4

Controller

This chapter covers

* Understanding the Controller Anatomy
* Storyboarding the Application.
* Mapping the Presentation Model
* Using Input from the Browser
* Passing View Metadata

The focus of the Model-View-Controller pattern is the controller. With this pattern, every request is handled by a controller and rendered by a view. Without the controller, presentation and business logic would move to the view, as we have seen with Web Forms. With the ASP.NET MVC Framework, every request routes to a controller, which is simply a class that implements the IController interface (see chapter 9). Microsoft provides the base class System.Web.Mvc.Controller to make creating a controller easy. The controller base class you choose is not crucial because most request processing goes into executing the ActionResult, which is the type that each action returns.‑­

An action is a method that handles a particular request. This method can take no parameters or many, but by the time the action method finishes executing, there ought to be one or many objects ready to be sent to the view, and the name of the view should be selected if the view does not follow the convention of having the same name as the action. Beyond that, the developer is in complete control regarding how to implement a controller and its actions. This chapter will explore controllers that use many actions and inherit from the System.Web.Mvc.Controller base class. The meat of the controller is the action.

4.1 The Anatomy of a Controller

A controller is simply a class that inherits from System.Web.Mvc.Controller that contains one or more methods that act as Actions. An action method is used to serve a single web requests. An action method normally returns an ActionResult and can take zero or many arguments. Parameters are resolved into the action method by the Model Binders, by using this automatic binding to complex types Action methods can focus on the concerns of controlling the application logic rather than spending time: translating input from the browser into Input Models or mapping Domain models into presentation models.

An well written action should have a clear purpose and a single responsibility. That responsibility is to accept input from the browser and coordinate the flow of the application. Along the way, the action should rely on application services to perform tasks such executing business logic, performing data access, or file I/O.

Listing 4.1 The SimpleController decides on ViewData and renders a view

using System.Web.Mvc;

namespace MvcInAction.Controllers

{

public class SimpleController : Controller

{

public ActionResult Hello()

{

ViewData.Add("greeting", "Hello Readers!");

return View();

}

}

}

4.2 Storyboarding the Application

Action methods exist to perform presentation coordination for a screen/page. This coordination is the Storyboard of the Application. Imagine drawing the flow of application screens on a whiteboard. Each place that a user has the ability to input information through a form or click a button, there are at least two possible outcomes. These are a successfully handle request or a request that could not be handled and results in additional input or a decision from the user. This alternate path could included dealing with data type validation business rule validation, or exception handling.

figure of a whiteboard view of a storyboard.

Focusing an Action to concentrate on wiring together the storyboard of the application has some good side effects. Actions tend to become smaller and focused. By moving business logic out of the Action and into supporting services, the actions are less complex and easier to test. A lean action should result in two possible outcomes: Happy Path ( a successfully processed request) or a alternate path. If an action starts branching to handle multiple alternate paths this is sign that the Action method is handling too much and some design should be put into the storyboard of the application.

4.3 Transforming a model to a view model

Show the index action. Walk through the mapping of a domain object to the view model.

Send the view model to the view through a View Result

4.3 Accepting Input

4.3.1 Handling the Successful processing of Action Input

Explain ModelState.IsValid as one branch in flow  
Show code from Edit Action

Explain Post Redirect Get

Diagram to explain PRG Pattern

TempData as a way to pass view data/success message

4.3.2 Handling the Failure processing of the Action Input

Show code from the failure branch of the Edit Action

Explain ModelState as a way to add additional error messages.

4.4 Summary