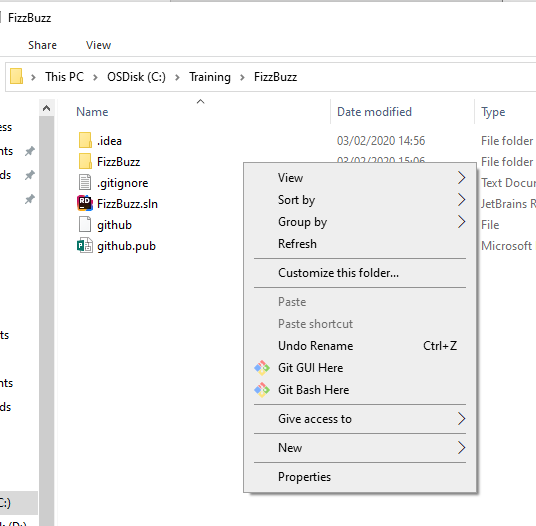
**Github**

**To open GitHub Bash , you can right click from your folder**

****

1. Create a github account
2. Create your github repository
3. Generate the Github SSH keys
4. Open Git Bash.
5. Paste the text below, substituting in your GitHub Enterprise email address.
   1. $ ssh-keygen -t rsa -b 4096 -C "*your\_email@example.com*"
6. This generates a public/private rsa key pair in /c/Users/HemNal/.ssh/id\_rsa
7. Press enter when you are prompted to Enter a file in which to save the key
8. Type enter when prompted to enter a secure passphrase.
9. Next add your SSH key to the ssh-agent

In your GitBash type the following to ensure the ssh-agent is running

HemNal@CAIMAN MINGW64 /c/Training/FizzBuzz (master)

$ eval $(ssh-agent -s)

Agent pid 140

1. Adding a new SSH key to your GitHub account

Type the following in GitBash to copy the SSH key into the clipboard

$ clip < ~/.ssh/id\_rsa.pub

Go to GitHub your account settings and paste the SSH key

**To add your program to github**

Create your project

Right click on your project folder and open github bash

$ git init This will create a git directory in your folder

C:/Training/FizzBuzz/.git/

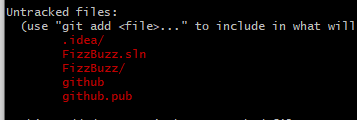
$ ls -a will list the folder

./ ../ .git/ .idea/ FizzBuzz/ FizzBuzz.sln

The command below links the local git to the remote git

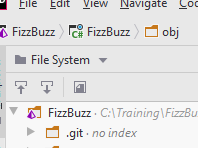
$ git remote add origin [git@github.com:hnalabanda/FizzBuzz.git](mailto:git@github.com:hnalabanda/FizzBuzz.git)

$git status will display the status of the files – shows all the un committed files

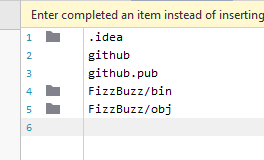


Create a gitignore file to ignore files that need not be added

On Rider , select File System

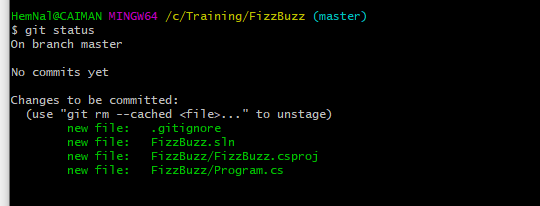


Create a .gitignore file and all the files that you want to ignore



Git add . adds the files to the staging

Now if you say $git status it shows the files to be committed



Next commit your files

git commit -m "Informative commit message goes here"

