Kuwait University

College of Computing Sciences and Engineering

Department of Information Science

COMP 2139 Web Application Development with C# .NET

Lab 2 Developing a Single-Page Web Application

Table of Contents

Laboratory #2 – Developing a Single Page Web Application

1. Laboratory Objective

The goal of this lab is to create a single page Tip Calculator web application using C# .NET.

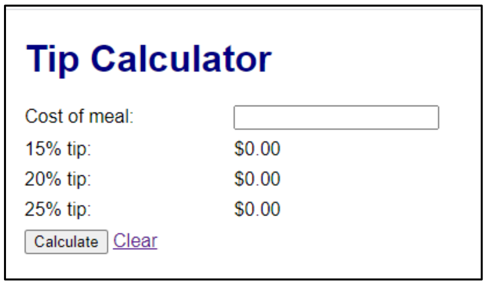
2. Laboratory Learning Outcomes:

After conducting this laboratory students will be able to:

a. Create a Git repository
b. Create a single page web application using C#.NET

3. Laboratory Instructions

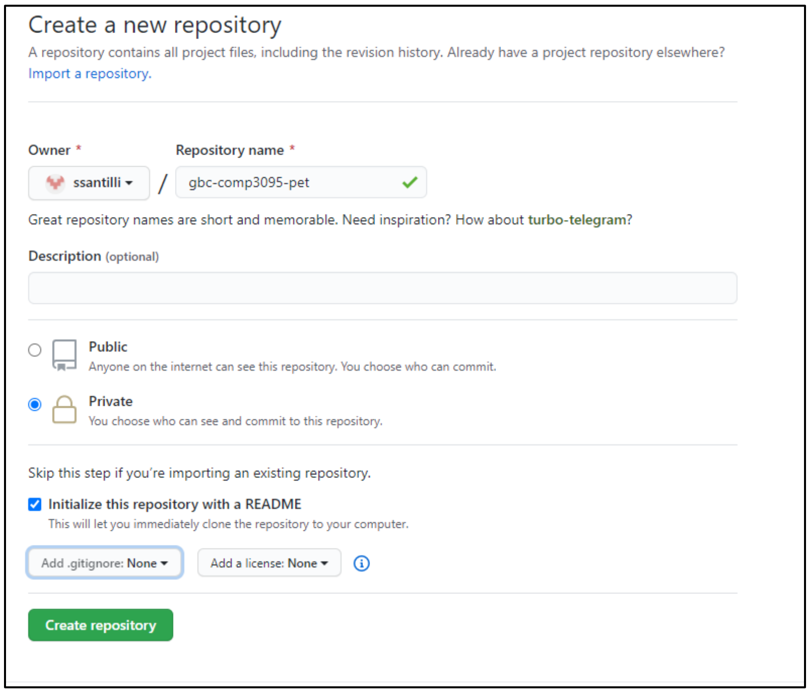
The objective of this lab is to create a Single Page Web Application – Tool Tip Calculator that meets the wireframe requirements as depicted below:



Tip Calculator

Cost of meal:	
15% tip:	$0.00
20% tip:	$0.00
25% tip:	$0.00

a. Start by navigating to GitHub (create a free account if you like), and create new project repository.



Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? Import a repository.

Owner * ssantilli / Repository name * gbc-comp3095-pet

Great repository names are short and memorable. Need inspiration? How about turbo-telegram?

Description (optional)

Public
Anyone on the internet can see this repository. You choose who can commit.

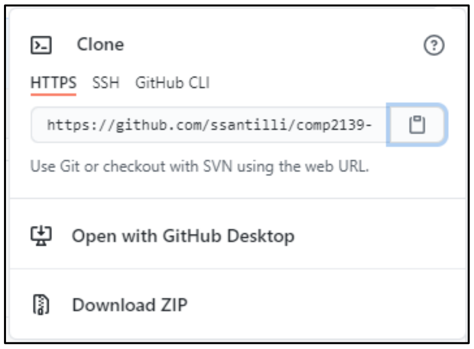
[✓ Private]
You choose who can see and commit to this repository.

Skip this step if you're importing an existing repository.

