**FINAL / FORMAL REQUIREMENTS FOR PROJECT 3 / FINAL**

**The solution “Project3\_FinalExam” contained in the “Project3\_StarterCode” folder in this lecture should be used as the starting point for creating “Project 3.”**

Some material in this section originally appeared in the lecture notes for Lecture 26: “Lecture\_26\_Transcirpt.docx.”

*This “final” statement of requirements contains some minor additions to those lecture notes.*

1. **“ist.rit.edu/api” endpoints from which you MUST retrieve data:**
   1. “about” – should be used on the “home page.” See **Scoring for the “about” endpoint** later in this section.
   2. “people/faculty” – must display the following information:
      1. “name”
      2. “username”
      3. “title”
      4. “office”
      5. “phone”
      6. “email”
      7. Picture – picture is obtained from the value of the “imagePath” property.
   3. “degrees”
      1. “undergraduate” must include:
         1. “degreeName”
         2. “title”
         3. “description”
         4. “concentrations”
      2. “graduate” must include:
         1. “degreeName”
         2. “title”
         3. “description”
         4. “concentrations” and for the last graduate degree returned in the array of “graduate:“ “availableCertificates”
   4. “minors” (for undergraduate) must include:
      1. “name”
      2. “title”
      3. “description”
      4. “courses”
   5. “employment” Note: See ‘Scoring for the “employment” endpoint’later in this section.
      1. “coopTable” in tabular form
      2. “employmentTable” in tabular form
      3. **15 Points extra credit:** use Chart.js to display the number of hires for each undergraduate degree. This will require you to write code to extract the *appropriate data from the “employmentTable.”*

*Continued next page.*

**Scoring for Endpoint Data**

1. Retrieving endpoint data and displaying it as “plain text” is necessary for a grade of “B.” See example below using “Undergraduate Programs:”

Graphical user interface, text, application, email

Description automatically generated

1. Retrieving endpoint data and displaying it using a jQuery plugin will add extra points to the project grade. See example below using “Undergraduate Programs:”  
   Graphical user interface, application

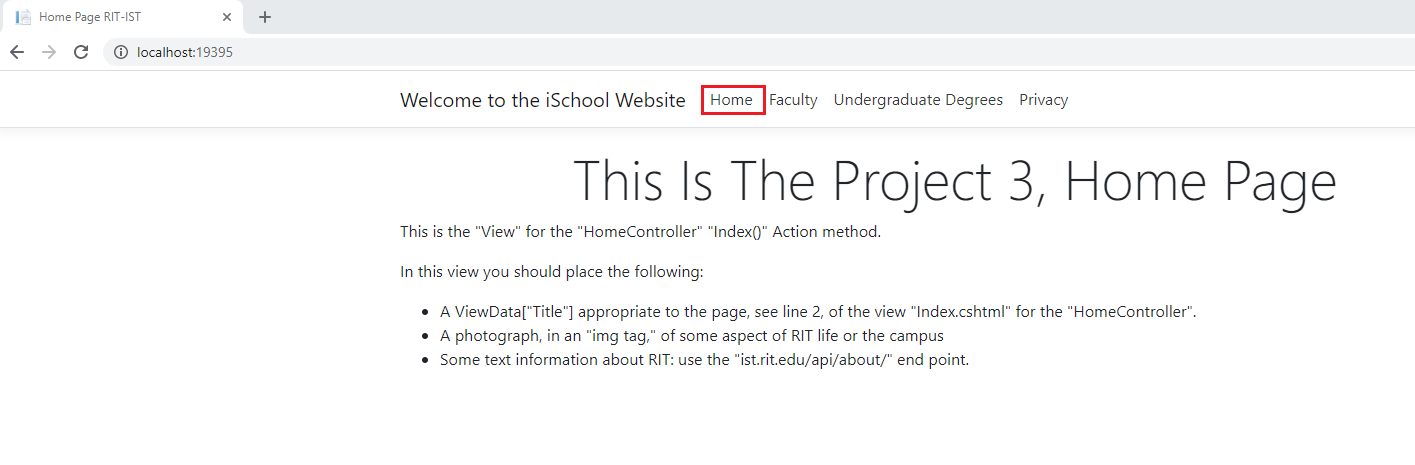
   Description automatically generated

**Scoring for the “employment” endpoint “coopTable” and “employmentTable.”**

This data **must** be displayed in tabular form. If you display them by constructing an HTML <table>, you will satisfy the requirements for a “B.” If you display them using jsGrid or a similar jQuery plugin for displaying tabular data in a formatted grid form, you will get extra credit.

