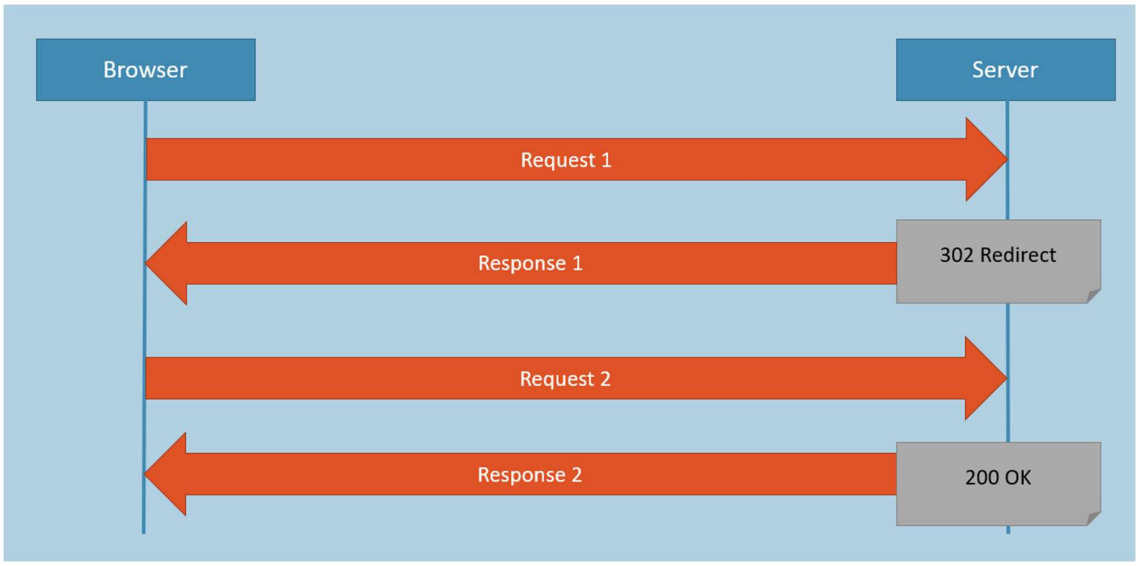
RedirectResult

This class's object represents redirection from an action method to other website.

It sends HTTP 302 response to the browser; so the browser sends another request to the specific url.

The Redirect( ) method creates and returns an object of "RedirectResult" class.

Syntax: `return Redirect ("url");`



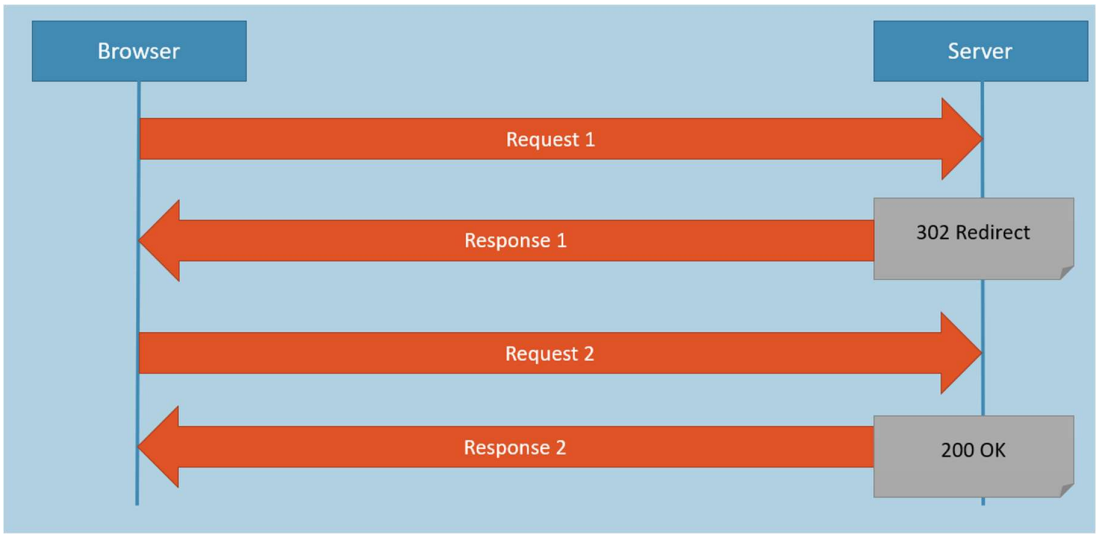
RedirectToRouteResult

This class's object represents redirection from one action method to another action method.

It sends HTTP 302 response to the browser; so the browser sends another request to the specific action method.

The RedirectToAction( ) method creates and returns an object of "RedirectToRouteResult" class.

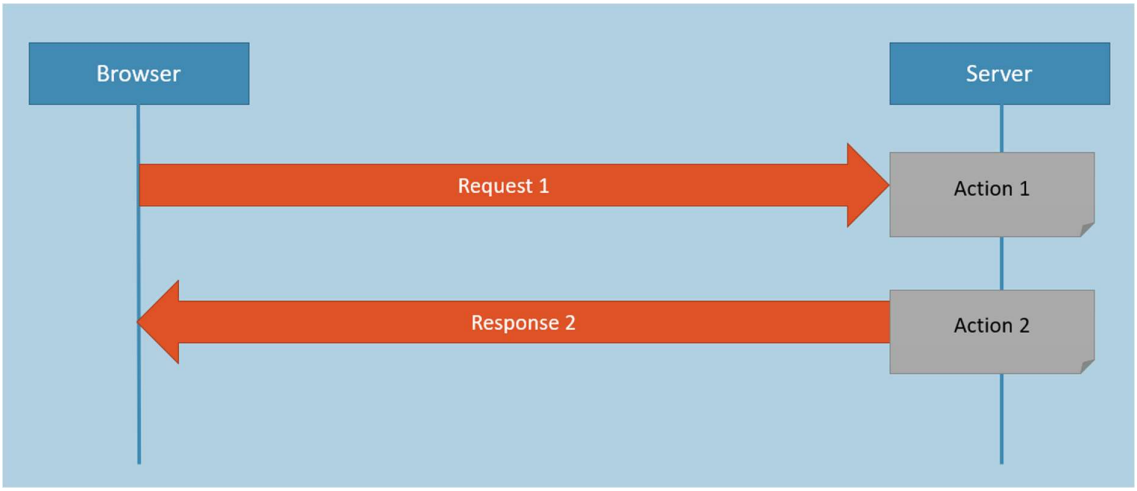
Syntax: `return RedirectToAction("action name", "controller name");`



Server.Transfer

This method transfer the current request itself to another action method.

Syntax: `Server.TransferRequest("url");`



View

View is a ".cshtml" file that contains presentation logic of the application.

Controller passes model object to view and view reads data from the same model object.

Typically, every action method contains a view.

Controller calls the View.

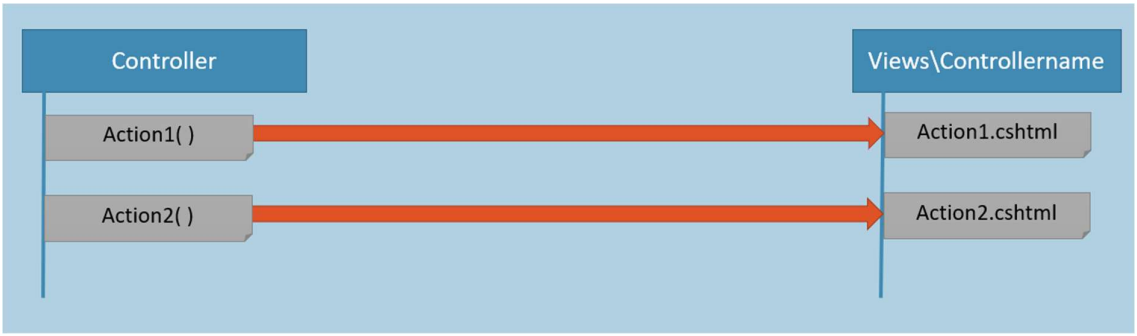
Location of View File

View file (.cshtml) should be present in the "Views\Controllername" folder.

It is recommended to keep view name same as "action name".

Invoking the view [if Action name and View name are same] : `return View( );`

Invoking the view [if Action name and View name are different] : `return View("view name");`



Presentation Logic in View

View.cshtml

```
1. HTML (for creating elements)
2. C# (for programming logic such as conditions and loops)
```

View Engines

View

```
1. HTML Code
2. @{  C# Code  }
3. HTML Code
4. @{  C# Code }
5. HTML Code
```

View Engine provides a set of syntaxes to write 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:

1. ASPX View Engine
2. Razor View Engine

ASPX View Engine

```
1. <%
2.   c#.net code
3. %>
```

Razor View Engine

```
@{
c#.net code
}
```

ASPX (vs) Razor

ASPX View Engine

```
<%
  c#.net code
%>
```

1. ASPX is older view engine, supported in MVC 1, 2, 3, 4.
2. File extension is .aspx.
3. Syntax is cumbersome, when used in real-world views.
4. You can't write the html tags in the code blocks.

Razor View Engine

```
@{
c#.net code
}
```

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

Razor - Expressions

Razor Expressions are used to render the value of a variable into the response.

`@variablename`

Razor - Code Blocks

Razor Code Blocks contains one or more lines of c#.net code.

You can use razor code blocks any no. of times in the view.

HTML tags also allowed in the code blocks.

C# code executes on the server; output will be rendered into response.

HTML code will be rendered into response as-it-is.

```
html code
@{
  c#.net code line 1
  c#.net code line 2
```

```
    …
}
html code
```

Razor - If

Razor If is used to check the condition and render some html content if the condition is true.

Even if u have single statement under if-block also the curly braces are must, u shouldn't avoid it like in c sharp.

```
@if (condition)
{
    html content here
}
else
{
    html content here
}
```

Razor - For

Razor-For is used to execute the code / content repeatedly, certain no. of times.

```
@for (initialization; condition; incrementation or decrementation)
{
    html content here
}
```

Razor - Foreach

Razor-Foreach loop is used to read elements from collection and arrays render the same to the response.

```
@foreach (var variable in collection)
{
    html content here
}
```

Razor - Literals

Razor-Literal is used to render some static text inside the razor code block / razor if / razor for.

`@: literal`

Razor - Comments

Razor Comments comment any code inside it (html / css / js / c#)

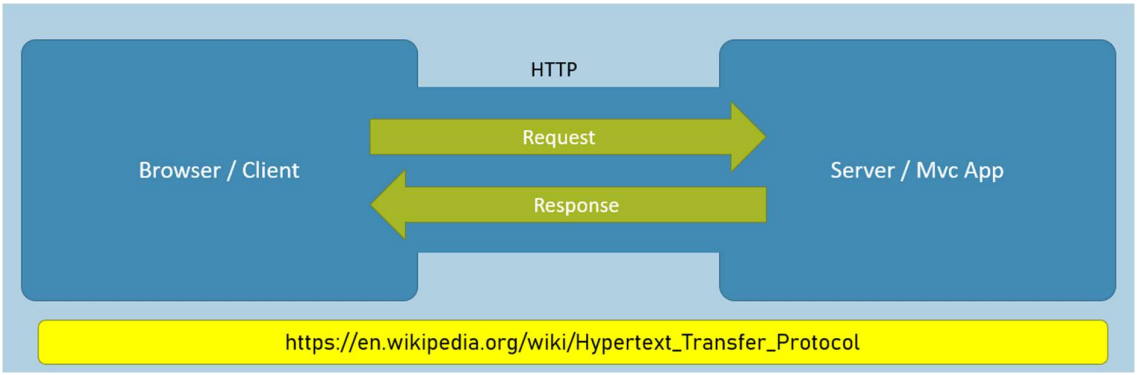
```
@*
   comment line 1
   comment line 2
```

4. …
5. *@

HTTP

HTTP is an application-protocol that defines set of rules to send request from browser to server and send response from server to browser.

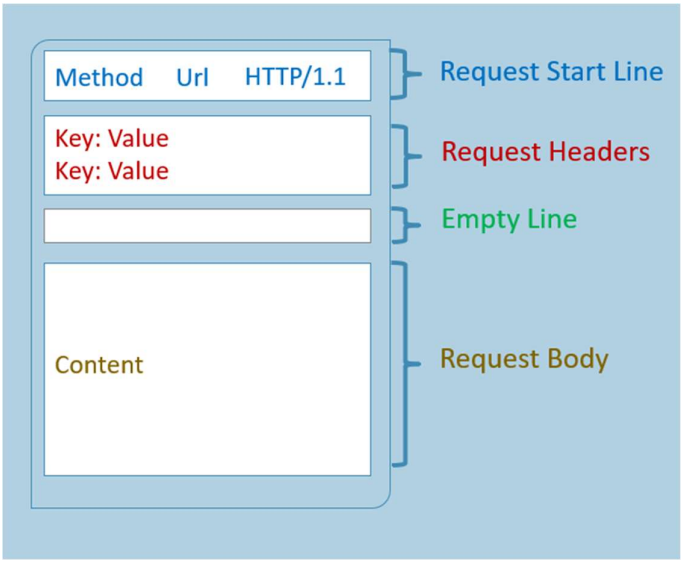
Initially developed by Tim Berners Lee, standardized by IETF (Internet Engineering Task Force) and W3C (World Wide Web Consortium).



HTTP Concepts

1. HTTP Request Message Format
2. HTTP Methods
3. HTTP Request Headers
4. HTTP Response Message Format
5. HTTP Response Headers
6. HTTP MIME Types
7. HTTP Status Codes

HTTP Request Message Format



Method: Get / Post

Url: Url of server program

Request Headers: Key/Value Pairs to send to server

Request Body: Content to send to server

Eg:

```
1. GET /customer/dashboard.html HTTP/1.1
2. Host: www.webitename.com
3. Accept: text/html
4. Accept-Language: en-us
5. Accept-Encoding: gzip, deflate
6. User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)
```

https://en.wikipedia.org/wiki/HTTP_message_body

HTTP Methods

GET

- Used to retrieve data from server: Ex: Search
- Parameters will be appended to the url as "query string. Eg: http://localhost:portno?parameter=value&parameter=value&...
- Parameters will be displayed in the address bar of the browser.
- Can pass limited no. of characters only. Maximum no. of characters per url: 2048 (depends on browser).
- Can be cached by the browser / search engines.
- Can pass only ASCII characters; can't pass Unicode characters.

