**Introduction**

Project 3, which will also take the place of the final exam, will produce a website like the one you built for Project 2, with some differences:

* The website will be built using Microsoft Visual Studio to construct an ASP.NET Core MVC website.
* We will leverage the following technologies:
  + Dependency injection
  + jQuery and jQueryUI
  + Creating “views” to display in the browser by “Mixing” C# and html on the “.cshtml” view pages
  + C# collections and LINQ
  + HTTP request / response using “HttpClient” to retrieve data from the “ist.rit.edu/api” endpoints
  + .NET Core async / await for HTTP requests
  + Newtonsoft.Json for processing data returned from the “ist.rit.edu/api” endpoints.
* You will be required to retrieve data from the following “ist.rit.edu/api” endpoints
  + about – there is text at this endpoint that will be useful for your “home page,” more on this later.
  + people/faculty
  + people/staff
  + degrees
    - undergraduate to include: degreeName, title, description, concentration
    - graduate to include: degreeName, title, description, concentration
  + minors (for undergraduate) to include: name, title, description, courses
  + employment/employmentTable – data to be displayed in a table, e.g. jsGrid will work

Completing all the above will guarantee a grade of “B.” I will have suggestions on how to enhance to project to get a higher grade.

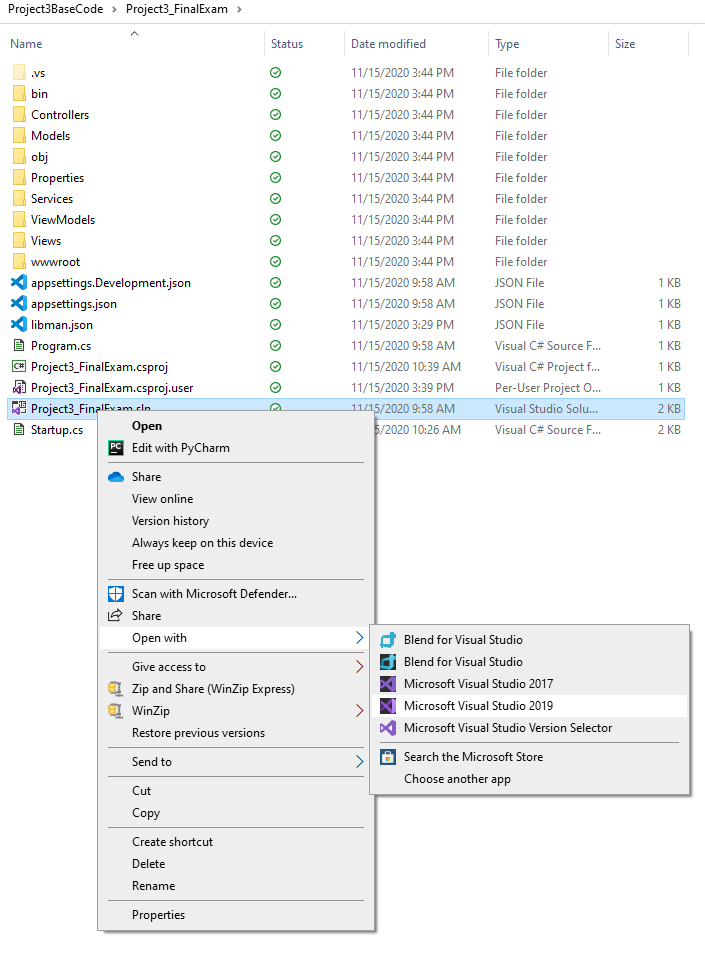
To get you started on all of this I have created a “base” solution containing a some of the requirements. Once you understand the “base” solution you should be able to easily enhance it to build out the full solution.

In this lecture we will start with the “base” solution and do the following:

* modify the “Layout.cshtml” Shared view to
  + expand the menu at the top of each web page
  + change the web page name
  + add menu items to the menu at the top of each web page
  + modify the copyright notice that appears at the bottom of each web page
  + add the jQuery / jQueryUI libraries to the solution
  + work with C# and HTML elements on the .cshtml view pages.

**Getting Started**

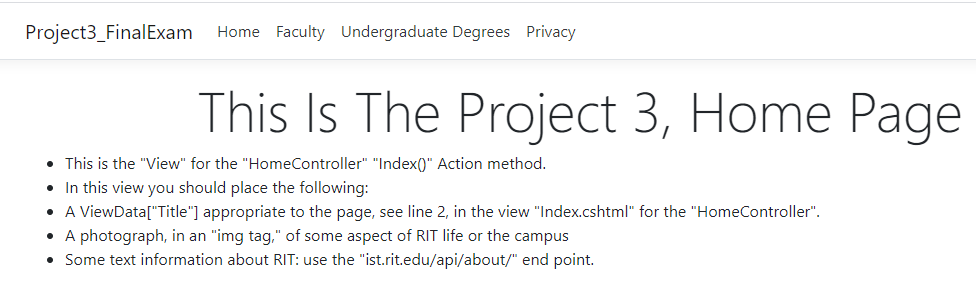
From the folder containing the content for this lecture open the folder “Project3BaseCode” and then open the folder “Project3\_FinalExam.” In “Project3\_FinalExam” find the file “Project3\_FinalExam.sln” and right click on it and click on it. From the dropdown menu that appears select “Open with” and from the next dropdown menu select “Microsoft Visual Studio 2019.”



This will open MS Visual Studio 2019 on your screen showing the “Projectr3\_FinalExam” base solution. Let us start out by running it. At the top of the screen click on the green “play” button to launch the website.



This will bring up the “default” home page for the website:



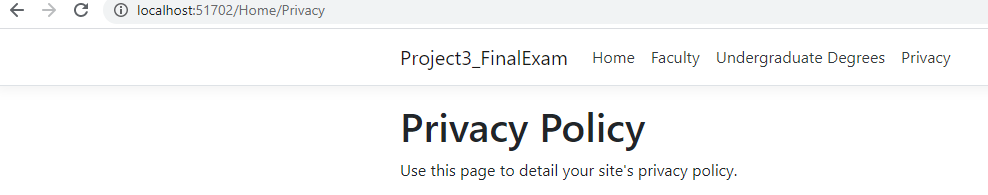
On the home page are very specific instructions on how you should modify it.

Along the top of the page we have:

* the “title” of the website – it is now set to the default value which is the name of the solution: “Project3\_FinalExam”
* to the right of the website title are the current menu items:
  + “Home”
  + “Faculty”
  + “Undergraduate Degrees”
  + “Privacy”

At the bottom of the page is the copyright notice which is the current year followed by the default copyright owner which, again, is the name of the solution: “Project3\_FinalExam.” To the right of the copyright notice is a link to the privacy notice for the website.

Clicking on the link to the privacy notice brings up the following web page:

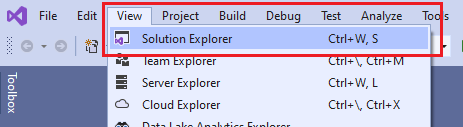


Notice that the URL ends with the “Privacy” action of the Home controller.

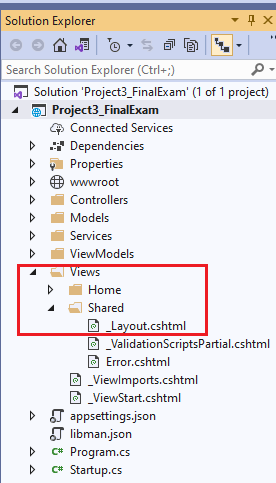
**Changing the “\_Layout.cshtml” File**

The “\_Layout.cshtml” file contains the HTML elements which set the values that appear on all web pages on our site. Each view is displayed withing the layout provided by this page. It is called a shared view because it is shared by all views in the stite.

In the “Solution Explorer,” should be showing on the right-hand side of the screen, Visual Studio for MacOS will display on the left-hand side; if it is not showing go the menu along the top of the page and click on “View” and from the dropdown menu click on “Solution Explorer.”



Go down the solution explorer until you find the “Views” folder. Expand that folder and then expand the “Shared” folder. Within the “Shared” folder double click on “\_Layout.cshtml.”



