Mega Tutorial

Isaac Weiss
Co-Op Emp OIT

This is the Campus Web Solutions **Mega Tutorial**, designed to get you started developing as quickly as possible. Table of Contents:

Overview

Specific Technologies:

- Bootstrap Multiselect
- Bootstrap Validation
- Other Technologies

Helpful Bookmarks (External)

- Jira
- Shared
- GitHub
- SharePoint

Overview

MVC

Our development is centered around the Model-View-Controller (MVC) pattern.

Model: This is the data. This could be tables in a database, classes that we make ourselves, or really anything that is designed to retain information for later use, whether that be updating it, deleting it, or just reading it. This is **storage**.

- Examples: tblPerson.cs, MyViewModel.cs

View: This is the .cshtml page that the user sees. It's called .cshtml because it is a C# HTML (we'll dive into these later) page, which uses Razor Markup. Think of it as the **frontend**. It's where HTML, CSS, and JS work together to create a page.

- Examples: Create.cshtml (and its components Create.css and Create.js)

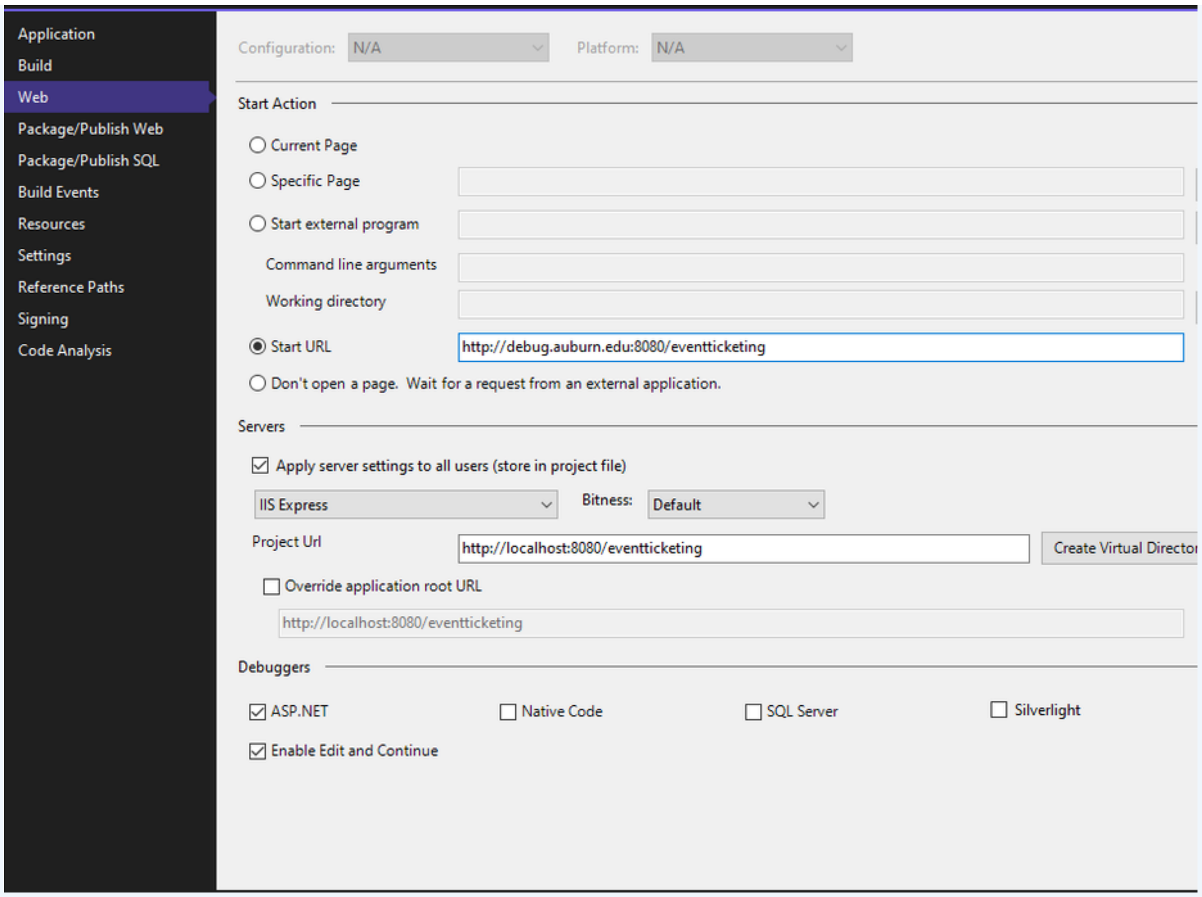
Controller: The controller acts as a traffic hub for data, directing it where it needs to go. It's also where developers will write most of their C#, **perform logic** on data, manipulate data, and validate user responses. The controller is a C# class which inherits Controller, so do not go around thinking that JavaScript is the Controller. JavaScript is the View

- Examples: HomeController.cs, AdminController.cs

Workflow

Receive a new story or defect: Go to its Jira page, bookmark it, and move the task out of backlog

1. Pull project down from Github: Open GitHub Desktop, Clone/Add the repository, navigate to Documents/GitHub/<repo> (*replace <repo> with whatever project you're pulling*), then navigate to the Project folder and click the .sln file (it opens the Solution)
2. Set up for debugging: There is a tutorial here, but these steps can be boiled down to two steps (in .NET framework...):
   a. Open your Solution, then go to Project tab (at the top) then click *<your solution>* Properties (it'll be near the bottom). Go to the Web tab. Triple-click the 'Project Url' input box, copy that to your clipboard. Now, in the 'Start Action' menu, select 'Start URL' and paste the link you just copied, replacing 'localhost' with 'debug.auburn.edu'. Save.

b. 

Configuration: N/A Platform: N/A

Start Action

- ○ Current Page
- ○ Specific Page
- ○ Start external program
- Command line arguments
- Working directory
- ● Start URL: http://debug.auburn.edu:8080/eventticketing
- ○ Don't open a page. Wait for a request from an external application.

Servers

- ☑ Apply server settings to all users (store in project file)
- IIS Express Bitness: Default
- Project Url: http://localhost:8080/eventticketing Create Virtual Director
- ☐ Override application root URL: http://localhost:8080/eventticketing

Debuggers

- ☑ ASP.NET ☐ Native Code ☐ SQL Server ☐ Silverlight
- ☑ Enable Edit and Continue

Notice how we're using the EventTicketing project in t

c. In Windows File Explorer, make sure you have Hidden Files available to view (View -> Check Hidden Items). Navigate to your Documents/GitHub/*<your solution>*/*<your project>*/.vs/*<your project>*/config/applicationhost.config, and open it. If you don't see it, you need to open your solution and build it. CTRL+F and search 'localhost'. Within a <bindings> tag group, you should see a <binding>tag with the port 8080 (usually). CTRL+D to duplicate this line, then replace 'localhost' with 'debug.auburn.edu' on one of the lines. The finished product will look like (we added the top <binding>). After that, run the program. Here's what the bindings looks like:

d.
```
<bindings>
   <binding protocol="http"
bindingInformation=":8080:debug.auburn.edu" />
   <binding protocol="http"
bindingInformation=":8080:localhost" />
   <binding protocol="https"
```

```
bindingInformation="*:44361:localhost" />
</bindings>
```

3. Develop the story using MVC (or find and fix the bug)
4. Code review: Send your code up the chain, once its approved it will be merged.
5. Code is published and sent to QA (out of your hands unless there's a defect): Once merged, let someone know to publish it (if you are publishing yourself, here is an excellent guide: Publishing to Dev/Test/Migr)

- Security -> UAT -> Prod: After it passes QA, it will go through Security, UAT (sometimes referred to as Test), then Release eventually