POST

- Used to perform insert / update / delete operations.
- Parameters are not sent in the url, but will be sent in "request body".
- Parameters will not be displayed in the address bar of the browser.
- Can send unlimited data.
- Can’t be cached by the browser / search engines.
- Supports Unicode characters / binary data.

https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol

HTTP Request Headers

1. Accept: Informs the server what type of content the client is expecting. Ex: text/html

2. Accept-language: Informs the server which language content the client is expecting. Ex: en-US

3. Content-Type: MIME type of request body (useful for POST and PUT).

Eg:

```
text/x-www-form-urlencoded [or] multipart/form-data
MyVariableOne=ValueOne&MyVariableTwo=ValueTwo
```

4. Content-Length : Length of request body (useful for POST and PUT). Ex: 100

5. Date : Date and time of request. Ex: Tue, 15 Nov 1994 08:12:31 GMT

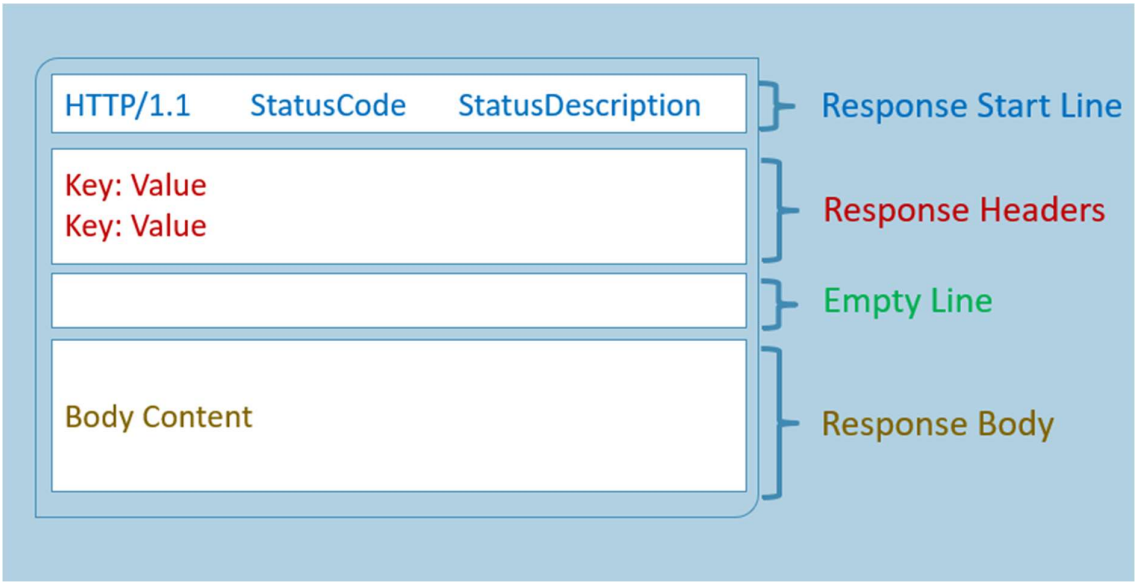
6. Host : Server domain name. Ex: www.website.com

7. User-Agent : Browser details. Ex: Mozilla/5.0 Firefox/12.0

8. Cookie : Contains cookies to send to server. Ex: x=100

https://en.wikipedia.org/wiki/List_of_HTTP_header_fields

HTTP Response Message Format



Eg:

```
HTTP/1.1  200  OK
Date: Sun, 18 Oct 2009 08:56:53 GMT
Server: Apache/2.2.14 (Win32)
Last-Modified: Sat, 20 Nov 2004 07:16:26 GMT
Content-Length: 44
Connection: close
Content-Type: text/html
<html><body><h1>It works!</h1></body></html>
```

StatusCode: 200 / 404 / 500 etc.

StatusDescription: OK / Not Found / Internal Server Error etc.

Response Headers: Key/Value Pairs to send to browser

Response Body: Content to send to browser

https://en.wikipedia.org/wiki/HTTP_message_body

HTTP Response Headers

Cache-Control: Indicates Browser cache should be enabled or not. Ex: max-age=60

Set-Cookie: Contains cookies to send to browser. Eg: x=100

Content-Type: MIME type of response body. Eg: text/plain

Content-Length: Length of response body. Eg: 100

https://en.wikipedia.org/wiki/List_of_HTTP_header_fields

HTTP MIME Types

- text/plain
- text/html
- text/css
- text/javascript
- application/json
- text/xml
- image/png
- image/jpeg
- audio/mp3
- video/mp4

https://en.wikipedia.org/wiki/MIME

https://en.wikipedia.org/wiki/Media_type

HTTP Status Codes

100: Continue

200: OK

201: Created

302: Redirection

304: Not Modified

400: Bad Request

401: Unauthorized

404: Not Found

500: Internal Server Error

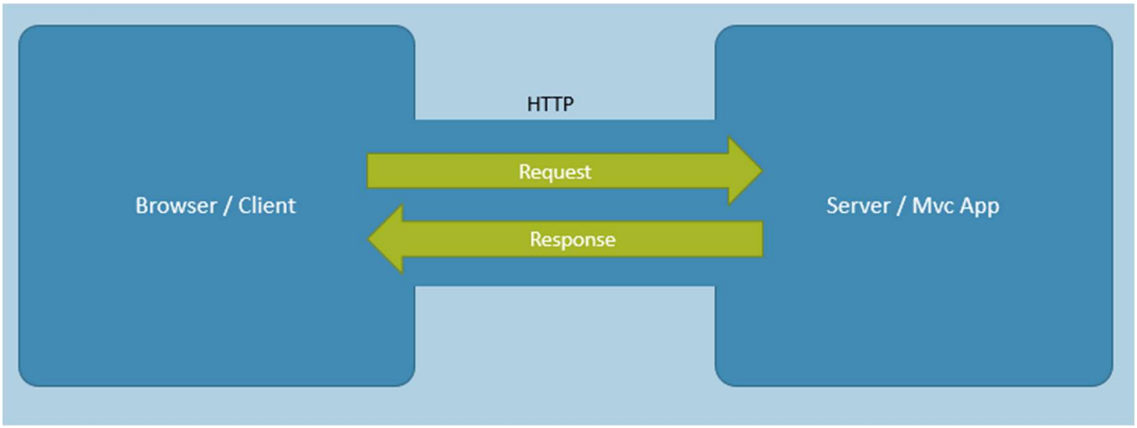
https://en.wikipedia.org/wiki/List_of_HTTP_status_codes

Request

The "Request" object contains request details sent from browser to server.

It is a member of “System.Web.Mvc.Controller” class.

It is of “System.Web.RequestBase” type.



Request Properties

1. Request.Url: Represents current url. Eg: http://localhost:1020/home/index

2. Request.PhysicalApplicationPath: Represents current folder path. Eg: c:\mvc\MyFirstApp

3. Request.Path: Represents current virtual path. Eg: /home/index

4. Request.Browser.Type: Represents current browser name. Eg: Chrome56

5. Request.QueryString: Represents current query string. Eg: Param1=Value1&Param2=Value2

6. Request.Headers: Represents request headers (key/value pairs).

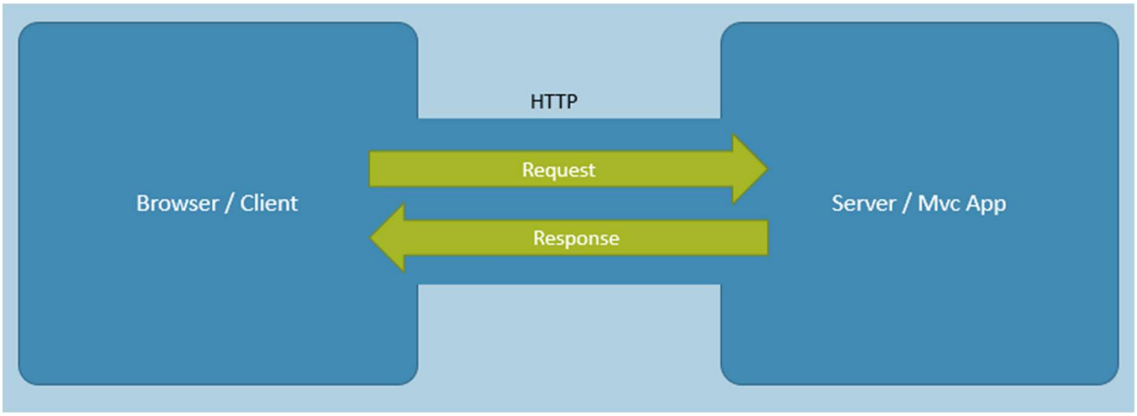
7. Request.HttpMethod: Represents http method of the request (GET / POST).

Response

The "Response" object contains the content that is to be sent from server to browser.

It is a member of “System.Web.Mvc.Controller” class.

It is of “System.Web.ResponseBase” type.



Response Properties

1. Response.Write( ): Sends given content to the browser.

2. Response.ContentType: Represents type of response content. Eg: text/html

3. Response.Headers: Represents response headers that can be sent from server to browser.

4. Response.StatusCode: Represents status of request. Eg: 200, 302 etc.

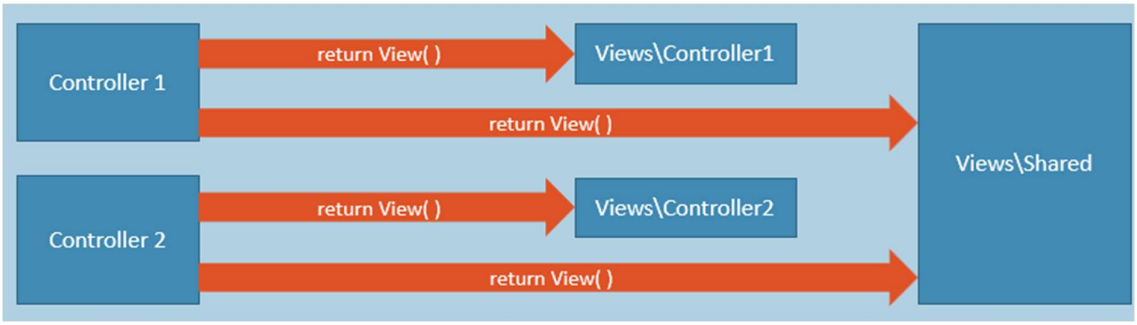
Shared Views

Shared views are present in the "Views\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; if it is not found, it will search in the "Views\Shared" folder.

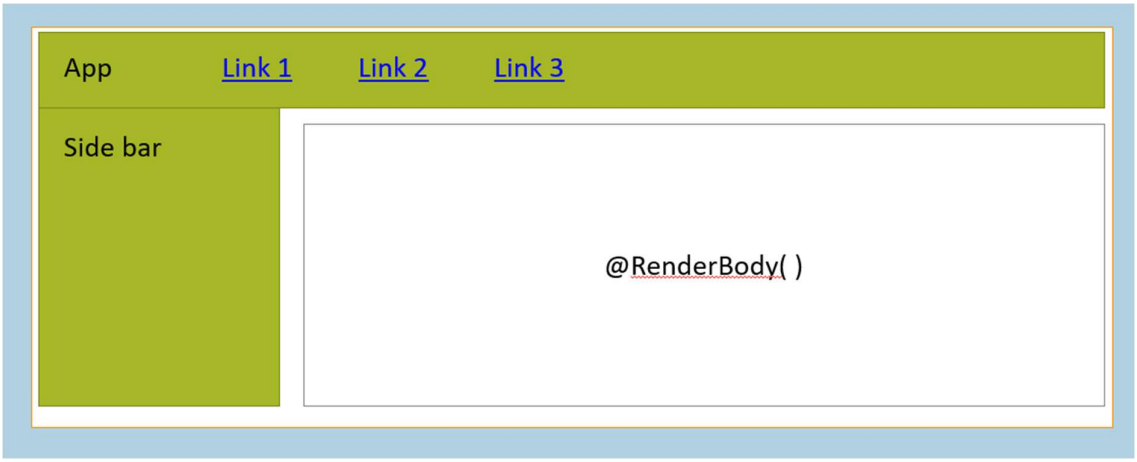


Layout Views

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.



Sections in Layout Views

Sections are used to display view-specific content in the layout view.

Sections are defined in the view and rendered in the layout view.

Layout View

```
1. @RenderSection(“section name”)
```

View

```
1. @section sectionname
2. {
3.     content here
4. }
```

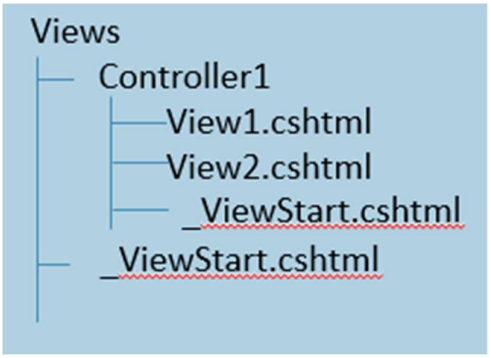
_ViewStart.cshtml

It defines the default layout view of all the views of a folder.

If it is present in "Views" folder, it defines the default layout view of all the 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 View --> Generate View Result à Response



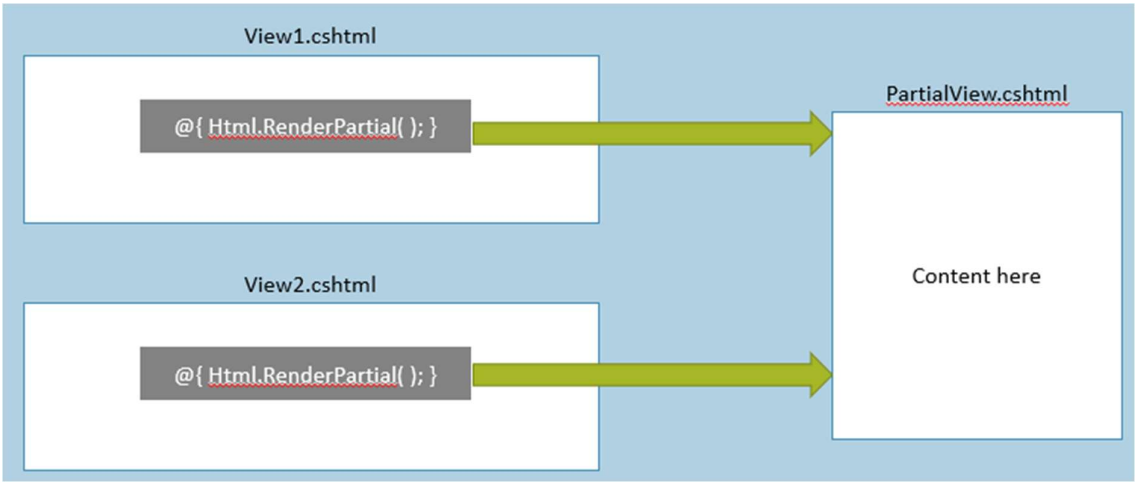
_ViewStart.cshtml

```
@{
  Layout = "Path of layout view";
}
```

Partial Views

Partial View is a small view that contains content that can be shared among multiple views.

