ASP.NET MVC

Estimated time for completion: 30 minutes

Overview:

In this lab you will be building a MVC (Model,View,Controller) web site to provide access to historical weather data from a series of locations all over the United Kingdom. The web site will contain an initial screen that will display all the locations. Each location will be represented by a hyperlink, clicking on the link will navigate to a table of data for the said location.

Goals:

- Learn how to build a simple Model View Controller Web Application

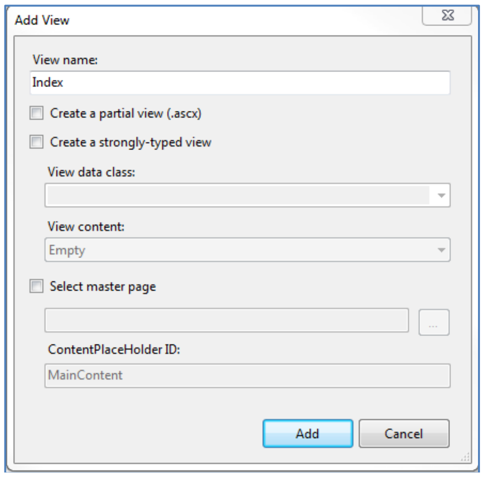
Lab Notes:

Part 1: Creating the initial Controller and View

In this part of the lab you will create the Weather controller. The weather controller will be responsible for orchestrating the model and view based on the user's action. In this first part you will ignore the model side of the system and just simply construct a single static view.

1. Open the **before\WebWeather.sln** solution. The solution contains two projects. The first is a class library called **WeatherLibrary** this provides access to the underlying weather data. The second is an MVC project called **WebWeather**. You will only modify code inside **WebWeather**.
2. Your first task is to create a controller that will provide access to the weather data. Add a controller by right clicking on the **Controllers** folder and selecting add controller. Call the controller **WeatherController**. The created controller will contain a generated action called Index. The code inside this action will be executed on receipt of the URL **/Weather**. The default implementation simply returns a View; the view name is not specified which means it will look for a view with the same name as the action, in this case Index.
3. Now create a simple static view for the Index action. Creating a View directly under the **Views** folder would work, but a more maintainable approach is to create a sub folder first with the same name as the controller. Create a folder called **Weather** and create a view

inside this folder called **Index**. Uncheck all the checkboxes.



View name: Index

- [ ] Create a partial view (.ascx)
- [ ] Create a strongly-typed view

View data class:

View content: Empty

- [ ] Select master page

ContentPlaceHolder ID: MainContent

4. Inside Index.cshtml, add a simple heading to the page.

```
<body>
    <div>
        <h1>Weather History</h1>
    </div>
</body>
```

5. Display the properties for the **WebWeather** project and navigate to the Web tab. Under the Start Action section select the Specific page option and supply a value of **Weather**.
6. Run the web project and confirm that the "Weather History" page is displayed.

After Part 1\WebWeather.sln

Part 2: Adding the Model

Static views are obviously not what we use MVC for; views typically render some dynamic content. The view obtains its data from the model. In this part of the lab you will construct a model to represent all the weather stations. Further you will modify the view you built in the previous part to render the information supplied by the model.

1. Create a model class for the model under the Models folder called WeatherStationsModel,
2. This model class will expose all the information required by the view, which in this case is the list of all weather stations. Add a property **IEnumerable<string> Stations** to the model class, and stub it out with a few made up stations using yield return.
3. Recompile the project, this will ensure that the new model class will be available with Intellisense inside the view editor.
4. Modify the first line of Index.cshtml to represent a strongly typed view. This will then provide intellisense when accessing the model.

```
@model WebWeather.Models.WeatherStationsModel
```

5. Inside the div of the view provide a foreach loop that iterates over the **Stations** property of the model outputting the station names

```
  @foreach (string station in Model.Stations){
              @station <br/>
  }
```

6. Running the code now will perform a NULL reference exception, as you have not initialised the model inside the controller action.
7. Modify the controller Index action to create an instance of **WeatherStationModel** and make it the active model by setting the **ViewData.Model** property to reference the model instance.

```
ViewData.Model = new WeatherStationsModel();
```

8. Re-run the project you will now see your dummy stations.
9. Instead of dummy values refactor the **WeatherStationsModel** so that it you will now use an instance of an **IWeatherService**. Modify your **WeatherStationsModel** class to accept an instance of such a service via a constructor, and replace your dummy values with a call to **GetWeatherStations**

```
public class WeatherStationsModel
{
  private IWeatherService service;

  public WeatherStationsModel(IWeatherService weatherService)
  {
    service = weatherService;
  }
  public IEnumerable<string> Stations
  {
    get { return service.GetWeatherStations(); }
  }
}
```