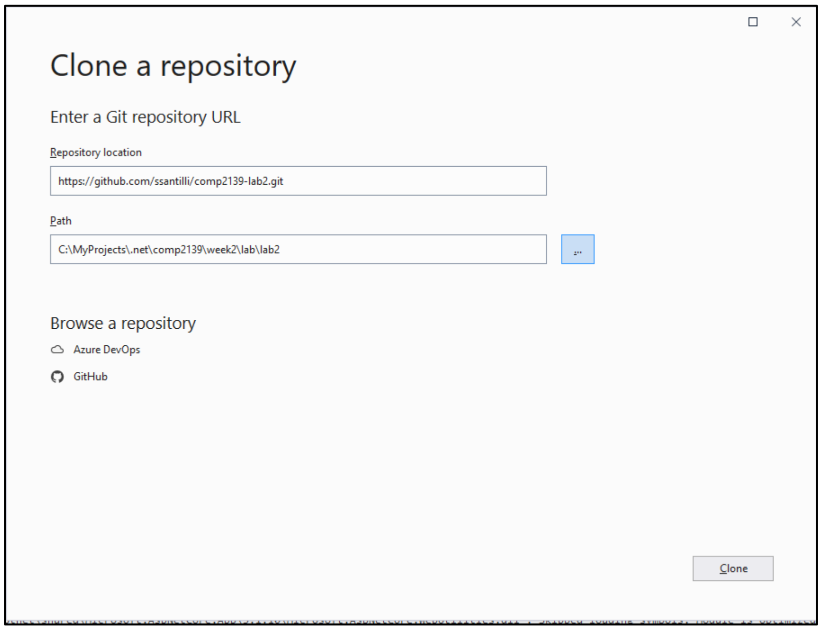
[✓] Initialize this repository with a README
This will let you immediately clone the repository to your computer.

Add .gitignore: None
Add a license: None

b. Clone the project (using ssh or https).


Clone
HTTPS
SSH
GitHub CLI
https://github.com/ssantilli/comp2139-
Use Git or checkout with SVN using the web URL.
Open with GitHub Desktop
Download ZIP

c. Launch Visual Studio, **File→Clone Repository**, entering the details for the repository you creating in the previous step:



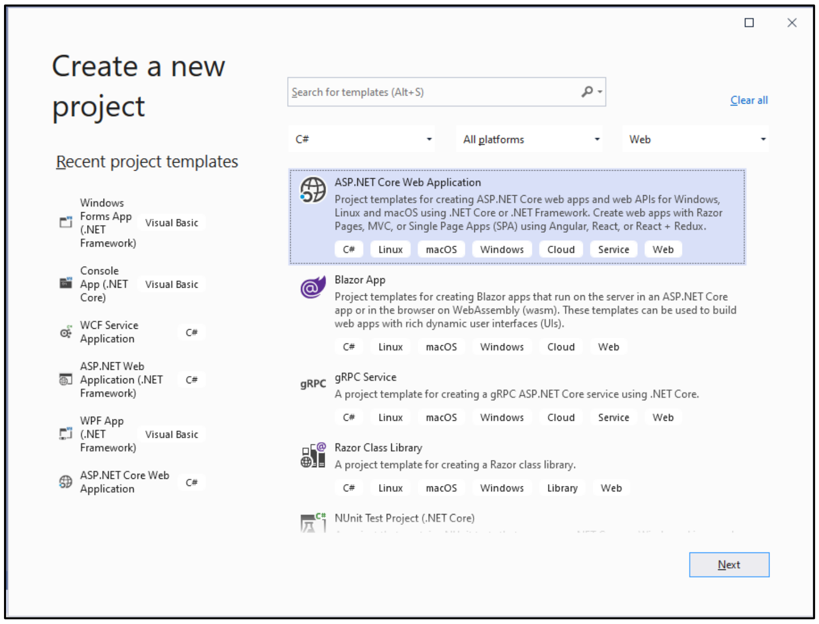
Repository location
https://github.com/ssantilli/comp2139-lab2.git

Path
C:\MyProjects\.net\comp2139\week2\lab\lab2

d. Click clone and Visual Studio will bring you to the location of your created project. This will effectively be the environment where your project will be created.

Creating the Tip Calculator Web Application

1. Within Visual Studio select **File→New→Project**, select **ASP.NET Core Web Application**.


Create a new project
Search for templates (Alt+S)
Clear all
C#
All platforms
Web
Recent project templates
Windows Forms App (.NET Framework) Visual Basic
Console App (.NET Core) Visual Basic
WCF Service Application C#
ASP.NET Web Application (.NET Framework) C#
WPF App (.NET Framework) Visual Basic
ASP.NET Core Web Application C#
Next

ASP.NET Core Web Application
Project templates for creating ASP.NET Core web apps and web APIs for Windows, Linux and macOS using .NET Core or .NET Framework. Create web apps with Razor Pages, MVC, or Single Page Apps (SPA) using Angular, React, or React + Redux.
C# Linux macOS Windows Cloud Service Web