Can be present in "Views\Controllername" folder or in "Views\Shared" folder.



View (vs) Partial View

View

1. View can contain a layout page.
2. _ViewStart.cshtml file will be called before the execution of view.
3. View can have html structured elements such as html, head, body etc.

Partial View

1. Partial view doesn't contain a layout page.
2. _ViewStart.cshtml file will not be called before the execution of partial view.
3. Partial view doesn't contain any html structured elements such as html, head, body etc.

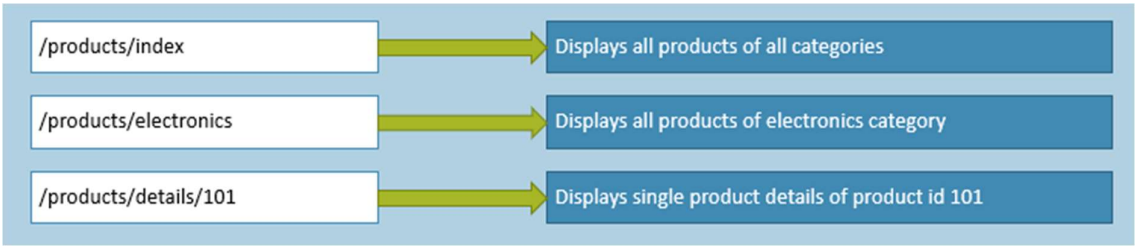
URL Routing

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. Eg: /folder/subfolder/search.aspx
- URLs are user-friendly. New URL. Eg: /products/search
- URLs are search-engine-friendly.



Route

Route 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.

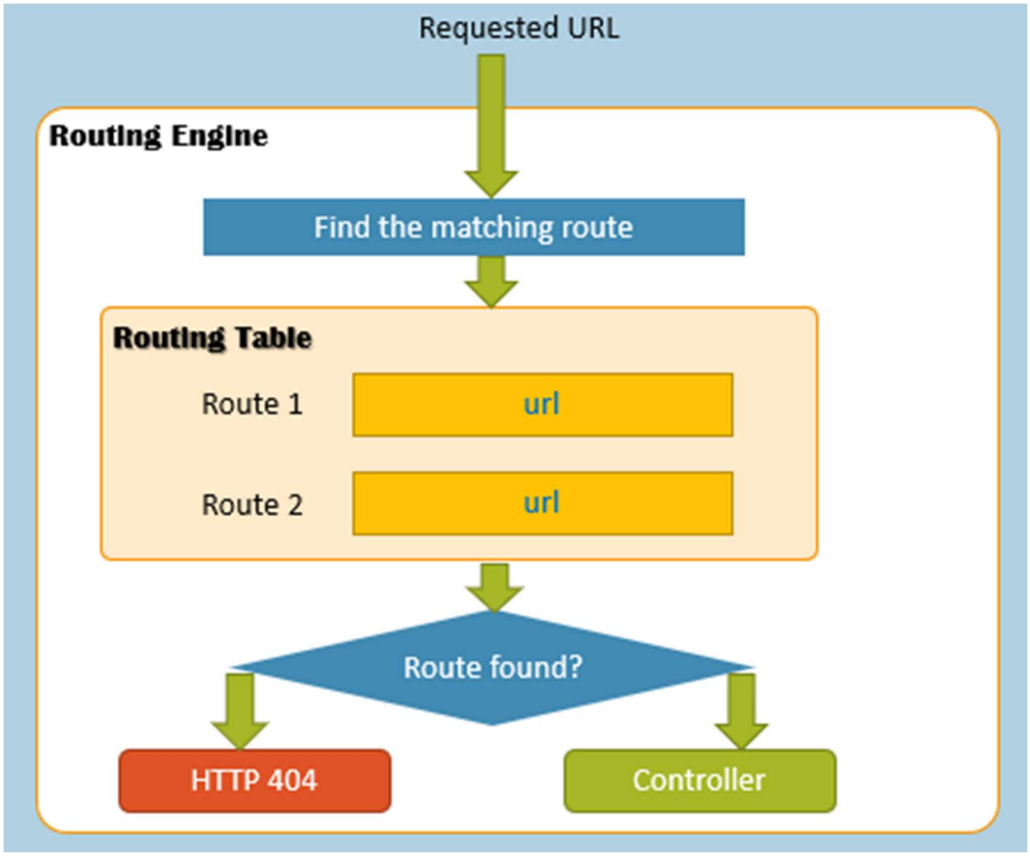
All the routes are stored in "Routing table", which will be stored in memory (RAM).

```
products/  details/  {productid}
```

Eg:

```
1. products/details/101
2. products/details/102
3. products/details/103
4. …
```

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 one of the routes in the RouteTable. If one matches, it goes to the corresponding controller.

URL Routing - Development

Global.asax

```
1. protected void Application_Start()
2. {
3.     RouteConfig.RegisterRoutes(RouteTable.Routes);
4. }
```

RouteConfig.cs

```
1. public static void RegisterRoutes(RouteCollection routes)
2. {
3.     routes.MapRoute(
4.      name: "route1",
5.      url: "{param1}/{param2}",
6.      defaults: new { param1 = "defaultvalue", param2 = "defaultvalue" },
7.      constraints: new { param1 = "constraint1", param2 = "constraint2" },
```

```
    );
}
```

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 that are added to the RouteCollection, will be stored in the RouteTable for the first request.

For each request, Routing Engine just searches for the matching route in the RouteTable.

Attribute Routing

Problems in Conventional Routing

- Very difficult to understand for the developers, which route is for which action methods.
- Very difficult to avoid conflicts among the routes [some times, we can't apply constraints].
- Overall, some routes for multiple action methods; some routes for specific action methods; Overall, it looks cumbersome.

RouteConfig.cs

```
protected static void RegisterRoutes( RouteCollection routes)
{
    routes.MapRoute("name", "url", defaults, constraints);
    routes.MapRoute("name", "url", defaults, constraints);
    routes.MapRoute("name", "url", defaults, constraints);
    …
}
```

Types of URL Routing

Conventional Routing

1. It is the traditional way of routing. `routes.MapRoute( "name", "url", { defaults } )`
2. Applicable for a specific / multiple action methods.
3. Routes will be cumbersome.
4. Enabled by default.
5. Supports parameters & constraints.

Attribute Routing

1. New and preferred way of routing:

```
["url"]
public ActionResult Methodname( )
{
}
```

2. Applicable for specific action method. Each action method must have "Route" defined.

3. Routes are very clearly understandable for the developers.

4. Should be enabled using: routes.MapMvcAttributesRoutes( );

5. Also supports parameters & constraints.

Models

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

```
public class ViewModel
{
    public dataType propertyName { get; set; }
}
```

Represents the structure of the data that you want to display to user.

Domain Model

```
public class DomainModel
{
  public dataType propertyName { get; set; }
}
```

Represents the structure of the data that you want to store in the database table.

Service Model

```
public class ServiceModel
{
    public returnType methodName { … }
}
```

Represents business logic (code that needs to be executed before inserting / updating etc.).

Strongly Typed Views

View that is associated to a specific model class is called as "Strongly Typed View".

Strongly typed views have to specify the model class name with @model directive at the top of the view.

Strongly Typed Views can receive model objects from the controller.

To receive model collections from the controller: `@model List<modelClassNameHere>`

View

```
1. @model modelClassNameHere
2. @Model.propertyName
```

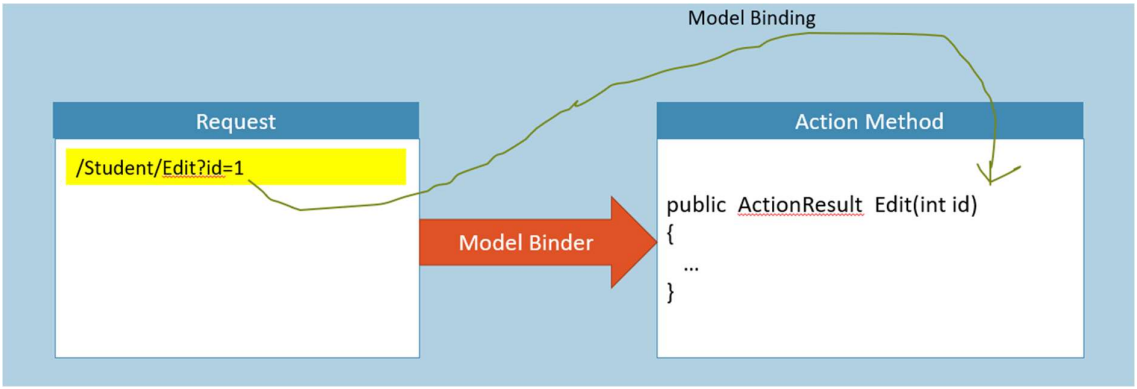
Model

```
1. class modelClassName
2. {
3. }
```

Model Binding

Process of “receiving values from different sources of the request and passing them as arguments to action method”.

Assigns values to different parameters of the action method automatically!



Model Binding with Complex Types

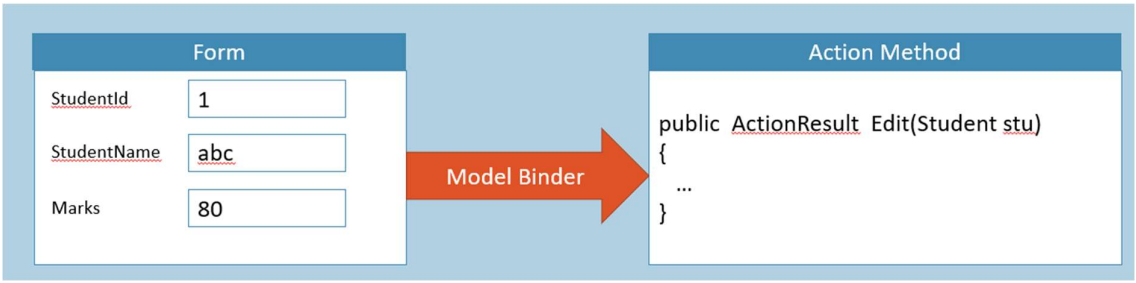
Model Binding can work with Complex Types.

Model Binding can automatically convert form field data or query string values to the properties of a complex type parameter of an action method.

Default values are null or zero (0).

Model Class

```
1. public class Student
2. {
3.     public int StudentId { get; set; }
4.     public string StudentMarks { get; set; }
5.     public double Marks { get; set; }
6. }
```



Sources of Model Binding

- Query String
- Form Data **Eg:** <input type="text" name="EmpName">
- Route Data
- JSON request body [in case of AJAX]

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.

Form

```
1. StudentId:   1
2. StudentName:   abc
3. Marks:  80
```

Model Class

```
1. public class Student
2. {
3.     public int StudentId { get; set; }
4.     public string StudentName { get; set; }
5.     public double Marks { get; set; }
6. }
```

Action Method (with Include)

```
1. public ActionResult Create([Bind(Include = "StudentId, StudentName")]
   Student stu)
2. {
3.     …
```

```
4. }
```

Action Method (with Exclude)

```
1. public ActionResult Create([Bind(Exclude = "StudentId, StudentName")]
   Student stu)
2. {
3. …
4. }
```

Custom Model Binders

Whenever the list of fields of the view and list of fields of model are different, you have to use "Custom Model Binders", to map exactly, which value of view should be stored in which property of model.

Form

```
1. D.No:    1-2/3
2. Street:    MG Road
3. Landmark:   Church
```

Custom Model Binder

```
1. public class CustomBinder : IModelBinder
2. {
3.    public object BindModel(ControllerContext controllerContext,
   ModelBindingContext bindingContext)
4.    {
5.       ...
6.    }
7. }
```

Model Class

```
1. public class Student
2. {
3.    public int StudentId { get; set; }
4.    public string StudentName { get; set; }
5.    public string Address { get; set; }
6. }
```

Action Method

```
1. public ActionResult Create( [ModelBinder( typeof(CustomBinder) ) ] Student
   stu)
2. {
3. …
4. }
```

Add Custom Model Binders to "Binders" collection in Application_Start( ) method.

Global.asax

```
public void Application_Start( )
{
    ModelBinders.Binders.Add( typeof(Student), new CustomBinder );
}
```

Introduction to 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.

```
class Modelclass
{
  public Propertyname { get; set; }
  public Propertyname { get; set; }
   …
}
```

```
Table: tablename
- Column1
- Column2
…
```

Object-Relational Mapping ORM

Eg:

```
class Employee
{
    public int EmpID { get; set; }
    public string EmpName { get; set; }
    public decimal Salary { get; set; }
    …
}
```

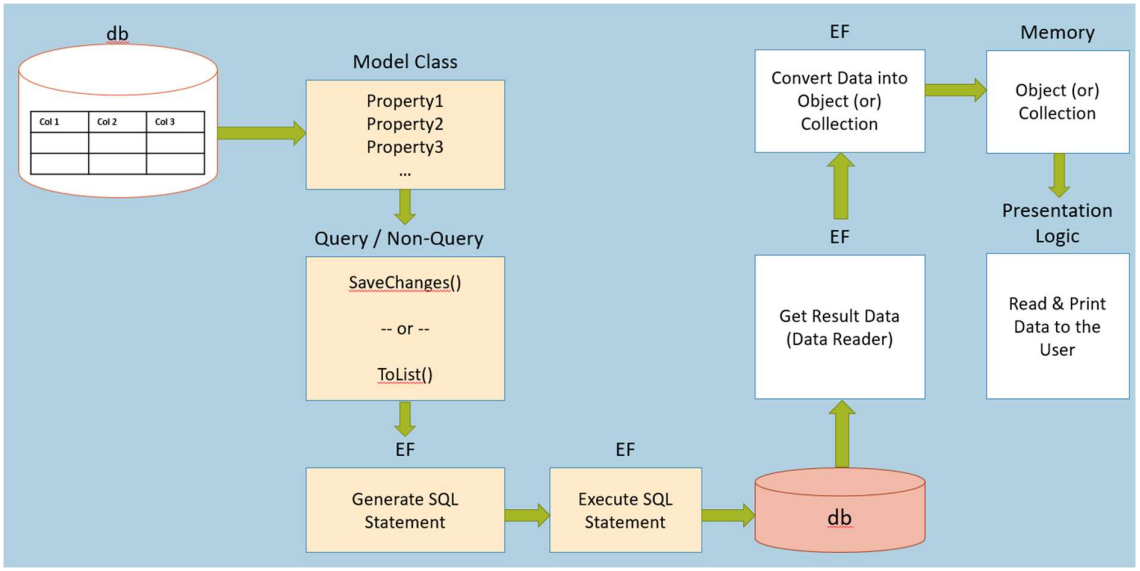
```
Table: Employees
- EmpID int
- EmpName varchar(200)
- Salary decimal
…
```

