ASP.NET MVC

MVC

- **.NET Framework** – A technology introduced in 2002 which includes the ability to create executables, web applications, and services using C# (pronounced see-sharp), Visual Basic, and F#.
- **ASP.NET** – An open-source server-side web application framework which is a subset of Microsoft's .NET framework. Their first iteration of ASP.NET included a technology called Web Forms.
- **ASP.NET WebForms** – (2002 – current) A proprietary technique developed by Microsoft to manage state and form data across multiple pages.

MVC

- ASP.NET MVC is Microsoft's framework for developing fast web applications using their .NET platform with either the C# or VB.NET language.
- MVC is an acronym that stands for:
  - **(M)**odel – Objects that hold your data.
  - **(V)**iew – Views that are presented to your users, usually HTML pages or pieces of HTML.
  - **(C)**ontroller – Controllers are what orchestrates the retrieving and saving of data, populating models, and receiving data from the users.

MVC

- An alternative to ASP .NET Web Forms
- Presentation framework
  - Lightweight
  - Highly testable
- Integrated with the existing ASP .NET features:
  - Master pages
  - Membership-Based Authentication

ASP .NET MVC Framework Components

- Models
  - Business/domain logic
  - Model objects, retrieve and store model state in a persistent storage (database).
- Views
  - Display application's UI
  - UI created from the model data
- Controllers
  - Handle user input and interaction
  - Work with model
  - Select a view for rendering UI

Advantages of MVC

- Advantages:
  - Easier to manage complexity (divide and conquer)
  - It does not use server forms and view state
  - Front Controller pattern (rich routing)
  - Better support for test-driven development
  - Ideal for distributed and large teams
  - High degree of control over the application behavior

Advantages of MVC

ASP.NET MVC has a separation of concerns. Separation of concerns means that your business logic is not contained in a View or controller. The business logic should be found in the models of your application. This makes web development even easier because it allows you to focus on integrating your business rules into reusable models.

Advantages of MVC

ASP.NET MVC provides testability out of the box. Another selling point is that ASP.NET MVC allows you to test every single one of your components, thereby making your code almost bulletproof. The more unit tests you provide for your application, the more durable your application will become.

Advantages of MVC

ASP.NET MVC has a smaller "View" footprint. With WebForms, there is a server variable called ViewState that tracks all of the controls on a page. If you have a ton of controls on your WebForm, the ViewState can grow to become an issue. ASP.NET MVC doesn't have a ViewState, thus making the View lean and mean.

Advantages of MVC

ASP.NET MVC has more control over HTML.
Since server-side controls aren't used, the View can be as small as you want it to be. It provides a better granularity and control over how you want your pages rendered in the browser.

ASP .NET MVC Features

- Separation of application tasks
  - Input logic, business logic, UI logic
- Support for test-driven development
  - Unit testing
  - No need to start app server
- Extensible and pluggable framework
  - Components easily replaceable or customized(view engine, URL routing, data serialization,…)

ASP .NET MVC App Structure

- URLs mapped to controller classes
- Controller
  - handles requests,
  - executes appropriate logic and
  - calls a View to generate HTML response
- URL routing
  - ASP .NET routing engine (flexible mapping)
  - Support for defining customized routing rules
  - Automatic passing/parsing of parameters

