Introduction to Routing & MVC

Install the routing package

1. Use the application you created in the earlier labs
2. Open the `project.json` file
3. In the `dependencies` section, add an entry for the "Microsoft.AspNetCore.Routing" package:

```
"dependencies": {
  ...,
  "Microsoft.AspNetCore.Routing": "1.0.0"
}
```

4. Save the file. Open the `Startup.cs` file
5. Add the routing services to the configuration in the `Startup.cs`:

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddRouting();
}
```

6. In the `Configure` method, create a `RouteBuilder` with a handler for the root of the site and add it to the middleware pipeline:

```
using Microsoft.AspNetCore.Routing;
...
public void Configure(IApplicationBuilder app, IHostingEnvironment env,
  ILoggerFactory loggerFactory)
{
    loggerFactory.AddConsole();

    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }

    var routeBuilder = new RouteBuilder(app);

    routeBuilder.MapGet("", context
      => context.Response.WriteAsync("Hello from Routing!"));

    app.UseRouter(routeBuilder.Build());
}
```

7. Run the site and verify your middleware is hit via routing (Ctrl+F5)
8. Add another route that matches a sub-path:

```
routeBuilder.MapGet("sub", context
  => context.Response.WriteAsync("Hello from sub!"));
```

9. Run the site and verify that your routes are hit for the corresponding URLs. Browsing to /sub (e.g. `http://localhost:8081/sub`) should display the message "Hello from sub!"

Capture and use route data

1. Add another route that captures the name of an item from the URL, e.g. "item/{itemName}", and displays it in the response:

   ```
   routeBuilder.MapGet("item/{itemName}", context
     => context.Response.WriteAsync($"Item: {context.GetRouteValue("itemName")}"))
   ```

2. Run the site and verify that your new route works. Browsing to "/item/monkey" should display the message "Item: monkey".

3. Modify the route to include a route constraint on the captured segmet, enforcing it to be a number:

   ```
   routeBuilder.MapGet("item/{id:int}", context
     => context.Response.WriteAsync($"Item ID: {context.GetRouteValue("id")}"));
   ```

4. Run the site again and see that the route is only matched when the captured segment is a valid number. Browsing to "/item/5" will work, but browsing to "/item/monkey" will now show a missing page (HTTP 404) error.

5. Modify the router to include both versions of the route above (with and without the route constraint)

6. Experiment with changing the order the routes are added and observe what affect that has on which route is matched for a given URL

   Note: Completed code for this section is found /Labs/Code/Lab3A.

Add MVC

1. Open `project.json` and add "Microsoft.AspNetCore.Mvc" to the `"dependencies"` section and save it:

   ```
   "dependencies": {
     ...,
     "Microsoft.AspNetCore.Mvc": "1.0.0"
   }
   ```

2. Add a "Controllers" folder to your application

3. Create a new class called "HomeController" in the new folder and add the following code:

   ```
   using Microsoft.AspNetCore.Mvc;

   public class HomeController
   {
     [HttpGet("/")]
     public string Index() => "Hello from MVC!";
   }
   ```

4. Replace the Routing middleware from the previous step with MVC services and middleware in `Startup.cs` as shown:

   ```
   public void ConfigureServices(IServiceCollection services)
   {
       services.AddMvc();
   ```

```
}

public void Configure(IApplicationBuilder app, IHostingEnvironment env,
  ILoggerFactory loggerFactory)
{
    loggerFactory.AddConsole();

    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }

    app.UseMvc();
}
```

5. Run the site and verify the message is returned from your MVC controller

6. If you have time, try the following:

- Change the controller to render a view instead of returning a string directly
- Play with the `[HttpGet("/")]` attribute to change the route the action method will match

Note: Completed code for this section is found /Labs/Code/Lab3B.