**Scoring for the “about” endpoint:**

The information returned from the “about” endpoint should be incorporated into the “Home” page for the project. If you click on the “Home” menu item across the top of the web page displayed by running the “Project3\_FinalExam” solution supplied as the starting point for Project 3, you will see the following:



You need to construct the “Home” page in accordance with the instructions displayed on the “default” version of the “Home” page as a necessary requirement for a “B” on the project.

You can choose from a variety of “RIT” photographs at:

<https://www.dropbox.com/sh/jysmitr9yratfwd/AACkBfFb_MFD_ylsd0ZId_a3a?dl=0>

You should create an “images” folder in the “wwwroot” folder of your solution and store phtotograph(s) from the above RIT website there.



For extra credit you can load several photographs into the “images” folder and use one of the many available jQuery “slide show” plugins to display them.

**Requirements for “\_Layout.cshtml” Shared View**

The “Project3\_FinalExam” solution supplied as the starting point for Project 3” has been modified with some of the changes required for Project 3.

The section of this “view” that you will have to modify as you continue the development of the solution is the navigation menu which is displayed across the top of each web page. Currently it looks like this:

A picture containing text

Description automatically generated

As you can see it already has the menu items for:

* “Home”
* “Faculty”
* “Undergraduate Degrees”

You will have to add the appropriate menu items for the remaining required endpoints. Doing so is necessary to obtain a “B” for the project.

*Continued next page.*

**Project Requirements Concerning the Use of Dependency Injection**

As has been stated many time, “Dependency Injection” is the most important topic for this section of the course, and it is the focus of “Project 3.” To satisfy the project requirements concerned with “Dependency Injection” you must create and use the constructs listed below.

***Note –*** *the code supplied for the “HomeController” “GetFaculty()” “IActionResult” all code related to it should be used as a model. You must be thoroughly familiar with the material contained in:*

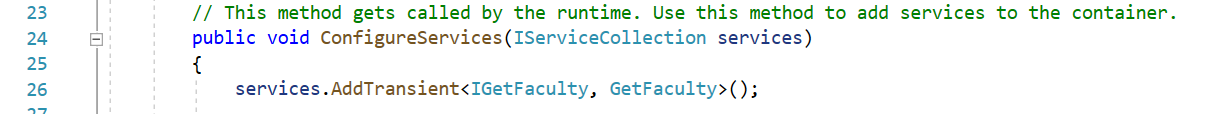
* *Lecture 21*
* *Lecture 24*
* *Lecture 24A*
* *Lecture 25*
* *Lecture 26*

*to succeed. They contain detailed, step by step instructions for the items listed below.*

*In particular, the MS Word document “Developing\_Dependency\_Injection” in “Lecture24-DependencyInjection” in the course “Content” contains detailed step by step instructions for what you must do to implement dependency injection for each api endpoint you are retrieving data from.*

At a “high level” for each endpoint from which you are retrieving data you will need:

For each api endpoint to be accessed you need to:

* Add a “model” for the endpoint data you are retrieving in the “Solution Explorer” “Models” folder.
* Using the above “model,” in the “Solution Explorer” “Services” folder create a “class” with the appropriate “method” to retrieve the endpoint data. Make sure the method is of type “async/await.”
* Refactor the “class” just created in the “Services” folder to extract an “Interface” for the class. This “Interface” abstracts the dependency implementation, i.e. our “Task<IActionResult> GetFaculty()” method will not be dependent on the “GetFaculty” class we defined but rather on any class implementing the “Interface” extracted from the “GetFaculty” class.
* Using the “Interface” for the class and a concrete class which inherits the interface register the “dependency implementation” with a new “services.AddTransient…” entry in the “Startup.cs” class – this creates a new “service” in the “service container.  
  The example for the “IGetFaculty” interface and the “GetFaculty” concrete class shown below registers the “IGetFaculty” service with the “GetFaculty” class. This will result in a new “service” named “IGetFaculty.” *When that service name is used in the “Constructor” of a class the ASP.NET Core framework assumes the responsibility of creating an instance of the dependency.*  
  
* In the “ViewModels” folder in the “Solution Explorer” create a “view model” for the endpoint data you are retrieving.
* In the “HomeController” create a new “Tack<IActionResult>” to call the method created to retrieve the endpoint data.

The above step can be one of the “trickiest” ones you will have to perform. Below is a screen shot of the “HomeController” with the “Dependency Injection” for the “GetFaculty()” “Task<IActionResult>” created; this is the “HomeController” in the supplied “starter code.”



