Building HTML Forms

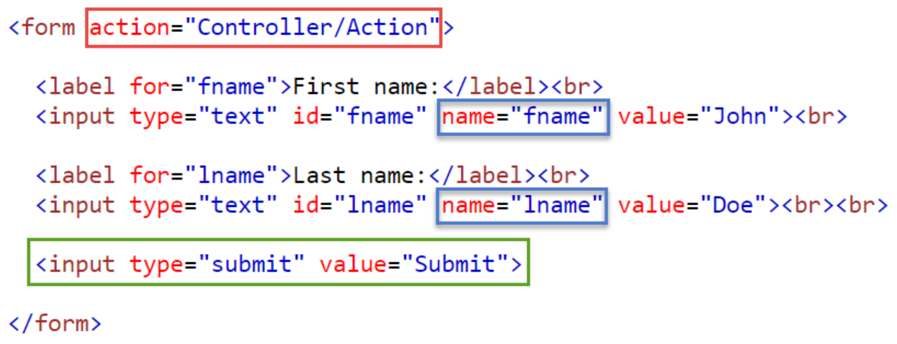
Background

Thus far our primary focus has been on link creation using various overloads of **Html.ActionLink()**. This helper method builds an anchor tag and takes us to various HttpGet Actions throughout our project. The main purpose of an HttpGet Action is to display a View for the user to see. Once the View is displayed it can be placed into 1 of 2 categories as follows.

- No user interaction expected (i.e. Index, Details)
- User interaction expected (i.e. Create, Edit, Delete)

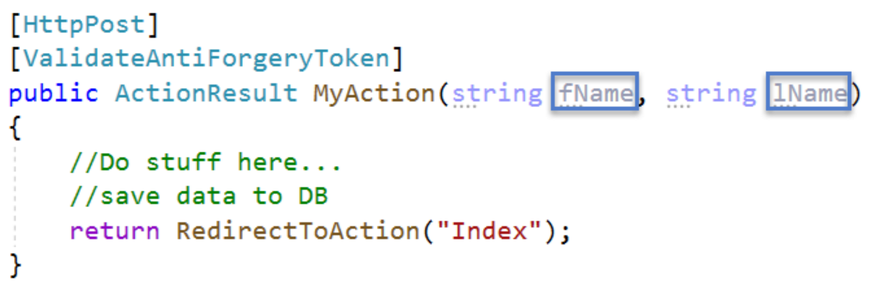
In the first category there will likely not be an HttpPost version of the Action but for Actions in the second category there is an expectation to collect user data and therefore will likely have an HttpPost. Once we begin focusing on collecting user input and forwarding it to an HttpPost we are solidly in the realm of the **Form**.

In HTML, forms are typically created using the **<form>** tag



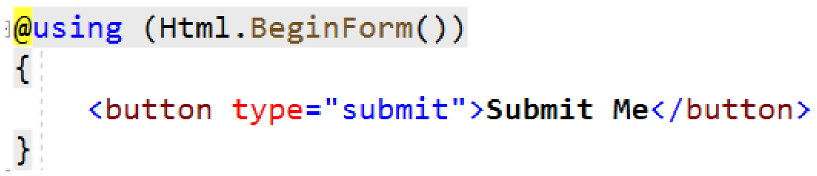
```
<form action="Controller/Action">

  <label for="fname">First name:</label><br>
  <input type="text" id="fname" name="fname" value="John"><br>

  <label for="lname">Last name:</label><br>
  <input type="text" id="lname" name="lname" value="Doe"><br><br>

  <input type="submit" value="Submit">

</form>
```

In the code snippet pasted above we see an Html form that contains 2 inputs and a button. When the button is clicked the form gets submitted to the Controller and Action specified in the action attribute. When this form data hits the HttpPost, the method should be expecting two variables with the names **fname** and **lname**.



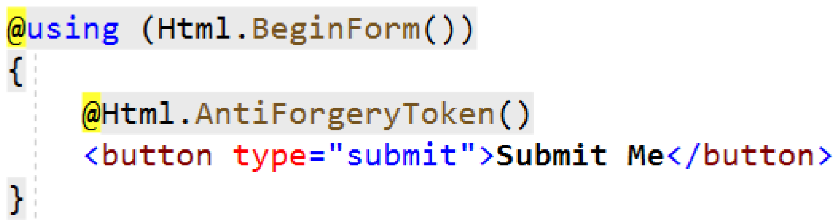
```
[HttpPost]
[ValidateAntiForgeryToken]
public ActionResult MyAction(string fName, string lName)
{
    //Do stuff here...
    //save data to DB
    return RedirectToAction("Index");
}
```

We will not be building Forms using this approach because we want to take advantage of the built in Html Razor helper method **Html.BeginForm()**. The BeginForm helper method builds a form for us behind the scenes using the <form> tag. A basic form in one of our views using this approach could look like this at first.