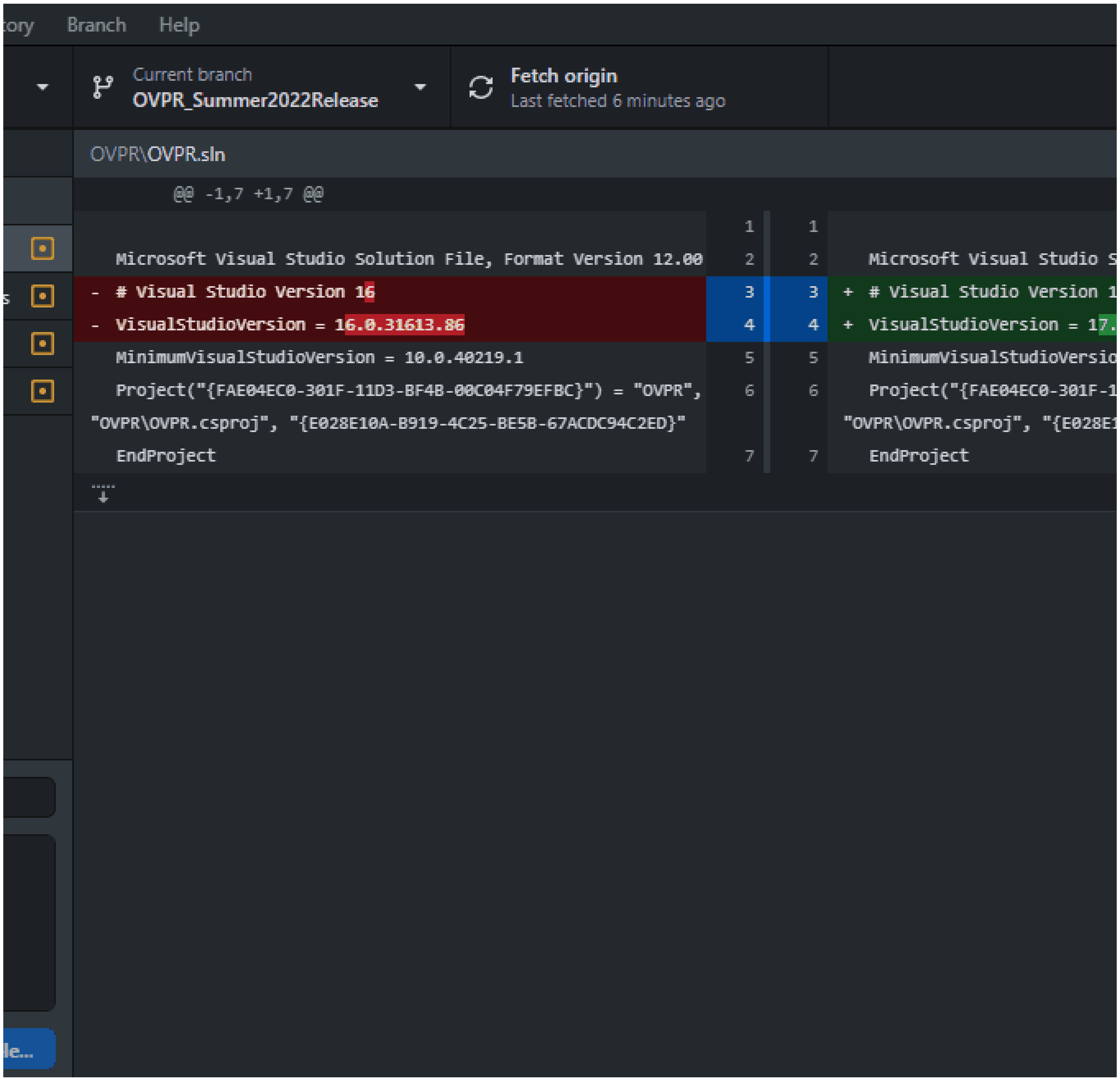
GitHub

GitHub:

We use GitHub as source control, or in other words, how we manage a bunch of people working on the same codebase at the same time. There's a lot of terminology at first but once you get it, you'll see it's rather simple. A branch is a unique copy of the codebase, and branches **allow you to develop features, fix bugs, or safely experiment with new ideas in a contained area of your repository.** Think of any of our GitHub repositories as a hierarchy with the 'main'/'master' branch at the very top (this is always the branch that prod (production) looks at). Most of the time, especially with projects that are being worked on heavily, there will be "Releases" planned, in which there will be a Release branch (ex. Release_Fall2022 branch). **We base our branches off the Release branch, and the Release branch is where we will eventually merge into.**

Let's walk through what to do when you get a new story. Say I get the story "OVPRS2022-81", which is the **story**, not the subtask. If you're confused about Jira, read here, but basically, just remember a story contains subtasks, and one of the subtasks is development (you track time on this, and move this to finished when done). Stories have orange lightbulbs as their symbol. Anyways, the first thing I want to do is go to GitHub Desktop, click Current Repository (it will dropdown), click Add -> Clone Repository... -> search OVPR and click clone. Make sure the local path it's cloning to is .../Documents/GitHub. It just makes your life easier. Anyways, you've cloned it, so you open your solution and set up for debugging (see above). Once you're ready to commit (and I suggest you use commits as checkpoints, it's probably reasonable to say you want to commit at least once a day), you'll notice on GitHub desktop you'll have a lot of working changes on the left. I almost always push after I commit.

The following gif is going to be me bringing these changes to a new branch using CWS's naming standards, then committing and pushing.

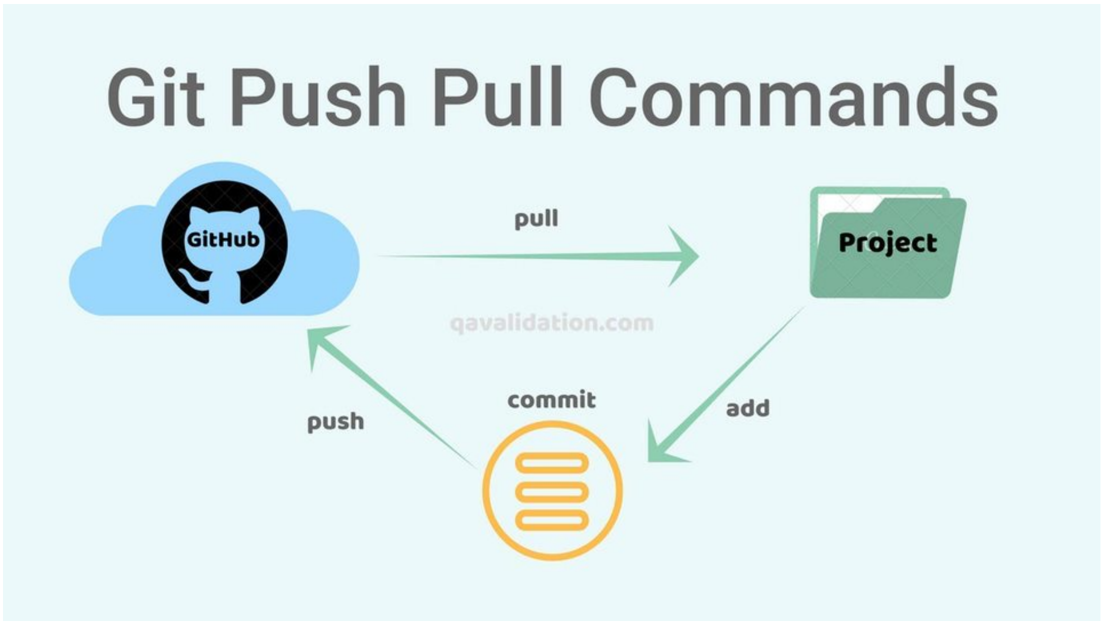


```
@@ -1,7 +1,7 @@

  Microsoft Visual Studio Solution File, Format Version 12.00
- # Visual Studio Version 16
- VisualStudioVersion = 16.0.31613.86
  MinimumVisualStudioVersion = 10.0.40219.1
  Project("{FAE04EC0-301F-11D3-BF4B-00C04F79EFBC}") = "OVPR",
"OVPR\OVPR.csproj", "{E028E10A-B919-4C25-BE5B-67ACDC94C2ED}"
  EndProject
```

```
  Microsoft Visual Studio S
+ # Visual Studio Version 1
+ VisualStudioVersion = 17.
  MinimumVisualStudioVersio
  Project("{FAE04EC0-301F-1
"OVPR\OVPR.csproj", "{E028E1
  EndProject
```

ting and pushing to your own branch does NOT affect master/main/release. Make sure you're on your branch

Commit vs Push: When you commit changes, you are staging them to be pushed to the remote (online) copy of your branch so that anyone in CWS could see your changes. You can still undo a commit (it's under where the commit button is). Once you push, you have updated your remote branch to match your local one. On the flip side of that, if you are switching to a branch, make sure you 'Fetch Origin' (basically, check for updates). GitHub usually does this for you, but you can never be too sure. If a branch has changes in the remote that aren't on your local, you can pull those.



I usually push as soon as I commit, but I see the utility of keeping many commits on your local before pushing. I'm just paranoid my computer will blow up and all my work will be reduced to NULL.

Pull Request: When you're ready to send your code to someone for code review, you can hit CTRL+R or Branch -> Create pull request. **Make sure you're merging into Release.** In the comment, add anything you feel should be said but also paste the link to the Jira story. Make sure you add reviewer(s) and assign yourself (so you get notifications). If someone requests changes, you can fix those on your branch, push those changes, then click the circular icon next to their name under the reviewers list to rerequest their review.

Merging: Sometimes you need to merge Release into your branch. Switch to your branch first. You can hit CTRL+Shift+M or go to the branch menu and click the button at the bottom for Choose a branch to merge into <your branch name>... If everything is fine with the merge, it'll give you the option to push. If there are conflicts, click the dropdown next to each conflict and open it in either Visual Studio Code or Visual Studio (or something that will show you the conflicts in an easy-to-understand fashion). "Accept from incoming" means you're overwriting your code with someone from Release. "Accept from current" means you're going to use your own code over whatever's in Release.

Stash: If you right-click the "x changed files..." tab above your changed files, you can 'stash' them, which basically discards the changes but GitHub keeps track of those changes just in case you change your mind later. You can restore the stash.

