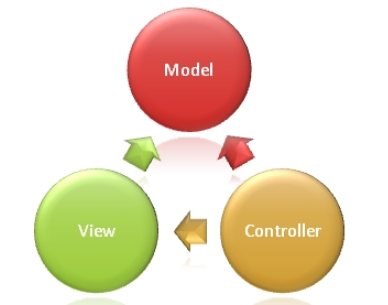
MVC

**ASP.NET MVC Overview**

The Model-View-Controller (MVC) architectural pattern separates an application into three main components: the model, the view, and the controller. ASP.NET MVC framework is a lightweight, highly testable presentation framework that (as with Web Forms-based applications) is integrated with existing ASP.NET features.



**Models**. Model objects are the parts of the application that implement the logic for the application s data domain. Often, model objects retrieve and store model state in a database.

**Views**. Views are the components that display the application s user interface (UI). Typically, this UI is created from the model data.

**Controllers**. Controllers are the components that handle user interaction, work with the model, and ultimately select a view to render that displays UI.

The MVC pattern helps you create applications that separate the different aspects of the application (input logic, business logic, and UI logic).

It easier to test applications than it is to test a Web Forms-based ASP.NET Web application. For example, in a Web Forms-based ASP.NET Web application, a single class is used both to display output and to respond to user input. Writing automated tests for Web Forms-based ASP.NET applications can be complex, because to test an individual page, you must instantiate the page class, all its child controls, and additional dependent classes in the application.

**Advantages of an MVC-Based Web Application**

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

**Advantages of a Web Forms-Based Web Application**

* It supports an event model that preserves state over HTTP, which benefits line-of-business Web application development. The Web Forms-based application provides dozens of events that are supported in hundreds of server controls.
* It uses view state or server-based forms, which can make managing state information easier.

# Understanding the ASP.NET MVC Execution Process

1. Requests to an ASP.NET MVC-based Web application first pass through the UrlRoutingModule object, which is an HTTP module. This module parses the request and performs route selection. The UrlRoutingModule object selects the first route object that matches the current request.

(Http module is executed with every request)

1. If no routes match, the **UrlRoutingModule** object does nothing and lets the request fall back to the regular ASP.NET or IIS request processing.
2. From the selected **Route** object, the **UrlRoutingModule** object obtains the **IRouteHandler** object that is associated with the **Route** object. Typically, in an MVC application, this will be an instance of **MvcRouteHandler**.

//

// Summary:

// Matches a URL request to a defined route.

[TypeForwardedFrom("System.Web.Routing, Version=3.5.0.0, Culture=Neutral, PublicKeyToken=31bf3856ad364e35")]

public class UrlRoutingModule : IHttpModule

{

…

//

// Summary:

// Matches the HTTP request to a route, retrieves the handler for that route, and

// sets the handler as the HTTP handler for the current request.

//

// Parameters:

// context:

// Encapsulates all HTTP-specific information about an individual HTTP request.

public virtual void PostResolveRequestCache(HttpContextBase context);

…

}

1. The **IRouteHandler** instance creates an **IHttpHandler** object and passes it the **IHttpContext** object. By default, the **IHttpHandler** instance for MVC is the **MvcHandler** object. The **MvcHandler** object then selects the controller that will ultimately handle the request.

public class MvcRouteHandler : IRouteHandler

{

…

//

// Summary:

// Returns the HTTP handler by using the specified HTTP context.

//

// Parameters:

// requestContext:

// The request context.

//

// Returns:

// The HTTP handler.

protected virtual IHttpHandler GetHttpHandler(RequestContext requestContext);

…

}

The module and handler are the entry points to the ASP.NET MVC framework. They perform the following actions:

* Select the appropriate controller in an MVC Web application.
* Obtain a specific controller instance.
* Call the controller's **Execute** method.

The following lists the stages of execution for an MVC Web project:

* Receive first request for the application
  + In the Global.asax file, **Route** objects are added to the **RouteTable** object.
* Perform routing
  + The **UrlRoutingModule** module uses the first matching **Route** object in the **RouteTable** collection to create the **RouteData** object, which it then uses to create a **RequestContext** (**IHttpContext**) object.
* Create MVC request handler
  + The **MvcRouteHandler** object creates an instance of the **MvcHandler** class and passes it the **RequestContext** instance.
* Create controller
  + The **MvcHandler** object uses the **RequestContext** instance to identify the **IControllerFactory** object (typically an instance of the **DefaultControllerFactory** class) to create the controller instance with.
* Execute controller - The **MvcHandler** instance calls the controller s **Execute** method. |
* Invoke action
  + Most controllers inherit from the **Controller** base class. For controllers that do so, the **ControllerActionInvoker** object that is associated with the controller determines which action method of the controller class to call, and then calls that method.
* Execute result
  + A typical action method might receive user input, prepare the appropriate response data, and then execute the result by returning a result type. The built-in result types that can be executed include the following: **ViewResult** (which renders a view and is the most-often used result type), **RedirectToRouteResult**, **RedirectResult**, **ContentResult**, **JsonResult**, and **EmptyResult**.