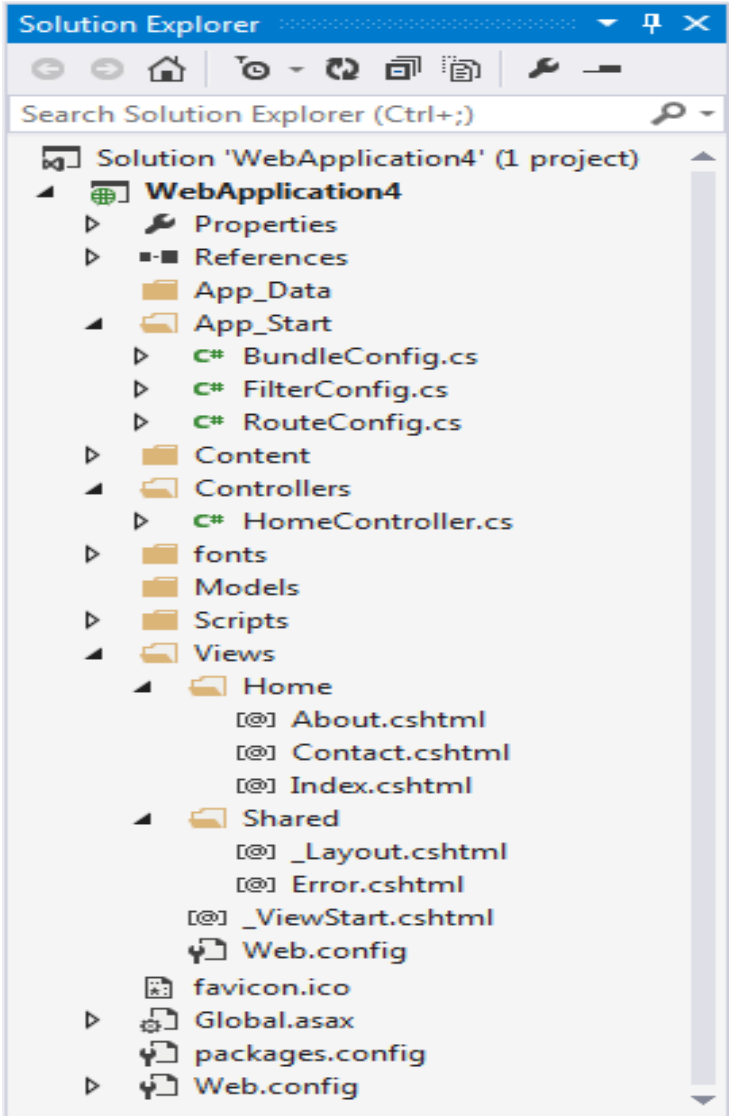
ASP .NET MVC App Structure

- No Postbackinteraction!
- All user interactions routed to a controller
- No view state and page lifecycle events

Layout of an MVC project

When you create a new MVC project, your solution should have the following structure in your Solution Explorer.

Layout of an MVC project



Layout of an MVC project

- **App_Data** – While I don't use this folder often, it's meant to hold data for your application (just as the name says). A couple of examples would include a portable database (like SQL Server Compact Edition) or any kind of data files (XML, JSON, etc.). I prefer to use SQL Server.
- **App_Start** – The App_Start folder contains the initialization and configuration of different features of your application.
  - BundleConfig.cs – This contains all of the configuration for minifying and compressing your JavaScript and CSS files into one file.
  - FilterConfig.cs – Registers Global Filters.
  - RouteConfig.cs – Configuration of your routes.

  There are other xxxxConfig.cs files that are added when you apply other MVC-related technologies (for example, WebAPI adds WebApiConfig.cs).
- **Content** – This folder is meant for all of your static content like images and style sheets. It's best to create folders for them like "images" or "styles" or "css".

Layout of an MVC project

- **Controllers** – The controllers folder is where we place the controllers.
- **Models** – This folder contains your business models. It's better when you have these models in another project, but for demo purposes, we'll place them in here.
- **Scripts** – This is where your JavaScript scripts reside.
- **Views** – This parent folder contains all of your HTML "Views" with each controller name as a folder. Each folder will contain a number of cshtml files relating to the methods in that folder's controller. If we had a URL that looked like this:

  http://www.xyzcompany.com/Products/List

  we would have a Products folder with a List.cshtml file. We would also know to look in the Controllers folder and open the ProductsController.cs and look for the List method.

Layout of an MVC project

- **Views/Shared** – The Shared folder is meant for any shared cshtml files you need across the website.
- **Global.asax** – The Global.asax is meant for the initialization of the web application. If you look inside the Global.asax, you'll notice that this is where the RouteConfig.cs, BundleConfig.cs, and FilterConfig.cs are called when the application runs.
- **Web.Config** – The web.config is where you place configuration settings for your application. For new MVC developers, it's good to know that you can place settings inside the <appsettings> tag and place connection strings inside the <connectionstring> tag.