The “\_Layout.cshtml” page will now appear and is available to edit.

The first item we will change is the “ViewData” array “Title” entry which is the “Title” for the website that can be accessed anywhere in the website. We want to change the website “Title” from “Project3\_FinalExam” to “Welcome to the iSchool Website.”

If you look at line 6, you will find:



Simply change the title to “Welcome to the iSchool Website.”



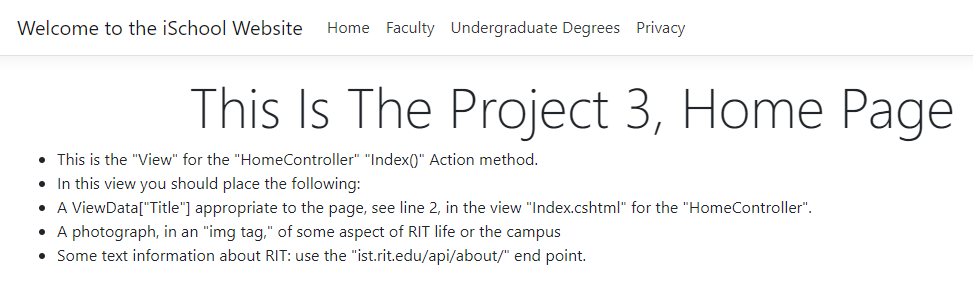
Next, we want to change the name of the website displayed on the left-hand side of the menu. We will also change this to “Welcome to the iSchool Website.” Go to line 14:



Change this line as follows:



If we start up our website, we now have:



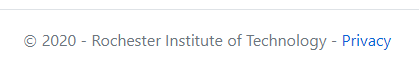
The other item we want to change on the layout page is the copyright notice, see line 46:



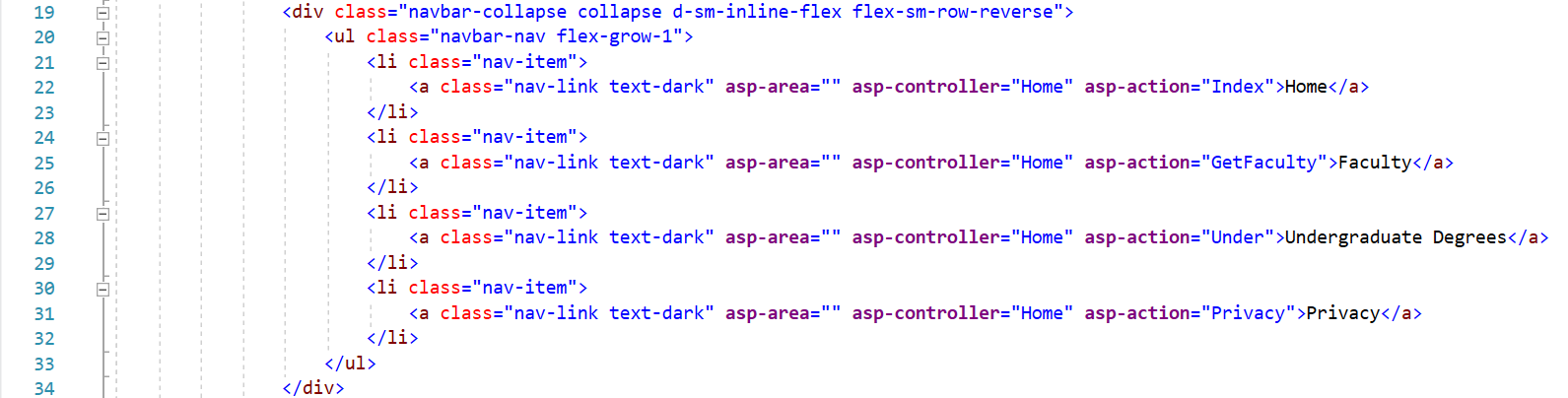
We will make the “Rochester Institute of Technology” the copyright owner. Change line 46, accordingly:



If we start up the website, we now have the following copyright notice at the bottom of the page:



Before leaving the layout page I want to draw your attention to the following code:



The section of code above is where the menu along the top of the web pages is built. Each menu item is a “list item” in an “unordered list.” The following portion of the “list item” for the “Home” Controller “Index” action item will help explain what is going on:



For a menu item the first segment above, “asp-controller,” indicates that the target of the menu item is the “Home” controller. The second segment, “asp-action,” indicates that the target is the “Index” action item within the “Home” controller. The “text content” “Home” is the menu option that will displayed at the top of the page. Clicking on “Home” will launch the “Index” action within the “Home” controller.

As we add new Action items within the “Home” controller we will return here to update the website menu.

**Reviewing What We Have Done**

Note – the base code is slightly different than the solution we started in “Lecture 25,” with respect to the views, in particular the “\_Layout.cshtml” Shared view and the view for the “Faculty.” There is also a small change to the Action items in the “Home Controller.”

The change to the views is the result of my updating to the very latest version of “Visual Studio Community Edition” which brought about some low-level changes in the way the views, in particular Bootstrap, are rendered. Please make sure that you are running the very latest version of Visual Studio.

In the “Home Controller” I have made “GetFaculty” an Action item. Originally, we were using the “Index” Action item to render the Faulty. However, I want the project to use a “home page” so rendering the “home page” is now the responsibility of the “index” Action item. Look at the “home page” when you start the project up – you need to create content there.

Use the code in Project3, solution in the “Project3BaseCode” folder as your starting point. If you have already created code for the assignment to add the “Undergraduate Degree” to the solution, you should be able to copy it over with no problems.

We started working on “Project 3,” in “Lecture24\_DependencyInjection” In that lecture we took a look at the basic layout of “Project 3,” and using the “people/faculty” ist.rit.edu/api endpoint developed a “Home Controller” Action Item to present a grid of images of the faculty on a web page.

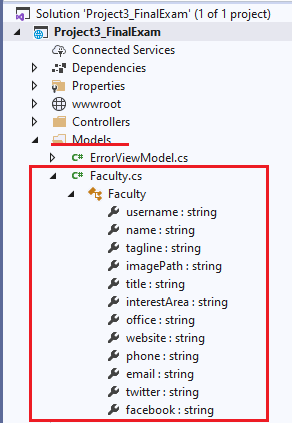
The key ideas of that work were understanding how to develop “loosely coupled” code by using “dependency injection” and implement asynchronous calls to “ist.rit.edu/api” endpoints using the “DOT NET Core” async / await pattern.

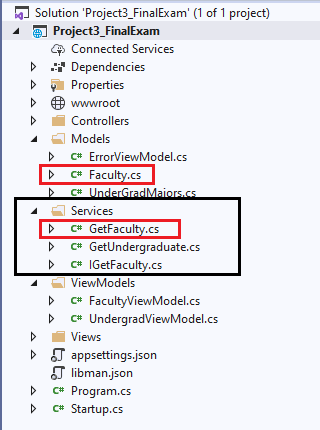
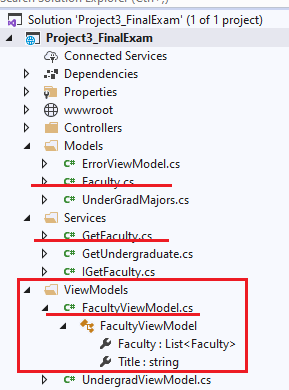
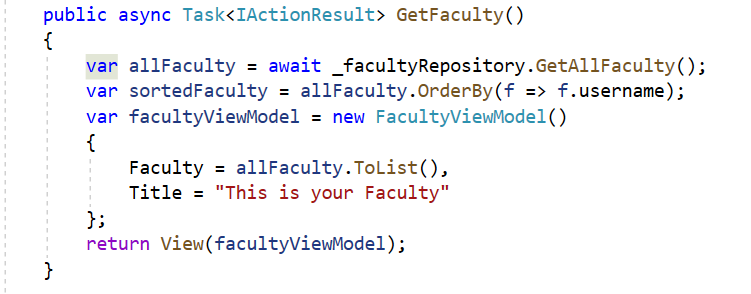
We also saw that the methods and constructs we created to present the faculty image grid could be applied to almost all the other api endpoints to display the information obtained from them.

