codecademy

ASP.NET: Page Models

asp-for

The `asp-for` attribute is used to connect an `<input>` element to a page model property.

The above example would be connected to a `Username` property in the page model:

```
public class IndexModel : PageModel
{
  [BindProperty]
  public string Username { get;
set; }
}
```

In the rendered HTML, attributes like `type`, `id`, and `name` will be added to the `<input>` tag:

```
Enter a username:
<input type="text" id="Username"
name="Username" >
```

```
<!-- In .cshtml file -->
Enter a username: <input asp-
for="Username" />
```

URLs in Razor Pages

In a default Razor Pages application, each view page's URL route is its path minus `.cshtml`.
For example, given the folder structure:

```
Pages
|-- About.cshtml
|-- About.cshtml.cs
|-- Index.cshtml
|-- Index.cshtml.cs
|-- Businesses/
   |-- Codecademy.cshtml
   |-- Codecademy.cshtml.cs
```

About.cshtml is available at

`https://localhost:8000/About`

Codecademy.cshtml is available at

`https://localhost:8000/Businesses/Codecademy`

Index.cshtml is available at

`https://localhost:8000/Index` OR

`https://localhost:8000`

OnGet() & OnGetAsync()

When a page model receives a GET request, its `OnGet()` or `OnGetAsync()` method is invoked. This typically happens when a user navigates to the page model's corresponding view page.
A page model must have either `OnGet()` or `OnGetAsync()`. It cannot have both.

```
public class ZooModel : PageModel
{
  public string FavoriteAnimal { get; set;
}

  // Sets FavoriteAnimal when page is
requested
  public void OnGet()
  {
    FavoriteAnimal = "Hippo";
  }
}
```

OnPost() & OnPostAsync()

When a page model receives a POST request, its `OnPost()` or `OnPostAsync()` method is invoked. This typically happens when a user submits a form on the page model's corresponding view page.

A page model must have either `OnPost()` or `OnPostAsync()`. It cannot have both.

```
public class IndexModel : PageModel
{
  public string Message { get; set; }

  public void OnPost()
  {
    Message = "OnPost() called!";
  }
}
```

Void Handler Methods

In a page model, synchronous handler methods like `OnGet()` and `OnPost()` that have no `return` statement will have a return type of `void`.

This results in the current page being rendered in response to every request.

```
public class IndexModel : PageModel
{
  public string Username { get; set; }

  public void OnGet()
  {
    Username = "n/a";
  }

  public void OnPost(string username)
  {
    Username = username;
  }
}
```

Task Handler Methods

In a page model, asynchronous handler methods like `OnGetAsync()` and `OnPostAsync()` that have no `return` statement will have a return type of `Task`.
This results in the current page being rendered in response to every request.

```
public class IndexModel : PageModel
{
  public string Users { get; set; }
  private UserContext _context { get; set;
}

  public IndexModel(UserContext context)
  {
    _context = context;
  }

  // Task return type
  public async Task OnGetAsync()
  {
    Users = await
_context.Users.ToListAsync();
  }

  // Task return type
  public async Task OnPostAsync(string
username)
  {
    _context.Users.Add(username);
    await _context.SaveChangesAsync();
  }
}
```

Handler Method Parameters

Page model handler methods, like `OnGet()`, `OnGetAsync()`, `OnPost()`, and `OnPostAsync()`, can access an incoming HTTP request's query string via its own method parameters.

The name of the method parameter(s) must match (case-insensitive) the name(s) in the query string.

```
// Example GET request
// https://localhost:8000/Songs?id=1
public async Task OnGetAsync(int id)
{
  // id is 1
}

// Example POST request
// https://localhost:8000/Songs?
songName=Say%20It%20Loud
public async Task OnPostAsync(string
songName)
{
  // songName is "Say It Loud"
}
```

