**Element Quiz Documentation**

**Section 1.1 Generating Report Overview**

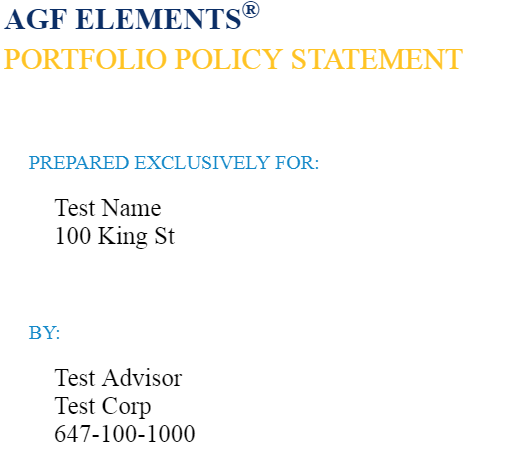
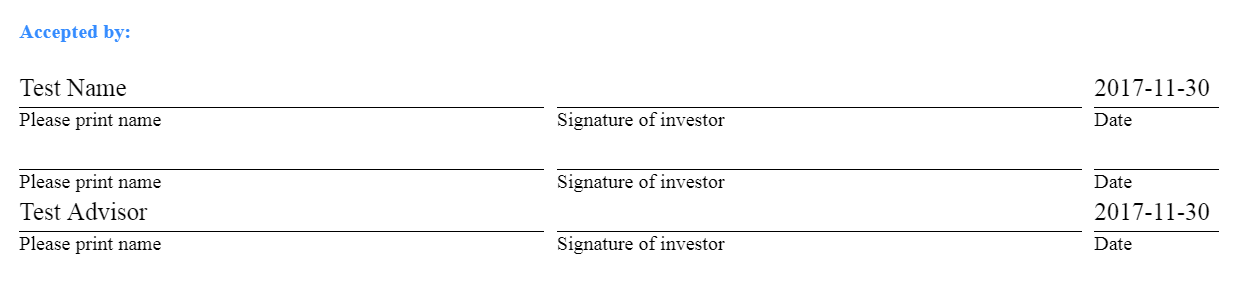
The report generating feature is used to save a PDF version of the portfolio report to user’s computer.

The report generating feature is divided into four sections: user input, information transfer between two sites, pie chart and table generation, and “print” (save as PDF). Detailed explanations can be viewed in the corresponding sections.

In this report, there are several sections requiring users to fill in when they finish their Element Quiz: client name, client address 1, client address 2, client address 3, advisor name, firm name, advisor phone number, date and class type.



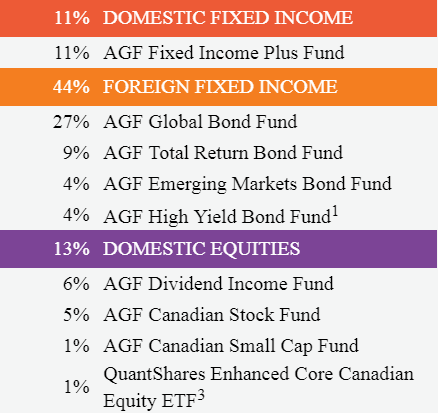
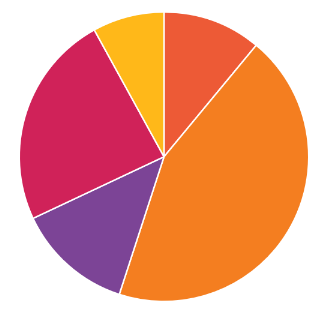
When user clicks on the “REPORT” button, a new webpage will open, and display the report:

The element quiz page and report page are two seperated web pages. In order to pass information from one site to another, query string is used. Example:

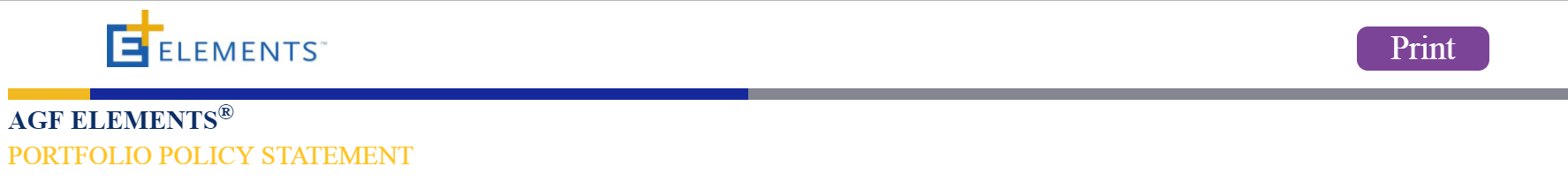
*“report-conservative.html?form/client-name=Test%20Name&form/address1=100%20King%20St&form/address2=&form/address3=&form/advisor-name=Test%20Advisor&form/firm-name=Test%20Corp&form/phone=647-100-1000&form/date=2017-11-30&form/class-name=Non-Corporate”*