*Continued next page.*

The procedure we followed was:

* We added a “Models” folder to the “Project3\_FinalExam” solution which contains a class for each api endpoint we are retrieving. The class contains all the properties and data types of the

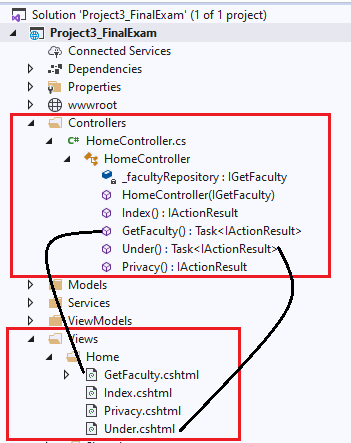
retrieved data. For example, in the “Models” folder we have a class for “Faculty:  


* To retrieve data from the api endpoints we needed to provide code to make the actual HTTP requests to the ist.rit.edu/api. The procedure we followed was to first create a “Services” folder in the solution in which we then created the endpoint request code for each class in the “Models” folder.  
  
* In order to pass the data returned by the, in this example, the “GetFaculty.cs” service to the view in which it will be displayed we created a “VewModels” folder in which we created a “View Model” for each view to be rendered by the “Home Controller.”  
  
* Finally we “tied it all together” by creating a “GetFaculty()” ActionResult in the Home Controller to initiate the api endpoint retrieval and pass it to the “View” which displays it on the web page:  
    
  Please note that the above code uses “dependency injection” to loosely couple the “GetFaculty()” routine from the actual implementation of the “GetFaculty()” service in the “Services” folder. You will have to do this for each Action Item in the Home Controller. How to do this is explained in detail in “Lecture24\_DependencyInjection” so it will not be repeated here. In fact, you had an assignment from that lecture to create a get undergraduate Action result along with the appropriate Model, Service and ViewModel using “Dependency Injection.”  
  The pattern of “Model,” “Service,” “ViewModel,” “Controller IAction” implement “Dependency Injection” for the “service” will be repeated over and over for each endpoint. Once you understand and master how to do it for one api endpoint you will be well on your way to building Project 3.
* The only difficult part once you have mastered the above is the creation of the views to display the data. In “Lecture 24” we went over how to create a view starting in the Home Controller IAction which will present the view. To get you started I have provided a view for the “GetFaculty() and “Under()” IAction items in the Home Controller. Note – “dependency injection” is not implemented for the “Under()” action item, that was part of the assignment from “Lecture 24.” ***Any Home Controller IAction item and associated service from the Service folder which does not utilize “dependency injection” will mean a grade of less than “B.”***

**Creating Views**

In the base code I have provided two “basic views” for the data, see the “Views” folder in the solution.

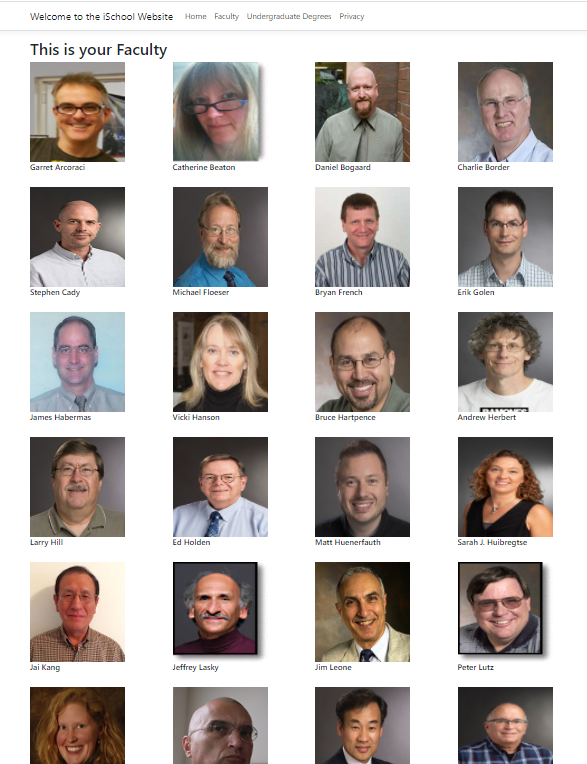
* “GetFaculty.cshtml”
* “Under.cshtml”



*Continued next page*

**GetFaculty.cshtml**

If we run the website and click on “Faculty” in the menu at the top of the page, we see; this is a partial screen shot:

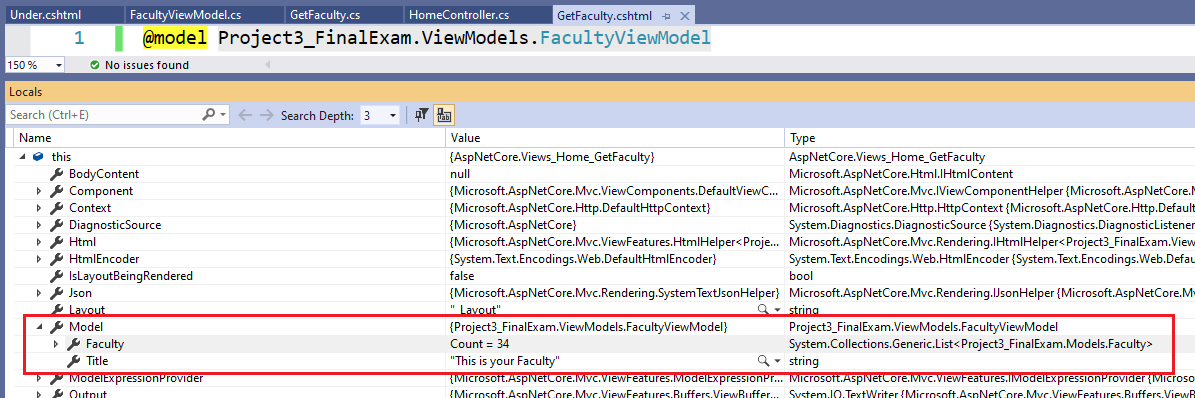


In the code I originally used in “Lecture 23” the faculty pictures were randomly sized and not arranged in an orderly manner. I fixed that up with some applied bootstrap. Let us examine the code on the view page.

The first line of the view page is:



The model is the “FacultyViewModel” defined in the “VewModels” folder and used by the “GetFaculty()” IAction in the HomeController to pass the data to be displayed in the view. If we put a breakpoint after line 1, we can see the model data passed to the view:



We create code in the view to display this data in any way we wish. To create the display shown above I wrote the following code:

*Continued next page*

**

Remember that any code that is preceded by @ indicates that what follows is interpreted as C# code. Code enclosed in

@{

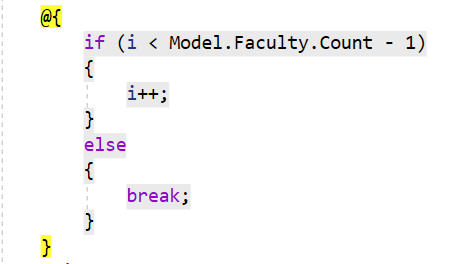
}

Indicates that what is inside the curly braces is C# code.

On line 3, of the above code we create an <h2> element with the text content being the “Title” property from the “FacultyViewModel.” Notice that we can create an HTML element and insert data using “C#” code. As seen in the screen shot for this view, line 3, will produce the heading “This is your Faculty.”

On line 5, is the start of a “for” loop, notice the “for” is preceded by a @. The “for” loop iterates over each “Faculty” item passed to the view in “Model.Faculty,” the 34, element array we have been working with for a while.

What is going on in this loop is the following:

* We create a <div> of class “row,” a Bootstrap row.
* We then proceed to create up to four Bootsrap <div class=”col-sm-3> columns each one of which will hold a faculty picture in the <div class ‘thumbnail”> below which is displayed the faculty members name in the <div class=”caption”>
* Notice that in the <div class=thumbnail”> I am creating what will be a unique id:  
    
  Similar to what we did in JavaScript I am creating a unique id by concatenating the character string “myDiv:” with the current value of “i” used for iteration in the loop. C# has a “String” method “.Concat” which performs concatenation of the strings enclosed in parenthesis. When I = 0, the id which will be created is “myDiv:0” Unlike JavaScript C# is type safe so it will “coerce” the integer i to a string, rather we have to use an explicit conversion the String .ToString() method. The reason for using the “:” in the id will be come apparent later when we enhance the view using jQuery and jQueryUI.  
    