Model Binding

In *model binding*, a page model retrieves data from an HTTP request, converts the data to .NET types, and updates the corresponding model properties. It is enabled with the `[BindProperty]` attribute.

```
public class IndexModel : PageModel
{
  [BindProperty]
  public string Username { get; set; }

  [BindProperty]
  public bool IsActive { get; set; }

  // Example POST
  // https://localhost:8000?
username=muhammad&IsActive=true
  public void OnPost()
  {
    // Username is "muhammad"
    // IsActive is true
  }
}
```

Append URL Segments

In Razor view pages (**.cshtml** files), the `@page` directive can be used to add segments to a page's default route. Use this feature by typing a string after `@page`.
For example, imagine the below code is from **About.cshtml**. Instead of `/About`, the new route would be `/About/Me`:

```
@page "Me"
```

```
@page "segment"
```

Append URL Parameters

In Razor view pages (**.cshtml** files), the `@page` directive can be used to add parameters to a page's route.

- The parameter(s) must be in between curly braces `{ }` after `@page`.
- Constraints, like `int` or `alpha`, can be added using colons `:`.
- A parameter can be marked optional using a question mark `?`.

Imagine the below code is from **Book.cshtml**. Instead of `/Book`, the new route could be `/Book/0` or `Book/1` or `Book/2` etc.:

```
@page "{id:int}"
```

Imagine the below code is from **House.cshtml**. The new route could be `/House` or `House/small` or `House/big` etc.:

```
@page "{size?}"
```

Imagine the below code is from **Song.cshtml**. Instead of `/Song`, the new route would be `/Song` or `Song/0` or `Song/1` etc.:

```
@page "{song:int?}"
```

```
@page {param}
```

asp-route-{value}

The `asp-route-{value}` attribute is used in `<a>` elements to add additional information to a URL route.

> `{value}` typically matches a property in a page model.
>
> The provided value will be added as a route segment or a query string, depending on how the route is defined.

```
<a asp-page="About" asp-route-
name="Joanne">About Joanne</a>
```

If the above `<a>` tag is in a **.cshtml** file, it would be rendered as this HTML:

```
<a href="localhost:8000/About?
name=Joanne">About Joanne</a>
```

However, if the About page has a route parameter, like `@page {name}`, then the same tag would be rendered as this HTML:

```
<a
href="localhost:8000/About/Joanne">
About Joanne</a>
```

Default Responses

In page models, a handler method with no `return` statement will respond to HTTP requests by sending back the associated page.
In the above example, `IndexModel` is associated with **Index.cshtml**. Neither `OnGet()` nor `OnPostAsync()` have `return` statements, so they both return **Index.cshtml**.

```
public class IndexModel : PageModel
{
  // Sends Index.cshtml
  public void OnGet()
  { }

  // Sends Index.cshtml
  public void OnPost()
  {  }
}
```

Page()

To return the view page associated with a page model, use `Page()` in the page model's handler methods.

This happens implicitly if the handler method has a `void` return type

If a handler method calls `Page()`, its return type is typically `IActionResult` or `Task<IActionResult>` (although others exist).

```
public class AboutModel : PageModel
{
  // Sends About.cshtml
  public IActionResult OnGet()
  {
    return Page();
  }
}
```

RedirectToPage()

To redirect users to a different Razor page within the application, use `RedirectToPage()` in the page model's handler methods.

If a handler method calls `RedirectToPage()`, its return type is typically `IActionResult` or `Task<IActionResult>` (although others exist).

The string argument passed to this method is a file path. `"/Index"` is a relative path and `"./Index"` is an absolute path.

```
public class IndexModel : PageModel
{
  // Sends Privacy.cshtml
  public IActionResult OnPost()
  {
    return RedirectToPage("./Privacy");
  }
}
```

NotFound()

To send a "Status 404 Not Found" response, use `NotFound()` in the page model's handler methods.

If a handler method calls `NotFound()`, its return type is typically `IActionResult` or `Task<IActionResult>` (although others exist).

```
public class EditModel : PageModel
{
  public async Task<IActionResult>
OnGetAsync(int? id)
  {
    if (id == null)
    {
      return NotFound();
    }

    // do something with id here

    return Page();
  }
}
```