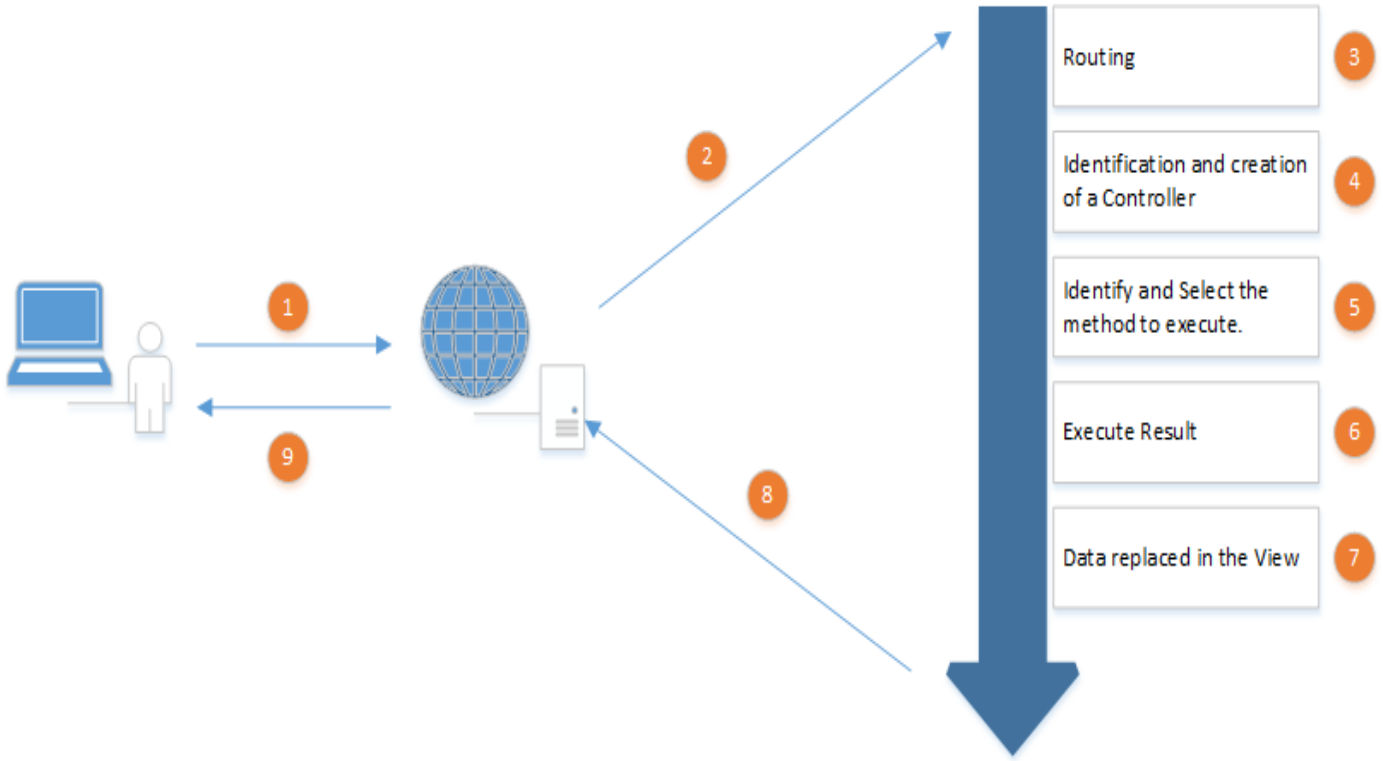
Now that you know where everything is located in your project, we can move forward with what is the process when an MVC application is initially called.

MVC App Execution

- **Entry points to MVC:**
  - UrlRoutingModuleand MvcRouteHandler
- **Request handling:**
  - Select appropriate controller
  - Obtain a specific controller instance
  - Call the controller's Execute method
- **Receive first request for the application**
  - Populating RouteTable
- **Perform routing**
- **Create MVC Request handler**
- **Create controller**
- **Execute controller**

MVC App Execution

- **Invoke action**
- **Execute result**
  - ViewResult, RedirectToRouteResult, ContentResult, FileResult, JsonResult, RedirectResult



Models

- Models are probably the easiest section to address first. Models are the objects that define and store your data so you can use them throughout the application.
- Models to be the equivalent of plain old CLR (Common Language Runtime) objects, or POCO's. A POCO is a plain class that holds structured data.
- one simple POCO (or model) would be similar to:

```
public class Customer
{
public string FirstName { get; set; }
public string LastName { get; set; }
public string Company { get; set; }
public IEnumerable<Order> Orders { get; set;
}
```

Models

Your business Model may look something like this:

```
public class Customer
    {    public string FirstName { get; set; }
         public string LastName { get; set; }
         public string Company { get; set; }
         public IEnumerable<Order> Orders { get; set; }
         public DateTime FirstMet { get; set; }

         public int CalculateAge(DateTime endTime)
         {       var age = endTime.Year - FirstMet.Year;
            if (endTime < FirstMet.AddYears(age))
               age--;
            return age;
         }
    }
```