## Technical Projects:

## 1) Added dollar signs in front of each of the input fields in the Lump Sum Analyzer.

## Approach #1:

## HOW: Went into charts.ftl and added <label for="${inputId}">$</label> at line 883.

## LEARNED: It seems like all the input boxes here are following a standard template defined by the <td class = "align-center colBaseValue end-input"> in charts.ftl. At line 884, it shows that the inputs are generalized by "${inputId}". So adding a label in front of it gave it to all instances of that class.

## QUESTION: Does this mean that, if I were to push my code, all other files pulling from charts.ftl would now get $?

Yes. Need a different approach.

Approach #2:

* HOW: Went into lump-sum-analyzer.js and created an array of all the id’s of the input fields. Then wrote the following to line 441:

**for** (**var** i = 0; i < inputFields.length; i++) {

**var** input = document.getElementById(inputFields[i]);

**var** text = document.createTextNode("$");

input.parentNode.insertBefore(text, input);

}

For styling purposes, also went to line 103 in lump-sum-analyzer.css and changed the width of the input fields to 55% (from 70% originally).

* LEARNED: How to create text nodes to insert text into HTML via JavaScript. Also, to get the parent node of a Node object, there’s actually a field/property called “parentNode.”

## 2) Get rid of the "Joint-Life Annuity" option for single people in the Lump Sum Analyzer.

Approach #1:

## HOW: In display-config.xml, I replaced Line 3243 -

## <column active="true" visible="true" name="annuity" formatType="currency" group="yourPensionOptions" allowZero="true" includeSign="true"/>

## <column active="false" visible="false" name="annuity" formatType="currency" group="yourPensionOptions" allowZero="true" includeSign="true"/>

* PROBLEM: Got rid of the whole line even when the user selected the marriage option.

Approach #2:

* HOW: In lump-sum-analyzer.js in the initHorizontalSliders function, added the lines below. This sees if currently the isMarried attribute is false, and looks for the input-name “annuity” in the outerHTML of the “this” object, which is being iterated over.

**if** (!LSA.isMarried && **this**["outerHTML"].includes("annuity")) {

**return** **true**;

}

## PROBLEM: It deleted just the sliders and all of them even if married.

Approach #3:

* HOW: In lump-sum-analyzer.js in the toggleMarried function, I added inside the “if” clause: $("tr[data-colname='annuity']").toggle();

Here, the “tr” refers to the type of node or object it is: a table row. If you notice, in the HTML it’s under the <tbody> identified by “YourPensionOptions,” and that itself is under its <table> parent.

I also added $("tr[data-colname='annuity']").hide(); in the initial script that is run after the document is ready, since the page is first loaded under “single”.

* LEARNED:
  + $(<name>) is used as a jQuery selector for various elements. For a library of different types of selectors, see: <http://www.w3schools.com/jquery/jquery_ref_selectors.asp>
  + To select on custom data attributes, can do one of the two:
    - Document.querySelectorAll(‘[data-<attr>]’); (Note: This returns a list that can be indexed using [].)
    - <http://stackoverflow.com/questions/4146502/jquery-selectors-on-custom-data-attributes-using-html5>
  + To find the attributes of an element, can just iterate over <element>.attributes.
  + To remove a row of a Table object, use .deleteRow(<index>). (Note: The rows of the table all shift over after the deletion.
  + To iterate over the properties (not attributes) of an object, use “for (var property in object) {}.”
  + To access and change a .xml file: <http://www.w3schools.com/xml/dom_httprequest.asp>

**3) Modify the “Singles” graph to not include an “Annuity” bar.**

Approach #1:

* HOW: In line 15 of lump-sum-analyzer.js, I removed the ‘Annuity’ label frm the singleCategory array. However, this only removed the label from the graph but not the actual data or bar visualization. To handle this, I went to the “createChartData” function in lump-sum-analyzer.js.

The data being returned is in a variable called “dataToReturn”—an array of data. And note that if the person is single, then the data is being returned as the variables yourPension, yourAnnuity, and yourLumpSum in dataToReturn[0]. Therefore, I just removed the yourAnnuity data variable and it worked.

* LEARNED: That the graphics and labels often are separately handled from the actual data handling itself.

**4) Modify the legend of the lump scenarios graph to ensure that it doesn’t show “Your Spouse” on the singles page.**

Approach #1:

* HOW: Was able to reference / select the “Your Spouse” legend item using a jQuery selector: $(“g[transform=’translate(8,21)’]”). However, in order to find when I should call .hide() or .show(), I had to use the Chrome Developer Tools debugger to follow until I found the very last call to LumpSumAnalyzer.drawScenariosGraph(response) on line 690. Since “Your Spouse” would always be showing there, I inserted the code:

**var** current = $('.marriage-option.selected').attr('id');

**if** (current == "single") {

$("g[transform='translate(8,21)']").hide();

} **else** {

$("g[transform='translate(8,21)']").show();

}

Note: Placing this in the toggleMarried function doesn’t work since toggleMarried is called before all the drawScenarios, which will just undo the hide().