Features of EF

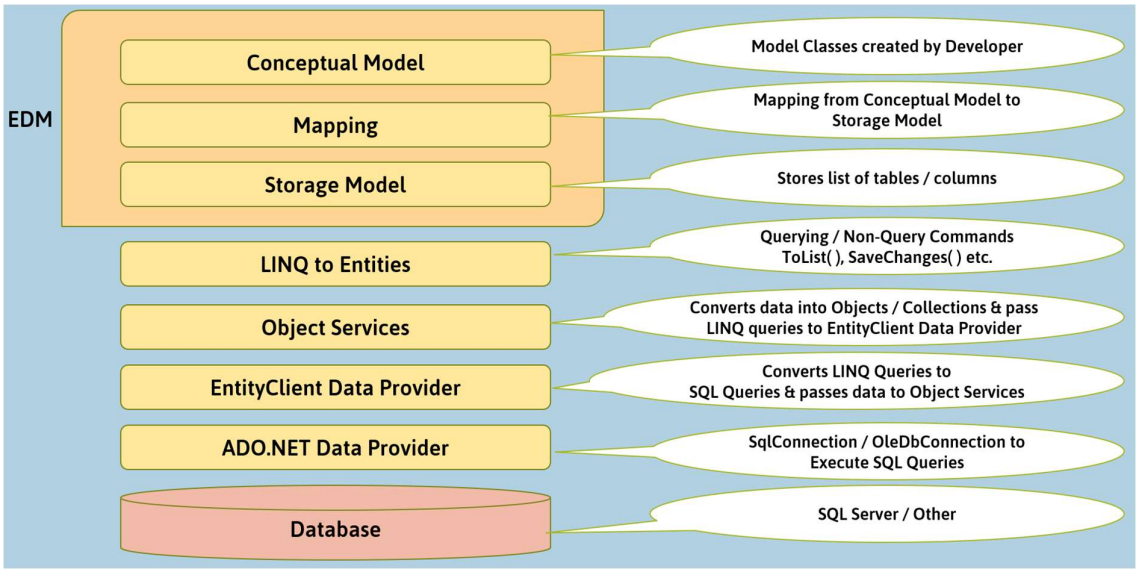
1. Modeling
2. Querying
3. Change Tracking
4. Saving
5. Concurrency
6. Transactions

7. Caching

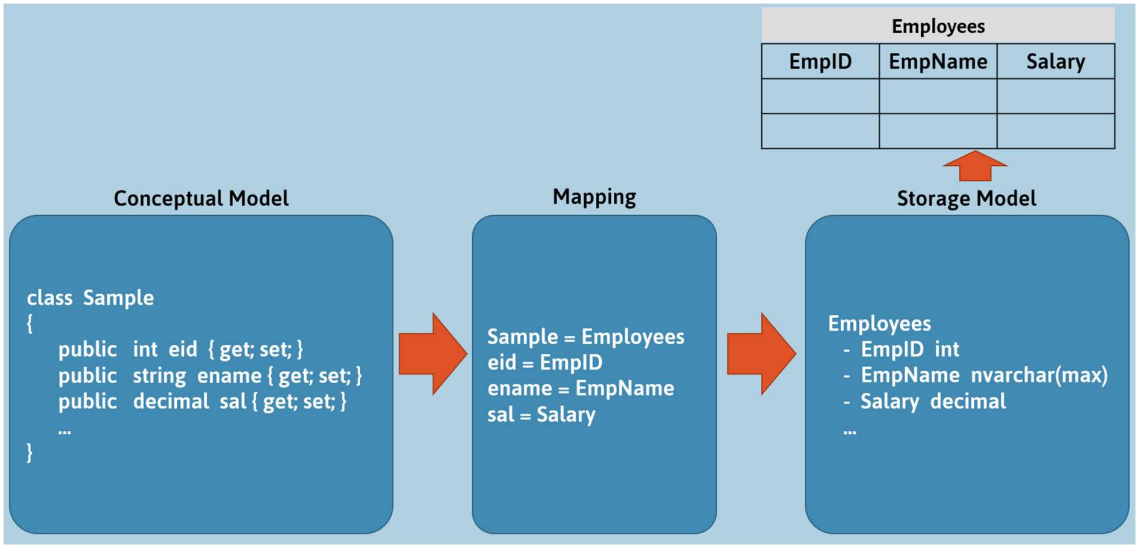
Entity Framework - Work Flow



Entity Framework - Architecture



Eg:



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.

```
1. using Modelclassname;
2.
3. class Classname : DbContext
4. {
5.     public DbSet<Modelclass> { get; set; }
6.     public DbSet<Modelclass> { get; set; }
7.     …
8. }
```

EF : Retrieving All Rows

Retrieving all rows and all columns from the database table

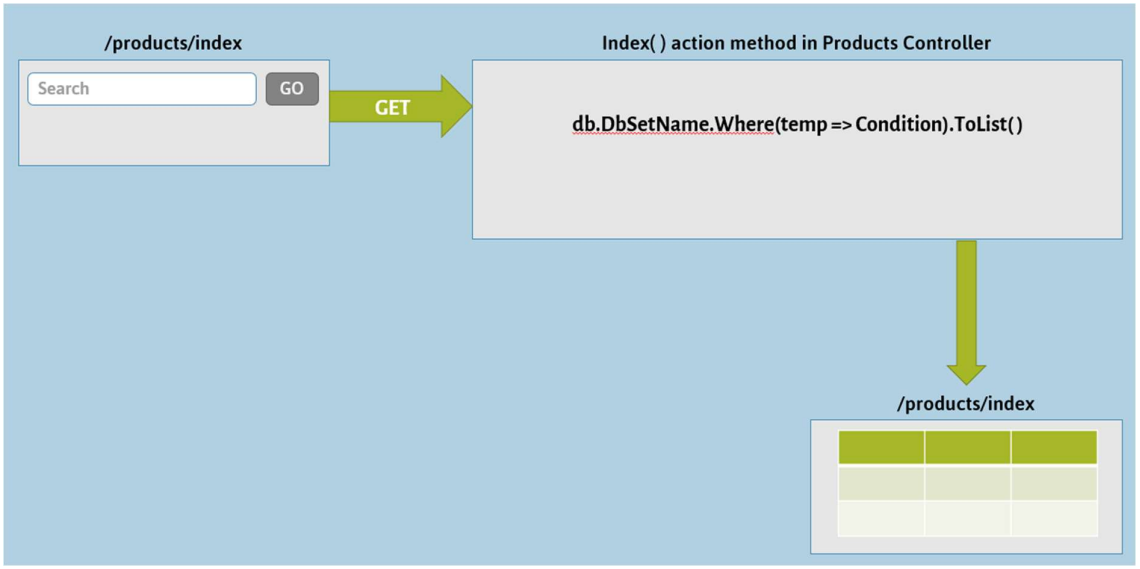
```
1. db.DbSetName.ToList( )
```

EF : Retrieving Multiple Rows Conditionally

Retrieving all rows and all columns from the database table

```
db.DbSetName.Where(temp => Condition).ToList( )
```

EF : Search

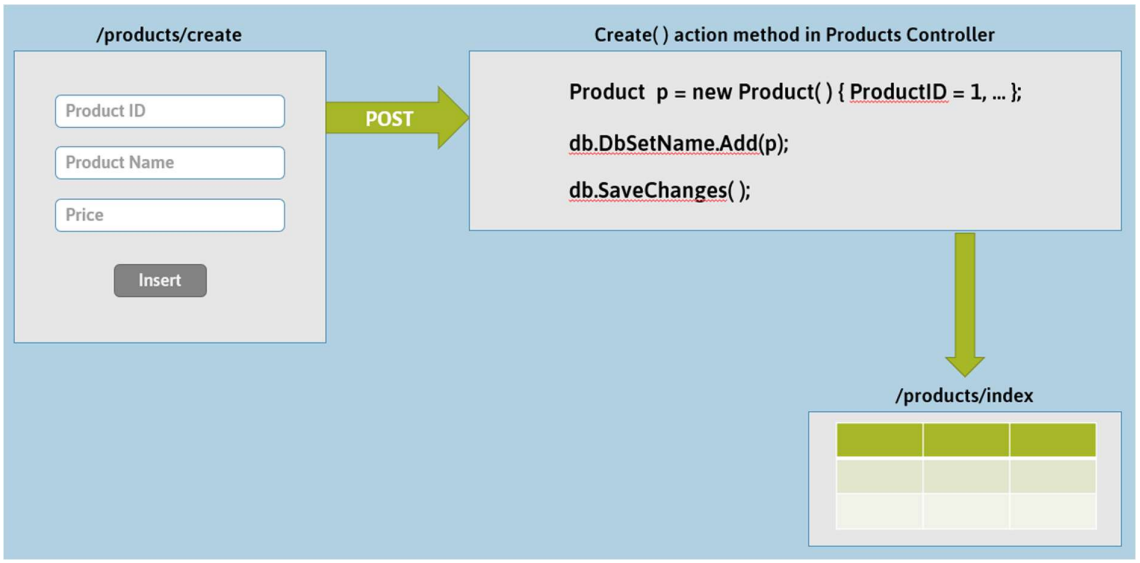


EF : Retrieving Single Row Conditionally

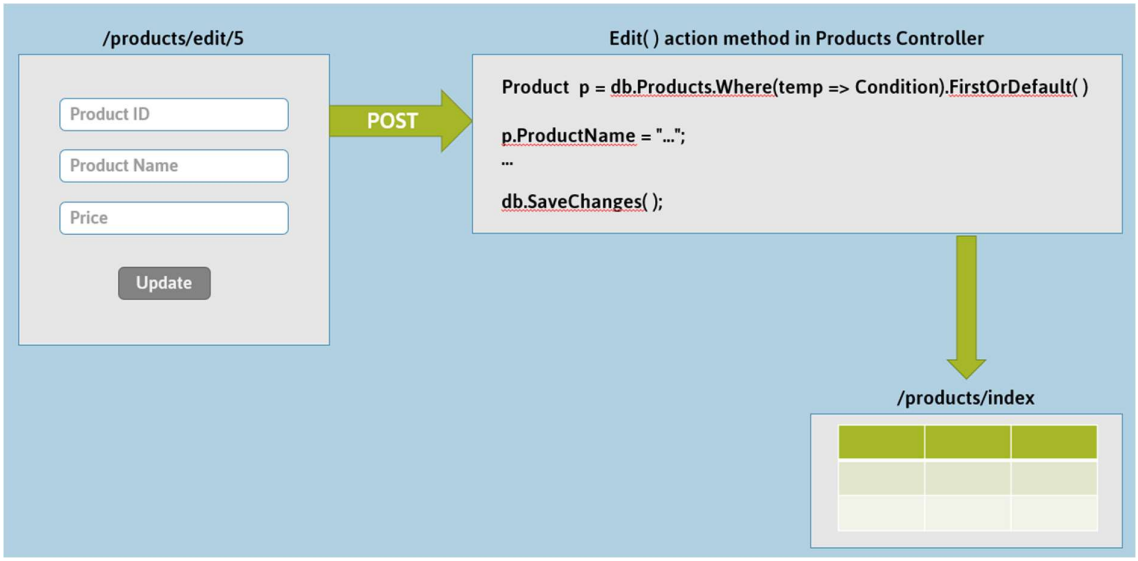
Retrieving single row and all columns from the database table

```
1. db.DbSetName.Where(temp => Condition).FirstOrDefault( )
```

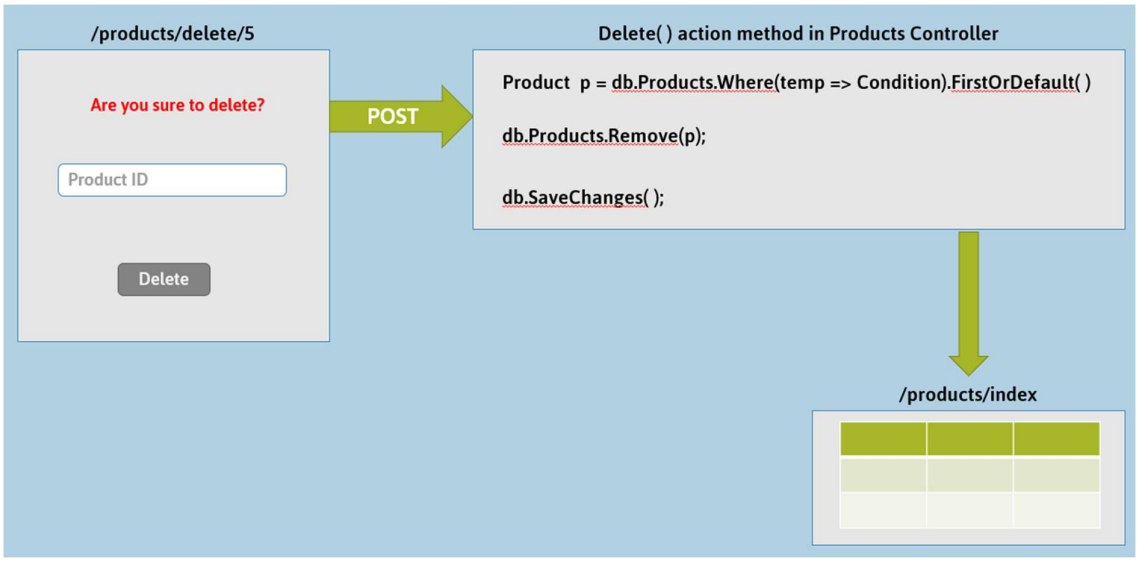
EF : Insert



EF : Update



EF : Delete



EF : Navigation Properties

Navigation Property is a complex-type property, with "virtual" keyword.

Navigation Property stores reference of parent model class's object, or vice versa.

Used to access parent table's data from child table's data, or vice versa.