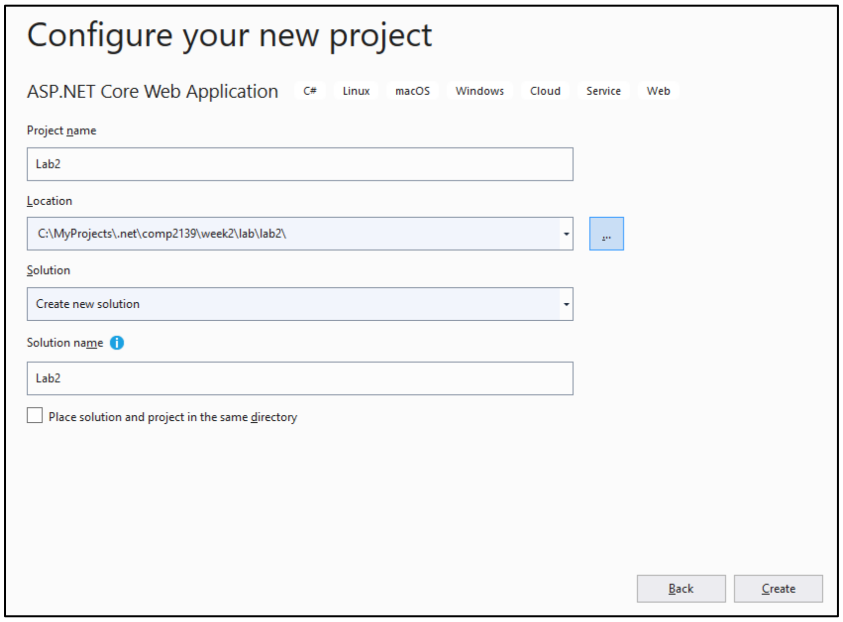
Blazor App
Project templates for creating Blazor apps that run on the server in an ASP.NET Core app or in the browser on WebAssembly (wasm). These templates can be used to build web apps with rich dynamic user interfaces (UIs).
C# Linux macOS Windows Cloud Web

gRPC Service
A project template for creating a gRPC ASP.NET Core service using .NET Core.
C# Linux macOS Windows Cloud Service Web

Razor Class Library
A project template for creating a Razor class library.
C# Linux macOS Windows Library Web

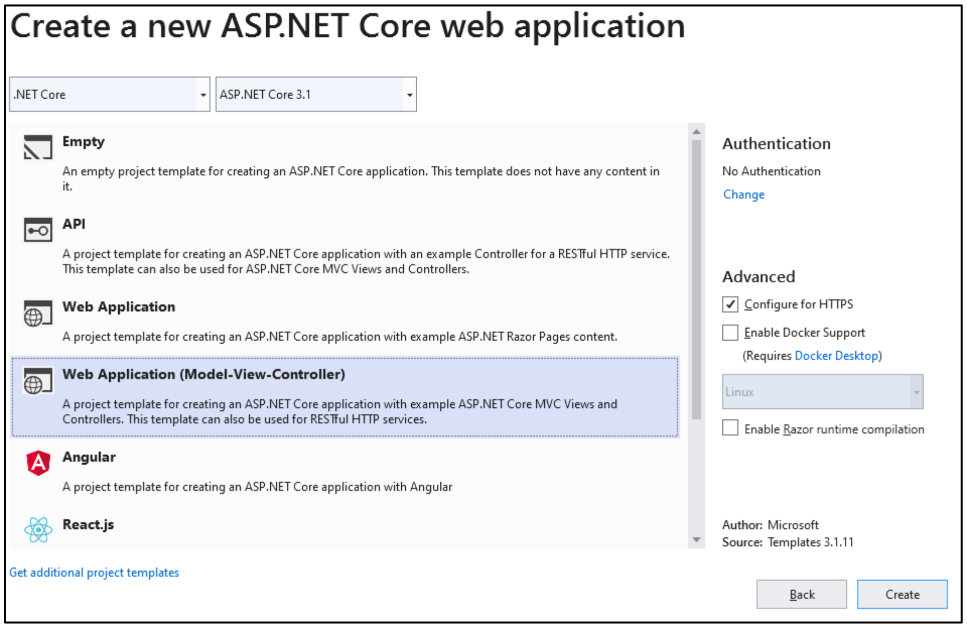
NUnit Test Project (.NET Core)

2. Fill in your desired project details for Project Name, and Location, then select create


Configure your new project
ASP.NET Core Web Application
C# Linux macOS Windows Cloud Service Web
Back
Create

Project name: Lab2
Location: C:\MyProjects\.net\comp2139\week2\lab\lab2\
Solution: Create new solution
Solution name: Lab2
☐ Place solution and project in the same directory

3. From the resultant page, select **Web-Application (Model-View-Controller),** then **Create**


Create a new ASP.NET Core web application

.NET Core | ASP.NET Core 3.1

Empty
An empty project template for creating an ASP.NET Core application. This template does not have any content in it.

API
A project template for creating an ASP.NET Core application with an example Controller for a RESTful HTTP service. This template can also be used for ASP.NET Core MVC Views and Controllers.

Web Application
A project template for creating an ASP.NET Core application with example ASP.NET Razor Pages content.

