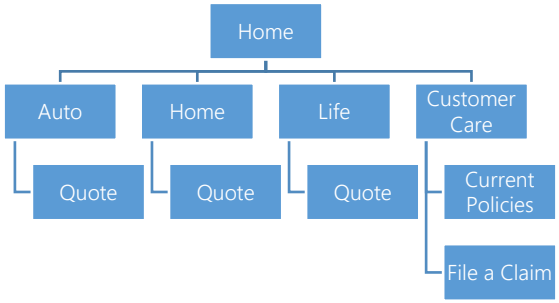
Lab: Building ASP.NET Controllers for MVC and Web API

Context: Fisher College is creating an insurance company for current students and alumni. They plan to sell Auto, Home, and Life insurance products and they have hired you to build the prototype application. You meet with the stakeholders and they want the following requirements:

- A home page
- Product sections for each product: Auto, Home, Life
    - Each product section should have a have a landing page
    - Each product section should have a quote page
- A customer care section
    - A page to see current policies
    - A page to file a claim



Exercise 1: Create the Project

1. Navigate to Your Workspace and run the Yeoman generator

```
cd C:\{your-workspace-path}

yo
```

2. Select the ASPNET generator
3. Select the Empty template
4. Name the project **FisherInsurance**
5. Restore, build, and run

```
dotnet restore

dotnet build
```

```
dotnet run
```

Browse to the site

What does the output of your terminal say?

Ctrl-C to stop the web server

6. Initialize you local Git repository and commit your changes (use a message like “initialize project”).
7. Add the ASP.NET MVC dependency in your project.json file:

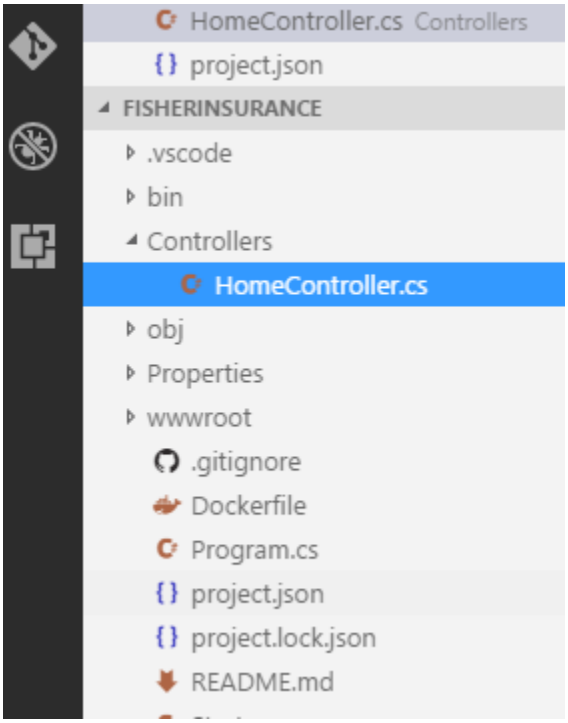
```
{
  "dependencies": {
    "Microsoft.NETCore.App": {
      "version": "1.1.0",
      "type": "platform"
    },
        "Microsoft.AspNetCore.Mvc": "1.1.0",
        "Microsoft.AspNetCore.Diagnostics": "1.1.0",
        "Microsoft.AspNetCore.Server.IISIntegration": "1.1.0",
        "Microsoft.AspNetCore.Server.Kestrel": "1.1.0",
```

8. Save your changes to the project.json and run the restore command
9. Open your startup.cs file and make the following changes:

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

```
        }

        public void Configure(IApplicationBuilder app, IHostingEnvironment env,
ILoggerFactory loggerFactory)

        {

            app.UseMvcWithDefaultRoute();

        }
```

10. Create a Controllers folder in the root of your project and add a HomeController.cs file to it:



11. Add the following code to the HomeController.cs:

```
using Microsoft.AspNetCore.Mvc;


public class HomeController : Controller

{
```

```
    public IActionResult Index()
    {
        return Ok("This is the index of the HomeController");
    }
}
```