* LEARNED: Through jQuery, can find an HTML element using pretty much any attribute, such as “transform” in this example. Also learned how to better use Chrome Developer Tools to follow this path.

Approach #2:

* HOW: Went into the drawScenariosGraph function in lump-sum-analyzer.js, and changed the “series” data. The series data was essentially an array of two Objects—one for “You” and one for “Your Spouse.” Consequently, it was just a matter of using an “if this.isMarried” to determine whether to only have one object or add the second “Your Spouse” object.
* LEARNED: It’s best to just go straight to the source of where the graphs are being created rather than to try handling each case that calls drawScenariosGraph.

**5) Modify the buttons for the timeline so that the spouse options don’t show up on the singles page.**

Approach #1:

* HOW: Essentially just needed to add two lines to the toggleMarried function in lump-sum-analyzer.js and two lines in the “afterDocumentReady” function. It’s the same logic as before:

$("div[value='spousePassAway']").toggle();

$("div[value='spousePensionDefaults']").toggle();

**6) Style the CSS so that the “Your Pension Defaults” buttons and such are separated from the ones for married couples.**

Approach #1:

* HOW: After selecting the buttons, via a CSS selector, set position : relative and play around with the “top” and “left” attributes to move the buttons around.
* PROBLEM: Because the buttons were all relative, they somehow began pushing each other around as you moved one. I couldn’t keep one in the place that I wanted it.

Approach #2:

* HOW: After selecting buttons via a CSS selector, I instead set position : absolute. This option means that the child elements (the buttons) would remain in an absolute position relative to their parent (the timeline). I then went and set the timeline position : relative.

Note: Also changed the button’s “toggle” to “fadeToggle” and the “hide” and “show” to “fadeOut” and “fadeIn” for aesthetics.

* LEARNED:
  + How to use a CSS selector (note that the attribute selector must come before the class selector (ex. div[value=”text”]): <http://www.w3schools.com/cssref/css_selectors.asp> and

<http://code.tutsplus.com/tutorials/the-30-css-selectors-you-must-memorize--net-16048>

* + How to position HTML elements using left, top, and margin-left, margin-top: <http://www.tutorialspoint.com/css/css_positioning.htm> and

<http://www.w3schools.com/css/css_positioning.asp>

**7) Convert from the singles and spouse buttons to two drop-down menus.**

Approach #1:

* HOW: Began by looking up a tutorial online of how to make a dropdown menu, which served as my starting template: <http://www.w3schools.com/howto/howto_js_dropdown.asp>

One key was to use the file “monthly-payments.ftl,” which allowed me to create my drop-down buttons as children of the “timelineWrapper” div on line 1. With this, I could use absolute positioning to position my buttons correctly.

To place the original four buttons within these two dropdown buttons, I placed the classes that create the original four buttons as children underneath the dropdown div. The template code would then search through these children and, through toggle(), would change their attribute to “display : block” from “display : none”. The rest of this was just CSS styling.

\*Note: One key piece of debugging was getting rid of the $(div[value=”spousePassesAway”]).hide() selectors in the beginning of the JavaScript, which were hiding the spouse buttons even when we clicked on the “Spouse Options” dropdown button.

* LEARNED:
  + How to make a dropdown menu and its buttons (via the link above).
  + The power of taking preexisting div classes representing elements and just placing them underneath other div classes to save work rather than reinventing those existing div classes.
  + How to round button corners using the “border-radius” property.
  + How to read “window.click” events and how “event.target.matches()” works to compare the click target to the element you’re trying to match with.
    - How “window.onload” runs the code immediately after the page loading.
  + How to splice a JavaScript array: <http://www.w3schools.com/jsref/jsref_splice.asp>

**8) Perform a backend query in order to get the ten-year annualized return of a ticker.**

Approach #1:

* HOW: Saw that there was a full request URL in the Aladdin API-Tester, and decided to make an XML HTTP Request to that URL via this code on line 321 in lump-sum-analzyer.js :

**var** name = $('#newTicker').val();

**var** xmlHttp = **new** XMLHttpRequest();

**var** spUrl = "http://localhost:7109/tools/services/portfolio-analysis?calculateExposures=true&calculatePerformance=true&positions=" + name + "~100&useCache=true";

xmlHttp.open("GET", spUrl, **false**);

xmlHttp.send(**null**);

From there, I parsed the output into a JSON using JSON.parse(xmlHttp.responseText). Then it was simply a matter of indexing into the right parts of the JSON using things such as [“resultMap”]. The structure of the JSON followed that shown in the API-tester. And I was able to get ten year annualized returns from this.