Web Application (Model-View-Controller)
A project template for creating an ASP.NET Core application with example ASP.NET Core MVC Views and Controllers. This template can also be used for RESTful HTTP services.

Angular
A project template for creating an ASP.NET Core application with Angular

React.js

Authentication
No Authentication
Change

Advanced
[✓] Configure for HTTPS
[ ] Enable Docker Support
(Requires Docker Desktop)
Linux
[ ] Enable Razor runtime compilation

Author: Microsoft
Source: Templates 3.1.11

Get additional project templates

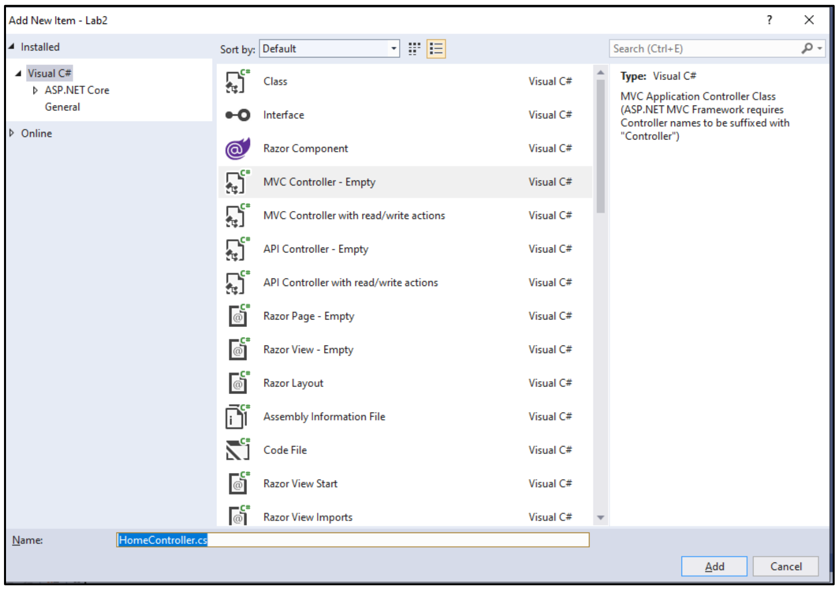
Back | Create

4. You should then be presented with your workspace to start creating your project.

Creating the Tip Calculator Controller

1. Navigate to your projects Controllers folder, deleting any default controllers Visual Studio has created for your as we will be creating a new one from scratch.

2. Right-Click on the **Controllers** folder, select **Add→New→Controller**, select an empty controller and give your controller a desired name.


Add New Item - Lab2

Installed
Visual C#
ASP.NET Core
General
Online

Sort by: Default

Class	Visual C#
Interface	Visual C#
Razor Component	Visual C#
MVC Controller - Empty	Visual C#
MVC Controller with read/write actions	Visual C#
API Controller - Empty	Visual C#
API Controller with read/write actions	Visual C#
Razor Page - Empty	Visual C#
Razor View - Empty	Visual C#
Razor Layout	Visual C#
Assembly Information File	Visual C#
Code File	Visual C#
Razor View Start	Visual C#
Razor View Imports	Visual C#

Search (Ctrl+E)

Type: Visual C#
MVC Application Controller Class (ASP.NET MVC Framework requires Controller names to be suffixed with "Controller")

Name: HomeController.cs

Add | Cancel

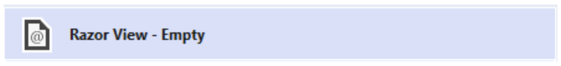
3. Select **Add** and a new empty controller should be added to your project.

Creating the Tip Calculator View

1. In the Solution Explorer, click on the **Views→Home** folder and delete default and .cshtml files within.

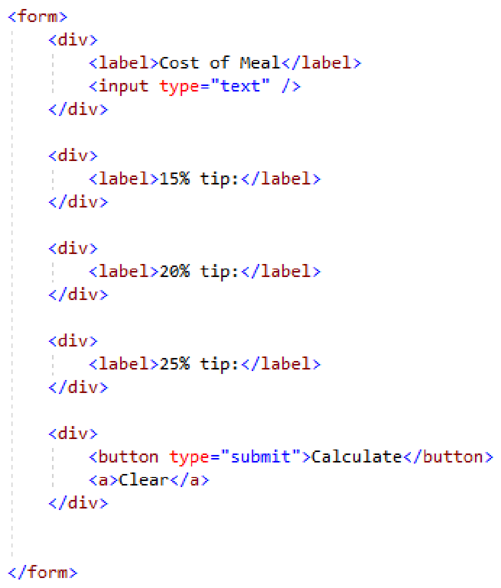
2. Again, in the Solution Explorer, right-click on the **Views→Home** folder and select **Add→View**

3. In the resulting dialog, select **Razor View - Empty**.



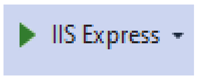
4. Given your View page a name and click **Add**