The first thing to notice is the statement on line 16:



We are declaring a “private readonly” member variable of type “IGetFaculty,” the “Interface” we refactored out of the “GetFaculty” class we created earlier. Marking a member variable “private” means that it cannot be referenced outside of the class it is defined in. *More importantly for our purpose is that marking it “readonly” declares it as a constant but allows the value of the constant to be declared at runtime.*

Lines 18 through 21 contain the “Constructor” for the “HomeController” class. The appearance in the “Constructor” of “IGetFaculty” means that the “Constructor” is passed an “instance” of the “IGetFaculty” service from the “service container,” in our code the instance is assigned the name “facultyRepository.” This instance is assigned to the private readonly member variable “\_facultyRepository.” Once the “Constructor” runs and the assignment is made to “\_facultyRepository” the value of “\_facultyRepository” cannot be changed. This ensures that any code in our “HomeController” which utilizes the method(s) contained in the instance of “IGetFaculty” passed to the “Constructor,” “facultyRepository,” will always be using the same instance of “IGetFaculty.”

**Reminder:** the mapping of an instance of a class which inherits the “IGetFaculty” “Interface” to the “IGetFaculty” “service” is performed in the “Startup.cs”

public void ConfigureServices(IServiceCollection services)

method using

services.AddTransient<IGetFaculty, GetFaculty>();

* The last step in the process is creation of the “View” to display the endpoint data. This is well documented in the material highlighted at the beginning of this document.  
  Lines 32 through 37 in the above code snippet show how to pass the “view model” to the view to be displayed to the end user.

Following the above steps for each endpoint for which you are required to display data will get the job done. Again, use the existing code for “GetFaculty” as a model*. Also, keep in mind that while the starter project contains code to display “undergraduate degrees” in a jQueryUI “Accordion” “Dependency Injection” for “undergraduate degrees” has not been implemented. You must complete that work.*

*Continued next page.*

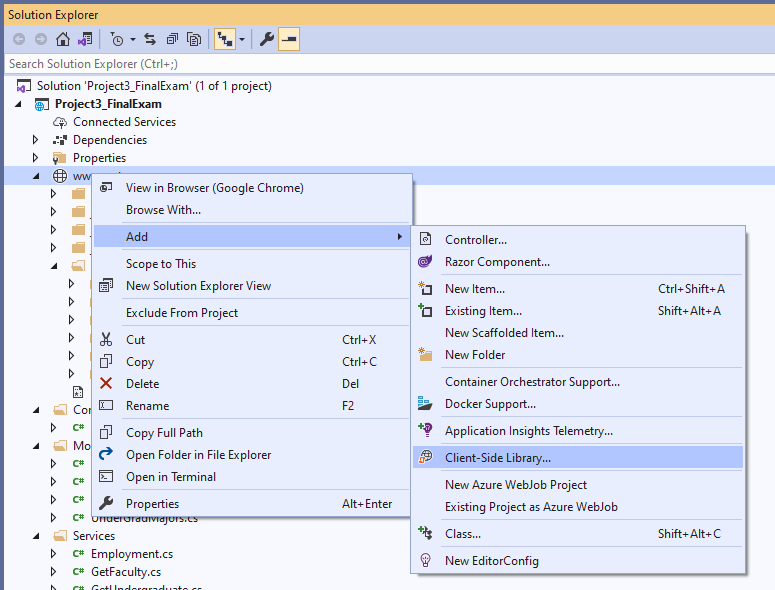
**jsGrid**

One of the requirements for Project 3, is to show the data obtained from the “coopTable” and “employmentTable” data obtained from the “employment” ist.rit.edu/api endpoint in a table. As was suggested for Project 2, the jQuery plugin “jsGrid” is good vehicle for displaying the data in a table. There are others available, if you want to take the time to investigate.

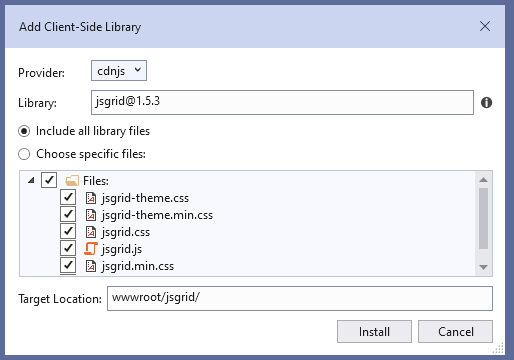
If you revisit ‘Lecture 17 – Widget Tutorial: “Accordian” & “Tab,”’ in the “CodeExamples” folder you will find the file “EmploymentGrid.html.” On the “.cshtml” page you create for the employment table you will have to add the appropriate <script> and <link> tags for “jQuery” and “jQueryUI” in front of the <script> block you use to create the “jsGrid.” Additionally, you are going to have to add <script> and <link> tags specific to “jsGrid,” see lines 17, through 21, “EmploymentGrid.html.”