Eg:

Model class of Child entity

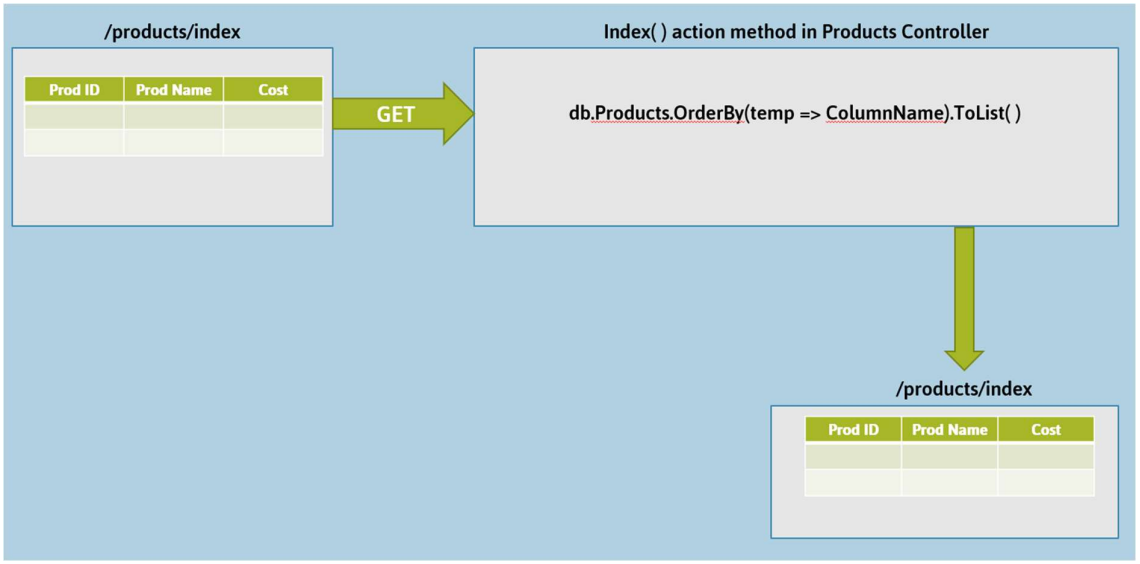
```
1. class Product
2. {
```

```
3.     public long ProductId { get; set; }
4.     public string ProductName { get; set; }
5.     public virtual Category Cat { get; set; }  //Refers to
   Category.CategoryName
6. }
```

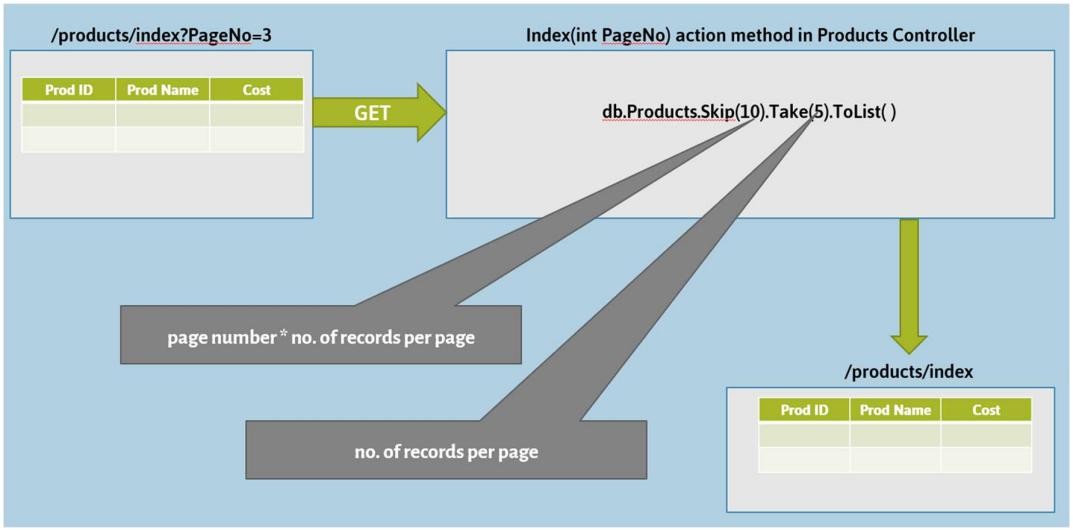
Model class of Parent entity

```
1. class Category
2. {
3.     public long CategoryId { get; set; }
4.     public string CategoryName { get; set; }
5. }
```

EF : Sort



EF : Paging



Calling Stored Procedures using EF

Note: This is for additional information for knowledge, as some students have asked this question, "how to invoke stored procedures in Asp .net MVC application using Entity Framework?".

Invoking Stored Procedures using Entity Framework

There are many ways to call stored procedures in EF.

One of the easier and better way, is provided here:

Step 1: Create procedure in SQL Server:

```
1. create procedure getProductsByBrandID
2. (@BrandID bigint)
3. as begin
4.   select * from Products where BrandID=@BrandID
5. end
```

Step 2: Invoke stored procedure directly using the object of DbContext:

```
1. SqlParameter[] sqlParameters = new SqlParameter[]
2.             {
3.                 new SqlParameter("@BrandID", 2)
4.                 //you can add more parameters here
5.             };
6.             List<Product> products = db.Database.SqlQuery<Product>("exec
   getProductsByBrandID @BrandID", sqlParameters).ToList();
```

EF : Model-First (vs) Database First (vs) Code First Approaches

1. Database First Approach

Developer has to create database first; model class will be auto-generated next.

2. Code First Approach

Developer has to create model class first, database will be auto-generated next.

3. Model First Approach

Developer has to design the model class first, database will be auto-generated next.

Database-First Approach

- Very popular, if you have DB designed by DBA's, developed separately by DB-Team or if you have existing DB and you want to migrate existing project into Asp.Net Mvc.
- Best for small-large projects, and the requirements are fully known.

- You will let EF create model classes automatically and you can make the necessary changes later.
- You can make changes either in ORM designer or use partial classes.
- Manual changes to database are possible because the database defines your model class. You can always update model classes from database.
- Best, if you want to use stored procedures.

Code-First Approach

- Very popular, because hard-core programmers like the grip in the model classes.
- Best for small-to-large projects, and the requirements are partially known (continuous changes in requirements).
- Full control over the code (model classes), because the model classes are not auto-generated code; so the developer can make any changes in the model classes, and easily map those classes to underlying database.
- General expectation is that you do not bother with DB. DB is just a storage with no logic. EF will handle creation and you don't want to know how it does the job.
- Manual changes to database will be most probably lost because your code defines the database.
- Not-good if you want to use Stored Procedures.

Model-First Approach

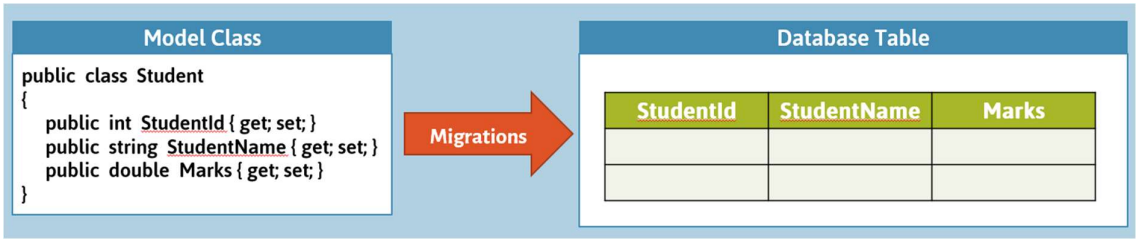
- Good only if you are designer fan (means, you don't like writing code or SQL).
- You will "draw" your model using VS ORM designer, thus both the database and model classes will be automatically generated in background.
- You will lose part of the control on both your model classes and database.
- Suitable for small projects and don't have strong DBA team.
- Manual changes to database will be most probably lost because your code defines the database.
- Not-much used in real-world applications.

Code-First Migrations

Use when you face problems with CreateDatabaseIfNotExists, DropCreateDatabaseIfModelChanges, and DropCreateDatabaseAlways.

Code-First Migrations are two types:

1. Automated Migration
2. Code-based Migration



Automated Migration in Entity Framework

Updates the model changes to database automatically.

Process flow:

1. Enable Migrations in Package Manager Console.
2. Set Database Initializer in DbContext.

Enable-Migrations Command FYI

```
Enable-Migrations [-ContextTypeName <String>]
[-EnableAutomaticMigrations] [-MigrationsDirectory <String>]
[-ProjectName <String>] [-StartUpProjectName <String>]
[-ContextProjectName <String>]
[-ConnectionStringName <String>] [-Force]
[-ContextAssemblyName <String>]
[-AppDomainBaseDirectory <String>] [<CommonParameters>]
```

Code-Based Code-First Migrations

Updates the model changes to database based on some automatic-generated classes.

Process flow:

1. Enable Migrations in Package Manager Console.
2. Add migration class.
3. Update-Database

Overriding the Conventions in Code-First Approach

Default Conventions in Code-First Approach

Model class + "s" or "es" --> Table name. Eg: `"Student" class --> "Students" table`

Schema name --> "dbo". Eg: `"Student" class --> "dbo.Students" table`

Property name --> Column name. Eg: `"StudentName" property --> "StudentName" column`

Property data type --> Column data type. Eg: `"string StudentName" property --> "StudentName varchar(max)" column`

Primary key --> "Id". Eg: `"Id" property à "Id" primary key column`

Overriding the Default Conventions in Code-First Approach

Model class + "s" or "es" --> Table name. Use: `[Table]`

Schema name --> "dbo". Use: `[Table]`

Property name --> Column name. Use: `[Column]`

Property data type --> Column data type. Use: `[Column]`

Primary key --> "Id" Use: `[Key]`

HTML Helpers

- 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.

Server code:

```
@Html.TextBoxFor (model => model.EmpName)
```

Browser (Client) - auto generated

```
1. <input type="text" name="EmpName">
```

List of HTML Helpers

1. **Html.ActionLink:** Hyperlink
2. **Html.TextBoxFor:** Textbox
3. **Html.TextAreaFor:** TextArea
4. **Html.CheckBoxFor:** CheckBox
5. **Html.RadioButtonFor:** RadioButton
6. **Html.DropDownListFor:** Dropdown
7. **Html.ListBoxFor:** Multi-Select ListBox
8. **Html.HiddenFor:** Hidden Field
9. **Html.PasswordFor:** Password TextBo
10. **Html.DisplayFor:** Plain text
11. **Html.LabelFor:** Label
12. **Html.EditorFor:** TextBox / TextArea / Numeric TextBox / Date TextBox etc.

Custom HTML Helpers

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.

Custom HTML Helper

```
1. public static class Classname
2. {
3.    public static MvcHtmlString Methodname(this HtmlHelper helper,
   arguments)
4.    {
5.       code here
6.       return new MvcHtmlString("html tags");
7.    }
8. }
```

View

```
@Html.Methodname( arguments )
```

Validations

Defined by using Data Annotations.

Data annotations are provided by System.ComponentModel.DataAnnotations namespace.

```
1. public class Employee
2. {
3.    [Required]
4.    public int EmpID { get; set; }
5. }
```

Data Annotations for Validations

1. **[Required]** Field is mandatory.
2. **[MaxLength]** Maximum no. of characters.
3. **[MinLength]** Minimum no. of characters.
4. **[Range]** Value should be within the min and max.
5. **[Compare]** Two fields must be same.
6. **[RegularExpression]** Pattern of value.
7. **[EmailAddress]** Email address only accepted.

Server Side Validation

ModelState.IsValid: Checks whether the model object satisfied all the validation rules

Client Side Validation

Three NuGet packages:

1. **jQuery :** Performs DOM manipulations
2. **jQuery.Validation :** Defines Validation Rules

3. **Microsoft.jQuery.Unobtrusive.Validation:** Shows / Hides Error Message

HTML Helpers for Client Side Validation

1. **ValidationMessageFor :** Displays error message
2. **ValidationSummary:** Displays validation summary

Custom Validation

- Used to implement user-defined validations.
- Create a class that extends ValidationAttribute 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.
- IsValid method returns either "Success" or "ValidationResult with error message".
- Best for database related validations.

```
1. public class SampleAttribute : ValidationAttribute
2. {
3.    protected override ValidationResult IsValid(object value,
   ValidationContext validationContext)
4.    {
5.       //your code here
6.       return ValidationResult.Success;
7.
8.       //or
9.       return new ValidationResult(this.ErrorMessage);
10.   }
11. }
```

Asp.Net Identity

"Asp.Net Identity" is a framework to store and manage user accounts in web applications.

It has four major components / packages:

1) Microsoft.AspNet.Identity.Core

Managing Users and Roles

2) Microsoft.AspNet.Identity.EntityFramework

Storing users information in database

3) Microsoft.AspNet.Identity.Owin

User Login and Logout & Social Login

4) Microsoft.Owin.Host.SystemWeb

Used to run OWIN based apps on IIS

OWIN- Open web interface. Used to mediate b/w framework components and web server

Installing Packages for Asp.Net Identity

```
Install-package Microsoft.AspNet.Identity.EntityFramework
Install-package Microsoft.AspNet.Identity.Owin
Install-package Microsoft.Owin.Host.SystemWeb
```

Connection String for Asp.Net Identity

```
<connectionStrings>
  <add name="DefaultConnection" connectionString="data source=servername;
user id=useridhere; password=passwordhere; initial
catalog=databasenamehere"
providerName="System.Data.SqlClient" />
</connectionStrings>
```

IdentityUser

The Microsoft.AspNet.Identity.EntityFramework.IdentityUser class represents the user details, which should be stored in the database.

Implements IUser<TKey> interface.

Requires an ID and Name field for every user.

```
public class ApplicationUser : IdentityUser
{
   public dataType YourProperty { get; set; }
   //You can add properties here
}
```

IdentityDbContext

The Microsoft.AspNet.Identity.EntityFramework.IdentityDbContext class is the DbContext for Identity, which is responsible for interaction between identity entity model and the database.

Implements IdentityDbContext<TUser> interface.

```
public class ApplicationDbContext : IdentityDbContext<ApplicationUser>
{
   public ApplicationDbContext( ) : base("DefaultConnection")
   {
   }
```

```
6.
7.    //You can add DbSet's here
8.
9. }
```

UserStore

The Microsoft.AspNet.Identity.EntityFramework.UserStore class DbContext for Identity, which is responsible for storing the identity data in the database, using IdentityDbContext.

Implements IUserStore<TUser, TKey> interface.

```
1. public class ApplicationUserStore : UserStore<ApplicationUser>
2. {
3.    public ApplicationUserStore(ApplicationDbContext dbContext) :
   base(dbContext)
4.    {
5.    }
6.
7.    //You can add your methods here
8.
9. }
```

UserManager

The Microsoft.AspNet.Identity.UserManager class DbContext for Identity, which is responsible for manipulating the identity data, based on the UserStore.