  Because we are creating four columns of thumbnails on each iteration of the loop we have to explicitly increment the value of i after each thumbnail is created to both insure that each thumbnail contains a unique id and that at the finish of creating the four columns the value of i is appropriately set for the next iteration of the loop. In order to prevent i from being set to a value greater than “Model.Faculty.Count” when it is incremented, I put in the C# code block:  
  

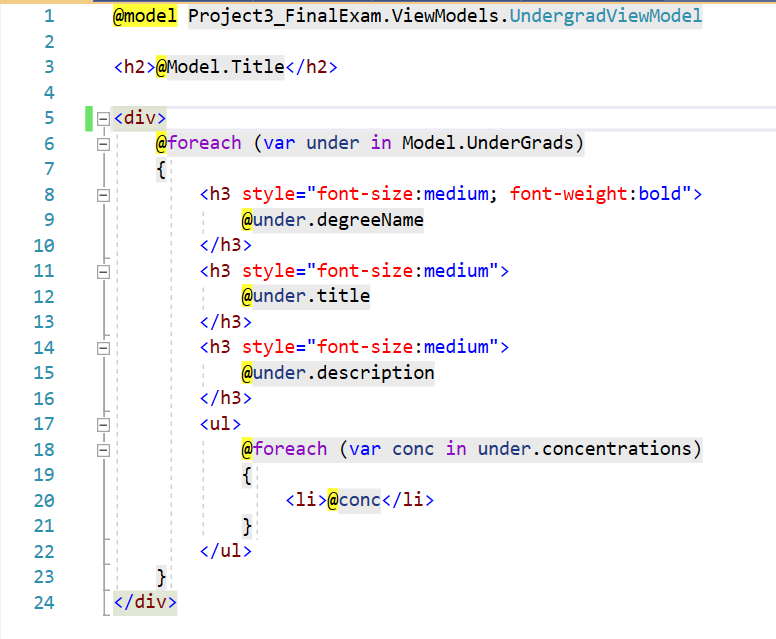
If it is “safe” to increment i, we do so. If not, we issue a “break” and immediately exit the “for” loop which then renders the page.

The <div>&nbsp;</div> on lines 78, and 79, are to provide some spacing between a faculty members name and the next row of images.

*Continued next page*

**Under.cshtml**

The view to display the Undergraduate degrees is a simpler one.

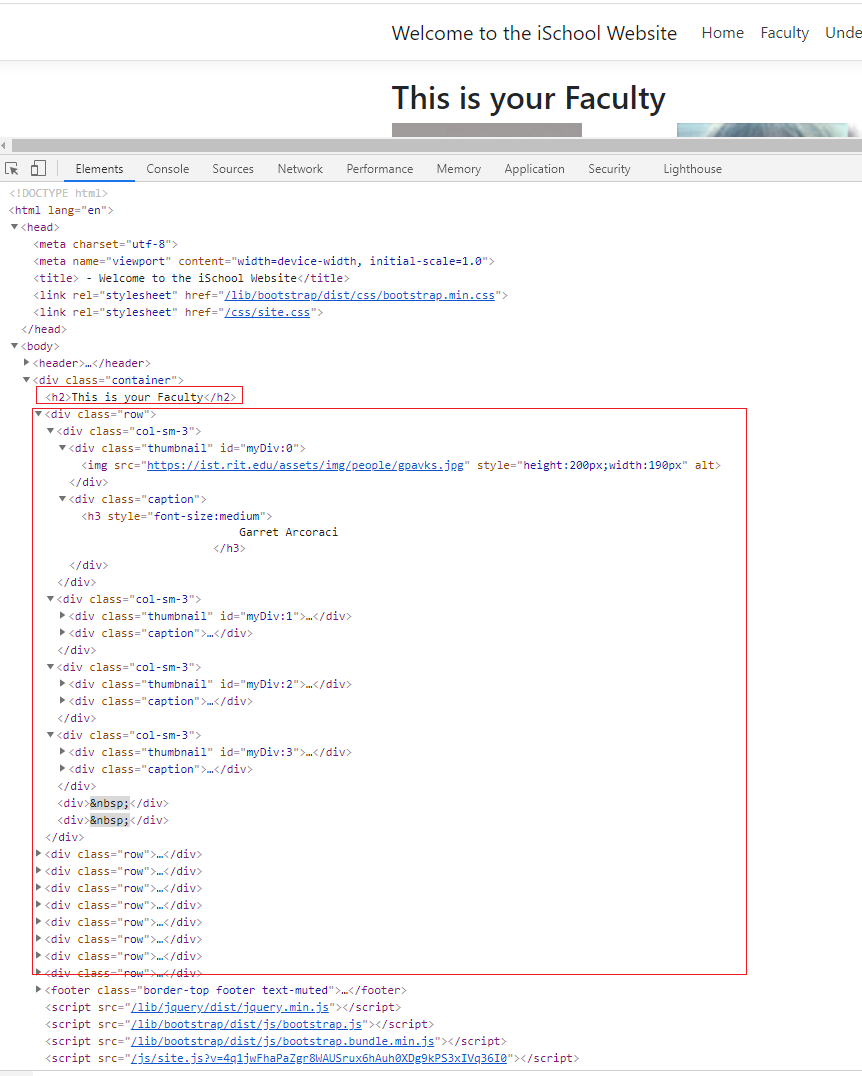


Using the information provided on how C# and HTML interact in the “GetFaculty.cshtml” view, it should be simple to work through what is happening.

What is interesting is to look at a .cshtml page when it is rendered to the browser.

*Continued next page*

Below is a partial screen shot of the “Elements” tab from “Developers tools” for the “GetFaculty.cshtml” page.



Notice that there is no C# code visible. The C# code generates the appropriate HTML tags and their content but is not actually rendered with the page in the browser.

**Enhancing Views With jQuery and jQuery UI**

We will now explore how to utilize jQuery and jQueryUI in our project.

In the Solution Explorer expand “wwwroot.” Remember that this is the folder used by the “MVC Middleware” to retrieve “static data.”

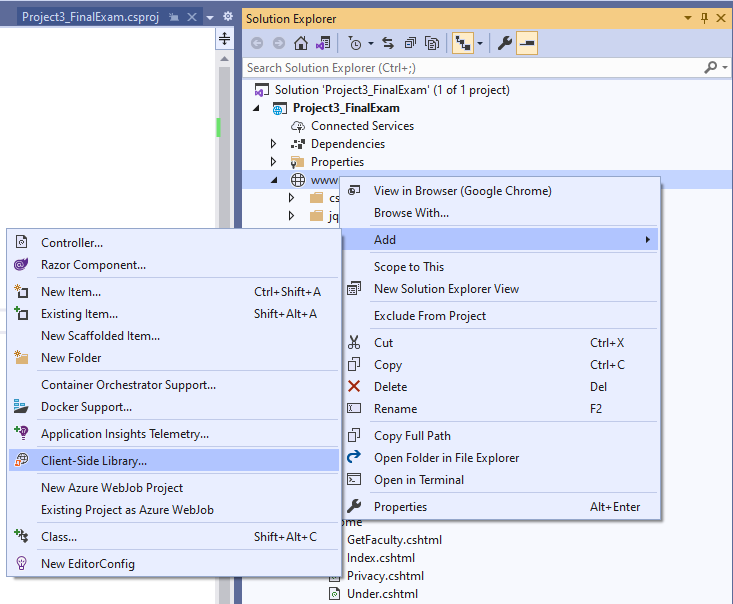
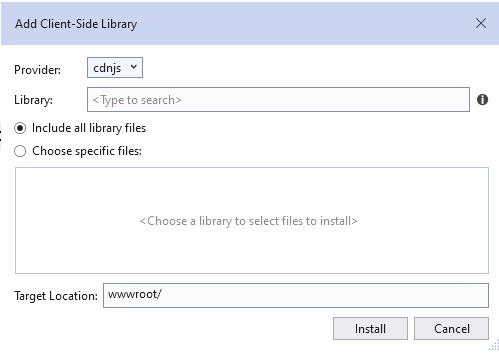
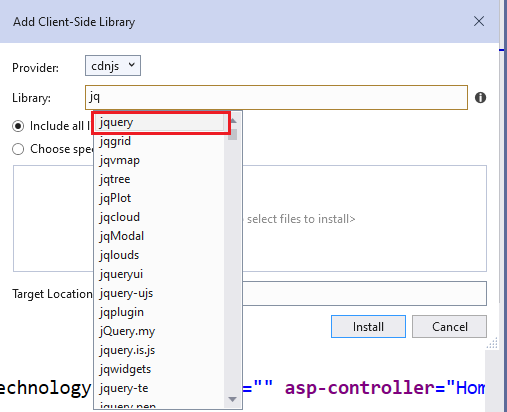
****