$git push will push to master

The first time you will get the error

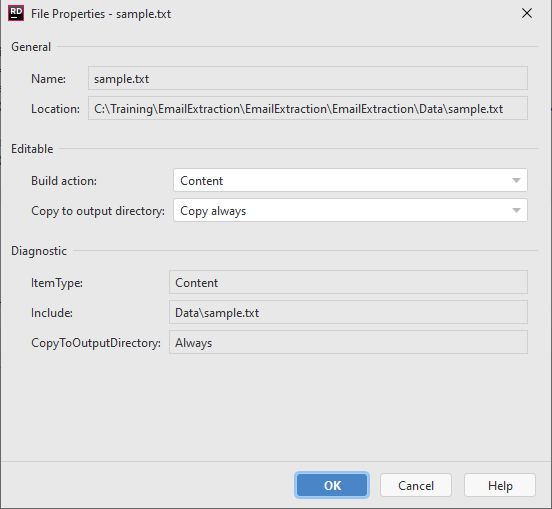
git push --set-upstream origin master , This is because there is no master branch in github yet

Next do git push --set-upstream origin master

Sometimes when you execute programs, some files may be looked for in the bin folder

To get around this :

Right click file , select properties



Select Build action – Content

Copy to output directory : Copy Always

Using GitKraken

Create Repository on Git

Create a solution on Rider pointing to c:\Training\ call solution SupportBank

Open GitKraken and click on init repo and copy the URL from Github and paste

Next go to FileSystem on Rider and create a .gitignore file

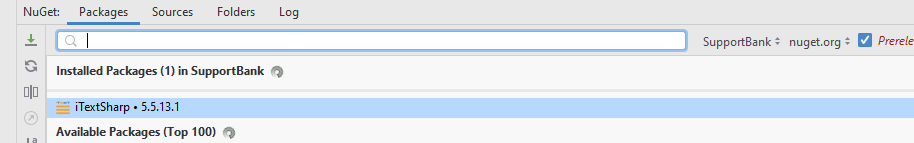
And add the files you do not want to see in GitKraken , Unstaged files list

New Method for classes

var newTransaction= new Transaction();  
newTransaction.TransactionDate = Convert.ToDateTime(values[0]);  
newTransaction.FromName = values[1];  
newTransaction.FromName = values[2];  
newTransaction.FromName = values[3];  
  
  
var newTransaction = new Transaction  
{  
 TransactionDate = DateTime.Now,  
 FromName = "Mike",  
 ToName = "Sasha",  
 FromNameAmount = "15",  
};

Object reference not set to an instance of an object. Is when object is null

To include a package , go to Tools> Manage NuGet Package and enter the package name to search



Instead of

if (readCommand == "All")  
{  
 table=PrintAll(bank);  
}  
else  
{  
 table=PrintAccount(readCommand,bank);  
  
}

# Working with REST APIs

First login into TFL API <https://api-portal.tfl.gov.uk/admin> and register

You will get an email

Go to the site and click on API Credentials and copy the App Id and App Key

And pass it on your api request

<https://api.tfl.gov.uk/StopPoint/Mode/bus/Disruption?app_id=b3be5063&app_key=1f49b0356145d2c72f9c8b638020c602>

#### Setting up our Dev Cert

Next, we’ll need to set up a dev certificate so that we can run the web app over HTTPS.

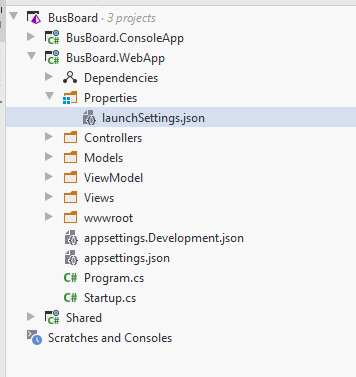
To do this, while in Rider, double press the Crtl key to bring up the 'run anything' window.

Now type dotnet dev-certs https --trust` into the window. This command will tell windows that it’s fine to trust the dev-cert that dotnet is going to use for our project.