5. The view will initially be empty, so we will need to create the view to match the wireframe requirements. Create a html form that matches the requirements of the wireframe provided, where we will not worry about the functionality until later in the lab.



```
<form>
    <div>
        <label>Cost of Meal</label>
        <input type="text" />
    </div>

    <div>
        <label>15% tip:</label>
    </div>

    <div>
        <label>20% tip:</label>
    </div>

    <div>
        <label>25% tip:</label>
    </div>

    <div>
        <button type="submit">Calculate</button>
        <a>Clear</a>
    </div>


</form>
```

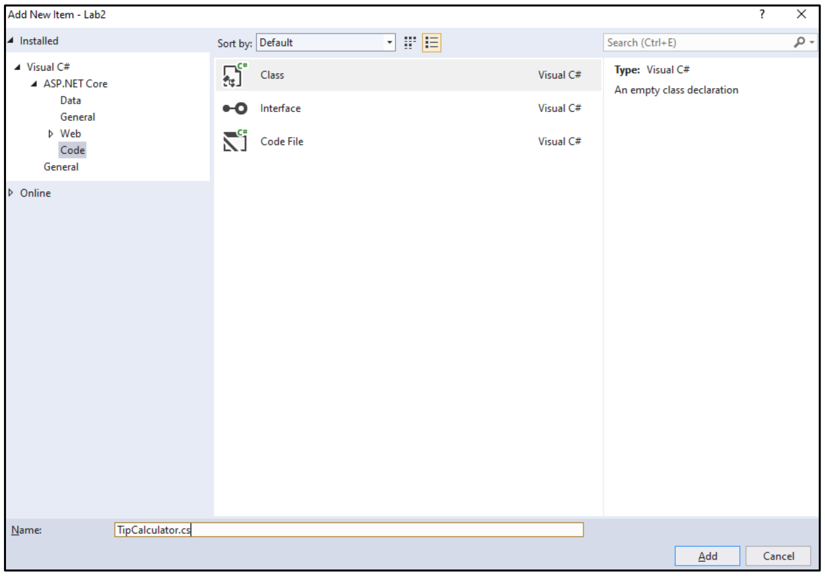
6. Run the application to validate that it renders okay. Press the "**play**" button to your application in the provided IIS Express server that comes bundled with Visual Studio Community.



Creating the Tip Calculator Model

1. In the solutions Explorer, open Models folder and delete any auto-generated classes within.

2. In the solutions Explorer, right-click the Models folder and select Add→Class



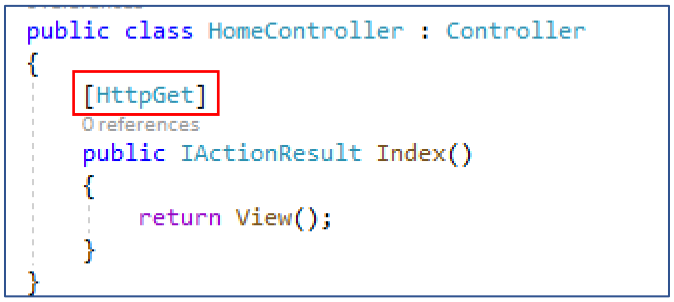
Name: TipCalculator.cs

3. Give your class a desired name (ex. TipCalculator.cs) and click **Add**. You now should have a new empty model class added to your project.

Refactor the Tip Calculator Controller

1. Refactor the Tip Calculator controller, index() method to handle the default case where the Tip Calculator page is first rendered and all of its values are calibrated to initial values.

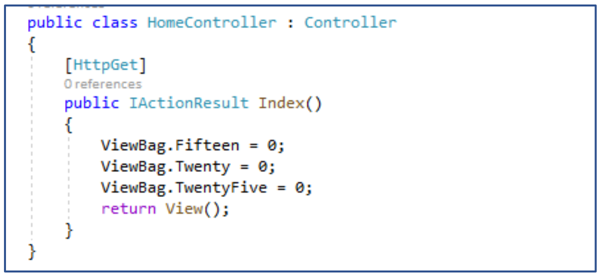
   To do this we need to modify the index() method in the controller to respond to the first HTTP GET request.



```
public class HomeController : Controller
{
    [HttpGet]
    0 references
    public IActionResult Index()
    {
        return View();
    }
}
```

2. Next, we will update the .NET ViewBag creating three new tip percentage attributes all set to a default value of 0, namely, **Fifteen**, **Twenty**, and **TwentyFive**. Lastly we will update the **index()** to return back to the **index.cshtml** view.

   The **ViewBag** property is automatically available to controllers and views. It uses dynamic properties to get and set values.