Implements IUserManager<TUser, TKey> interface.

```
1. public class ApplicationUserManager : UserManager<ApplicationUser>
2. {
3.    public ApplicationUserManager(IUserStore<ApplicationUser> store) :
   base(store)
4.    {
5.    }
6.
7.    //You can add your methods here
8.
9. }
```

View Models

ViewModel is a class that defines structure of the data that you want to display or read in the view.

View binds to ViewModel

Doesn't contain business logic.

Controller passes ViewModel objects to te view.

Always use ViewModels in Strongly Typed Views.

View (associated to View Model)

```
@model ModelClassNameHere

@Model.propertyName
```

View Model

```
class ModelClassName
{
}
```

Areas

Area represents part of the project. Eg: Admin

Area contains its own set of controllers, models and views.

Area Registration

```
using System.Web.Mvc;

public class AreanameAreaRegistration : AreaRegistration
{
    public override string AreaName
    {
        get
        {
            return "name here";
        }
    }
}x
```

Folder Structure

ProjectFolder

- Areas
- Areaname
- Controllers
- Models
- Views

Filters

Filters Execute at a specific situation, while executing an action method.

Authentication Filters

- Executes first, before any other filter, to check whether the user is a valid user or not.
- Implemented using IAuthenticationFilter

Authorization Filters

- Executes first, before any other filter, to check whether the user has permission to execute the action method or not.
- Implemented using IAuthorizationFilter

Action Filters

- Executes before and after of action method execution. You can modify the ViewBag, before sending it result.
- Implemented using IActionFilter

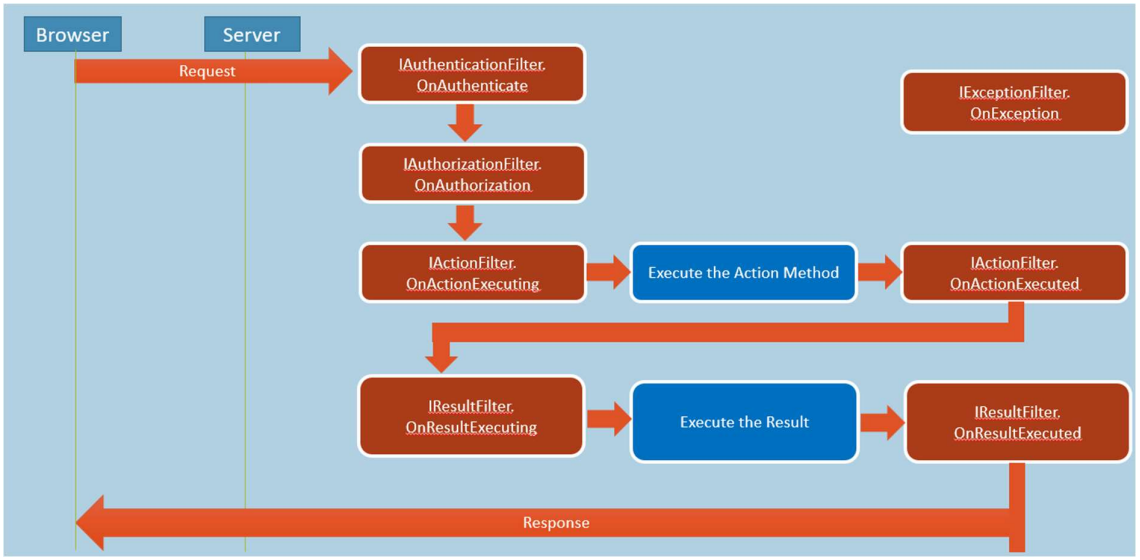
Result Filters

- Executes before and after of result execution. You can modify the result.
- Implemented using IResultFilter

Exception Filters

- Executes when an exception is raised while executing the action method or result. You can store exception log.
- Implemented using IExceptionFilter

How Filters Execute?



Authentication Filter

IAuthenticationFilter

Used to check whether the current working user is logged-in or not.

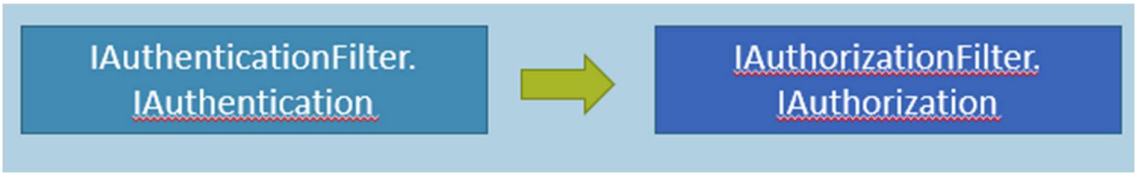
```
1. interface System.Web.Mvc.Filters.IAuthenticationFilter
2.     OnAuthentication(AuthenticationContext authenticationContext)
```

Authorization Filter

IAuthorizationFilter

Used to check whether the current working user has permission to access the action method, or not.

```
1. interface System.Web.Mvc.IAuthorizationFilter
2.   OnAuthorization(AuthorizationContext authorizationContext)
```

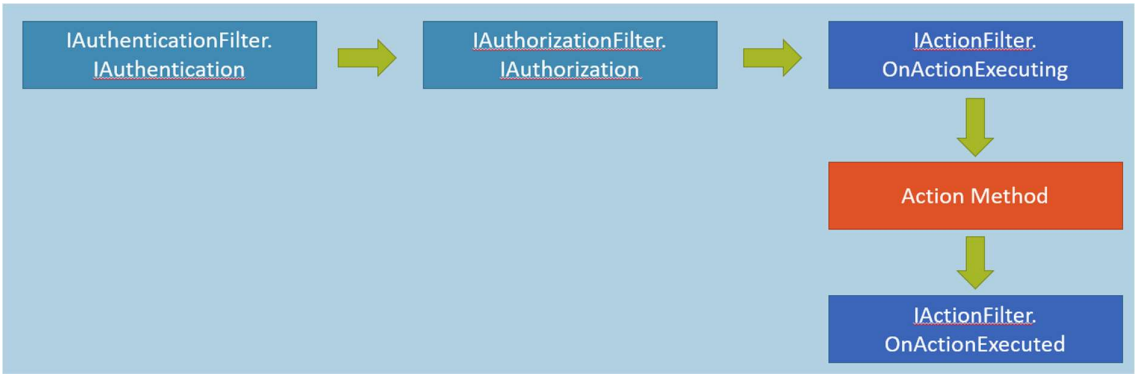


Action Filter

IActionFilter

Executes before and after of action method execution. You can modify the ViewBag, before sending it result.

```
1. interface System.Web.Mvc.IActionFilter
2.     OnActionExecuting(ActionExecutingContext actionExecutingContext)
3.     OnActionExecuted(ActionExecutedContext actionExecutedContext)
```

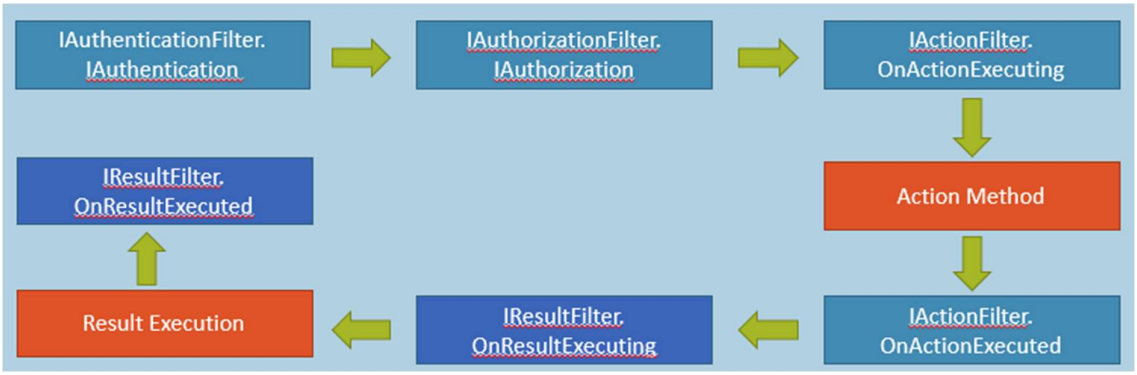


Result Filter

IResultFilter

Executes before and after of result execution. You can modify the result.

```
1. interface System.Web.Mvc.IResultFilter
2.     OnResultExecuting(ResultExecutingContext resultExecutingContext)
3.     OnResultExecuted(ResultExecutedContext resultExecutedContext)
```

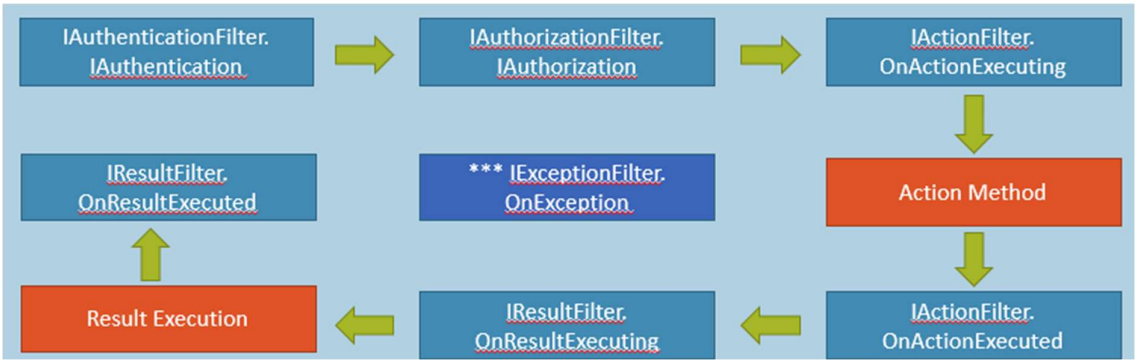


Exception Filter

IExceptionFilter

Executes when an exception is raised while executing the action method or result. You can store exception log.

```
1. interface System.Web.Mvc.IExceptionFilter
2.     OnException(ExceptionContext exceptionContext)
```



Global Filters

Global Filters gets applied to all controllers by default.

Global Filters are registered in FilterConfig.cs at App_Start folder.

The FilterConfig class has a method called RegisterGlobalFilters, which receives an argument of type GlobalFilterCollection, in which you can add any no. of built-in or custom filters.

The RegisterGlobalFilters method of FilterConfig.cs is invoked in Application_Start( ) method of Global.asax file, which gets called on the first request of the application.

FilterConfig.cs

```
1. public class FilterConfig
2. {
3.    public static void RegisterGlobalFilters(GlobalFilterCollection filters)
4.    {
5.       filters.Add(new YourFilterClassNameHere());
6.    }
7. }
```

Calling RegisterGlobalFilters method

Calling RegisterGlobalFilters method in Global.asax

```
1. public class MvcApplication : System.Web.HttpApplication
2. {
3.    protected void Application_Start()
4.    {
5.       FilterConfig.RegisterGlobalFilters(GlobalFilters.Filters);
6.    }
7. }
```

Filter Overrides

Filter Overrides are used to exclude a specific action method or controller from the global filter or controller level filter.

Types of Filter Overrides:

1. [OverrideAuthentication]
2. [OverrideAuthorization]
3. [OverrideActionFilters]
4. [OverrideResult]
5. [OverrideException]

SampleController.cs

```
1. public class SampleController : Controller
2. {
3.    [OverrideAuthentication]
4.    public void ActionMethodName()
5.    {
6.       …
7.    }
8. }
```

Built-in Filters

Authorization Filters

1. **[ChildActionOnly]** : Accepts only child requests (not direct requests).
2. **[ValidateAntiForgeryToken]** : Accepts only the requests from same domain. Useful for CSRF security
3. **[Authorize]** : Accepts only if the user has logged-in.

Action Filters

1. **[ActionName]** : Defines name of action, which is different than method name.
2. **[NonAction]** : Defines that the method is not an action method.
3. **[HttpGet]** : Accepts only GET request.
4. **[HttpPost]** : Accepts only POST request.
5. **[OutputCache]** : Caches result of action method & delivers the same for subsequent requests.

Result Filters

(No filters)

Exception Filters

1. [HandleError] : Redirects to custom error page, when exception is raised.

Output Cache

Purpose: Improving page loading time of a view.

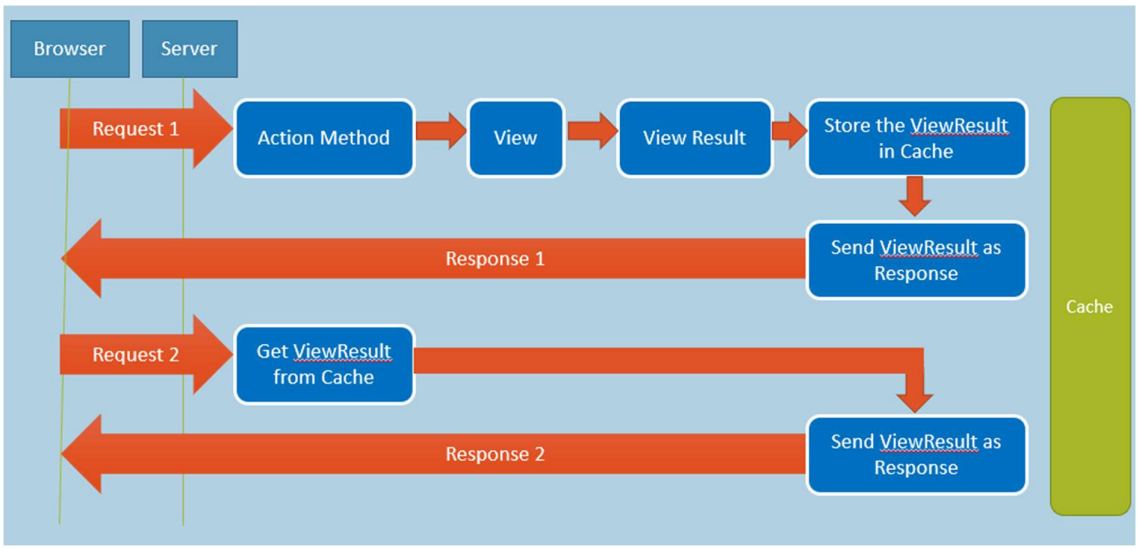
Caches (stores) result of the action method in cache memory and delivers the same for all subsequent requests of any client.

Things to keep in mind:

- Avoid caching contents that are unique per user.
- Avoid caching contents that are accessed rarely.
- Use caching for contents that are accessed frequently.

```
1. [OutputCache(Duration = seconds)]
2. public ActionResult Methodname( )
3. {
4. }
```

Execution flow of OutputCache



ActionName

ActionName attribute allows us to specify a different action name than the method name.

```
1. [ActionName("action name here")]
2. public ActionResult Methodname( )
3. {
4. }
```

NonAction

Indicates that a public method of a Controller is not an action method.

```
1. [NonAction]
2. public ActionResult Methodname( )
3. {
4. }
```

ChildActionOnly

Child Actions can only be invoked from within a view, you cannot invoke a child action method via user request (URL).