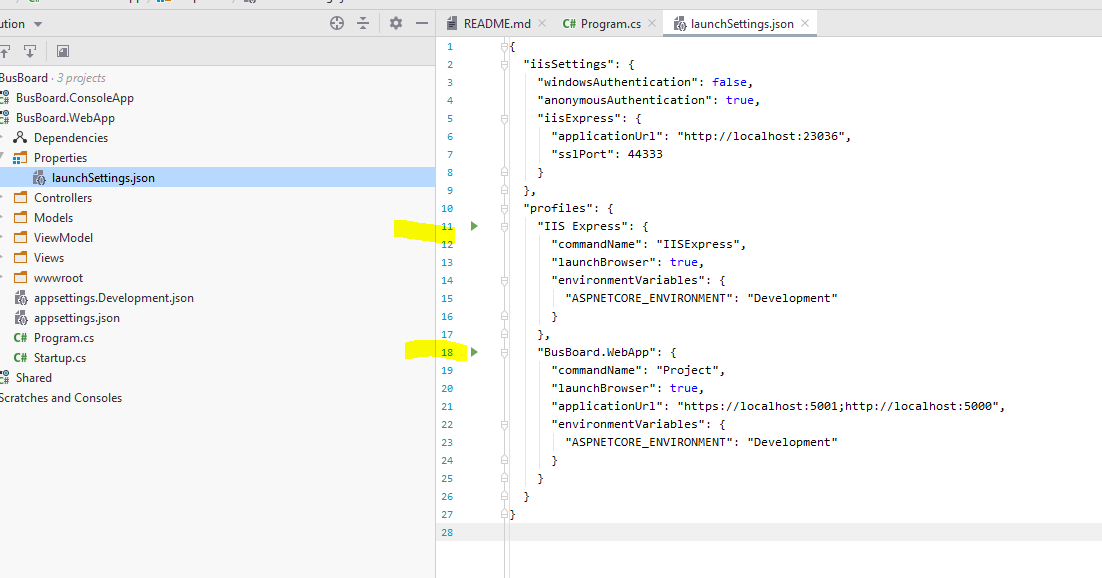
Windows should give you a popup to confirm that is OK.

Once that’s all done, you’ll need to restart your computer before it takes effect.

This is the structure of the web app



The launchSettings.json



MVC

### **Model**

The Model component corresponds to all the data-related logic that the user works with. This can represent either the data that is being transferred between the View and Controller components or any other business logic-related data. For example, a Customer object will retrieve the customer information from the database, manipulate it and update it data back to the database or use it to render data.

### **View**

The View component is used for all the UI logic of the application. For example, the Customer view will include all the UI components such as text boxes, dropdowns, etc. that the final user interacts with.

**MVC Application**

**Create a ASP.NET Core Web Application**

Select an empty template to start with – VS would have already created a StartUp.cs file for you

the **Controller** acts as the middleman - it will combine your **Model** with a **View** and serve the result to the end-user.

**Views**

display the model data

**Controllers**

Classes that handle browser requests. They retrieve model data and call view templates that return a response. The controller handles route data and query-string values, and passes these values to the model.

For example, https://localhost:5001/Home/Privacy has route data of Home (the controller) and Privacy (the action method to call on the home controller).

https://localhost:5001/Movies/Edit/5 is a request to edit the movie with ID=5 using the movie controller.

The MVC pattern helps you create apps that separate the different aspects of the app (input logic, business logic, and UI logic)

The UI logic belongs in the view. Input logic belongs in the controller. Business logic belongs in the model.

For example :

using Microsoft.AspNetCore.Mvc;

using System.Text.Encodings.Web;

namespace MvcMovie.Controllers

{

public class HelloWorldController : Controller

{

//

// GET: /HelloWorld/

public string Index()

{

return View();

}

//

// GET: /HelloWorld/Welcome/

public string Welcome()

{

return "This is the Welcome action method...";

}

}

}

Every public method in a controller is callable as an HTTP endpoint. Such as <https://localhost:5001/HelloWorld>

The MVC Controller will call the **Index()** method , which will try to return the default view by calling the View() method.

**Views:**

View have a folder called Views

It's also customary to have one folder per Controller, so inside the new Views folder, let's create a folder called **Home**, for the HomeController

Folder called Posts for the PostsController

The HomeController’s index method returns a view

### **How does it work?**

Thanks to the default routing mechanisms found in the ASP.NET MVC framework, the root URL is automatically routed to the HomeController's **Index()** method (don't worry about routes just yet, we will talk about them soon). With a call to the View() method, a number of locations are searched to find a View with a matching name, in this case \[project root]\Views\[name of controller]\Index.cshtml. This view is then interpreted (as it may very well contain Razor code) and then returned as output to the browser.

