**Adding a Data Model**

The data model is the most important part of any ASP.NET Core application. The model is the representation of the real-world objects, processes, and rules that define the subject, known as the *domain*, of the application. The model, often referred to as a *domain model*, contains the C# objects (known as domain objects) that make up the universe of the application and the methods that manipulate them. In most projects, the job of the ASP.NET Core application is to provide the user with access to the data model and the features that allow the user to interact with it.

**Conventions**

The convention for an ASP.NET Core application is that the data model classes are defined in a folder named Models.

A single controller class can define multiple action methods, and the convention is to group related actions together in the same controller.

**How is razor view-engine choose the right view for a specific action?**

The Razor view engine will use the name of the action method when looking for a view file.

**“a” element**

The addition to the listing is an *a element* that has an *asp-action attribute*. The attribute is an example of a tag helper attribute, which is an instruction for Razor that will be performed when the view is rendered. The asp-action attribute is an instruction to add an href attribute to the a element that contains a URL for an action method. This tag helper tells Razor to insert a URL for an action method defined by the same controller for which the current view is being rendered.

*<a asp-action="RsvpForm">RSVP Now</a>*

**Good practice**

There is an important principle at work here, which is that you should use the features provided by ASP.NET Core to generate URLs, rather than hard-code them into your views. When the tag helper created the href attribute for the a element, it inspected the configuration of the application to figure out what the URL should be. This allows the configuration of the application to be changed to support different URL formats without needing to update any views.

**Biulding the Form**

…

**@model** Asp.Net.Models.GuestResponse

…

<form asp-action="RsvpForm" method="POST">

<div>

<label asp-for="Name">You name:</label>

<input asp-for="Name">

</div>

<div>

<label asp-for="Email">You email:</label>

<input asp-for="Email">

</div>

<div>

<label asp-for="Phone">You phone:</label>

<input asp-for="Phone">

</div>

<div>

<label>Will you attend?</label>

<select asp-for="WillAttend">

<option value="">Choone an option</option>

<option value="true">Yes</option>

<option value="false">No</option>

</select>

</div>

<button type="submit">Submit RSVP</button>

</form>

**@model**

The @model expression specifies that the view expects to receive a *GuestResponse* object as its view model.

**How to associate view elements with the model property**

I have defined a *label* and *input* element for each property of the *GuestResponse* model class (or, in the case of the *WillAttend* property, a *select* element). Each element is associated with the model property using the asp-for attribute, which is another tag helper attribute. The tag helper attributes configure the elements to tie them to the view model object.

**asp-for attrbiute**

The asp-for attribute on the label element sets the value of the for attribute. For example, the asp-for attribute on the input element sets the id and name elements. This attribute extracts the name of the specified model property into the rendered HTML.

**asp-action attribute**

Of more immediate use is the asp-action attribute applied to the form element, which uses the application’s URL routing configuration to set the action attribute to a URL that will target a specific action method, like this:

*<form method="post" action="/Home/RsvpForm">*

**Receiving Form Data**

As things stand, clicking the Submit RSVP button just clears any values you have entered in the form. That is because the form posts back to the RsvpForm action method in the Home controller, which just renders the view again.

**How to process submitted form data**

To receive and process submitted form data, I am going to use an important feature of controllers. I will add a second RsvpForm action method to create the following:

* A method that responds to HTTP GET requests: A GET request is what a browser issues normally each time someone clicks a link. This version of the action will be responsible for displaying the initial blank form when someone first visits /Home/RsvpForm.
* A method that responds to HTTP POST requests: By default, forms rendered using Html.BeginForm() are submitted by the browser as a POST request. This version of the action will be responsible for receiving submitted data and deciding what to do with it.

**Good Practice**

Handing GET and POST requests in separate C# methods helps to keep my controller code tidy since the two methods have different responsibilities. Both action methods are invoked by the same URL, but ASP.NET Core makes sure that the appropriate method is called, based on whether I am dealing with a GET or POST request.

**HttpGet and HttpPost Attributes**

I have added the HttpGet attribute to the existing RsvpForm action method, which declares that this method should be used only for GET requests. I then added an overloaded version of the RsvpForm method, which accepts a GuestResponse object. I applied the HttpPost attribute to this method, which declares that the new method will deal with POST requests.

***[HttpGet]***

*public ViewResult RsvpForm() => View();*

***[HttpPost]***

*public ViewResult RsvpForm(GuestResponse guestResponse) => View();*

**Model Binding**

A useful ASP.NET Core feature whereby incoming data is parsed and the key/value pairs in the HTTP request are used to populate properties of domain model types.

**What is actually doing the model binding feature?**

Eliminates the grind of dealing with HTTP requests directly and lets you work with C# objects rather than dealing with individual data values sent by the browser. The GuestResponse object that is passed as the parameter to the action method is automatically populated with the data from the form fields.

**How to access the value of a propertyin the domain object?**

*@Model.<PropertyName>*

**Adding the Thanks View**