Child Action Method

```
1. [ChildActionOnly]
2. public ActionResult Methodname( )
3. {
4. }
```

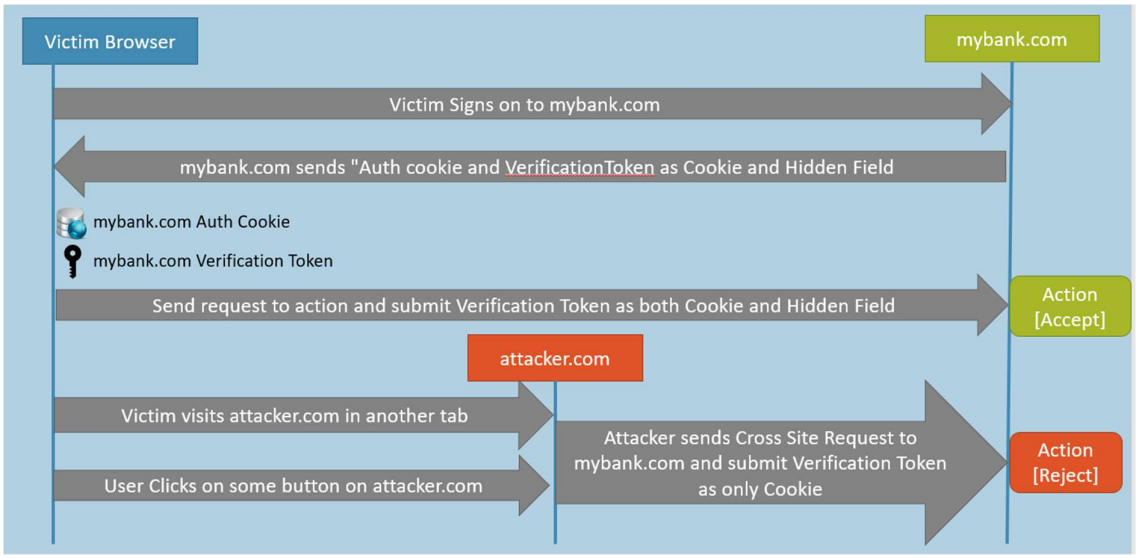
Invoking Child Action Method in the View

```
@Html.RenderAction("action name", "controller name")
```

CSRF

What is CSRF?

CSRF / XSRF (Cross Site Request Forgery) is a process of sending a request to actual website, from a page present in attacker website, based on the logged-in session of actual website user.



How to Implement CSRF Security?

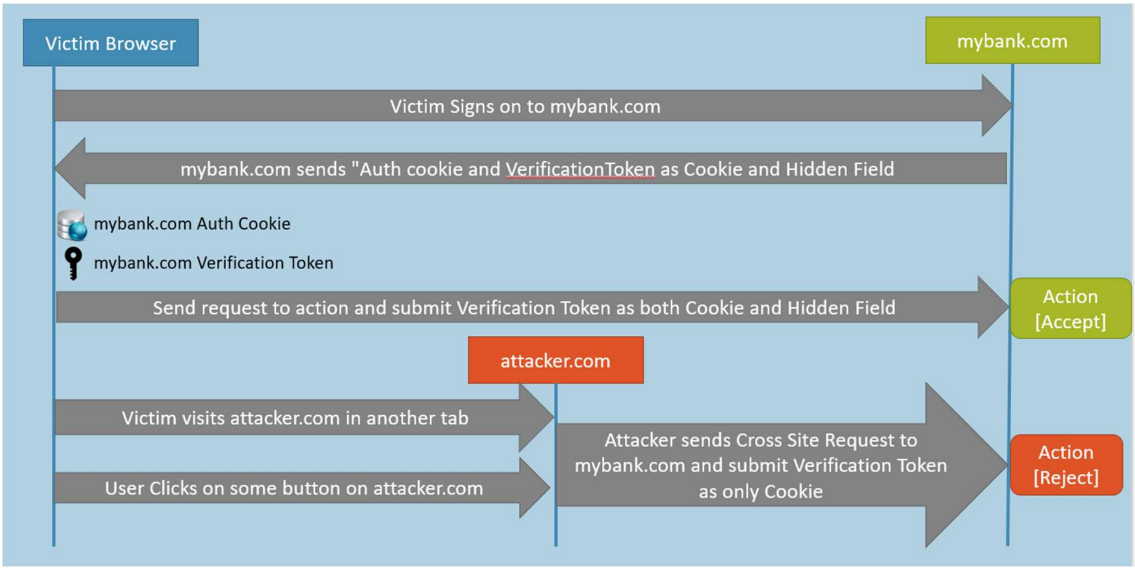
View

```
@Html.AntiForgeryToken( )
```

Controller

```
1. [ValidateAntiForgeryToken]
2. public ActionResult Method( )
3. {
4. }
```

How CSRF Security Works?



Exception Handling

Used to display custom error page, instead of default exception page, when an exception is raised.

Types of Exception Handling:

1. IExceptionFilter

2. HandleError

3. Http Errors

4. Application_Error

HandleError

HandleError is a pre-defined filter, which implements "IExceptionFilter".

Advantage: Ready-to-use

Disadvantage: Doesn't support to add custom logic to generate logging of exceptions.

Can detect errors of action methods only.

FilterConfig.cs

```
1. public class FilterConfig
2. {
3.     public static void RegisterGlobalFilters(GlobalFilterCollection filters)
4.     {
```

```
        filters.Add(new HandleErrorAttribute());
    }
}
```

Http Errors

We can handle HTTP Error Status Codes such as 404, 500 etc.

Web.config

```
<customErrors mode="RemoteOnly">
    <error statusCode="404" redirect="/controller/action" />
    <error statusCode="401" redirect="/controller/action" />
    <error statusCode="500" redirect="/controller/action" />
    …
</customErrors>
```

Application_Error

Global Error handler.

Used to handle other exceptions that raises beyond the action methods & views.

Global.asax

```
protected void Application_Error( )
{
    Exception exc = Server.GetLastError();

    //store exception log
    Response.Redirect("~/Controller/action");
}
```

Service Pattern

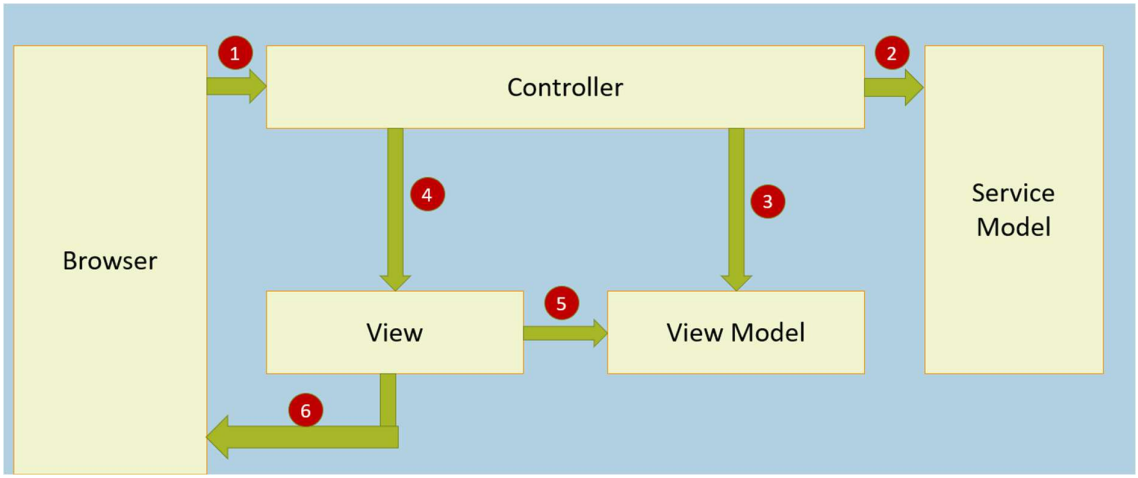
Service Pattern is a concept of writing Service classes that contain business logic.

Goals:

- Separation of business logic from controllers.
- Make the code of controllers clean and clear.

MVC Architecture with Service Pattern

Controller invokes the Service.



Creating a Service

For every table, it is recommended to create a service class.

Eg: Customers table --> Customers Service

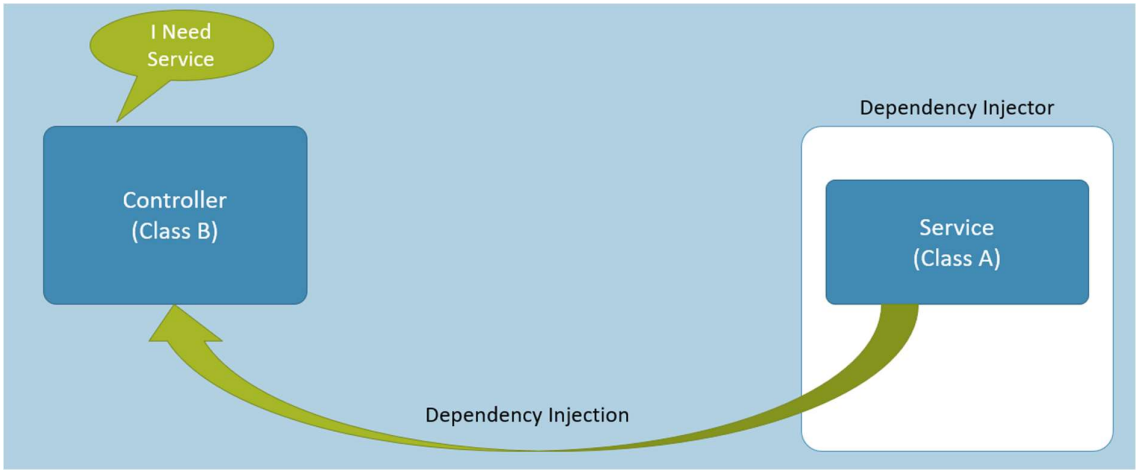
Methods:

- GetCustomers( )
- GetCustomerById( )
- InsertCustomer( )
- UpdateCustomer( )
- DeleteCustomer( )

Dependency Injection

Dependency Injection is a process of creating object of repository / service class at run time, and passing its references to the service class / controller, through it's constructor.

Advantage: The controller can change its repository; so we can perform unit testing on controller.

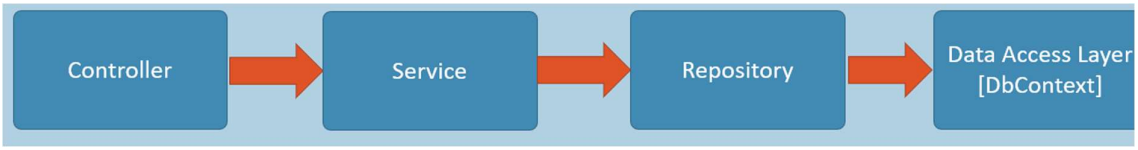


Repository Pattern

Repository sits between Controller and Data Access Layer.

Repository contains methods for CRUD operations.

Repository makes controller directly depend on Data Access Layer.



IRepository

IRepository interface defines the list of methods that needs to be present in repository class

Eg:

Employee Repository Interface

```
1. public interface IRepository
2. {
3.     List<Employee> GetAll();
4.     Employee GetById(int id);
5.     void Insert(Employee entity);
6.     void Update(Employee entity);
7.     void Delete(Employee entity);
8. }
```

ViewData

ViewData is used to transfer the data from controller to view.

Controller sets data into ViewData and View gets data from ViewData.

ViewData's lifetime is "per request". That means ViewData object will be re-initialized for each request.

ViewData internally uses Dictionary. Uses key value and that key shouldn’t be duplicate simply it’s a string.

ViewData supports to send any primitive values, objects and collections also.

View Data Usage in Controller

```
ViewData["key"]
```

ViewBag

ViewBag is used to transfer any type of data from controller to View.

ViewBag is "dynamic" based. That means no need of declaring members of ViewBag.

The "ViewBag.key" syntax returns "null" if the key is not found.

ViewBag's lifetime is "per request".

Differences Between Viewdata and Viewbag

1.While transferring the complex data using ViewData we have to manually type caste it and also need to check for null values all these are neglected in Viewbag becoz its dynamic type

Controller

```
ViewBag.key = value;
```

View

```
@ViewBag.key
```

TempData

Used to transfer the data from one action method to another action method, in case of redirection.

Accessible from among those all action methods, if action methods redirects from one to other.

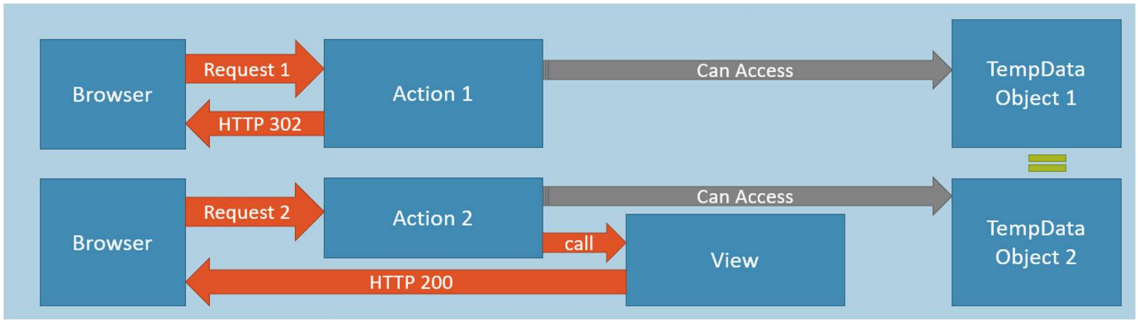
Item will be deleted automatically, at the end of request, when we retrieve the item from TempData.

It is Dictionary-based. . Uses key value and that key shouldn't be duplicate simply it's a string.

It internally uses Session.