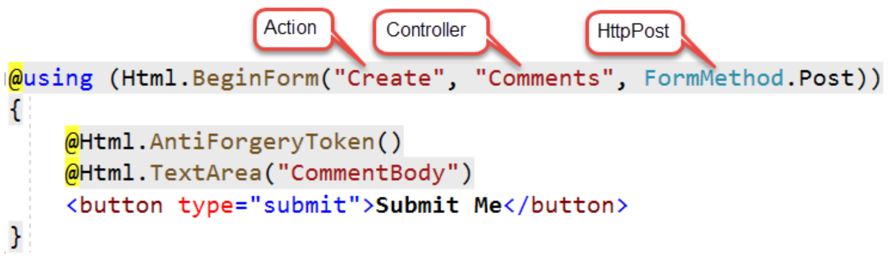
```
@using (Html.BeginForm())
{
    <button type="submit">Submit Me</button>
}
```

The code above creates a form with a single button whose only responsibility is to submit the form when it is clicked. This is helpful but our forms will always include a security mechanism called an **AntiForgeyToken**. An Anti-Forgery Token is a long string embedded into the form before it is rendered to help prevent **CSRF** attacks. We use another helper method to allow this token to be embedded and it changes the basic form to the following.



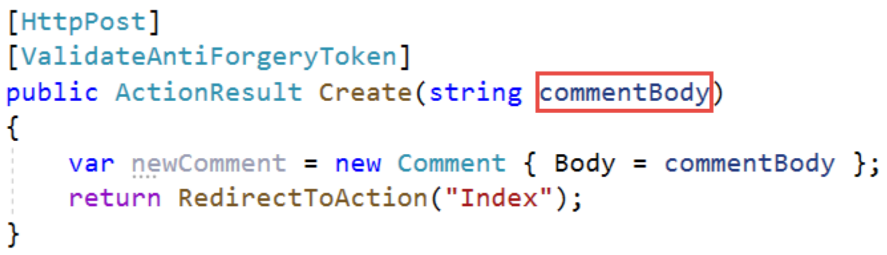
```
@using (Html.BeginForm())
{
    @Html.AntiForgeryToken()
    <button type="submit">Submit Me</button>
}
```

Now we need to tell the form where to submit its data when the submit button is clicked, which Controller and which Action. We tell a form where to submit its data by including the name of the Action, the name of the Controller and whether we want to send data to an HttpPost or HttpGet...usually an HttpPost. For example, if we wanted to submit data inside a form to the Comments Controller and the HttpPost Create Action when the submit button is clicked, we can do that as follows.



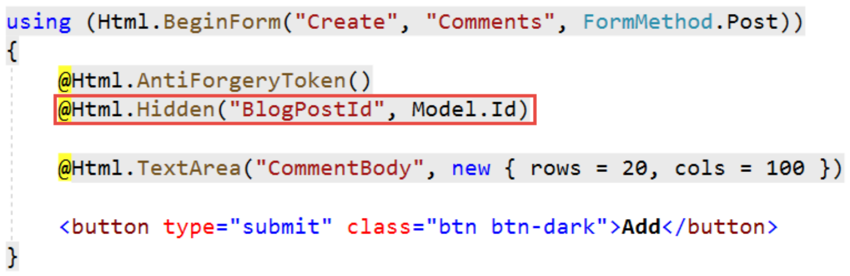
```
@using (Html.BeginForm("Create", "Comments", FormMethod.Post))
{
    @Html.AntiForgeryToken()
    @Html.TextArea("CommentBody")
    <button type="submit">Submit Me</button>
}
```

The above form will send a piece of data to the **Comments** Controller and the **Create** Action with the name **CommentBody**. The associated HttpPost would accept the incoming CommentBody as follows.



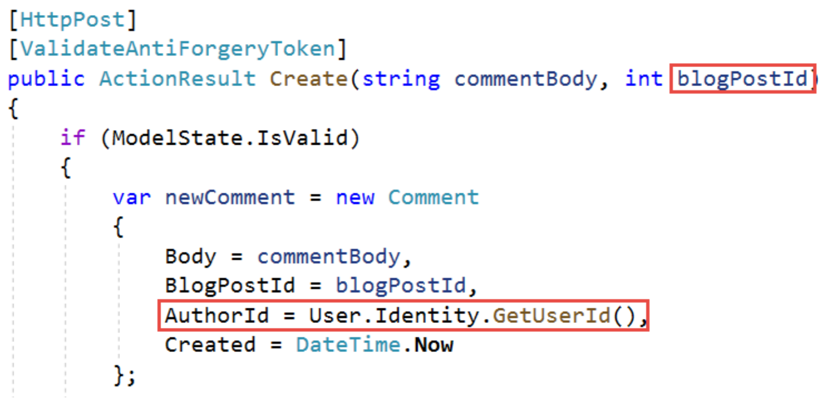
```
[HttpPost]
[ValidateAntiForgeryToken]
public ActionResult Create(string commentBody)
{
    var newComment = new Comment { Body = commentBody };
    return RedirectToAction("Index");
}
```

This is a great start but unfortunately the Body of the Comment is not the only information required in order to save it to the Database. In addition to the Body we also need to record which Blog Post the Comment belongs to and who wrote it. We can make a small change to the Comment form to embed the Blog Post's Primary Key and give it a special name of **BlogPostId** so that we pass it into the HttpPost as the Foreign Key.



