Blog MVC Project

BlogPosts Controller Walkthrough

In this handout, we will be walking though the code in the BlogPostsController using Visual Studio's debugging tools. This controller should have been scaffolded from the BlogPost model. The purpose of this exercise is to teach you how to read and understand code that has been given to you.

First, drop a break point by every "return" line in your controller. This will pause the program when it runs. Use the "Step Over" arrow to continue running the program.

Loading the Index

Run your application. In the URL bar, type in "/BlogPosts" after "localhost:… ", so you should go to the URL: "localhost:…/BlogPosts".

What action is this URL calling?

Creating An Item

On the Index page, click on the link "Create New". What action is this calling? How is this reflected in the URL?

On the Create page, fill in the form's input boxes and click "Create".

What action is being called in the controller?

Hover over "blogPost" in every line of the code in this action. Click on the arrow. Note what is being attached to the blogPost. Where is this blogPost coming from? Does the blogPost ever change?

Try creating another blog post, but enter text for the "Created" input box. How does this change what code is run in the action?

Create a few more blog posts.

Editing An Item

On the Index page, click on "Edit" next to one of your blog posts. What action is being called now? Is this a GET or POST action?

Note the "int? id" being passed in as a parameter to this action. How does this differ from what is passed into the Create GET action?

What is this id being used for? Hover over id in each line of the code where it is used. What line of code is returning a specific blog post from the database?

Change a line in the Description on the Edit page and click "Save". What action is this calling? Compare this action the Create POST action. What line of code is different?

Passing One Model vs. A Collection of Models

Go back or refresh the Index page. Hover over db.Posts. Click on the arrow and then click on the arrow by Results View. Click on the arrow by each item. What is being returned to the view?

Click on the “Details” link of a blog post. What action is being called? Note, unlike Edit and Create, Details only has a GET action. Why is this?

Compare the Details GET action to the Edit GET action. What are the differences in the code?

Open up two files from the Solution Explorer: Views/BlogPosts/Index.cshtml and Views/BlogPosts/Details.cshtml. Notice the @model at the top of these .cshtml files. Why is the model different for the Index vs. the Details? Walk through loading the Index and a Details page again.

Notice the foreach loop in Index.cshtml. Why is a foreach loop necessary? Above the foreach loop, type: @Model. Reload the Index page. What does this show above the table of blog posts?

What is being passed into the Index view? What is being passed into the Details view?

Understanding Razor Syntax

In Index.cshtml, which line of code creates a link for going to the Create page? Hover over this code and note what parameters are being passed into this method.

What line of code creates a link for going to the Details page of a specific blog post? How is this different than the link for creating a new blog post? What parameters are passed into this method? Why is there an extra parameter for this method?

Open Views/BlogPosts/Edit.cshtml. Compare this to Details.cshtml. How are these views coded differently? In Details.cshtml, what line of code displays the title of the blog post? In Edit.cshtml, what line of code creates the input textbox for displaying the title on a blog post?

Note that both Razor methods are creating HTML based on the property of the model that has been passed in. Navigate to an Edit page in the browser. Right click on the text box next to “Title” and select “Inspect”. What is the id and name of this input box? What is the value? How does this correspond to the model’s property and value?

Final Thoughts

Do you understand every line of code that Visual Studio has scaffolded for you in your BlogPostsController? Now is a good time to write comments above the code to explain what is happening. You can refer back to these comments in later projects.