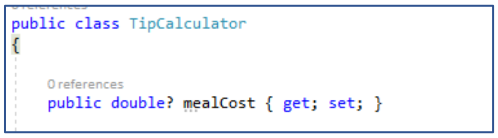


```
public class HomeController : Controller
{
    [HttpGet]
    0 references
    public IActionResult Index()
    {
        ViewBag.Fifteen = 0;
        ViewBag.Twenty = 0;
        ViewBag.TwentyFive = 0;
        return View();
    }
}
```

Refactor the Tip Calculator Model

1. Refactor the Tip Calculator Model, to include a meal cost attribute, an attribute that holds the principal cost of the meal, to which the application will be calculating the tip amount for.

   The data type for meal cost should be a **double** (nullable):

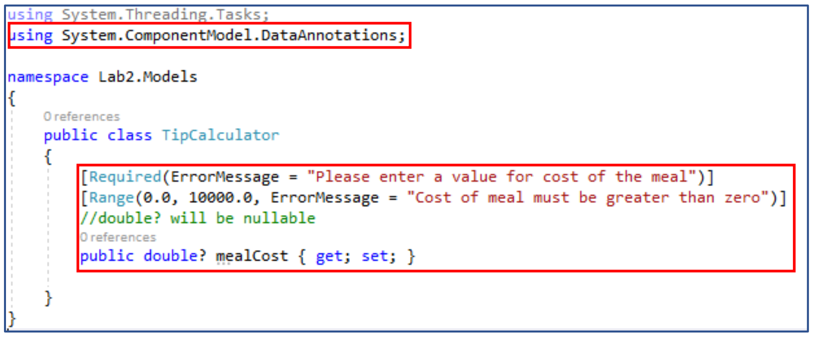


```
public class TipCalculator
{

    0 references
    public double? mealCost { get; set; }
```

   We also want to create the getter (accessor) and setter (mutator) for the attribute so the view layer can call these to get/set the value respectively.

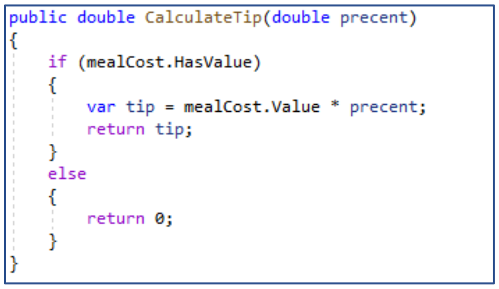
2. We will need to specify some validation on this mealCost parameter:
   a. The cost of the meal (mealCost) is a **mandatory/required** field, necessary if we desire the application to calculate a tip amount
   b. The value of mealCost must be between **$0** and **$10,000.00**

   To set the validation (**Required** and **Range**) we will need to include the **DataAnnotations** library



```
using System.Threading.Tasks;
using System.ComponentModel.DataAnnotations;

namespace Lab2.Models
{
    0 references
    public class TipCalculator
    {
        [Required(ErrorMessage = "Please enter a value for cost of the meal")]
        [Range(0.0, 10000.0, ErrorMessage = "Cost of meal must be greater than zero")]
        //double? will be nullable
        0 references
        public double? mealCost { get; set; }

    }
}
```

3. We can further refactor the model to include a method that does the actual calculation of determining the tip amount of the **mealCost** based on the percent passed as a parameter via the users (Tip Calculators) view form.



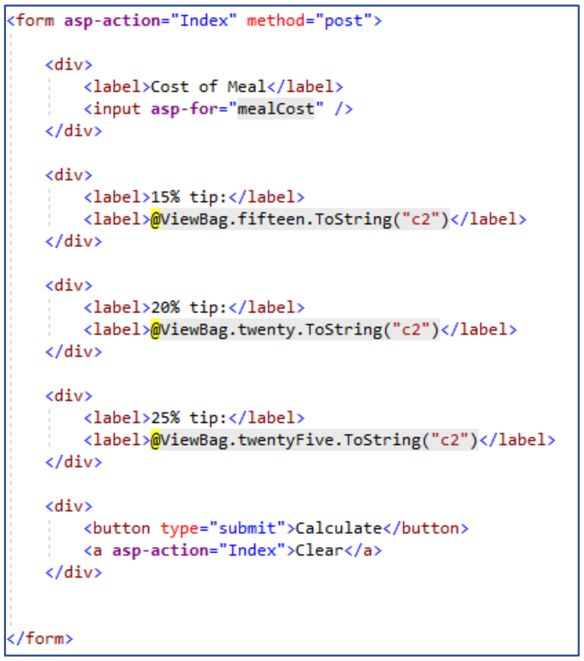
```
public double CalculateTip(double precent)
{
    if (mealCost.HasValue)
    {
        var tip = mealCost.Value * precent;
        return tip;
    }
    else
    {
        return 0;
    }
}
```

Refactor the Tip Calculator View

1. Lets refactor the **index.cshtml** view to bind the index form parameters with the variables we will be using in the model and controller for calculating the tip amount.