Notice that there are one pie chart and three tables in a single report, for example:

All the data in the table and pie chart is dynamic, which means the data is not directly wirtten in the HTML. There is a JSON file (data definition) used to store all the necessary data. Functions are implemented in order to populate data into correct position.

On the top left of the report, there is a “Print” button:



This print is based on browser’s print functionality. When user click on this, a pop-up will show to let user select the print location( printer’s name or save as PDF). The print version has slightly different format with screen (web site) version. Two CSS stylesheets are liniked to the HTML homepage (the report page).

**Section 1.2 Report Generator Interface and User Input**

As shown in the overview, this is the user input interface:



**Structure Overview**

This form is simply use the HTML <form> tag, here is the HTML code:



(main/sliding.html)

<form> tag works with <input> tag to create a HTML form for user input. Notice that there is a “type” attribute in the <input> tag. This attribute specify the type of <input> element to display. In this piece of code, there are two types of <input>: text and radio.

Text is the default type of <input>. It will be shown as a text box with single-line text field, and its default width is 20 char.



Radio displays a radio button. User can select the option they want.



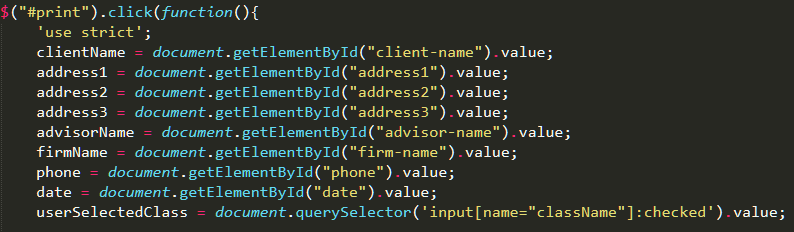
By default, all the option is unselected, so a new attribute “checked” is used to make the first radio selected.

Note that there is another attribute called “name”. In this code piece, the two options are grouped together, so they have the same name attribute.User can only select one option of the grouped options.

The “value” attribute contains the value of <input> tag. For the deault text box, it will return the value user typed in. For the ratio button, the value is pre-assigned. The value of the attribute will be returned.

**Function Overview**

When user clicks on “REPORT” button, the “click()” function will be triggered



(main/function.js)

All the user’s input is collected by “document.getElementById().value” except for the radio button.

“document.querySelector(selector)” will return the first matching element based on the selector. Since the user can only pick one option, there is only one matching in the documentation.

The data is collected, now they need to be parsed into query string in order to transfer to another page without backend script.

**Section 1.3 Query String and Data Passing between Sites**

**URL Creation**

Since this quiz app is designed to use frontend tech only, we cannot use backend language such as PHP to pass data between two sites. Query string is used here.

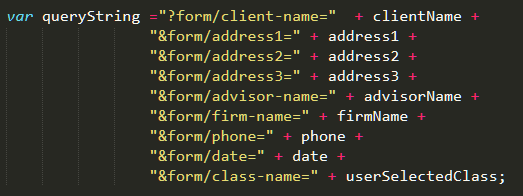
Query string is the part of URL which containing data that does not belong to hierarchical path structure.

Example:

/ElementsQuiz/report-page-yield/report-yield.html?form/client-name=query%20string%201

hierarchical path data name and its value

It is easy to generate query string: just use string append. Here is the code:

 (main/function.js)

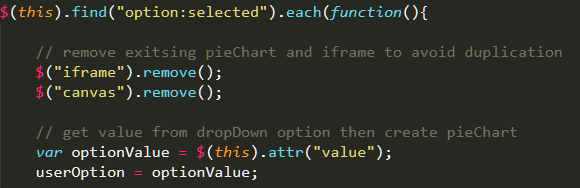
Query string is start with “?”, following with parameters’ names and values. Each parameter is separated by “&”.

Now, we need to open the correct report page (there are five different ones) based on the dropdown selection or user’s quiz result.