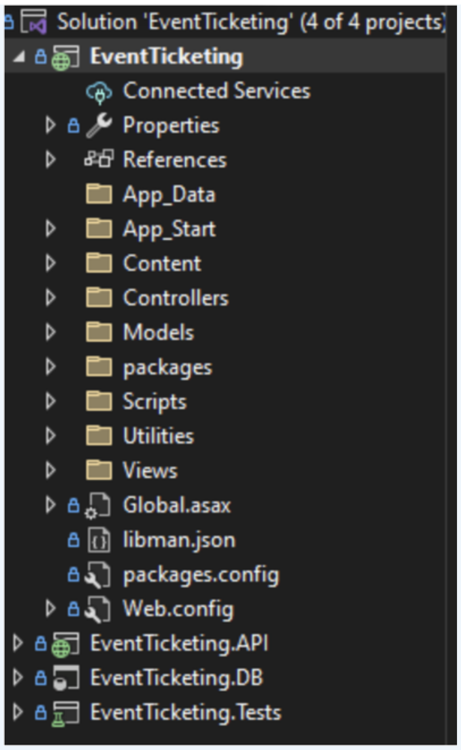
Advanced GitHub: Some nice shortcuts are CTRL+Shift+N for a new branch, CTRL+R for pull request, and CTRL+Shift+M for merging into your branch. Also, if you have a pull request you can quickly get to your branch on the branches menu by clicking the 'pull requests' tab. You can click the "History" tab on the main GitHub desktop page to see the previous commits, and right clicking on any of them gives options for reverting, etc. You can hold Shift and select the top and bottom commits for a range, or hold CTRL and click for specific commits. Some options for you are:

- Revert: Reverses all changes from a commit
- (batch) Cherry pick: You can take the changes from a specific commit (or many) and also apply it to a different branch.
- (batch) Squash: Combine multiple commits into one. This is just to clean up the history.

Visual Studio

The application you are going to use the most here at CWS. I'm using Visual Studio 2022 for this tutorial, and using EventTicketing as an example, so things might be slightly different for different projects.

First, assuming you've set up debug and can run the project, notice there is a Solution Explorer on the right (sometimes left) with all the project files. CTRL+ALT+L to show the Solution Explorer if it's hidden.



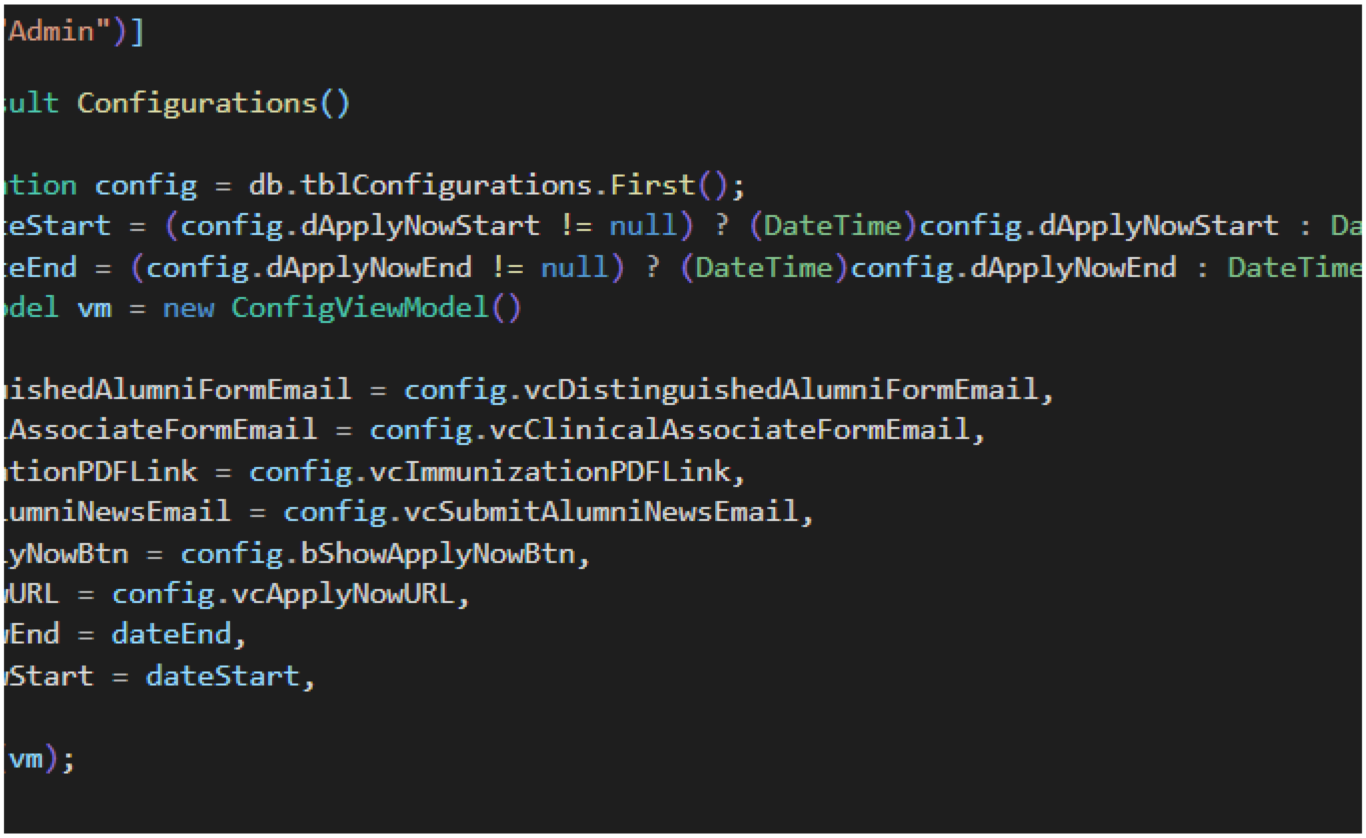
May look different for .NET Core

Let's explain these:

- References contains all of your NuGet packages (if these have yellow caution triangles and you can't run your project- uh oh! NuGet Errors: Fixing and Committing).
- App_Data and App_Start are project files and we don't usually modify them.
- Content has a CSS folder (contains CSS files) and a JS folder (contains JS files).
- Controllers contains all the controllers.
- Models has both the database tables (Entity Framework maps our databases to actual C# classes in our project) and any ViewModels we create, as well as any other user-defined classes.
- packages holds more NuGet information.
- Scripts has some bootstrap files, so that might not be in other projects.
- Utilities has helper classes/methods.
- Views has all the views and partial views (they start with '_').
- Global.asax has the method that runs when the application is started, including which Masterpage to use.
- The web.config is what is modified when dropping down the 'Debug' and selecting other environments (if your project has transformations set up, otherwise Web.config Transformations Setup. It is also where the connection string is stored, so if you need to know what database a project is looking at, look in the specific web config for that environment (drop down web config and select the specific one you need, i.e. web.QA.config.

Controllers

Remember that the controller is the backend of the application. There are really two main types of methods we have at CWS. An ActionResult method that returns a View (this is what is called when going to a specific URL, like "cws.auburn.edu/Nursing/Home") and an ActionResult method that returns a Json (this is typically when users submit forms, and the Json contains information such as if the request went through successfully). These do not redirect a user, unlike the "return View()" methods.



```
'Admin")]

sult Configurations()

ation config = db.tblConfigurations.First();
teStart = (config.dApplyNowStart != null) ? (DateTime)config.dApplyNowStart : Da
teEnd = (config.dApplyNowEnd != null) ? (DateTime)config.dApplyNowEnd : DateTime
odel vm = new ConfigViewModel()

uishedAlumniFormEmail = config.vcDistinguishedAlumniFormEmail,
lAssociateFormEmail = config.vcClinicalAssociateFormEmail,
ationPDFLink = config.vcImmunizationPDFLink,
lumniNewsEmail = config.vcSubmitAlumniNewsEmail,
lyNowBtn = config.bShowApplyNowBtn,
wURL = config.vcApplyNowURL,
wEnd = dateEnd,
wStart = dateStart,

(vm);
```

Example Controller method from Nursing/AdminController

You have the ability to create ViewModels in the Models/ViewModels folder, and after that, you can use that as a custom object that can hold whatever data you need to send to a page. The ViewModel can be called within the HTML to display data. Think of the ViewModel as the messenger between the Controller and the View.

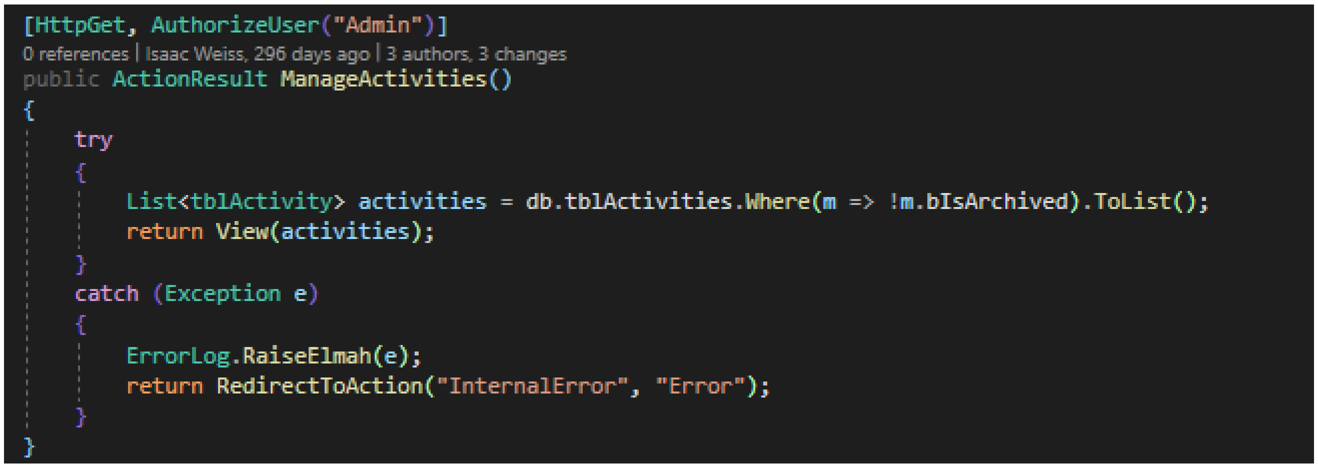
Entity Framework has made all of our database tables into C# objects so that we can use them as such in our controller. Above, notice how I am getting a tblConfiguration and reading values from it, which I place into the ViewModel which is then sent to the View.