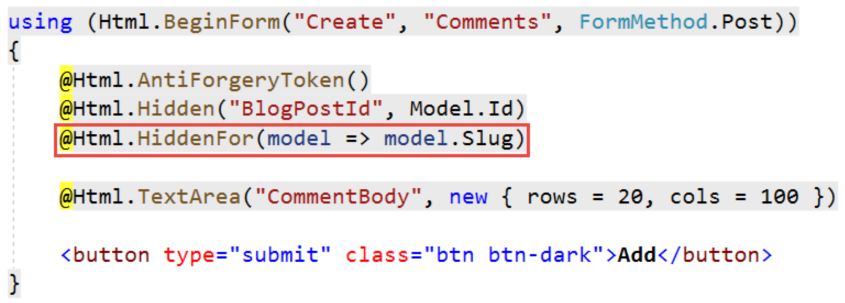
```
using (Html.BeginForm("Create", "Comments", FormMethod.Post))
{
    @Html.AntiForgeryToken()
    @Html.Hidden("BlogPostId", Model.Id)

    @Html.TextArea("CommentBody", new { rows = 20, cols = 100 })

    <button type="submit" class="btn btn-dark">Add</button>
}
```

Following the pattern, the new signature of the Create HttpPost would need to be updated as follows.



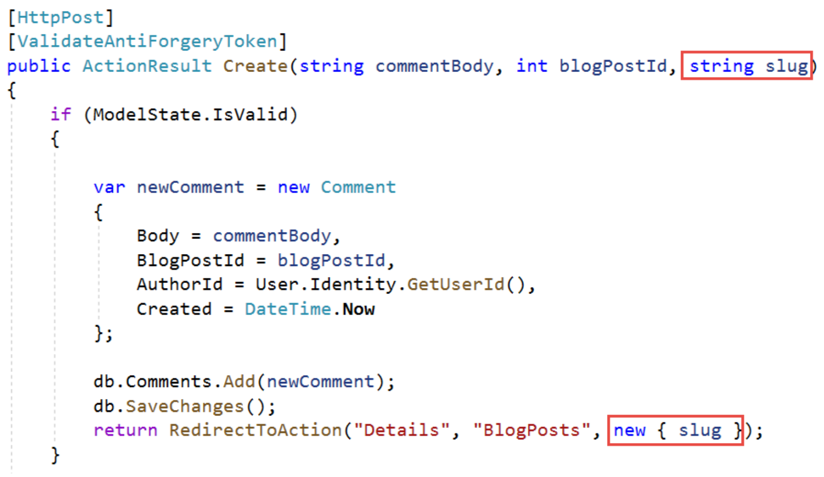
```
[HttpPost]
[ValidateAntiForgeryToken]
public ActionResult Create(string commentBody, int blogPostId)
{
    if (ModelState.IsValid)
    {
        var newComment = new Comment
        {
            Body = commentBody,
            BlogPostId = blogPostId,
            AuthorId = User.Identity.GetUserId(),
            Created = DateTime.Now
        };
```

The body of the HttpPost Create will use the incoming data to populate properties of the new Comment and will also have to **programmatically set the Author's id** by leveraging the built in **User.Identity.GetUserId()** method. Once the new Comment data has been saved to the Database, we want to redirect the user back to the Blog Post Details Action so they can see the new comment listed. In order to build a link that passes the Slug as a Route Value we need access to the Slug. The highlighted code below embeds a hidden input in the form that is named Slug.



```
using (Html.BeginForm("Create", "Comments", FormMethod.Post))
{
    @Html.AntiForgeryToken()
    @Html.Hidden("BlogPostId", Model.Id)
    @Html.HiddenFor(model => model.Slug)

    @Html.TextArea("CommentBody", new { rows = 20, cols = 100 })

    <button type="submit" class="btn btn-dark">Add</button>
}
```

And the updated HttpPost changes to the following.



```
[HttpPost]
[ValidateAntiForgeryToken]
public ActionResult Create(string commentBody, int blogPostId, string slug)
{
    if (ModelState.IsValid)
    {

        var newComment = new Comment
        {
            Body = commentBody,
            BlogPostId = blogPostId,
            AuthorId = User.Identity.GetUserId(),
            Created = DateTime.Now
        };

        db.Comments.Add(newComment);
        db.SaveChanges();
        return RedirectToAction("Details", "BlogPosts", new { slug });
    }
```

The final line of code is using the Slug passed in from the form to redirect the User back to the Blog Post Details Action.