In “Lecture25 Everything You Need To Know For Project3” it was explained how to add the “jsQuery” and “jsQueryUI” libraries to the “wwwroot” folder so that the code in the website can obtain the libraries via the “middleware pipeline.” To make use of “jsGrid” we have to also add the “jsGrid” library to the “wwwroot” folder. To do so, follow the same procedure that was outlined in Lecture 25.

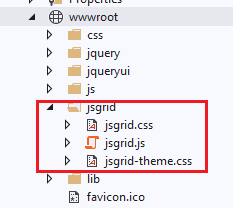
In the Visual Studio “Solution Explorer” right-click on “wwwroot” and from the dropdown menu click on “Add” and on the next dropdown menu click on “Client-Side Library…”



In the “Add Client-Side Library” panel that appears type “jsgrid” into the “Library:” text box, <Type to search> and click on the highlighted “jsgrid” that appears below the textbox you type into. You should see the following



Then click on “Install.” You will now see “jsgrid” under the “wwwroot” folder. In the screen shot below I have expanded the “jsgrid” entry.



You can now include jsGrid on you “.cshtml” pages the same way “jQuery” and “jQueryUI” were included. Again, see Lecture 25, if you need to review all of that. To create the page with the employment data table you will have to:

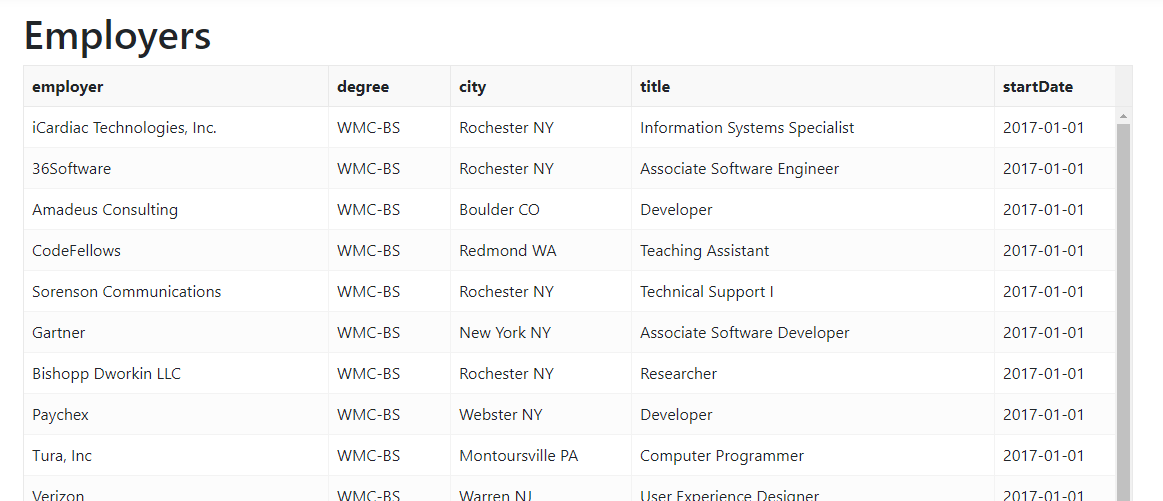
* Create a model for the employment data in the “Model” folder.
* Create a get employment data class in the “Service” folder.
* Create an interface class for the employment data class.
* Create a view model for the employment data in the “ViewModels” folder
* Create an employment data action item in the “HomeController” using dependency injection – don’t forget to add the appropriate mapping for the dependency injection in the “Startup.cs” class.
* Create the “.cshtml” view to display the employment table.
* Finally, update the website menu on the “\_Layout.cshtml” file in the “Veiws/Share” folder to add a menu entry for the employment table.

Hint – in the “.cshtml” view use @Html.Raw(Json.Serialize(Model.[employtment information]))

to add the employment data to “jsGrid.” In the $(“jsGrid”).jsGrid({… block you would have

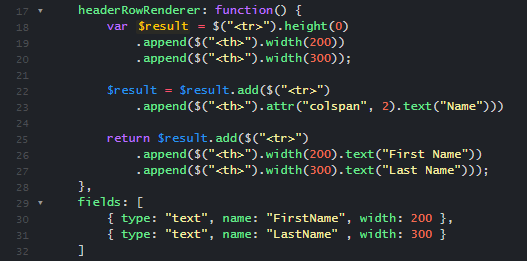
data: @Html.Raw(Json.Serialize(Model.[employment information]).