With this, it was then just a matter of changing the IRR graph options in “getIrrGraphOpts” so that it would graph a line representing the ten year annualized return for the S&P 500 using the following code:

fullOpts.yAxis.plotLines = [

{

value : spValue,

color : 'orange',

dashStyle : 'shortdash',

width : 2,

label : {

text : name + ' Ten Year Annualized

}

}

];

* LEARNED: How to make an XML Http Request to a URL via JavaScript and to parse the output into readable / usable format using the JSON.parse method. Also learned how to take and manipulate HighCharts options / graph features via JavaScript (as shown above).

Approach #2:

* HOW: Was shown how ApiTester.js used an .ajax query to get the backend data. What was particularly interesting was that the URL used was just “/tools/json/performance-data.” And then Spring would map this via the @RequestMapping to the PortfolioCalcController.java file which would then handle everything else including inputting the hostname. Consequently, I then made the following .ajax query in lump-sum-analyzer.js:

$.ajax({

type : 'GET',

url : '/tools/json/performance',

data : {

"identifiers" : tickerName,

"useCache" : "true"

},

dataType : 'text',

success : **function**(response, status, xhr) {

**var** response\_json = JSON.parse(response);

**var** tenYearAnnualized = response\_json["resultMap"] ["RETURNS"][0]["latestPerf"]["tenYearAnnualized"];

LumpSumAnalyzer.plotTickerToIrr(tickerName, " Ten Year Annualized Return ", tenYearAnnualized);

},

error : **function**(response) {

console.log(response);

window.alert("Failed to retrive financial information.");

},

});

And because in a .ajax request you aren’t allowed to actually return anything, I had to define another function called plotTickerToIrr that would then take the ticker and ten-year-annualized return data and actually plot it to the graph.

NOTE: Also made a global variable irrTickerValues—an associative array. It basically works like a dictionary and allowed me to store the ticker values I got so that they no longer had to be requeryed each time.

* LEARNED:
  + How to make a .ajax request as well as all the different options available: <https://www.sitepoint.com/use-jquerys-ajax-function/> and <http://stackoverflow.com/questions/15576548/how-to-pass-parameters-in-get-requests-with-jquery>. Especially important was the “data” option, in which you include query-specific request parameters.
  + How to use the JSON.parse() method to take a text response from a backend query and parse it into an easiliy indexed JSON structure.

**9) Generalize the backend query to take in and remove any ticker line of ten-year annualized returns. Therefore, we need an input box for users to put in ticker names.**

Approach #1:

* HOW:
  + Tried adding another horizontal slider on the left. In order to get it to even appear on the left without erroring, I had to do the following:

Insert this at line 3249 in display-config.xml, imitating what occurred with the other horizontal sliders:

<column active="true" visible="true" name="tickerName" group="yourPensionOptions" cellTemplate="input" />

I then had to go into lump\_sum\_analyzer.xml and insert this at line 11:

<input name=*"tickerName"* baseValue=*"SPY"* />

However, this just created another horizontal slider. But once you hit submit, nothing renders because I suppose the code can’t handle the input to the new slider. It seemed more trouble than it was worth, so I went with another approach (besides, re-rendering the whole page for each change to the slider seemed unnecessary.

* LEARNED: A bit more regarding how XML files create new elements in websites. In this case, how to create columns and the purpose of the attributes such as “visible,” “name,” “group,” and “cellTemplate.” In addition, I learned how this line in display-config.xml mapped to the input line that needed to be specified in lump-sum-analyzer.xml.

Approach #2:

* HOW:
  + Recognized that the current IRR chart corresponded to a “tab” element in display-config.xml with the name “required-irr.” In fact, every tab element has been automatically synced with a .ftl file with a similar name. However, since the “required-irr” tab element had an attribute called “chartDisplayOnly” I wasn’t able to make anything more than just the HighCharts chart. So to include HTML elements, I instead used a tab called “irr” and modified an existing “irr.ftl” file and included input boxes with buttons that allowed you to take in any ticker that was inputted and add that to the HighCharts graph.

After that, I was able to just grab the value in text input box and perform the corresponding backend query.

* LEARNED:
  + How to better interpret XML files and write HTML in correspondence with XML elements. Also learned how to grab text input / values from text boxes using jQuery.