ViewBag and ViewData are like ViewModels, except you don't have to create anything. You can just place something in the ViewBag/ViewData and it'll still be there in the View.

Typically, we use a lot of helper methods if something is going to be used lots of times. This is usually in Utilities.

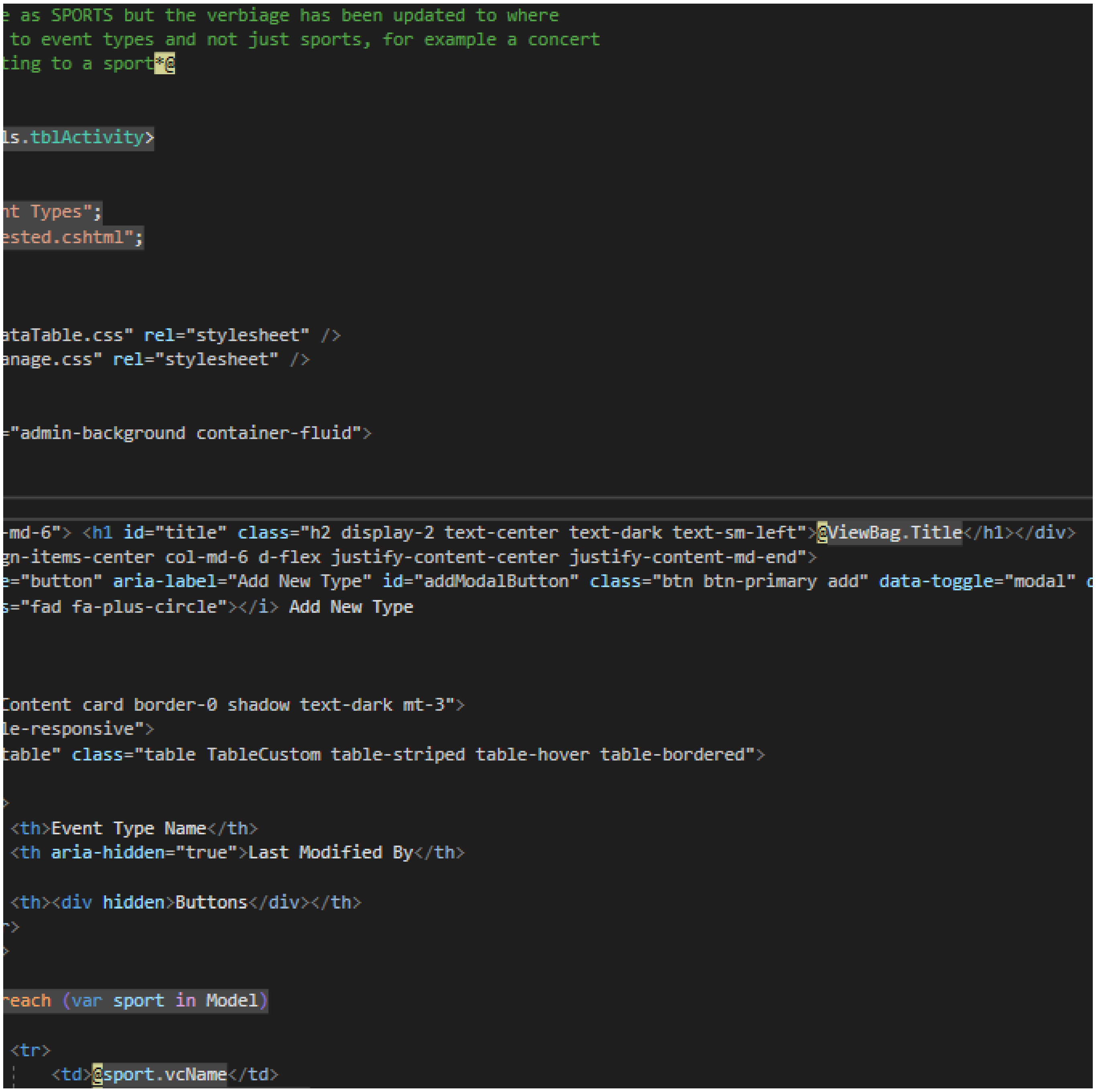
This is the flow of a typical MVC page:

The user goes to cws.auburn.edu/EventTicketing/Admin/ManageActivities. They are directed to the controller method ManageActivities in the Admin Controller.



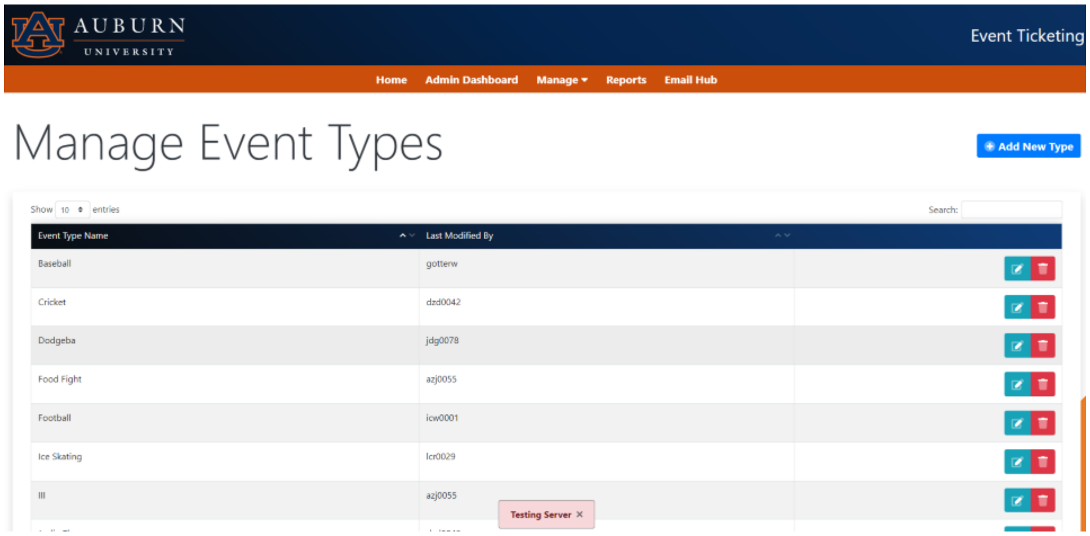
```
[HttpGet, AuthorizeUser("Admin")]
0 references | Isaac Weiss, 296 days ago | 3 authors, 3 changes
public ActionResult ManageActivities()
{
    try
    {
        List<tblActivity> activities = db.tblActivities.Where(m => !m.bIsArchived).ToList();
        return View(activities);
    }
    catch (Exception e)
    {
        ErrorLog.RaiseElmah(e);
        return RedirectToAction("InternalError", "Error");
    }
}
```

The controller method grabs the list of tblActivity, sends it to the View. You can typically use anything as a model with MVC pages, so this controller method just uses List<tblActivity>. The return View(activities) then directs to the ManageActivities.cshtml view in the Views/Admin folder.



```
e as SPORTS but the verbiage has been updated to where
 to event types and not just sports, for example a concert
ting to a sport*@


ls.tblActivity>


nt Types";
ested.cshtml";



ataTable.css" rel="stylesheet" />
anage.css" rel="stylesheet" />


="admin-background container-fluid">



-md-6"> <h1 id="title" class="h2 display-2 text-center text-dark text-sm-left">@ViewBag.Title</h1></div>
gn-items-center col-md-6 d-flex justify-content-center justify-content-md-end">
="button" aria-label="Add New Type" id="addModalButton" class="btn btn-primary add" data-toggle="modal" d
s="fad fa-plus-circle"></i> Add New Type



Content card border-0 shadow text-dark mt-3">
le-responsive">
table" class="table TableCustom table-striped table-hover table-bordered">

>
 <th>Event Type Name</th>
 <th aria-hidden="true">Last Modified By</th>

 <th><div hidden>Buttons</div></th>
r>
>

each (var sport in Model)

 <tr>
     <td>@sport.vcName</td>
```

Just FYI, the Layout at the top is specifying that we are going to use Nested- the Auburn header and footer (but for most projects, this is already defined in the _ViewStart, or at least should be). Razor is everything with @, basically just C# within the HTML. Allows foreach blocks, setting variables, etc. Razor looks at the Model, and from then on you can access the model within Razor blocks. I will explain the View in the next section, but for now just know it returns a page that looks like below.