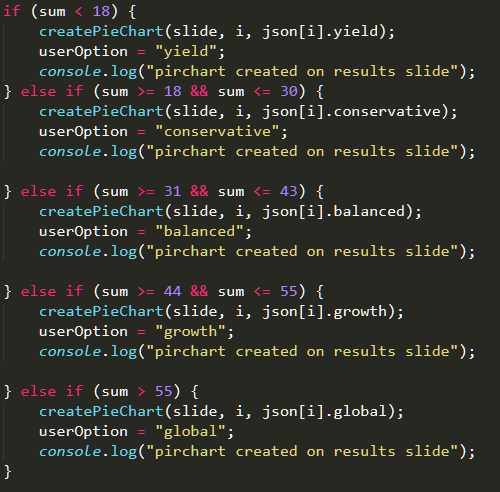


(main/function.js)

“userOption” is pre-declared variable (in the most outer scope). it will be assigned in “option:selected” selector:

 (main/function.js)

And when creating result slide:

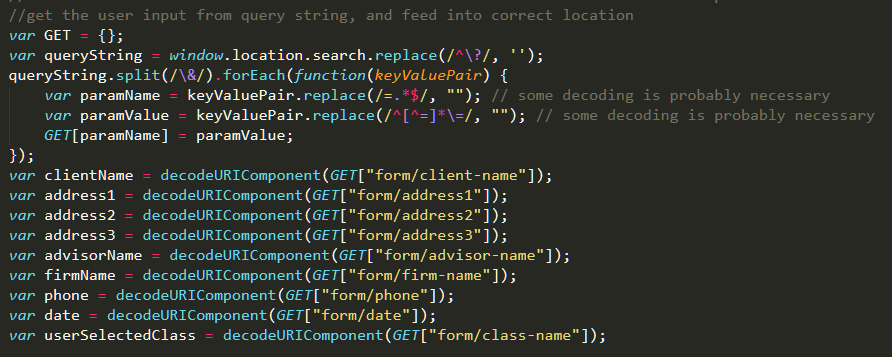
 (main/function.js)

Notice that in order to open the new page, “window.open()” is called. Modern browser will only open the page if it is triggered by trusted click events (user direct clicking), otherwise it will be blocked. Therefore, if an asynchronous tracking service (till this point, the click event may be sent by tracking server) such as Google Analytics is asked to be added, be careful with “window.open()”.

**Data Decoding and DOM Update**

Some Functions in report site are needed to deal with the query string.

**Decoding function:**



(report-page/dataBinding.js)

There are two processes of decoding: converting all key (parameter’s name) value (parameter’s value) pairs into an object “GET”, and converting the parameter’s value to a human readible string.

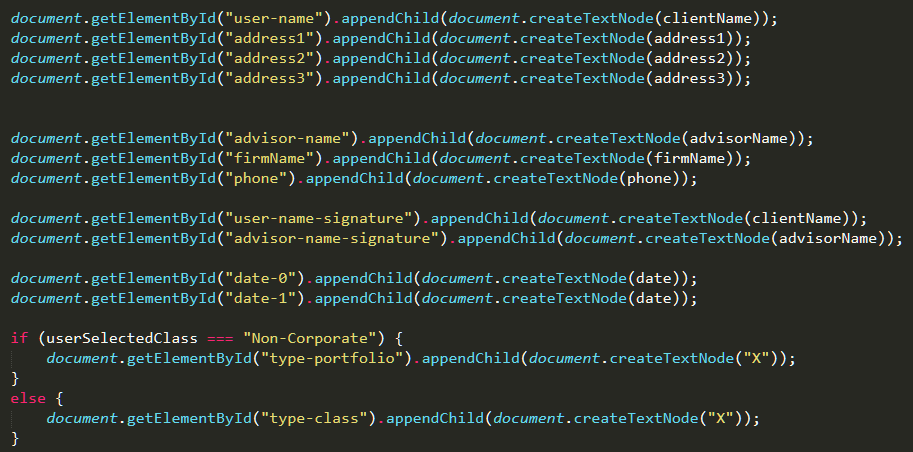
**Converting query string to an object:**

1. Declare an object “GET”
2. “location.search()” will return the query string from the URl (start with “?”)
3. “replace(/^\?/, “”)” wil return a string where the first parameter is replaced by the second parameter. In this case, the first parameter is a regular expression (regex), wrpped by two slashes (/).
   1. “^” matches the beginning of the string.
   2. “\” is the escape character.
   3. “\?” will match “?”

This “replace()” will remove the “?” at the beginning of the string.