12. Run your project and verify that it works

```
dotnet build

dotnet run
```

Exercise 2: Add the Remaining Controllers

1. Create the controller for Auto
    - There should be a default route for 'index'
    - There should be a secondary route for 'quote' (e.g. /auto/quote that resolves to the Quote( ) method of the class)
2. Create the controller for Home
    - Pay attention to the naming. You already have a HomeController for the home page! You will need to think of a different name.
    - There should be a default route for 'index'
    - There should be a secondary route for 'quote'
3. Create the Controller for Life
    - There should be a default route for 'index'
    - There should be a secondary route for 'quote'
4. Create the controller for Customer Care
    - There should be a default route for 'index'
    - There should be a secondary route for 'claims (e.g. /customercare/claims that resolves to the Quote( ) method of the CustomerCareController class
5. Run your application and verify that each of these routes work:
    - http://localhost:5000
    - http://localhost:5000/home
    - http://localhost:5000/home/index
    - http://localhost:5000/auto
    - http://localhost:5000/auto/quote
    - http://localhost:5000/customercare/claims
    - What is the URL for to Home Insurance quote?
6. Commit your changes to Git with a meaningful message.

Exercise 3: Customize the URL's with Attribute Routing

After a review with the UX team, they do not like URL for the CustomerCareController. The have asked if you can change the route to 'customers'. Also, they would like the ability to file a claim and check on claim status.

1. Open the CustomerCareController and add the following attribute to the class:

```
[Route("customer")]

public class CustomerCareController : Controller

{

    public IActionResult Index()

    {

        return Ok("This is the index of the CustomerCareController");

    }

}
```

2. The UX team wants the URL for filing a claim to be '/claim/fileclaim', but your team lead says the naming convention for the method must be 'NewClaim( )'. Make the necessary changes.
3. The UX team wants the URL for checking status to be '/claim/myclaims, but your team lead says the naming convention for the method must be 'ClaimHistory( )'. Make the necessary changes.
4. Run your application and verify the following URL's work:
    - http://localhost:5000/customer/fileclaim
    - http://localhost:5000/customer/claimstatus
5. Commit your changes to Git with a meaningful message.

Exercise 4: Creating Web API Controllers

The demo with the Dean and his advisors went very well and they would like to add mobile functionality. You have been asked to build a backend for the mobile app to use.

1. Navigate to Your Workspace and run the Yeoman generator

```
cd C:\{your-workspace-path}

yo
```

2. Select the ASPNET generator
3. Select the Web API Application template
4. Name the project **FisherInsuranceApi**
5. Restore and build

```
dotnet restore

dotnet build
```

6. Delete the ValuesController and checkin your code to Git.
7. Create a new AutoController:

```
 [Route("api/auto/quotes")]

    public class AutoController : Controller

    {



    }
```

8. Create an action in the AutoContoller to get a submit a quote:

```
    // POST api/auto/quotes

        [HttpPost]

        public IActionResult Post([FromBody]string value)

        {

            return Created("", value);

        }
```

9. Create an action under the AutoContoller to get a submitted quote by Quote ID:

```
 // GET api/auto/quotes/5

        [HttpGet("{id}")]

        public IActionResult Get(int id)

        {

            return Ok("The id is: " + id);

        }
```

10. Create an action under the AutoContoller to update a submitted quote by Quote ID:

```
          // PUT api/auto/quotes/id

        [HttpPut("{id}")]

        public IActionResult Put(int id, [FromBody]string value)

        {

            return NoContent();

        }
```

11. Create an action under the AutoContoller to delete a submitted quote by Quote ID:

```
// DELETE api/auto/quotes/id

        [HttpDelete("{id}")]

        public IActionResult Delete(int id)

        {

            return Delete(id);

        }
```

12. Create a HomeController and repeat steps 7-10.
    - Pay attention to the routing
13. Create a LifeController and repeat steps 7-10.
    - Pay attention to the routing
14. Create a ClaimsController and repeat steps 7-10.
    - Pay attention to the routing. You are not handling quotes anymore, you are handling claims.

Exercise 5: Upload your code to GitHub.

Use the lessons you learned in Lab 1 to push FisherInsurance and FisherInsuranceApi to Github. Submit the link to the root of you account in the Assignments section in Carmen. Your profile should show three projects, helloapi, FisherInsurance, and FisherInsuranceApi.