As you can see when we create an ASP. NET Core solution with Visual Studio the “jQuery” is automatically incorporated. It is available for inclusion in any code created in the solution using the usual “<script>” tags. However, what is not included is the “jQueryUI” library.

To utilize jQuery and jQueryUI we will add them to the “wwwroot” folder from which they can be accessed anywhere in our solution.

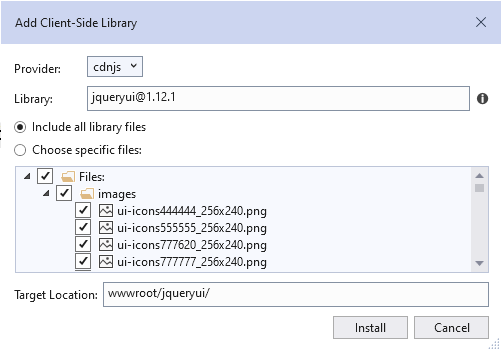
*Continued next page.*

We can add jQuery to our solution as follows:

* With your mouse click on the arrow to the left of  
    
  in the Solution Explorer
* Then right click on “wwwroot” and from the dropdown menu that appears click on “Add” and then click on “Client-Side Library…”  
  
* The “Add Client-Side Library” panel will appear  
  
* On the line “Library:” start typing “jqu” you will see a list of “auto-complete” choices appear below the textbox:  
    
    
  Click on “jquery” and then “Install”

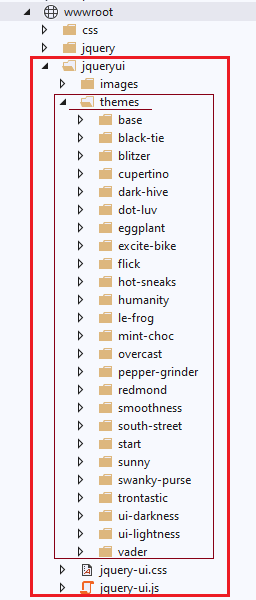
*Continued next page*

Follow the same procedure to install jQueryUI



*Continued next page.*

We now have jQueryUI available, notice all the themes included:

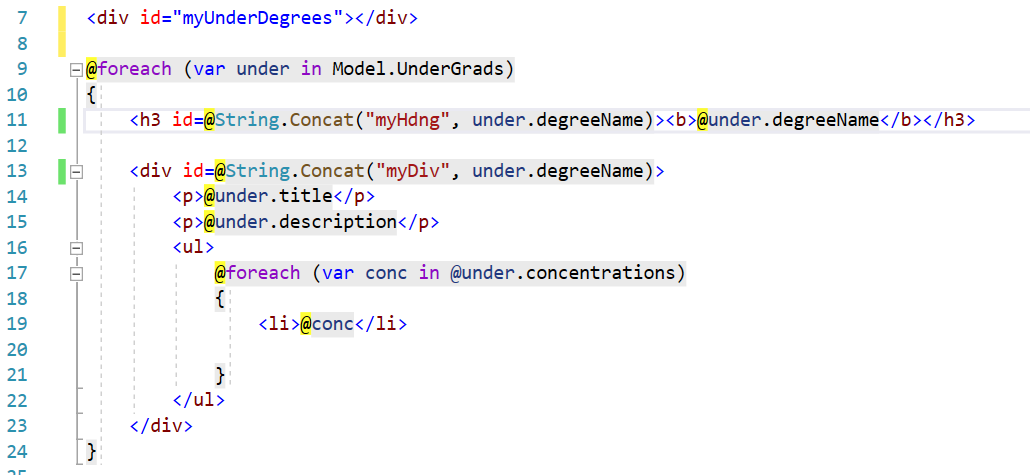


With both jQuery and jQueryUI available let us enhance our two views to make use of jQueryUI widgets. We will start with the simpler task of rebuilding the “Under.cshtml” view to use a jQueryUI “Accordion” to display the majors just as in Project 2.

Delete everything in the “Under.cshtml” view from line 5, to the end.

We are going to be doing the same thing we did in “Project 2,” to build the Accordion except that we will be building our “heading” / “div” pairs using C# and HTML. We will then add the <script> and <link> tags for the jQuery / jQueryUI libraries, which are obtained from “wwwroot” and a <script> block to the end of the page to compose the “Accordion.”

The code below is what we will add to the “Under.cshtml” page to get started.



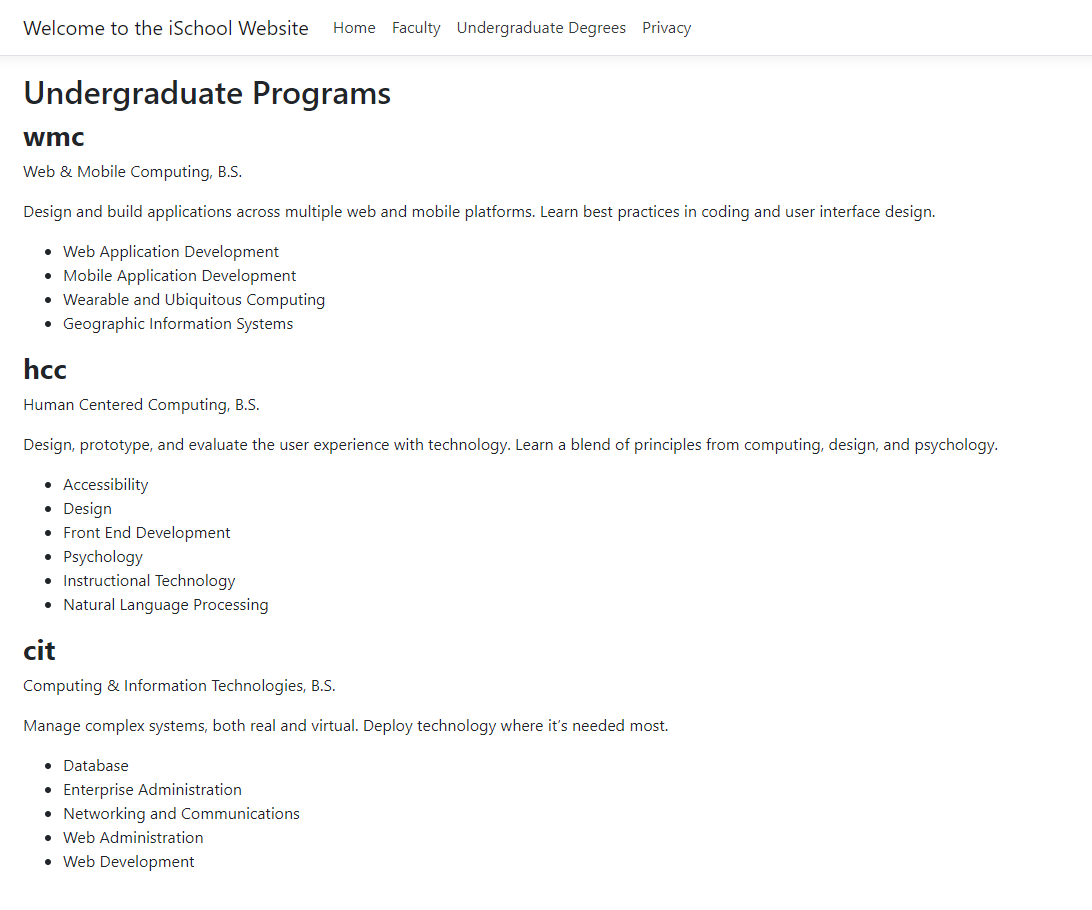
Even with the interspersal of C# and HTML it should look familiar.