When you succeed in performing the above clicking on the menu item for the employment table will bring up the following grid (partial screen shot):



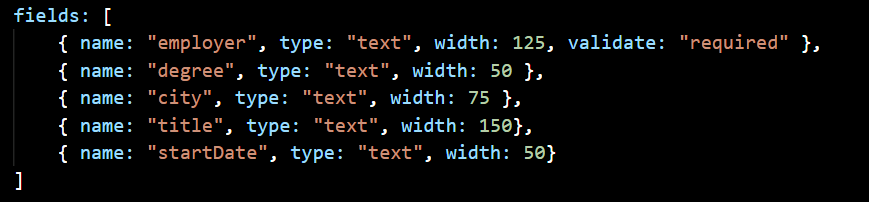
As you saw in Project 2, the column headings come from the properties in the employment table model. Adding “custom” column names on a “jsGrid” table takes some work. You have to create a function that returns html for the header with the function being assigned to the “jsGrid” “headerRowRenderer” property.

You can see an example of such a function at “[jsGrid Complex Header](https://jsfiddle.net/tabalinas/k9935wn3/)” In the lower left quadrant of the example you will find:



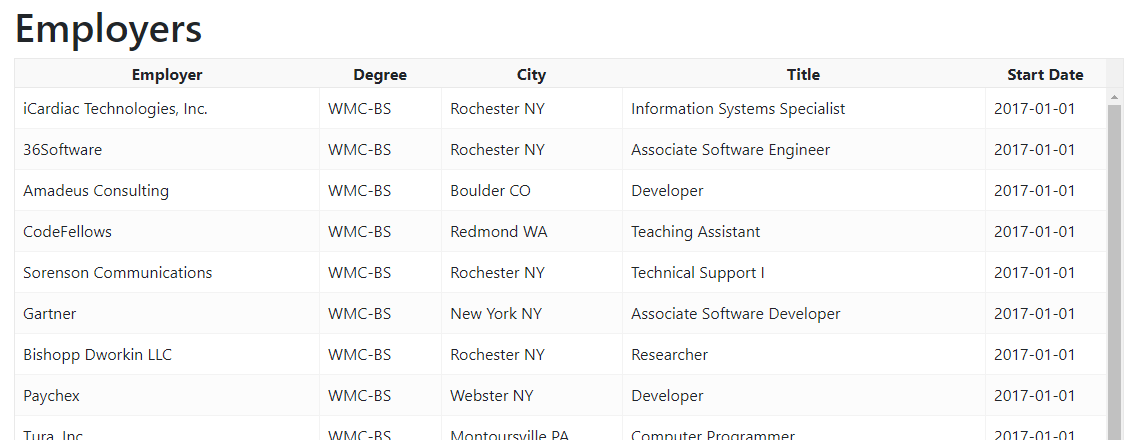
“Throw out” the code on lines 22, and 23.

What you need to do is to add one “.append…” for each table column that appears in the “fields” array in the “jsGrid” block being used for the employment table. The widths you use are the widths shown in the “fields” array. Below is the “fields” array from the jsGrid example referred to above. The heading text you use should be based on the “name” property in the “fields” array, e.g. for “startData” you should use “Start Date.”



One final hint – you can insert a “.css” to add some styling to each custom column header, it can be inserted right before the “.text(…” You can add styling to align the text, enhance the font, etc.

With the “headerRowRenderer” function added the grid can now look like this, partial screen shot:



If you want an alternative to jsGrid, investigate jQuery jQGrid. It is also available as a “Client-Side Library.” Your familiarity with jQuery and jQueryUI may make it easier to implement. Some samples can be found at:

<https://www.js-tutorials.com/jquery-tutorials/beautiful-jqgrid-example-demo-using-bootstrap-jquery-ui/>

**Conclusion:**

These lecture notes and the ones from previous lectures cover everything you must do to satisfy the requirements for Project 3. The inclusion of appropriate graphics on the various view pages, they are required on the home page, will of course make your website more professional looking – remember the view pages are basically html pages so including graphics is straight forward. Taking the time to place “meaningful headers” on the employment grid will also help make the website look more professional.

**A final note – please do not overuse the jQueryUI accordion.**

As Tom Waits said: *“A gentleman is someone who can play the accordion, but doesn’t.”*

Be creative. Any questions – please do not hesitate to reach out to me.