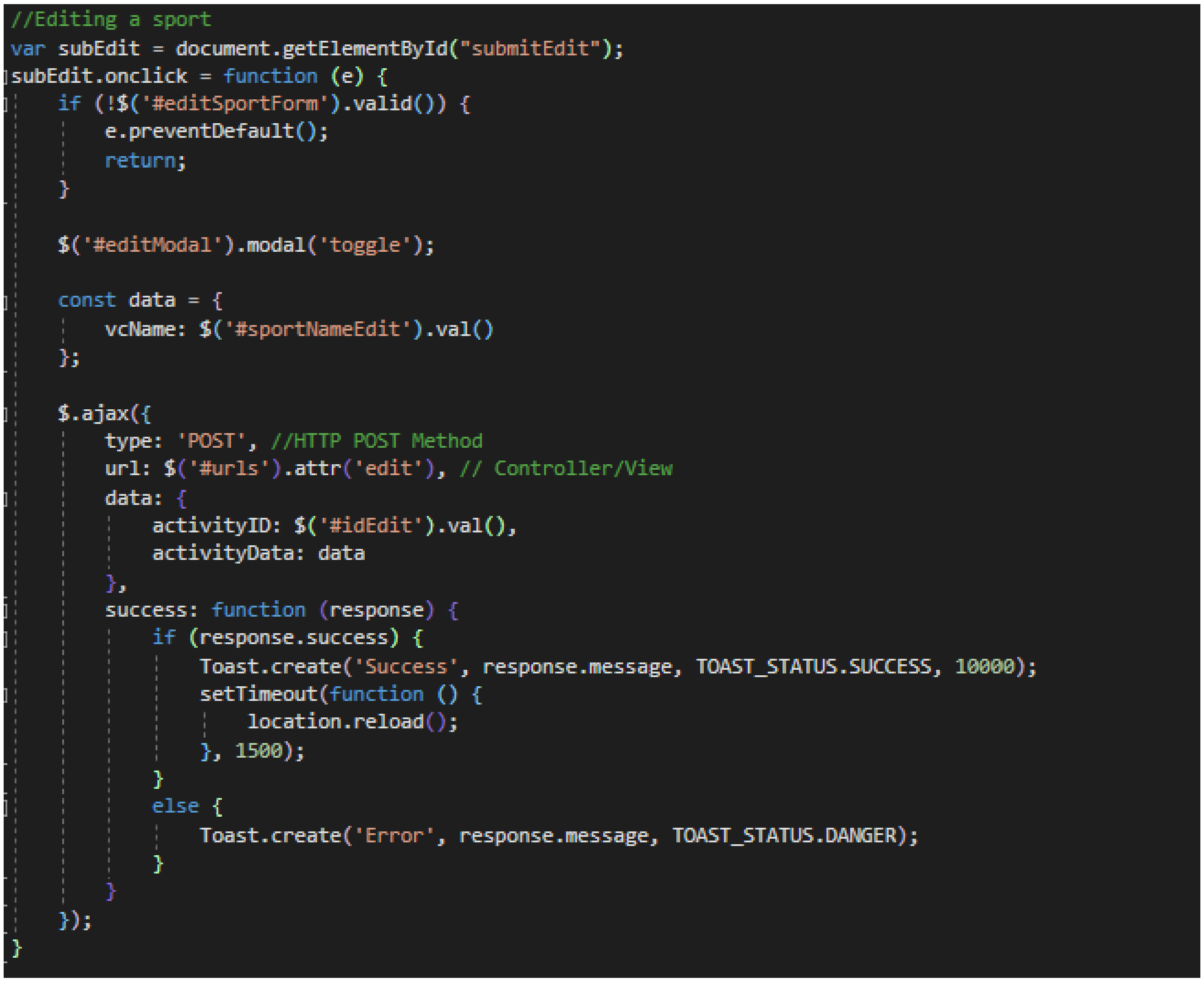
Manage Event Types

Show 10 entries

Search:

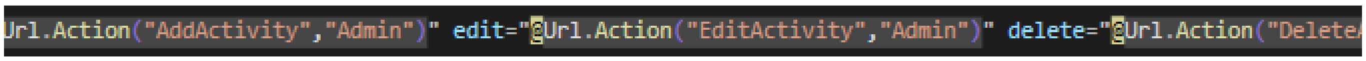
Event Type Name	Last Modified By	
Baseball	gotterw	
Cricket	dzd0042	
Dodgeba	jdg0078	
Food Fight	azj0055	
Football	icw0001	
Ice Skating	lcr0029	
lll	azj0055	

Now, clicking on the Edit button brings up a modal (a popup that dims the rest of the screen) and clicking submit goes to the URL "Admin/EditActivity" using Ajax, within the JavaScript. Here is that. By the way, see if you can't make these pages dynamic and not require a 'location.reload()' after each successful ajax. It will make your testing much smoother.



```
//Editing a sport
var subEdit = document.getElementById("submitEdit");
subEdit.onclick = function (e) {
    if (!$('#editSportForm').valid()) {
        e.preventDefault();
        return;
    }

    $('#editModal').modal('toggle');

    const data = {
        vcName: $('#sportNameEdit').val()
    };

    $.ajax({
        type: 'POST', //HTTP POST Method
        url: $('#urls').attr('edit'), // Controller/View
        data: {
            activityID: $('#idEdit').val(),
            activityData: data
        },
        success: function (response) {
            if (response.success) {
                Toast.create('Success', response.message, TOAST_STATUS.SUCCESS, 10000);
                setTimeout(function () {
                    location.reload();
                }, 1500);
            }
            else {
                Toast.create('Error', response.message, TOAST_STATUS.DANGER);
            }
        }
    });
}
```

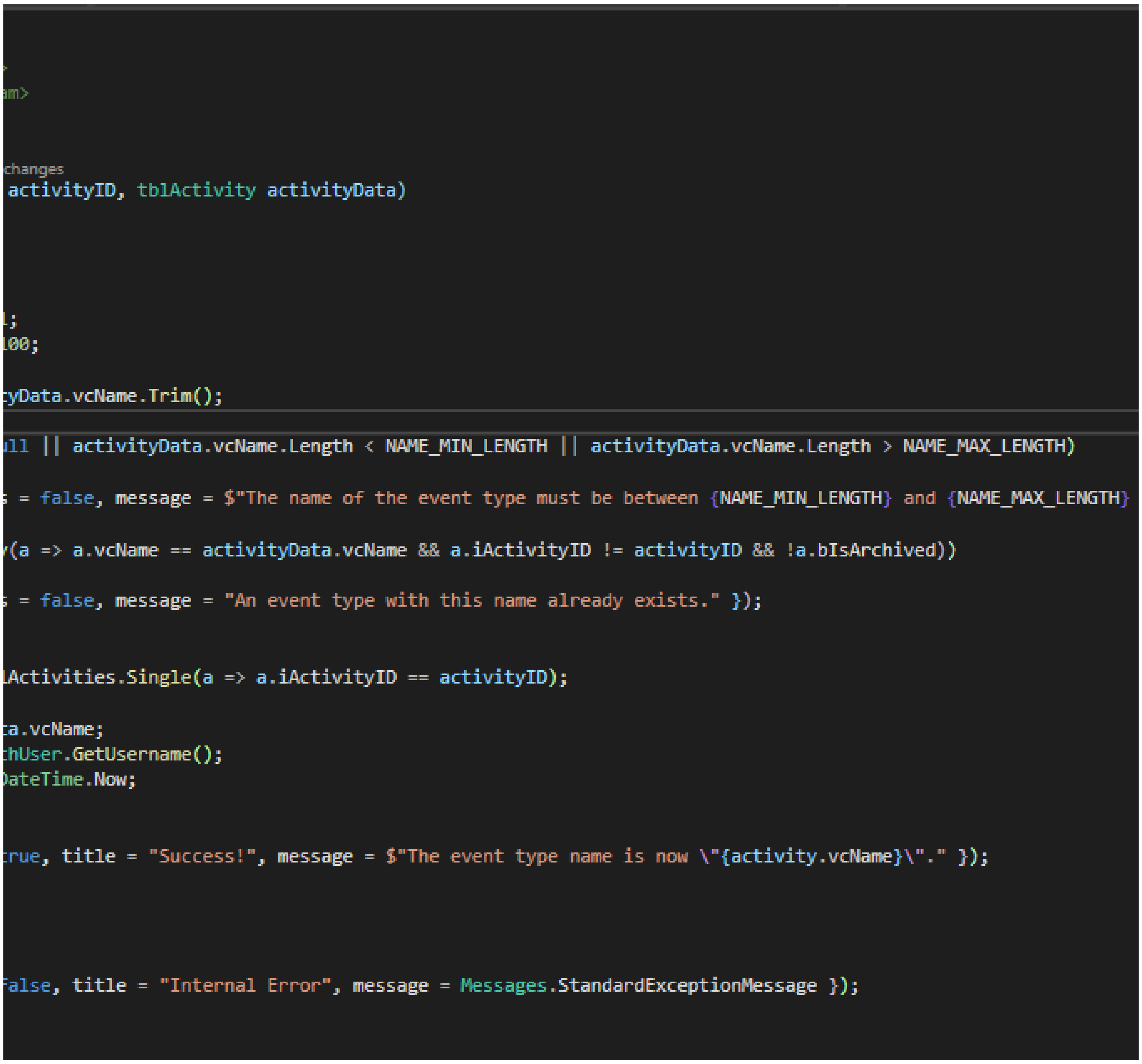
First the submitEdit button is collected (it has the ID of submitEdit) and this checks if the form surrounding the button is valid using Bootstrap Form Validation. After validated, it makes a data object that will be sent into the Controller. **Important:** whatever the parameter is for the controller method that accepts this Ajax call, it needs to match exactly the data specified in the Javascript. The two parameters for EditActivity are "activityID" and a tblActivity named "activityData". We are using the const data in the javascript as a placeholder for the tblActivity (we don't need to fill all the fields of a tblActivity, but since we are specifying a vcName, it will send a mostly empty tblActivity to the controller with just that name changed). **Make sure the names and types match**! We call ajax with $.ajax()... **Type GET vs POST:** Use POST for ajax. Get is like when going to a URL, a GET request is called for that URL. Post means we are sending data to the controller. There will be problems if you try to use GET (for example, running out of allocated space within the Ajax request and the whole page breaks).



```
Url.Action("AddActivity","Admin")" edit="@Url.Action("EditActivity","Admin")" delete="@Url.Action("Delete
```