Things to notice:

* On line 7, is the <div> we will associate the jQueryUI Accordion with
* In the “foreach” loop the unique id for the <h3> and <div> tags is created through the concatenation of the string “myHdng” and the “degreeName,” which is unique on each iteration, and the string “myDiv” and the “degreeName.”

If we run the code with just those changes, the output we get is:

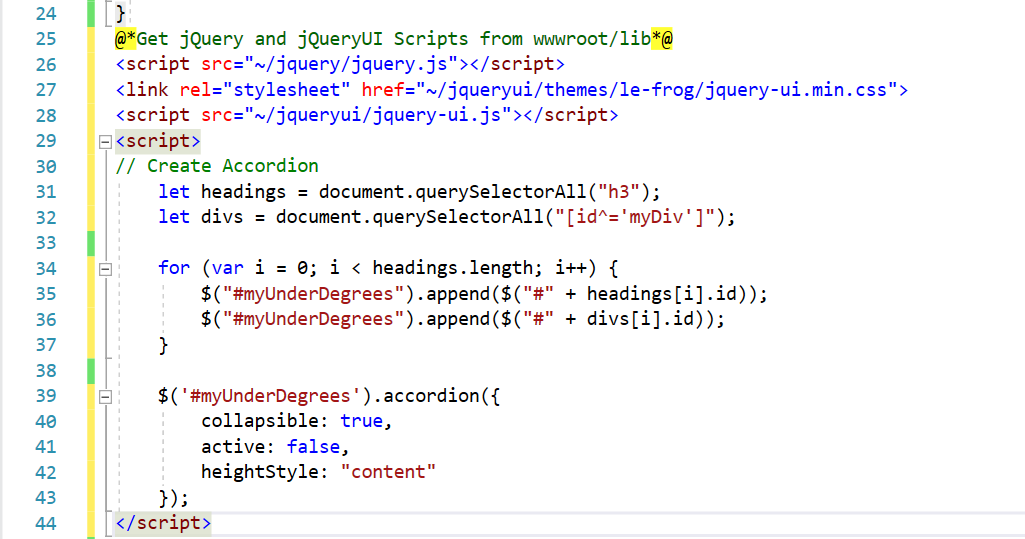
*Continued next page.*



Now we will add the jQuery / jQueryUI Code to produce the “Accordion.”

*Continued next page*

The scripting we will use to build the Accordions from the “header / div” elements above is as follows:



The code should be placed following line 22, as shown.

The src / href for all our jQuery / jQueryUI constructs is “wwwroot.” Typing “~/…” in front of each construct automatically points the code to the “wwwroot” folder independent of the actual file configuration of the machine it is running on.

We need to append to the div with id “myUnderDegrees” the heading / div pairs we just constructed. We will first “collect” them into a “nodeArray,” remember those, with the statements on line29, and line30.

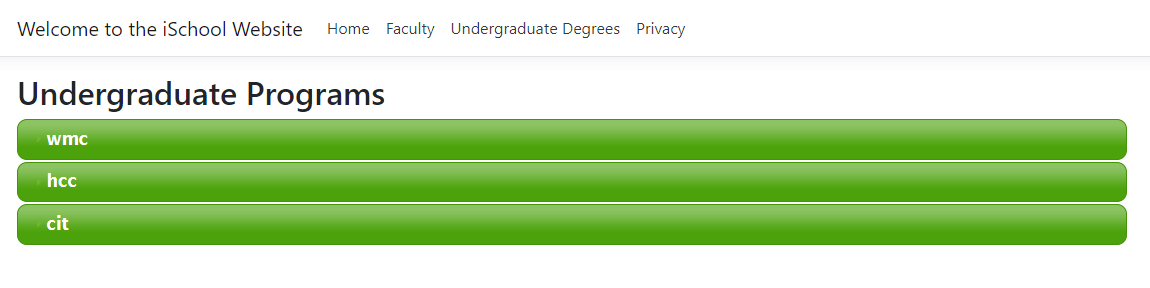
On line 29, we simply use “querySelectorAll(“h3”)” because the only “h3” headings on the page are the ones created for the “degreeName.”

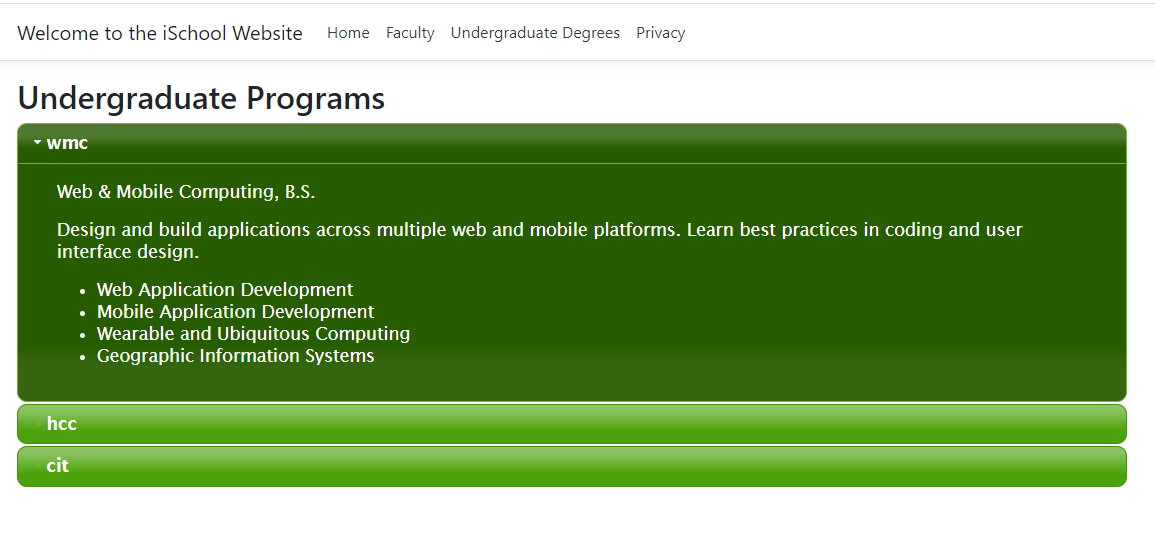
On line 30, we need to use a more selective querySelector. There are multiple <div> tags on the page but we only want the ones with an “id” that begins with “myDiv.” As such we are using the querySelector “[id^=’myDiv’]” which will retrieve all divs whose id starts with “myDiv.”

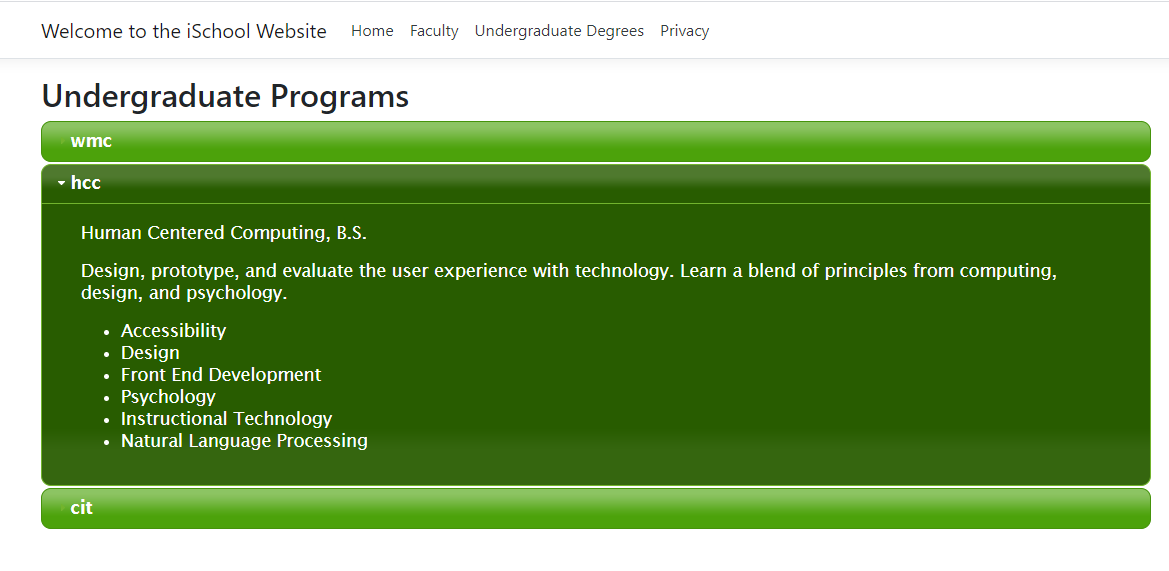
We can use the length of the “headings” nodeArray, the “divs” one would work as well since they are the same length, to iterate over the “headings” and “divs” nodeArrays to “glue” the “heading / div” pairs to the “myUnderDegrees” <div>.