   We will…

   - Bind the form to the **index()** method using **HTTP Post** (next section below)
   - Bind the view to the model using **@TipCalculator**
   - Display the values for the calculated percentages (**fifteen**, **twenty**, **twentyFive**)
   - Convert the calculated values to **currency**, 2 decimal places
   - Bind the form "**clear**" action to invoke the index method on the controller to re-calibrate all values to their original state.



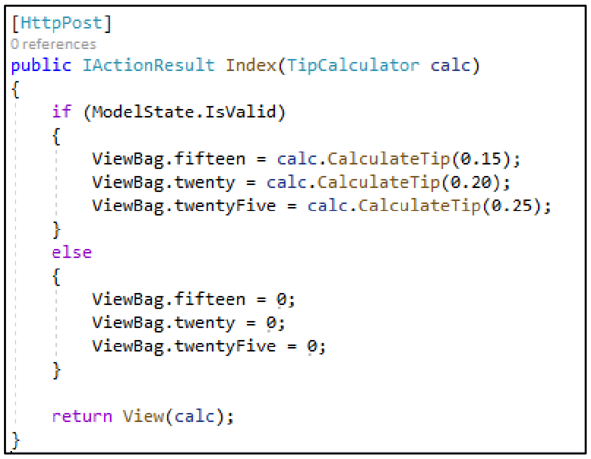
```
<form asp-action="Index" method="post">

    <div>
        <label>Cost of Meal</label>
        <input asp-for="mealCost" />
    </div>

    <div>
        <label>15% tip:</label>
        <label>@ViewBag.fifteen.ToString("c2")</label>
    </div>

    <div>
        <label>20% tip:</label>
        <label>@ViewBag.twenty.ToString("c2")</label>
    </div>

    <div>
        <label>25% tip:</label>
        <label>@ViewBag.twentyFive.ToString("c2")</label>
    </div>

    <div>
        <button type="submit">Calculate</button>
        <a asp-action="Index">Clear</a>
    </div>


</form>
```

2. Add an error validation summary to the page that display a list of errors that occurred during validation, this can be added where ever desired, though ideally above the form.

```
<div asp-validation-summary="All"></div>
```

Add Index() HTTP Post method

1. Create a second method in the controller that accept the **TipCalculator** model and updates the calculated values for **fifteen**, **twenty** and **twentyFive** based the **meal cost** stored in the model.



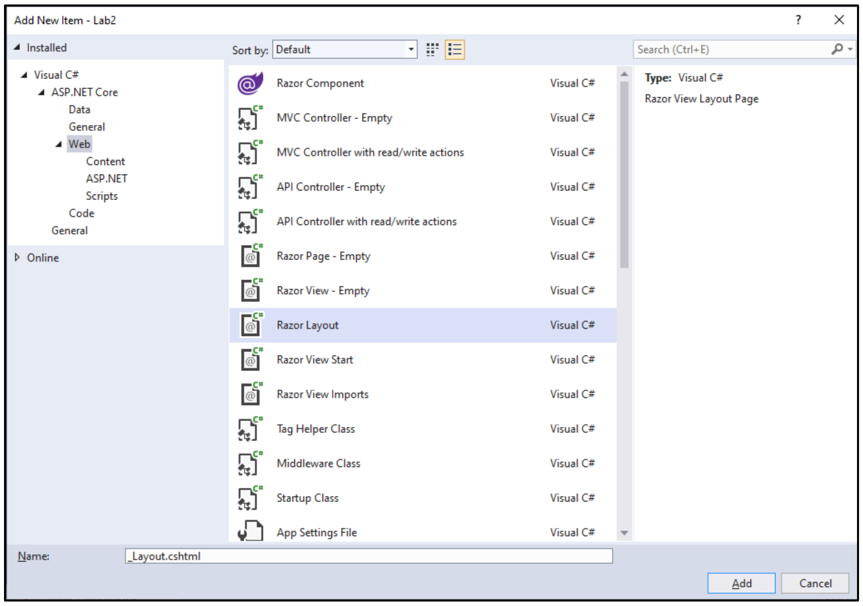
```
[HttpPost]
0 references
public IActionResult Index(TipCalculator calc)
{
    if (ModelState.IsValid)
    {
        ViewBag.fifteen = calc.CalculateTip(0.15);
        ViewBag.twenty = calc.CalculateTip(0.20);
        ViewBag.twentyFive = calc.CalculateTip(0.25);
    }
    else
    {
        ViewBag.fifteen = 0;
        ViewBag.twenty = 0;
        ViewBag.twentyFive = 0;
    }

    return View(calc);
}
```