We create a span with id="urls", then call the jquery syntax to get a specific attribute

At this point, we go to the controller.



```
>
am>


changes
 activityID, tblActivity activityData)




l;
100;

tyData.vcName.Trim();

ull || activityData.vcName.Length < NAME_MIN_LENGTH || activityData.vcName.Length > NAME_MAX_LENGTH)

s = false, message = $"The name of the event type must be between {NAME_MIN_LENGTH} and {NAME_MAX_LENGTH}

y(a => a.vcName == activityData.vcName && a.iActivityID != activityID && !a.bIsArchived))

s = false, message = "An event type with this name already exists." });


lActivities.Single(a => a.iActivityID == activityID);

ta.vcName;
thUser.GetUsername();
DateTime.Now;


true, title = "Success!", message = $"The event type name is now \"{activity.vcName}\"." });




false, title = "Internal Error", message = Messages.StandardExceptionMessage });
```

The Pink bar denotes validation (checking if input is valid), the Yellow mark denotes us getting the tblActivity that we want to modify from the database. The Blue bar denotes us modifying the tblActivity, and finally calling db.SaveChanges().
Important: Methods need a try catch, and within the catch raise to Elmah (our error logging system, you go to Shared and click Errors). Once we edit the data (or fail) we go back to the ajax, within the success function. We use information to create a Toast (popup that disappears after some time) and call reload on the page. That's a typical process for a CRUD page.

C#

If you know Java, C# should come pretty naturally. Actually, C# with the addition of LINQ makes things a lot more intuitive. LINQ adds features to C#, most importantly abilities to manipulate lists and filter with lambda expression. Example of a LINQ statement:

The m is the lambda parameter, and its a stand-in for one tblActivity. db.tblActivities is a DbSet<tblActivity>, calling .Where means you are trying to only return tblActivity that match the follow lambda expression (!m.bIsArchived), and ToList() turns it into a list.

Also make sure you use foreach, while, etc. But try not to have nested loops.

By the way, if you want to **log SQL statements** as Entity Framework is running methods (such as Where, First, etc), then put this line above where you want the SQL to start logging:

```
dbo.Database.Log = s => System.Diagnostics.Debug.WriteLine(s);
```

Databases

Updating EDMX in Visual Studio:

Clicking on Models/<project name>.edmx will bring up a diagram in framework projects. This is your database. If you need to update this, you can do so by right-clicking and "update from database". If that's not working, CTRL+A and deleting all, then updating again should be good. Make sure no tables you don't want to change are modified if you do that though.

SSMS (Microsoft SQL Server Management Studio) is how we check or edit our databases. Common databases:

- OITSSG-001\CWSDEVSQL2016
- OITSSG-001\CWSTESTSQL2016
- SQL2016CL-CWS (**warning**: this is usually prod)
- OITSS9\MSSQL09,3433 (**warning**: this is also usually prod)

You can then drop-down Databases, drop down the Database you want, and then on the table you have several options.

If you select top 1000 rows, that will provide those rows in read only, and you can use SQL to filter even more.

If you select edit, you bring up the edit row screen. CTRL+3 will bring up the SQL edit pane, where you can type your "Wheres" or whatever SQL you need to use to filter those rows. This editing is live immediately, and can be done by using the down/up arrows to leave the row that is edited.

If you select design, you see this screen.

This is where tables can be modified by adding fields, adding foreign key relationships (right click -> relationships), ordering columns and more.

Advanced SSMS:

- Use F5 while on a Select screen to run the query. CTRL+R while on the Edit SQL pane to run that query.
- There are Views (virtual tables) and stored procedures (custom sql queries). Both can be modified from SSMS.
- Stored procedures can be ran by right clicking and selecting Execute. If you wish to modify a stored procedure, you just have to make your changes and Run (click F5). Make sure you test by testing the actual query (between 'Begin' and 'End') in a separate window.
- Its good to pin tabs you are working on, or even save them to a common folder. You can right click a tab and close all to the right.
- A table needs an identity column if you want it to autoincrement its primary key (its unique ientifier). This is in table properties.
- Fields can have default fields. Usually do not check the allow nulls box.

HTML

There's a lot to get to here. I'm just going to briefly cover this topic.

This is HTML, specifically a modal. The aria- is for accessibility. Divs are used to denote blocks (spans are in-line). Bootstrap (an extension that helps with CSS) has many classes that ease with making web development easier, and those are the classes you see. With HTML, a class is used multiple times on a page, but an ID only has **one** element. With buttons, the type matters a lot. A type submit will sometimes attempt to reload the page. Change it to type="button" if you get into trouble. Inputs always have an associated label. You can use class="sr-only" for things you don't want showing up visually.

There's a section head at the top for linking your CSS, and a section scripts at the bottom for linking JS.

Example table. There must be an equal number of <th> (table headers) and <td> (table definitions). As you can, Razor is being used to dynamically create HTML rows.

JavaScript

JavaScript:

The wizard. You have events, such as click, that when fired can reach a method within your own code if you define it. Let's have a button with id="test". In jQuery, we would get this button using $('#test'). If it had class test, then $('.test') would get all elements that have that class. Here are some examples of jQuery in action.

As for jQuery methods, there's also:

- .each : iterates through an array
- target.before(someElement) : places someElement before target
- target.after(someElement) : opposite of above
- target.prepend(someElement) : adds someElement as the first child element of target
- target.append(someElement) : adds someElement as the last child element of target
- target.wrap(someElement) : wraps target with someElement (so target becomes a child of someElement)

Also, if you have Javascript inside a .cshtml, you can extract that to its own Chrome Inspector debug window by adding

```
//# sourceURL=test.js
```

As the first thing under that script tag. Then going to Inspect (F12 on google chrome) -> Sources -> pick the folder "no domain" and your file will be in there. You can set breakpoints by clicking to the left of the source.

This is Chrome DevTools. I recommend looking up good shortcuts because knowing how to use this quickly is a lifesaver when working on bugs. First of all, the colors coding. From bottom left to top right:

- Green arrow is console. You can type things down here and press enter to evaluate.
- The underline light purple clears the console.
- The underlined pink adds watches, meaning you can see a variables value any time
- The orange bar is where you can edit CSS.
- The blue underline is clicking :hov to force states (focus, hover, these are mouse events), or clicking .cls to add or remove classes from an element (very helpful when trying to fix something with CSS).
- By the way, clicking the arrows next to Layout will bring up something called "Event Listeners", which will show what is called in the javascript from your files that are associated with whatever element you have selected. (click "show ancestors" because sometimes the event will be attached to the document, like when we do document.on('click', '#test', function () {}), etc.)
- In the top left, you have the select element (CTRL+Shift+C), or the responsive button next to, to make sure things are scaling with different screen sizes.
- **Also,** when an element is selected in HTML, you can type $($0) in the console to make that object a jQuery object (and then manipulate it in the console if you want)

JavaScript is also where we control modals, toasts, etc. There's a lot to do with JavaScript, so make sure to read some good tutorials. Honestly, use https://www.w3schools.com/js/default.asp for JavaScript tutorials or really anything else.

Controllers

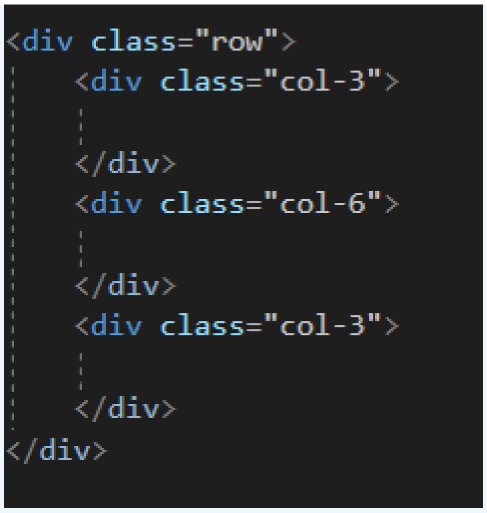
There's a lot here as well. You still use # or . to find elements in HTML with id or class, and you can use the same specifying that you use inside a jQuery (ex. $('button[disabled]) in JS would be button[disabled] { // some styles } in CSS).

A lot of the time, Bootstrap (CSS library) has classes that make a lot of CSS for you. Here's a list of **VERY HELPFUL** classes that bootstrap provides.