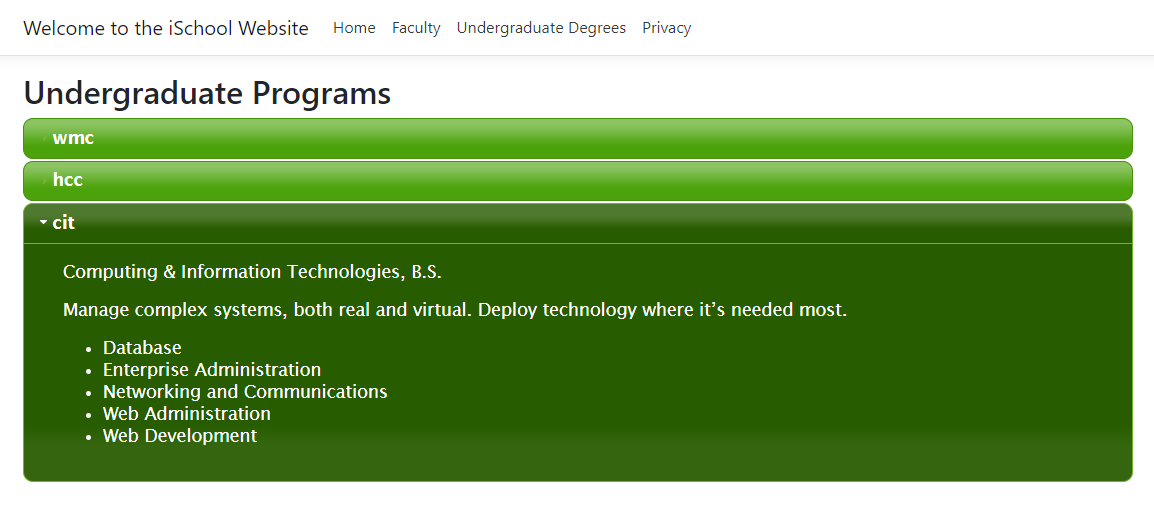
With that done, we just create the “accordion” with the code starting on line 39.

When we run the new code, we have:



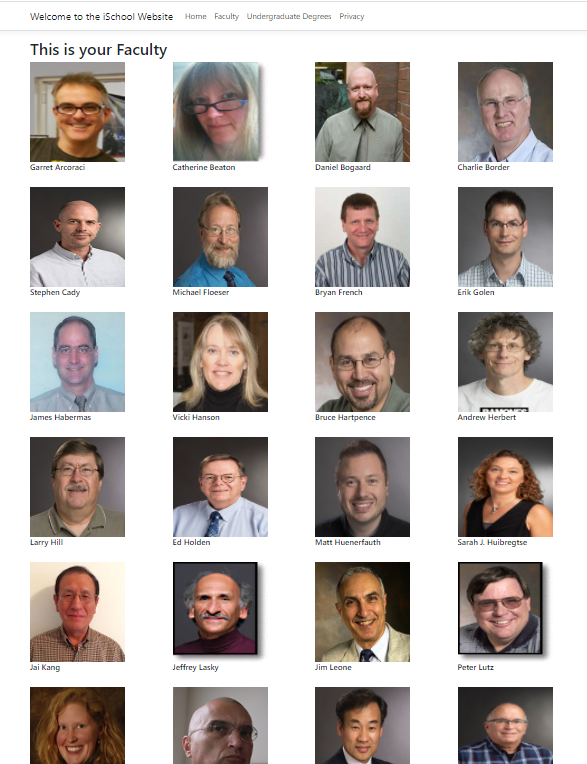




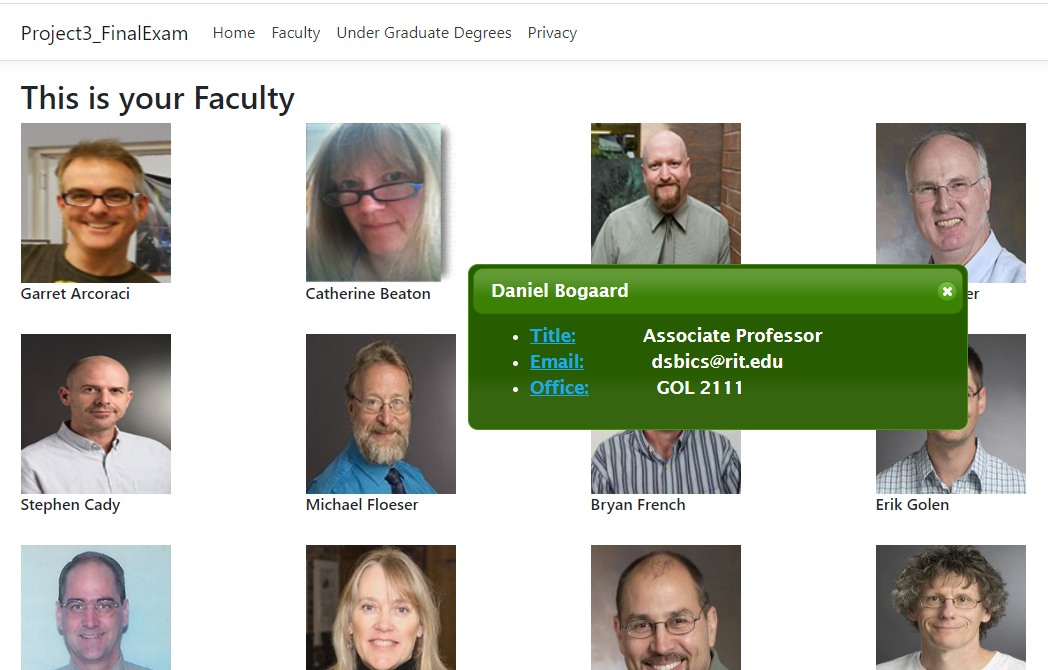


*Continued next page*

We will now enhance the “GetFaculty.cshtml” with jQuery / jQueryUI. Right now, when we select “Faculty” from the menu we see:



We would like to enhance this view so that when we click on one of the faculty members pictures, we see a jQueryUI “Dialog” widget as follows:



We will accomplish this in two steps:

* In the code already on the “GetFaculty.cshtml” page we will add an “onclick=” event to each thumbnail.
* We will create the appropriate script at the bottom of the page as we did for the “Under.cshtml” page.

*Continued next page*

At the end of each



on the page we will add the following



Each <div class=”thumbnail”… should now look as follows:



All the rest of our work involves adding the appropriate script at the end of the page.