Create Shared Layout

1. Navigate to the Solution Explorer → Shared folder and delete any auto-generated files within.

2. Right click on the Views/Shared folder and select AddNew Item

3. Select the ASP.Net Core → Web category, select Razor Layout item, and click the Add button



Add New Item - Lab2

Installed
- Visual C#
  - ASP.NET Core
    - Data
    - General
    - Web
      - Content
      - ASP.NET
      - Scripts
    - Code
  - General
- Online

Sort by: Default

Search (Ctrl+E)

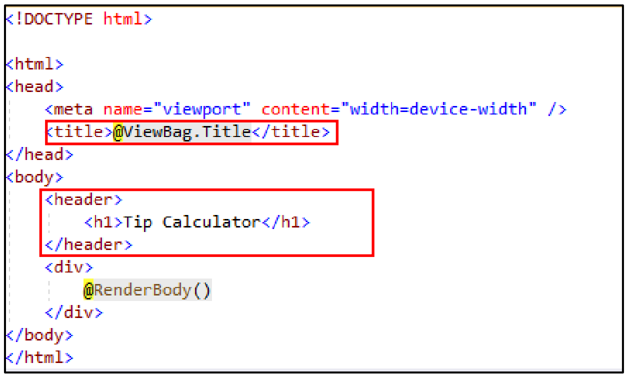
Razor Component	Visual C#
MVC Controller - Empty	Visual C#
MVC Controller with read/write actions	Visual C#
API Controller - Empty	Visual C#
API Controller with read/write actions	Visual C#
Razor Page - Empty	Visual C#
Razor View - Empty	Visual C#
Razor Layout	Visual C#
Razor View Start	Visual C#
Razor View Imports	Visual C#
Tag Helper Class	Visual C#
Middleware Class	Visual C#
Startup Class	Visual C#
App Settings File	Visual C#

Type: Visual C#
Razor View Layout Page

Name: _Layout.cshtml

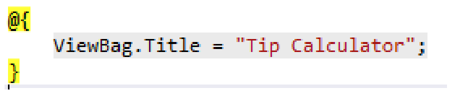
The layout generated will allow us to define a consistent layout structure for our application.

4. Lets update the _Layout.cshtml to include a h1 header title.



```
<!DOCTYPE html>

<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>@ViewBag.Title</title>
</head>
<body>
    <header>
        <h1>Tip Calculator</h1>
    </header>
    <div>
        @RenderBody()
    </div>
</body>
</html>
```

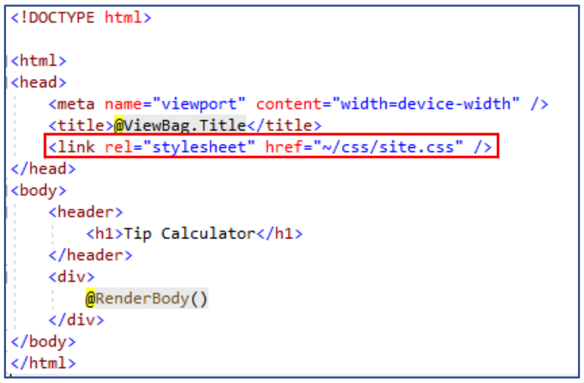
5. Set the **ViewBag.Title** in the **index.cshtml** page



```
@{
    ViewBag.Title = "Tip Calculator";
}
```

Create CSS Styling for Layout (Optional)

1. To create and add any css styling to your layout, you can drop the css in the wwwroot/css directory in your Solution Explorer and include it in your layout.

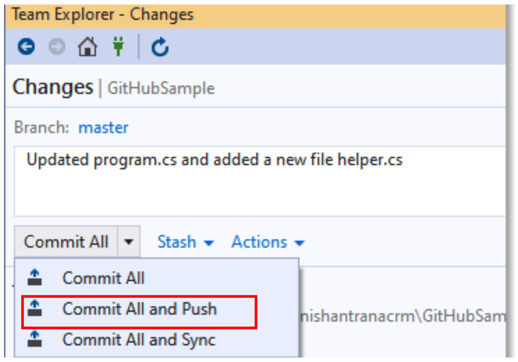


```
<!DOCTYPE html>

<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>@ViewBag.Title</title>
    <link rel="stylesheet" href="~/css/site.css" />
</head>
<body>
    <header>
        <h1>Tip Calculator</h1>
    </header>
    <div>
        @RenderBody()
    </div>
</body>
</html>
```

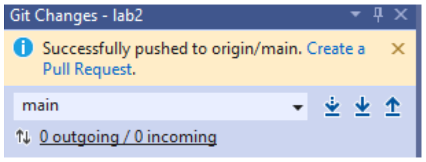
Run and test the Application, then Commit

1. Run the application and test to make sure it functions as expected (correct any errors).

2. If all is well, commit the code to your code GitHub repository. To commit the code, navigate to the git changes tab, and select **commit all and push**



Updated program.cs and added a new file helper.cs



Successfully pushed to origin/main. Create a Pull Request.