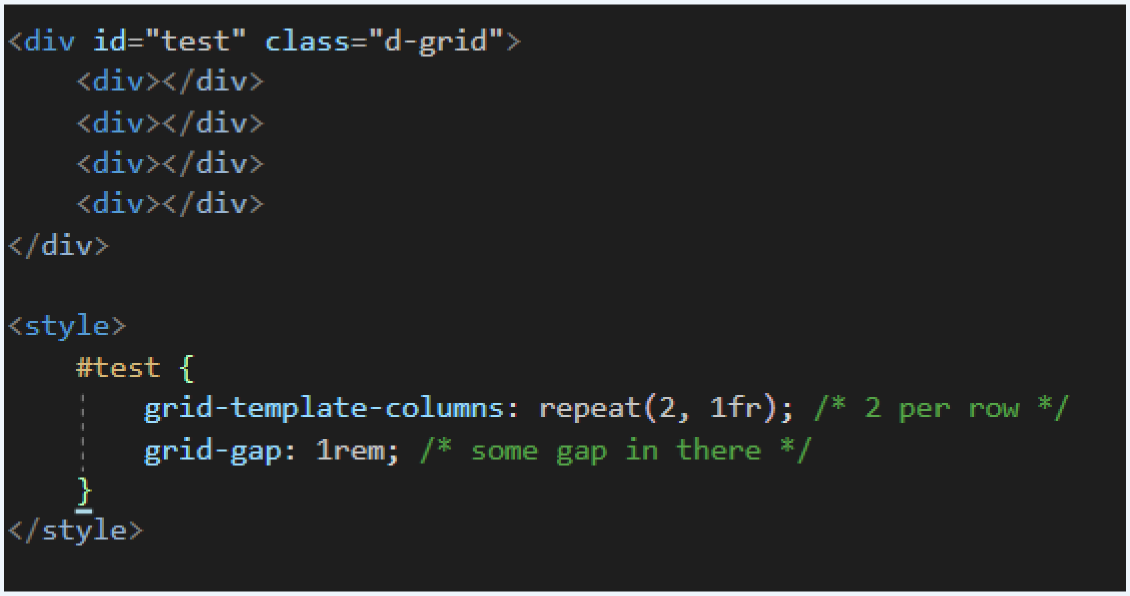
- Display: d-flex, d-none, d-grid, etc.
- Margin: mt-3 (margin top 3 == 1rem), mx-auto (margin left and right auto, centers stuff), mb-5 (margin bottom 5 == 3rem)
- Padding: same as above
- text-white, text-danger, text-primary
- btn-primary, btn-block, btn-danger
- container-fluid (goes around an object to make it responsive)
- card, card-caption
- position-relative, position-absolute, etc.
- justify-content-center, etc
- w-100 (width 100%), w-75, h-100, h-75, mh-75 (min height 75%)

There are also font awesome icons, in an <i> tag that have the class "fad fa-pencil", or something similar.

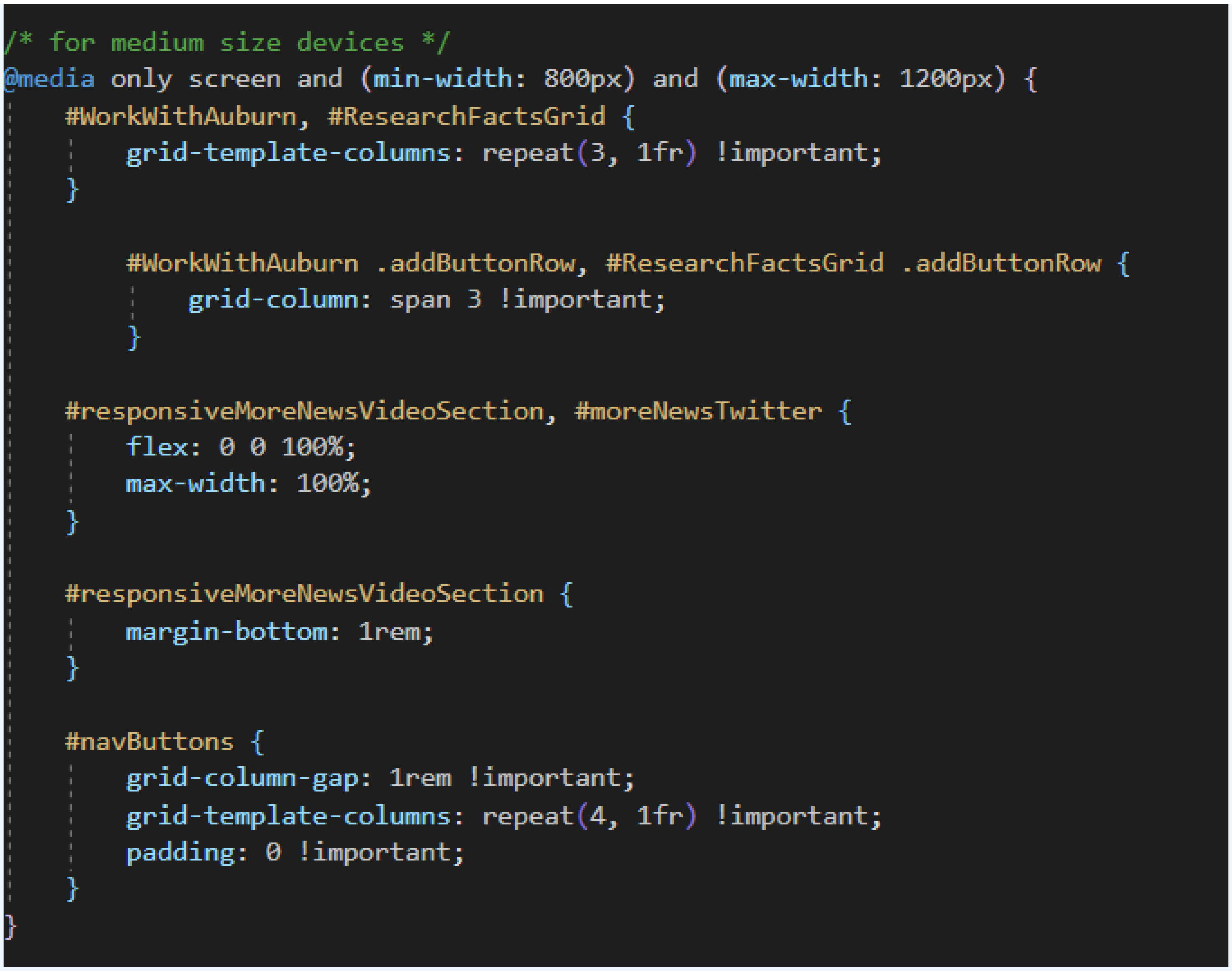
You can define a div with class "row", which will constitute a row on a page. Within that row, it's broken up into 12 columns, which you can specify. Here is an example of making a row with three divs. The first one stretches from 0 to 25%, the second from 25% to 75%, and the third from 75% to 100%.


```html
<div class="row">
    <div class="col-3">

    </div>
    <div class="col-6">

    </div>
    <div class="col-3">

    </div>
</div>
```

Grids are used variously. For a grid, you could have:


```html
<div id="test" class="d-grid">
    <div></div>
    <div></div>
    <div></div>
    <div></div>
</div>

<style>
    #test {
        grid-template-columns: repeat(2, 1fr); /* 2 per row */
        grid-gap: 1rem; /* some gap in there */
    }
</style>
```

Media queries let you apply styles only for screens of a certain size



```
/* for medium size devices */
@media only screen and (min-width: 800px) and (max-width: 1200px) {
    #WorkWithAuburn, #ResearchFactsGrid {
        grid-template-columns: repeat(3, 1fr) !important;
    }

        #WorkWithAuburn .addButtonRow, #ResearchFactsGrid .addButtonRow {
            grid-column: span 3 !important;
        }

    #responsiveMoreNewsVideoSection, #moreNewsTwitter {
        flex: 0 0 100%;
        max-width: 100%;
    }

    #responsiveMoreNewsVideoSection {
        margin-bottom: 1rem;
    }

    #navButtons {
        grid-column-gap: 1rem !important;
        grid-template-columns: repeat(4, 1fr) !important;
        padding: 0 !important;
    }
}
```

This means screens with a min-width of 800px and max-width of 1200px

Remember specificity - CSS will override a style if it has more specificity. That means if I have

```
<div id="test" class="testClass">
```

and

```
#test { color: black; }

.testClass { color: white }
```

Then the color will be black because ID has more specificity.

Additional Information

Some good projects to look at for these tips in action are Event Ticketing and OVPR. Here are additional tips that were not mentioned above:

Helpful Visual Studio Shortcuts:

- Shift+Alt+W (inside a .cshtml, when highlighting an element) -> surround by div
- CTRL+M+O - collapse all regions
- CTRL+M+L - expand all regions
- F12 - go to definition (mouse cursor over method name)
- CTRL+K+D - format whole document
- CTRL+R+R - refactor, when mouse cursor over a variable name
- CTRL+D - duplicate line
- CTRL+K+C - comment out line
- CTRL+K+U - uncomment out line
- CTRL+C, CTRL+X, CTRL+V, etc. when not highlighting will just act on a specific line

How to Update Stored Procedure in your edmx (columns have been added/removed):

- EDMX -> Model Browser -> Function Imports
- Double click on the stored procedure -> Update -> OK -> Save

How to Revert File(s) in your Branch to the File(s) from Release:

- open git command line in Visual Studio
- type "git restore -s [Release_Fall2022] [Nursing/Nursing.Web/Views/Admin/Index.cshtml] (change the parts in square brackets, remove the brackets)

Miscellaneous:

- Hot reloading in Visual Studio will only help for some controller changes. If its javascript or CSS, you have to hard reload the page.
- There are good extensions (especially Viasfora and output enhancer) for Visual Studio.
- Breakpoints set in Visual Studio can be made conditional, among other options, by right clicking it.
- You can change the environment database by selecting the dropdown at the top, left of IIS Express.

