The Learning House

427 S 4th Street #300

Louisville KY 40202

Introduction to ASP.NET MVC

.NET Cohort

Coding Bootcamp

SOFTWARE GUILD

Why Do We Care?

- ASP.NET MVC is a web development framework that utilizes the Model-View-Controller (MVC) architecture and the existing ASP.NET platform.
- ASP.NET launched in 2002 and caused a large paradigm shift for web development. The web forms model attempted to hide both HTTP and HTML from developers who traditionally came from the desktop world.
  - Web Forms, then and now, use Server Controls to simulate a stateful UI.

Model-View-Controller

MVC is a standard design pattern that many applications follow.

Models are objects that represent application's data domain. They are used to pass information between the UI and other layers.

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

Controllers are the classes that handle user interactions. They move data between the UI and other layers, enforcing rules and workflows.

MVC is a Popular Pattern

The MVC pattern encourages separation between input logic, business logic, and UI logic, which helps with testing and organization.

This separation pays off on larger teams because developers can work on the UI, business logic, and input logic concurrently.

What is Web Forms?

Web Forms was created to ease desktop developers into web development.

It uses server controls to render HTML and contains event modeling much like a desktop application.

The biggest source of comfort is that Web Forms uses something called the ViewState to create the illusion of stateful web pages.

Stateful?

The web is stateless by default. This means that once code is completed on the server and a response is rendered to the browser, all server code objects are destroyed.

This allows the web to be highly scalable. It also means that the server, by default, does not “remember” what data was sent to which clients.

Web Forms puts hidden encoded text into HTML pages that is used to restore data when the page is posted back to the server.

Weighing the Options

Web Forms

- Supports an event model that preserves state over HTTP, which benefits line-of-business Web application development
- Uses view state or server-based forms, which can make managing state information easier
- Works well for small teams of Web developers and designers who want to take advantage of the large number of components available for rapid application development
- Generally, less complex for application development because the components are tightly integrated and usually require less code than the MVC model

MVC

- Easier to manage complexity by dividing an application into the model, view, and controller. Allows custom URLs and Routing, which make it ideal for SEO.
- Provides better support for test-driven development (TDD)
- Works well for Web applications that are supported by large teams of developers and designers who need a high degree of control over the application behavior

Can I Migrate to MVC From Web Forms?

- Absolutely. The two technologies can coexist in the same application, so it is possible to migrate existing applications gradually.
- This, of course, is much easier if your existing application applied good object-oriented principles where the domain model, business logic, and data access layers were partitioned.

MVC Execution Process

A browser executes a request to a server via a URL.

The server uses a UrlRoutingModule class, which parses the request and attempts to select a route object that matches the current request.

If no matches are found, the routing module does nothing and lets the server handle the process (e.g. serve up a static HTML page or issue a 404 Not Found error).

UrlRoutingModule's Responsibility

The UrlRoutingModule, upon finding a matching route, obtains an IRouteHandler which figures out which controller will handle the request. So, the workflow looks like this:

1. RoutingModule checks URL for a pattern match
2. If match is found, select handler associated with that request
3. Handler loads the specific controller class for the request and calls its execute method

MVC Server Life Cycle

Stage	Details
Receive the first request for the application	In the global.asax file, routes are registered with the RouteTable object. The RouteTable is stored until the application is restarted.
Perform routing	The UrlRoutingModule compares the URL to the RouteTable collection. If a match is found, it creates a RouteData object and a RequestContext object.
Create MVC request handler	The MvcRouteHandler object creates an instance of MvcHandler and passes it the RequestContext.
Create controller	The MvcHandler uses the RequestContext to identify which IControllerFactory object to instantiate the right controller instance.
Invoke action	The controller calls the Action Method associated with the URL.
Execute result	The controller provides a response, typically a ViewResult (HTML page), RedirectResult, ContentResult, or JSONResult.

RouteConfig.cs

Here is the starter code for a new MVC site in the RouteConfig.cs file:

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

Notice that the URL specifies 3 segments. Controller is the first segment, action is second, and id is the optional third segment.

{controller}/{action}/{id}

Request URL	Segment Variables
http://mysite.com/admin/index	Controller=admin Action=index
http://mysite.com/index/admin	Controller=index Action=admin
http://mysite.com/banana/apples	Controller=banana Action=apples
http://mysite.com/admin	Controller=admin Action=index (index is the default in routeconfig)
http://mysite.com/admin/index/banana	Controller=admin Action=index Id=Banana

Understanding Controllers

A controller is responsible for handling how a user interacts with your MVC Application. A controller contains the flow control logic that determines what response to send back to a user who makes a request.

A controller is just a class that inherits from Controller. It communicates with our business layer, and passes model data to views.

Controller Duties

Controllers should contain very little logic, only the bare minimum required to return the right view or redirect the user to another action.

Models are in charge of storing data properties and determining validity.

Your business layer is in charge of saving, retrieving, enforcing rules, and doing calculations for your application.

Gut Check

- Which type of component controls the UI of the application as HTML and CSS?
- Which type of component is responsible for storing data?
- Which type of component is responsible for making decisions about which view to display?
- How does the application decide which controller to invoke for a given HTTP request?