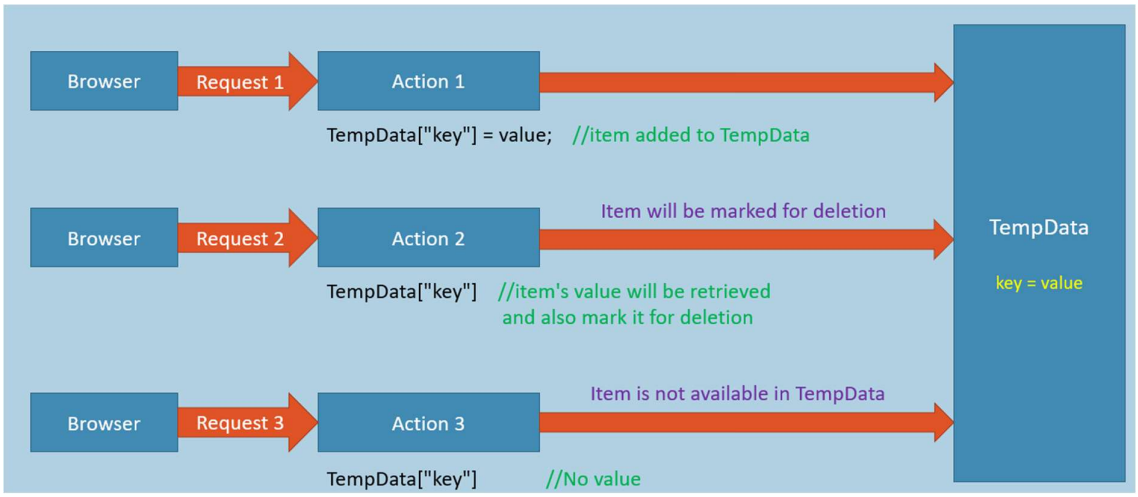
TempData value must be type casted before its usage like viwedata i.e., to a string or class type, and also need to check for null values

Syntax: `TempData["key"]`



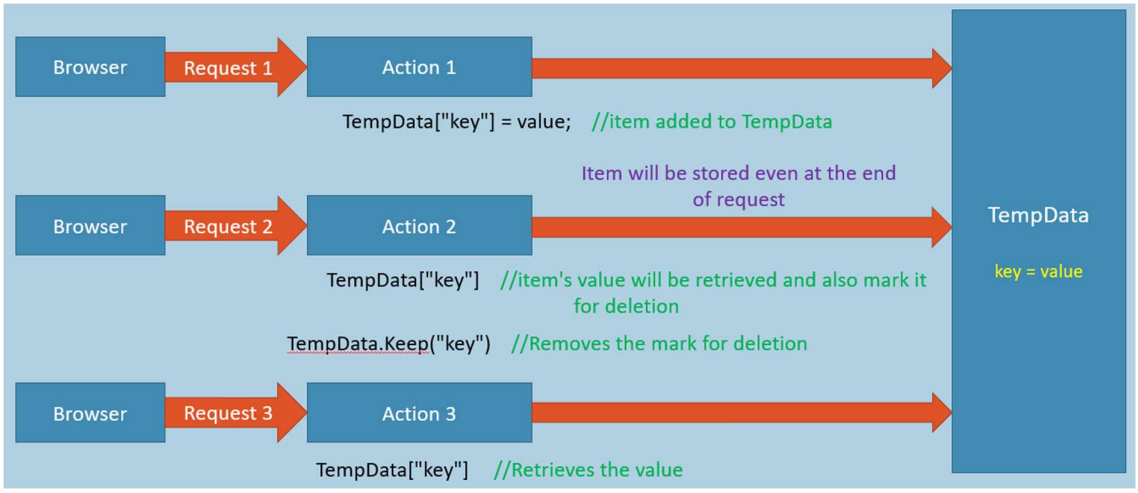
TempData["Key"]

TempData item will be deleted marked for deletion automatically, when we retrieve the value from TempData and will be deleted at the end of the request.



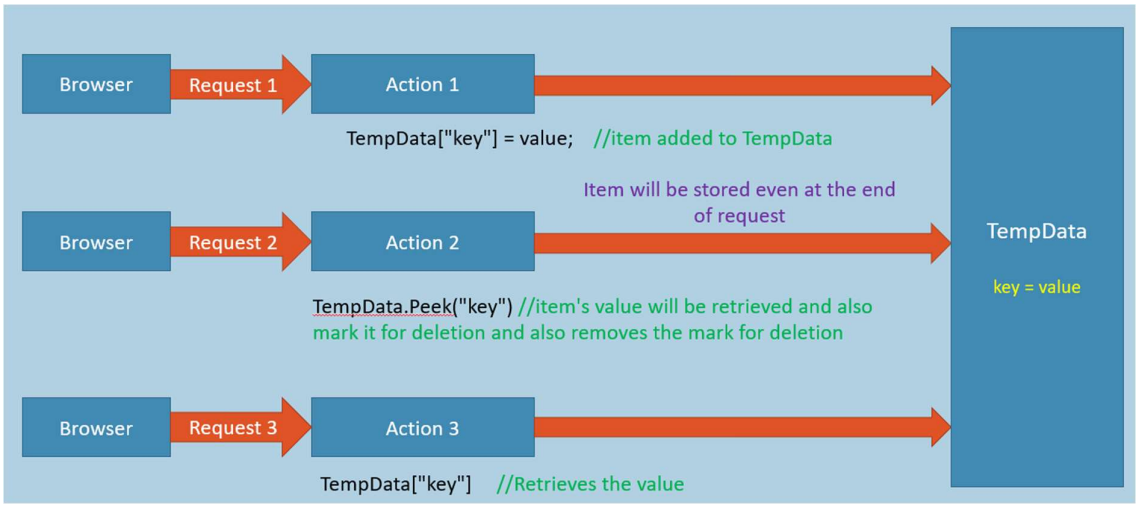
TempData["Key"] and then TempData.Keep("key")

The TempData["key"] statement marks the item for deletion; and then the Keep( ) method removes the mark for deletion of item, so that the item will not be deleted at the end of the request. You can access the value in the next request also.



TempData.Peek("key")

The Peek( ) method retrieves the value of item but not mark it for deletion; so that the item will not be deleted at the end of the request.

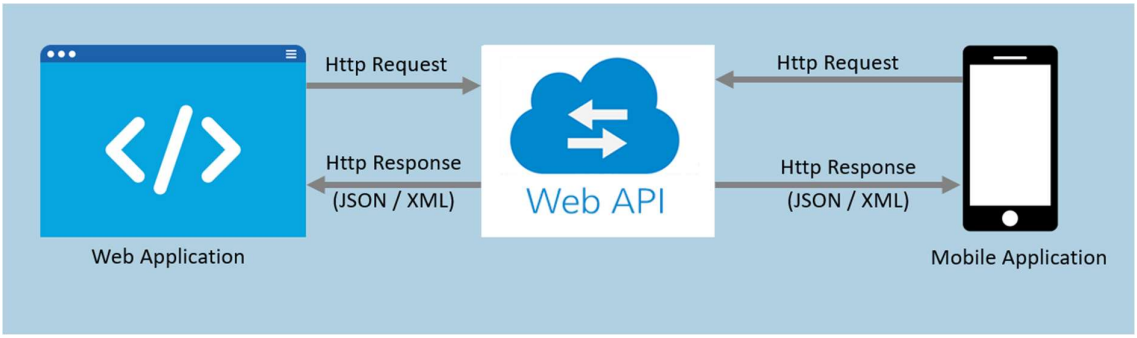


Web API

"Web Api" Framework is used to create HTTP services, that can be called from any application, using HTTP protocol.

Web Api Service / Http Service executes when the browser sends request using AJAX. It performs CRUD operations on database table.

MVC response will be on Html format but webApi response will be in JSON FORMAT .



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

1. **Get :** Used to retrieve / search data from server
2. **Post :** Used to insert data to server
3. **Put :** Used to update data on server
4. **Delete :** Used to delete data from server

Json basically it's a string contains collection of key value pairs

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;
        }
    }
}
```

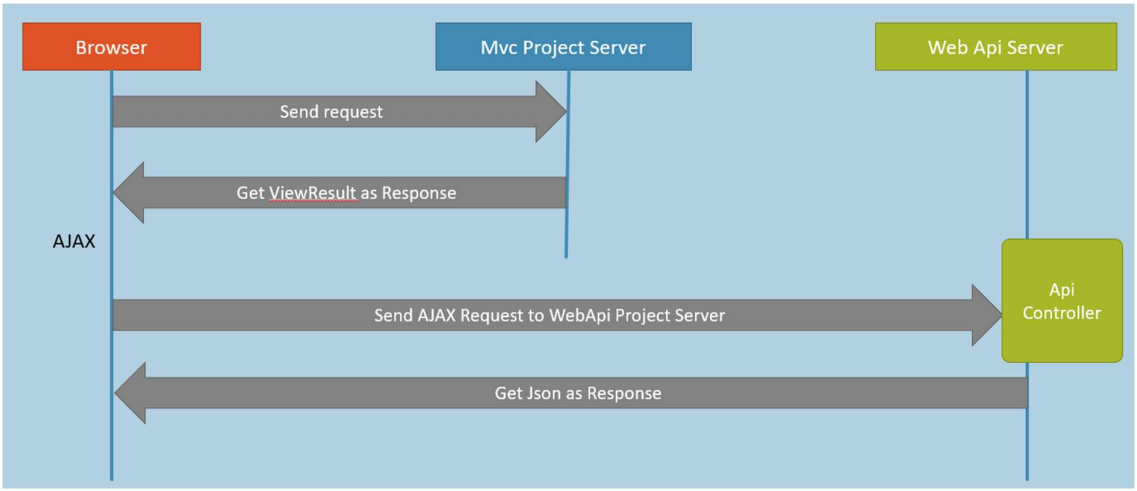
WebApiConfig.cs

It provides routing for Web API Controllers / Services.

```
1. using System;
2. using System.Web.Http;
3.
4. namespace namespacename
5. {
6.    public static class WebApiConfig
7.    {
8.       public static void Register(HttpConfiguration config)
9.       {
10.         config.Routes.MapHttpRoute(
11.            name: "DefaultApi",
12.            routeTemplate: "api/{controller}/{id}",
13.            defaults: new { id = RouteParameter.Optional }
14.         );
15.      }
16.   }
17. }
```

CORS

CORS (Cross-Origin-Resource-Sharing) is a concept of creating Web API Service in one server and invoking it from another server.



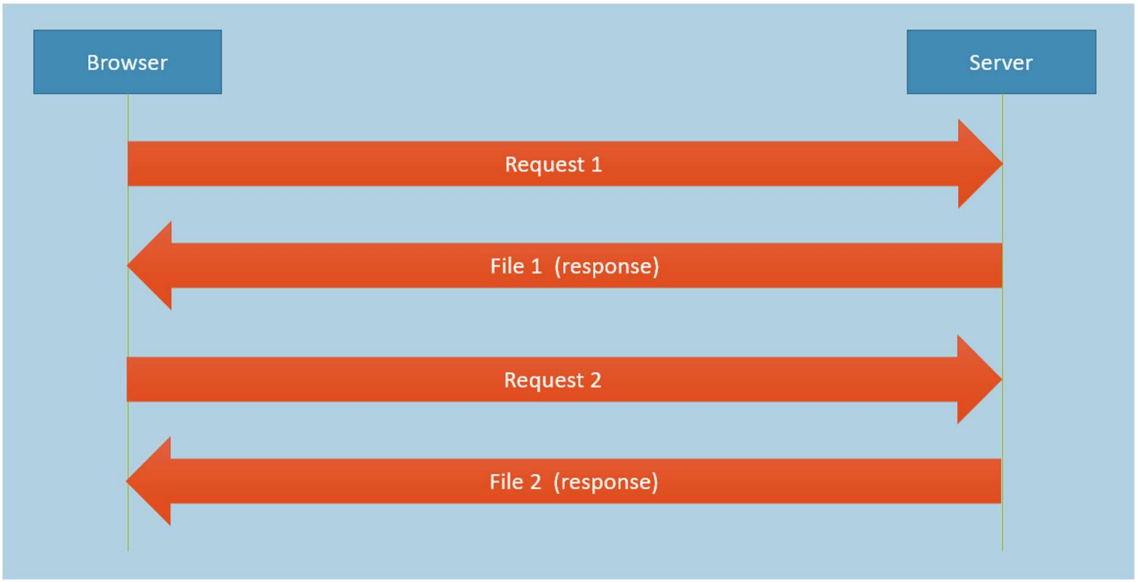
How to Implement CORS?

```
install-package Microsoft.AspNet.WebApi.Cors
```

Bundling and Minification

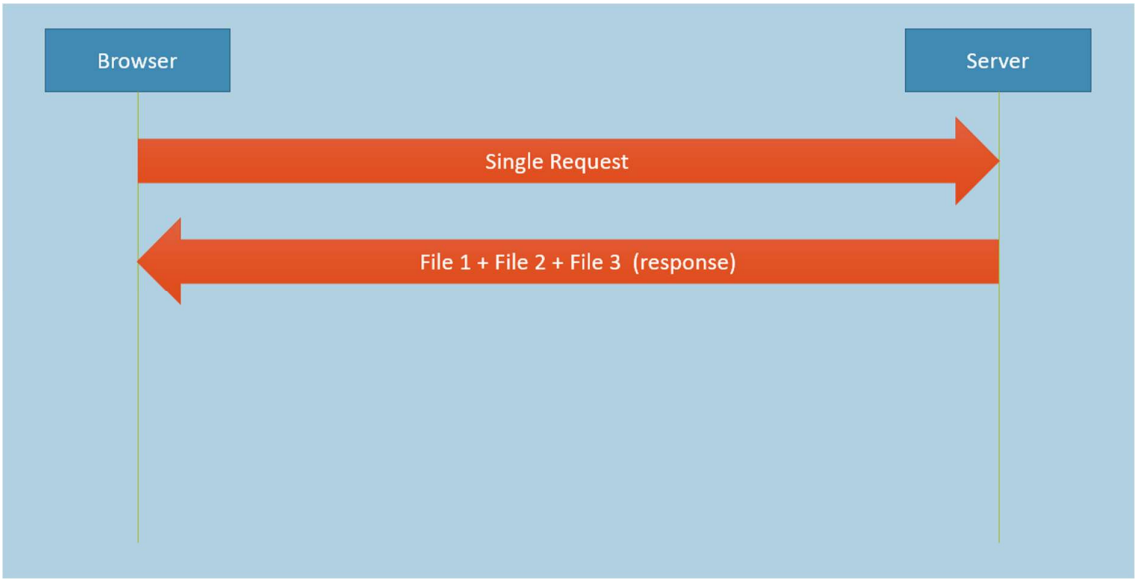
1. Traditional Web Loading

One request will get only one file.



2. Loading with Bundling

One request will get multiple files.



Bundling and Minification

Used to load multiple css files / js files with a single request.

Also minifies (compresses) the css / js files.

Types of Bundles:

- **ScriptBundle:** Contains javascript files (.js files).
- **StyleBundle:** Contains css fles (.css files).

Microsoft.AspNet.Web.Optimization

This NuGet Package needs to be installed in the project, to use bundling concept.