- You can say [HttpPost] and [AuthorizeUser("Admin")] above methods for authentication.
- I recommend you having your Visual Studio tabs on the left (stacked vertically) and making use of the "Pinned" feature.
- If you want to move data between databases (*not* the schema, like tables or etc.) then you can go to Tools->Sql Server->New Data Comparison in Visual Studio

Specific Technologies

Bootstrap Multiselect

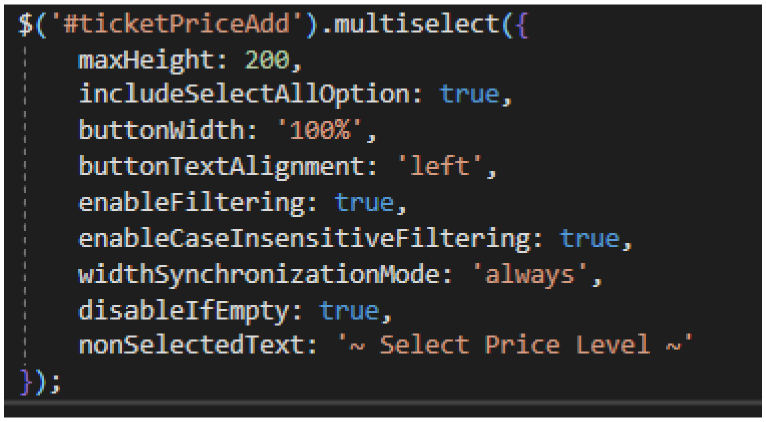
Long version here. Short version:

1. Link CSS and JS to your page

```
<link rel="stylesheet" href="https://unpkg.com/bootstrap-multiselect@1.1.0/dist/css/bootstrap-multiselect.css" />
```

```
<script
src="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/js/bootstrap.min.js" integrity="sha384-JjSmVgyd0p3pXB1rRibZUAYoIIy6OrQ6VrjIEaFf/nJGzIxFDsf4x0xIM+B07jRM" crossorigin="anonymous"></script>
<script src="https://unpkg.com/bootstrap-multiselect@1.1.0/dist/js/bootstrap-multiselect.js">
</script>
```

2. Add multiple="multiple" to your select tag (in HTML)
3. Initiate your multiselect in the document ready in your JS file:



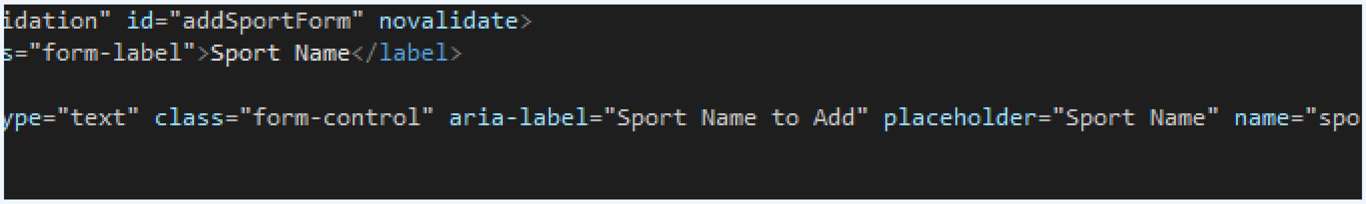
```
$('#ticketPriceAdd').multiselect({
    maxHeight: 200,
    includeSelectAllOption: true,
    buttonWidth: '100%',
    buttonTextAlignment: 'left',
    enableFiltering: true,
    enableCaseInsensitiveFiltering: true,
    widthSynchronizationMode: 'always',
    disableIfEmpty: true,
    nonSelectedText: '~ Select Price Level ~'
});
```

Initating the multiselect

4. Go to https://davidstutz.github.io/bootstrap-multiselect/ for more info

Bootstrap Validation

Long version here. Short version:

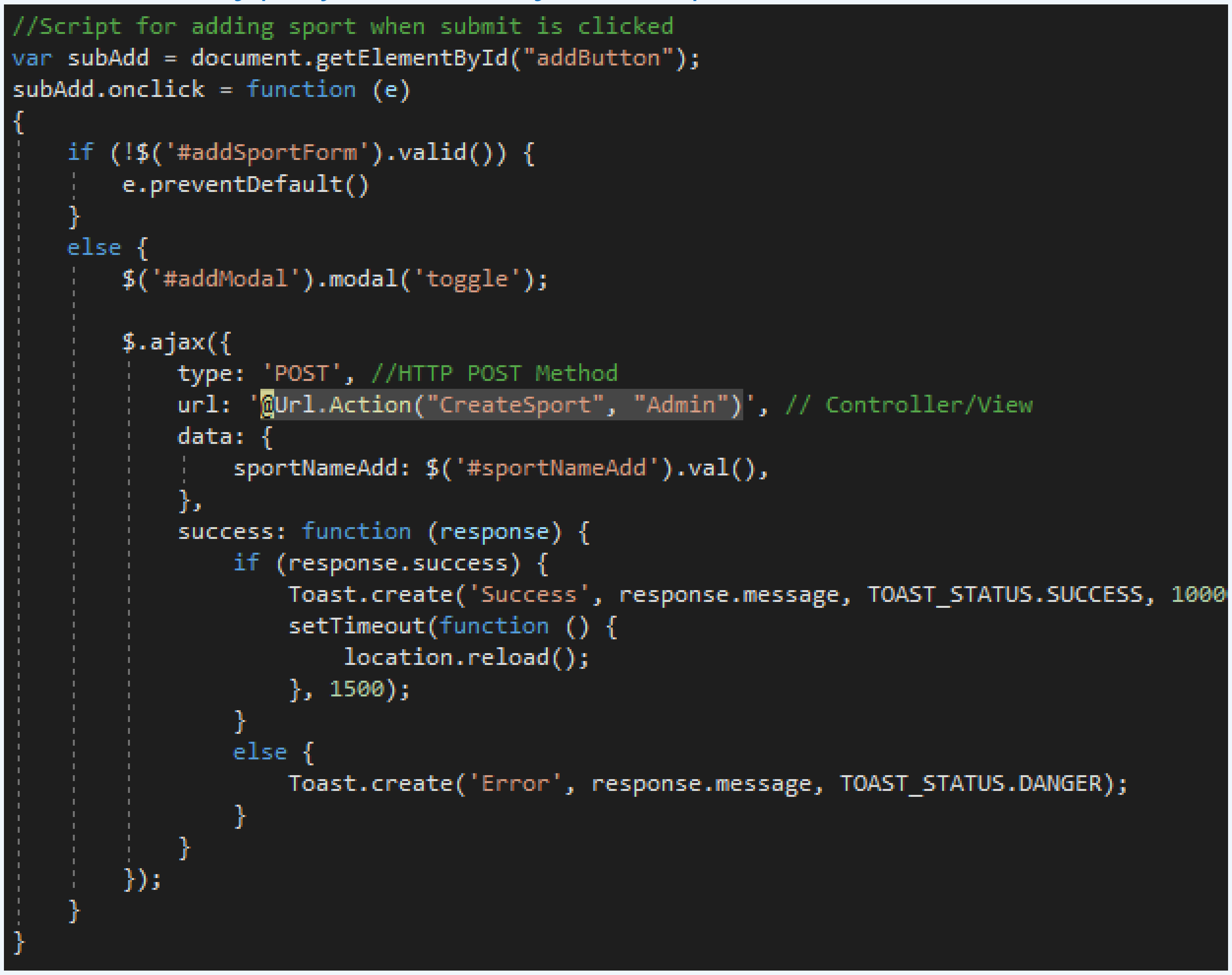


```
idation" id="addSportForm" novalidate>
s="form-label">Sport Name</label>

ype="text" class="form-control" aria-label="Sport Name to Add" placeholder="Sport Name" name="spo
```

1. Your form needs to look like the one above. The form needs **class="needs-validation"** and **novalidate**. The input needs a **name="*name*"** and a **required.**
2. Make sure your submit button doesn't have data-dismiss="modal" and make sure it does have **type="submit"**
3. Copy this CSS

```
input.error {
    border: solid 2px #FF0000;
}
label.error {
    color: #FF0000;
}
```

4. Link JS <script src="https://cdnjs.cloudflare.com/ajax/libs/jquery-validate/1.14.0/jquery.validate.min.js"></script>



```
//Script for adding sport when submit is clicked
var subAdd = document.getElementById("addButton");
subAdd.onclick = function (e)
{
    if (!$('#addSportForm').valid()) {
        e.preventDefault()
    }
    else {
        $('#addModal').modal('toggle');

        $.ajax({
            type: 'POST', //HTTP POST Method
            url: '@Url.Action("CreateSport", "Admin")', // Controller/View
            data: {
                sportNameAdd: $('#sportNameAdd').val(),
            },
            success: function (response) {
                if (response.success) {
                    Toast.create('Success', response.message, TOAST_STATUS.SUCCESS, 1000
                    setTimeout(function () {
                        location.reload();
                    }, 1500);
                }
                else {
                    Toast.create('Error', response.message, TOAST_STATUS.DANGER);
                }
            }
        });
    }
}
```

5. Add required JavaScript. The difference here is that you are calling **.valid()** on a form. Only modification I would make to the above image is put the modal toggle inside the ajax success.

Other Technologies

Written by Isaac Weiss on 12/16/22