10. The controller now needs to pass in an instance of **IWeatherService** when constructing the model. The **WeatherLibrary** contains a single implementation that will read the data from CSV files. Use the following lines in your controller class to create a CSV based instance of this service.

```
private IWeatherService weatherService;
        //
        // GET: /Weather/
public WeatherController()
{
     weatherService = new
   CsvWeatherService(System.Web.HttpContext.Current.Server.MapPath("~/Weather
   Data"));
}
```

11. Supply an instance of IWeatherService when constructing the model inside the Index action.
12. Recompile and run the project. You will now see a large number of weather stations.

Solution: After Part 2\WebWeather.sln

Part 3: More Action Handling

In this part of the lab you will extend the Weather controller to provide an action to display the historic weather data for a given station.

1. Add a new action to your controller to handle the request for viewing weather data for a given station, the method will take a single parameter that identifies the station to view. Call this method List.

```
public ActionResult List(string id)
```

2. The action will ultimately make use of the weather service to retrieve the sample data for the supplied station. To implement this behaviour you will first create a model that holds all the date required for the action. Create a new model called **WeatherStationModel**. This model will need to contain two properties, one that is the name of the station the other a property representing the stream of data from the **GetWeatherSample** method found on the **IWeatherService**.

```
public class WeatherStationModel
{
   IWeatherService service;

    public WeatherStationModel(IWeatherService service , string station )
    {
       this.service = service;
       Name = station;
    }
    public string Name { get; private set; }

    public IEnumerable<WeatherSample> Samples
    {
      get { return service.GetWeatherSamples(Name); }
    }
 }
```

3. Create an instance of the model and make it the active model inside the List action method.

```
  ViewData.Model = new WeatherStationModel(weatherService, id);
```

4. Recompile
5. Now create a strongly typed view for this action. The responsibility of the view will be to display the name of the station, and then output a table containing a row for each of the samples.

```
@model WebWeather.Models.WeatherStationModel
@using WeatherLibrary

<body>
    <div>
        <h2>@Model.Name</h2>

        <table border="1">
            <tr>
             <th>  Year     </th>
             <th> Month </th>
             <th>  Min      </th>
             <th>  Max      </th>
             <th> Rainfall  </th>
            </tr>

      @foreach (WeatherLibrary.WeatherSample sample in Model.Samples)
      {
          <tr>
              <td>@sample.Year</td>
              <td>@sample.Month</td>
              <td>@sample.MinTemperature</td>
              <td>@sample.MaxTemperature</td>
              <td>@sample.RainFall</td>
          </tr>
       }
        </table>
    </div>
</body>
```

6. The URL for this action would then take the form **/Weather/List/id**. With the id part varying based on the station being requested. Run the project and then manually navigate to a given weather stations data via the browsers URL field.

```
http://Server Address/Weather/List/aberporth
```

Solution: After Part 3\WebWeather.sln

Part 4: Invoking actions

In the previous part you created a new action to provide the weather station data. In this part you will modify the view displaying the list of stations to provide hyperlinks to take the user to the weather data for that station.

1. Open the **Views\Weather\Index.aspx** file. You now want replace the text for each weather station with a hyperlink. It may be tempting to start adding an html anchor tag and build the URL up. Whilst this would work it results in the view being tightly coupled to the URL structure. Far better to have the framework generate the URL based on the controller and action you wish to invoke.

```
~~<a href="/Weather/List/<%: station %>">~~
```

2. To create a hyper link that invokes the action use the **Html.ActionLink** helper method. The **ActionLink** method takes the text to display, the name of the action to execute against the current controller and finally a collection of additional input parameters. The method will return a string containing the necessary HTML tags ready to be inserted into the output of the view

```
@Html.ActionLink( station , "List" ,
           new RouteValueDictionary{ { "id" , station } } )
```

3. A short hand form of this uses anonymous types as a convenient way to build a dictionary. Reflection is utilized at runtime to determine the parameter set.

```
@Html.ActionLink( station , "List" , new { id = station } )
```

4. Modify the index.aspx to use whatever form of **ActionLink** you prefer, and re-run the application. You will now be presented with a page of hyperlinks, clicking on them will take you to the corresponding weather stations data.

Solution After Part 4\WebWeather.sln

Part 5: Extra bits (Optional)

This part of the lab is optional, and is designed to make you think for yourself, extend the application to implement the following features.

- Group the Weather Stations by their first letter
- Provide paging support for the weather data

Solution After \WebWeather.sln