Approach #3:

* HOW: Although the previous approach worked and allowed us to display the ten-year annualized return of any ticker, it wasn’t very realistic from a user standpoint. The main things that needed to be compared to were just checkboxes for each of the major indices such as the S&P 500 and Russell 2000. Other than that, the same things were displayed.
* LEARNED:
  + Learned how to create checkboxes in HTML and about the .checked attribute to see if a checkbox was checked already: <http://stackoverflow.com/questions/901712/how-to-check-if-a-checkbox-is-checked-in-jquery>.
  + Also learned how to make, space, and color borders in CSS.
  + Didn’t end up using this, but learned that you can define your own HTML attributes for an element and then check them or set them using the .attr method.
  + Learned that the <span> tag is best for things that you don’t want to have their own line, like small words you just want to insert.
  + Learned that a space in a jQuery selector means that you’re looking at all descendants of that element rather than the typical format without a space.
  + Learned that you can also use “.click(function(event)” vs “.click(function()” since the former allows you to then use the “event.currentTarget” feature. Here, it allowed us to grab the “value” attribute of what we were clicking.
* QUESTION: The “chartDisplayOnly” attribute seems highly limiting. I was thinking about defining a new macro that would work for chartDisplayOnly and then making a required-irr.ftl file, but definitely wouldn’t have been the most efficient coding. Is there any way around chartDisplayOnly?

**10) Make an adjustment to the y-axis of the IRR graph. Make an interface that allows the user to adjust the range as needed and reset it.**

Approach #1:

* HOW: To make the needed interface, I just went onto line 14 of irr.ftl and typed out multiple <input> tags, some of which were “type = text” to input the range #’s and others were “type = submit” for buttons.

When the inputs were placed and the submit button was pressed, then used jQuery to select the input boxes and used .val() to get the range numbers. It was then quite simple to access the chart and use the “.setExtremes(x, y)” function in the HighCharts API to set the y-axis max and min according to the user input.

Also, used the FormatUtil.cleanAndParse function to filter out invalid inputs for the range, such as a string. Strings would cause the cleanAndParse function to return a NaN. I created a small <p> element that I would change the innerHTML of to show an error message.

To reset, it was essentially just changing the .customIrrRange attribute back to false and doing .setExtremes(null, null). The code is shown below:

$("#rangeSubmit").click(**function**() {

**var** min = $("input[name='min']").val();

**var** max = $("input[name='max']").val();

**if** (isNaN(min) || isNaN(max) || (max.length == 0) || (min.length == 0)) {

$("#badRange").html("Please insert a valid range.");

} **else** **if** ((min < -100) || (max > 100)) {

$("#badRange").html("Please keep your range within -100 to 100.");

} **else** {

$("#badRange").html(**null**);

**var** chart = $('#irrGraph').highcharts();

chart.yAxis[0].setExtremes(min, max);

LumpSumAnalyzer.customIrrRange = **true**;

}

});

$("#reset").click(**function**() {

**var** chart = $('#irrGraph').highcharts();

LumpSumAnalyzer.customIrrRange = **false**;

$("#badRange").html(**null**);

chart.yAxis[0].setExtremes(**null**, **null**);

});

NOTE: Defined a global variable called customIrrRange that indicated whether or not the user had adjusted the range. Until the user adjusted the range, the chart would autoadjust itself. However, the customIrrRange permission would be overridden if the user tries to draw a line that is NOT visible within the user’s set y-axis max’s and min’s.

**11) Create a new section at the bottom of the right-hand side that will contain the different buttons through which one can submit information to iRetire.**

Approach #1:

* HOW:
  + Went into display-config.xml and made a new tab on line 3352 called “toIRetire.” This would represent the new section on “columnRight”. I then placed HTML elements underneath as buttons that would link to different JavaScript functions I’d implemented: submitLumpToIRetire, submitPensionToIRetire, submitYourPensionSpouseLump, and submitYourLumpSpousePension.
  + In webtools\_en.properties, I then created lines 2033-2050 to essentially name the different elements and buttons I’d created.
  + Made a new file called toiretire.ftl, which was necessary to create any new tab by the way the system was set up.
  + The next step was to implement the functions in JavaScript:
    - One key variable was LumpSumAnalyzer.iRetireUrl on line 4, which I changed to ‘/tools/iretire#/allstep-2?’.
    - A helper function I created was getIndividualValue(), which took in two arguments: the metric you wanted (such as pension amounts) and who you wanted it for (spouse, self, or both).
    - Then, within each of the JavaScript functions linked to the buttons, it was simply a matter of using getIndividualValue() to get the needed pension or lump sum values.
  + Lastly, given those values acquired by the functions, then you concatenate onto the URL via the parameters “additionalIncome” and “savings”.
* LEARNED:
  + How to make a new tab and HTML elements via XML documents.
  + How to set URL parameters that will translate into actual effects upon the endpoint website (e.g., ? = query).
  + The window.open(url, “\_blank”) command to open up a new window for iRetire.

## Miscellaneous:

## 1) Added