**Models:**

In the MVC architecture, the **Model** is generated by the **Controller** and then passed to the **View**, which outputs the relevant data to the user.

Classes that represent the data of the app. The model classes use validation logic to enforce business rules for that data. Typically, model objects retrieve and store model state in a database.

In a cshtml file

Create a variable called count

To include that within an id put a – in front of the @ because otherwise it is thought to be an @ in an email for example.

<img class="likes-image" src="/Images/dislike.png"/>  
<a id="rem-@count">Click here to remove this post</a>

**React**

**If you have images place it in the folder Public**

If chrome does not refresh , then do Ctrl + Shift + R it will then force page to be reloaded

To use router installreact-router-dom

**If you get this error**

Module not found: Can't resolve 'react-router-dom' in 'C:\Training\myface-react-app\myface-react\src'

This code returned error:

*useEffect*(  
 () =>   
 *fetch*(`https://localhost:5001/users/${id}`)  
 .then(result => result.json())  
 .then(data => setAllPosts(data))  
 , []  
);

This was fixed by

*useEffect*(  
 () => {  
 *fetch*(`https://localhost:5001/users/${id}`)  
 .then(result => result.json())  
 .then(data => setAllPosts(data))  
 } , []  
);

**Css Naming convention**

Use dashes instead of camel case profile-class instead of profileClass

**Razor**

You can still use Request.QueryString[“test”]

**HTML Encoded**

Razor expression is HTML encoded. But if you want raw html , use the **html.Raw**

@{  
    var helloWorld = "<b>Hello, world!</b>";  
}  
<p>@helloWorld</p>  
<p>@Html.Raw(helloWorld)</p>

This will output <b>Hello, world!</b>

**Calculations**

@(35+5)

## **Multi-statement Razor blocks**

Put the @ followed by curly bracket

@{  
    var sum = 32 + 10;  
    var greeting = "Hello, world!";  
    var text = "";  
    for(int i = 0; i < 3; i++)  
    {   
 text += greeting + " The result is: " + sum + "\n";  
    }  
}

## **Html tags inside Razor**

You can have html tags inside a Razor

You can also have plain text , you can use @: or <text> tags

@{  
    var helloWorld = "This is a code block...";  
      
   
    @:This is plain text!  
    <br><br>  
    <text>This is plain text as well, and we can  
    even span multiple lines, if needed!</text>  
}

## **Commenting inside Razor**

@{  
    @\*  
 Here's a Razor server-side comment  
    \*@  
  
    // C# style single-line comment  
  
    /\*   
 C# style multiline comment  
 It can span multiple lines  
    \*/      
}

## **Variables in Razor**

Variables can be declared with the @

@if(DateTime.Now.Year == 2042)  
{  
    <span>The year 2042 has finally arrived!</span>  
}

Similarly you can have loops

**For Loops**

<ul>  
 @for (int i = 0; i < names.Count; i++)  
 {  
 <li>@names[i]</li>  
 }  
</ul>

**For Each**

<ul>  
 @foreach (string name in names)  
 {  
 <li>@name</li>  
 }  
</ul>

**While Loop**

<ul>  
 @{   
 int counter = 0;  
 }  
 @while(counter < names.Count)  
 {  
 <li>@names[counter++]</li>  
 }  
</ul>

**Do while**

<ul>  
 @{  
 counter = 0;  
 }  
 @do  
 {  
 <li>@names[counter++]</li>  
 } while (counter < names.Count);  
</ul>

**Break**

@if(i >= 2)    
 {    
 <li>...and so on</li>    
 break;    
 }

**Switch**

@switch()

{case something:

<span>dfd</span>

Break;

Case

}

**Functions in Razor**

Let’s import a model which is a list of WebUser

@model List<HelloWorld.Models.WebUser>

@{

Void RenderUserInfo(HelloWorld.Models.WebUser)

{<div>

@user.LastName

</div>

}

//have a foreach loop to go through all the webusers

Foreach(var user in Model.OrderBy(x=>x.FirstName))

{RenderUserInfo(user)

}

}

**Functions that return value can be written like**

The universe is **@GetAnswer**