Before adding the script, we need to add a div to serve as the target for the “Dialog” widget:



Now we can add the following code after the above div:



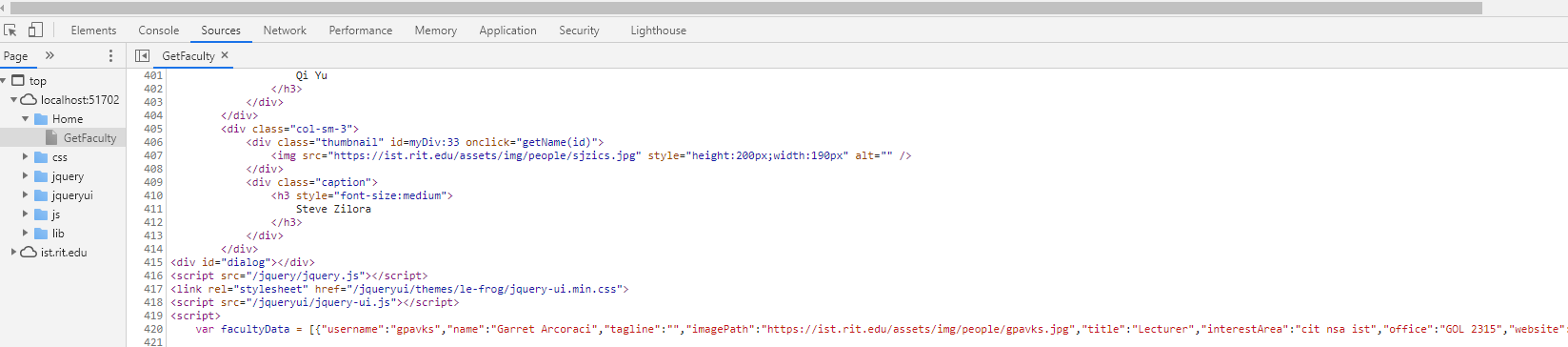
On line 88, is code that is probably new to you.



The “@Html.Raw” method provides access to the HTML on the page that the <script> block is at the bottom of. This gives us access to the “Model.Faculty” containing the array of ‘people/faculty’ which we then Serialize into Json so that our JavaScript can work with it.

Let us start up the program. After the “Home” page appears open the “Developers tools” and click on the “Sources tab.”

Expand the “Sources” window so that the <script> tab followed by “var facultyData…” is visible.



It is a little hard to read in this document so below is a copy of part of the first entry in “facultyData.”

**var facultyData = [{"username":"gpavks","name":"Garret Arcoraci", "tagline":""," imagePath":"https://ist.rit.edu/assets/img/people/gpavks.jpg","title":"Lecturer",**

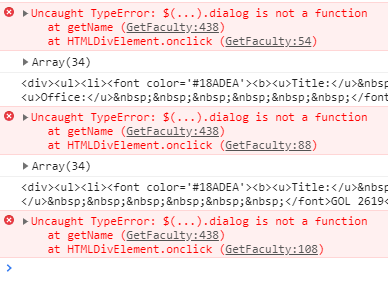
The point is that the JavaScript variable “facultyData” is the array returned from the “ist.rit.edu/api/people/faculty” endpoint.

“function getName(id) {…” is called when the user clicks on a faculty members picture. The code at line 92,



is why we set the start of the “id” for each “thumbnail” div to “myDiv:” By splitting the “id” on the character “:” we get a two element array where the first element is the string “myDiv” and the second element is the value of “i.ToString()” at the time the “id” was created. More specifically, it is the index in the “facultyData” array of the faculty member whose picture was clicked. We set “facultyElement” to that index and use it to retrieve from “facultyData” the data we want to display in the “Dialog” widget.

With that code in place let us run the website again and select “Faculty” from the menu. I have clicked on three different pictures and nothing has happened. If we open the console, we find three errors:



What is happening here is something that can occur because ASP.NET Core loaded libraries multiple times from multiple places. We did not have any issues with the “Under.cshtml” accordion because all our jQuery / jQueryUI libraries were loaded and run in the script during page load time – remember web pages are loaded and run from top to bottom.

If you look at the “Sources” tab from the scripting down, you’ll notice that before running our script block, we load the various jQuery / jQueryUI libraries, then the script block is loaded and run and following that some more scripts are loaded by the “\_Layout.cshtml” page that contains the “GetFaculty.cshtml” page.



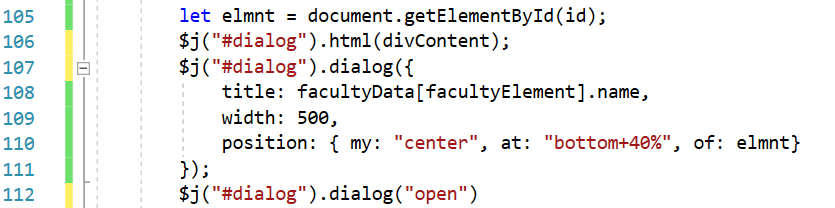
The issue here is that one of the last things that gets loaded is the “jquery.min.js” library. That comes from a different location than the “jquery.js” loaded right before the script block in our “GetFaculty.cshtml” view. When you click on a faculty member’s picture the “getName(id) function is invoked at which time a copy of jQuery other than the one used to create the script block is “active.” This leads to a library conflict which is why the “.dialog” is not being recognized as a jQueryUI widget. This is not an uncommon problem. The work around for it is to add a special jQuery method as the first line of our script block:



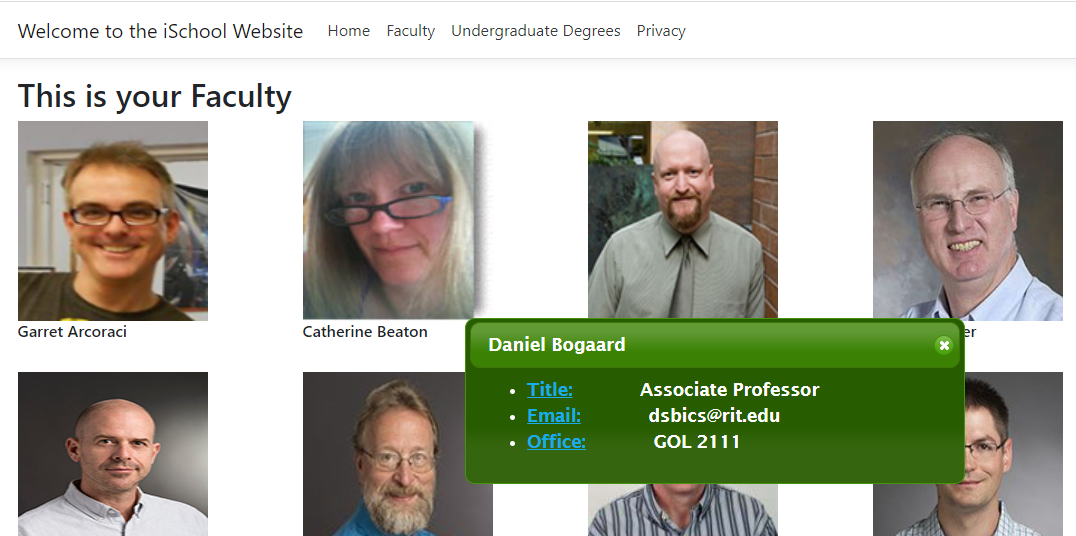
What this method does is tell the script to avoid jQuery conflicts by mapping and $j command to the currently active jQuery library regardless of any libraries that may be loaded later.

To use this “noConflict” method we change all jQuery $ prefixed methods in our script block to be prefixed with $j.

The code for the “Dialog” widget now is:



With that change we again run our code and this time:



*Continued next page*

**Conclusion**

You now have everything, well almost, you need to complete Project 3. Again, review the exact requirements of what you are expected to produce which are at the start of this document. If you have any questions, do not hesitate to contact me.

**THE PROJECT IS DUE TUESDAY, MAY 11, AT 10:30AM – THAT IS THE DEADLINE.**

**IT IS THE ENDING TIME FOR THE FINAL EXAM FOR THIS COURSE AS SCHEDULED BY THE REGISTRAR.**

**THE DROP BOX WILL LOCK AT 10:35AM, NO SUBMISSIONS WILL BE ACCEPTED AFTER THEN.**

**THERE CAN BE NO EXTENSIONS OF ANY KIND FOR ANY REASON**

By the way, the one thing you are missing is how to utilize jsChart for the Employment table. You will load it like the way you loaded jQueryUI – that will be the subject of next Tuesday’s lecture.