1. “split(/&/)” will separate the whole query string and return an array of string, which is the word between “&”. “forEach” will loop through this array, and do the following things:
   1. “replace(/=.\*$/, “”)” will match all the substrings start with “=” till the end of string, and replace this substring with empty string (this is the parameter name)
   2. “replace(/^[^=]\*\=/)” will match all the substrings start from the beginning, till the first occurance of “=”, and replace it with empty string (this is the parameter value).
   3. Insert the key value pair into the GET object
2. When the loop is done, each information variable is assigned from GET object. Since those information strings may contain special character such as white space, “decodeURIComponent()” is used here to convert special character code into human readible format.

**DOM updating function:**



(report-page/dataBinding.js)

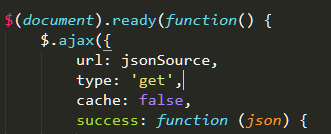
This part is simply use JavaScript DOM function to put data into the correct location.

HTML Element often consists of both element node and text node, so we need to use “createTextNode()” to create the text node, and append it to the correct location (decided by getElementById()).

**Section 1.4** **Report Data Table and Chart Construction**

The report page is constructed first before user input feed. Most part is static HTML except the data table and pie chart.

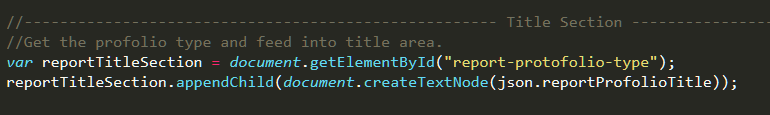
All data is stored in the JSON file. When the report DOM is ready, Ajax request is sent to get JSON.



**1.4.1 The title section**



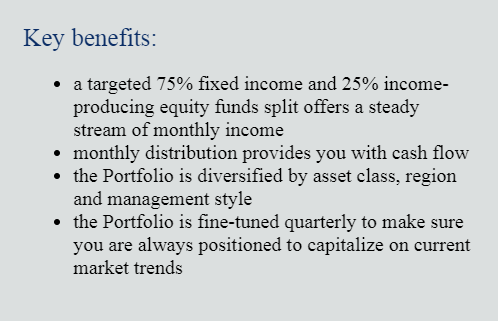
Is feed by



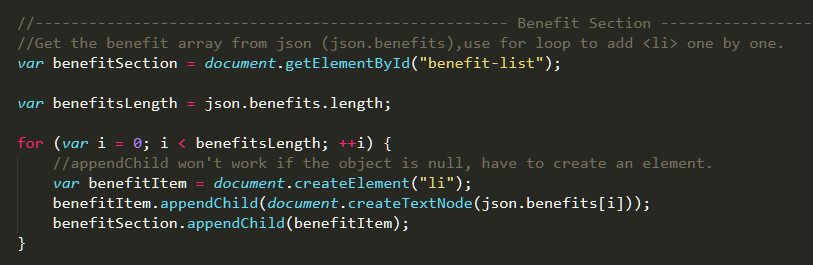
Here is the JSON:



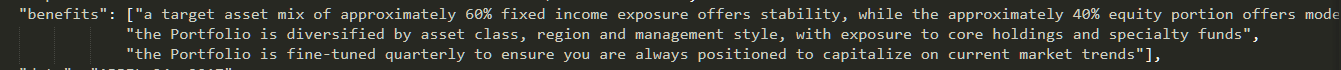
**1.4.2 The benefit section:**



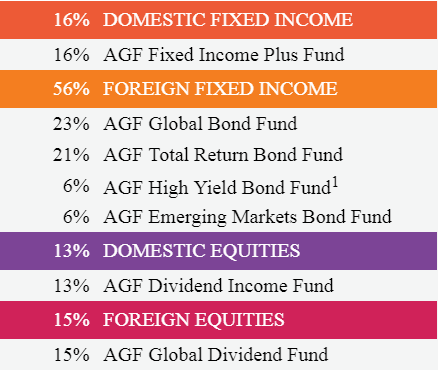
Is controlled by this piece of code:



All the benefits is stored in an array structure:



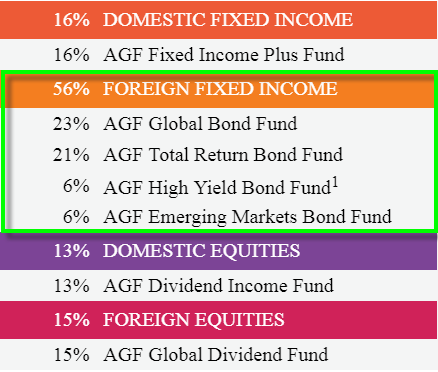
**1.4.3 The table section**

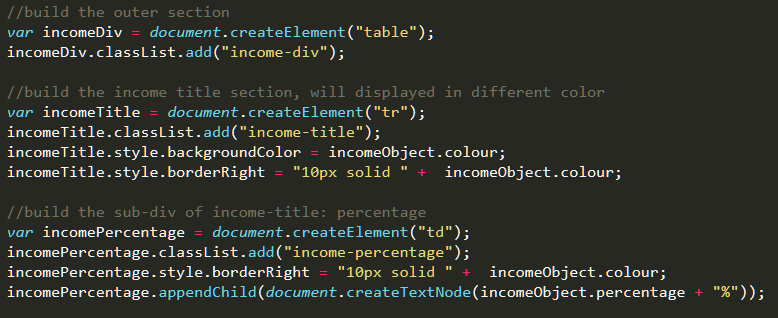


the table section is built by a helper function:

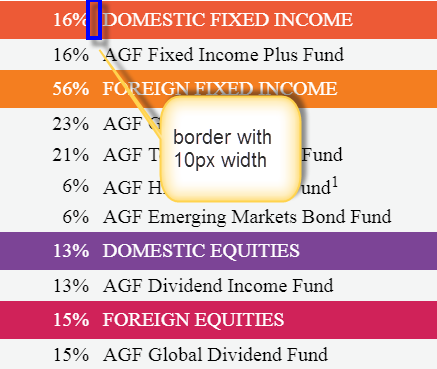


This function will consume the element node (divToBeAppend), and an object called incomeObject. Note that this function only build one “income title” with its sub fund:

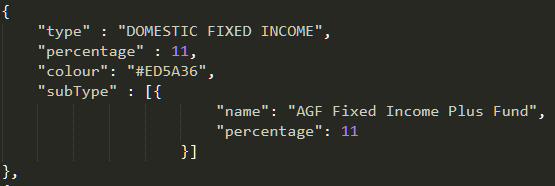
 this big “table” actually contains four tables.



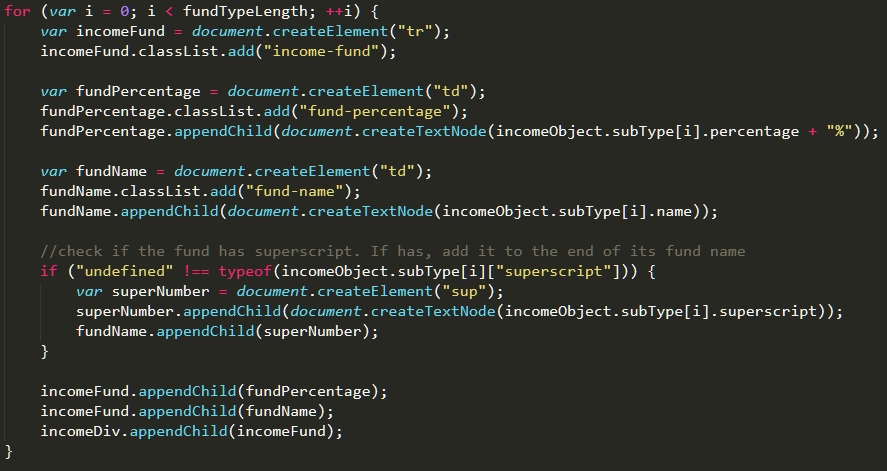
This section is used to construct the table. Table head background color is from JSON. “border-right” is added in order to separete percentage number and income/fund name.



The fund row is still stored in an array of fund objects:

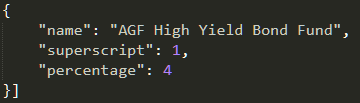
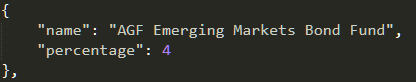
 an example of incomeObject

Those code is used to build the fund row:

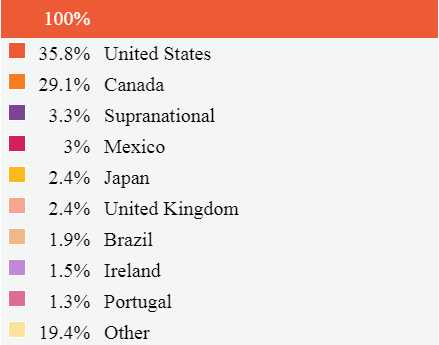


Some fund name may contain superscript, so an if statement is here to check the fund object. If property “superscript” is defined, then append its value to the end of fund name:

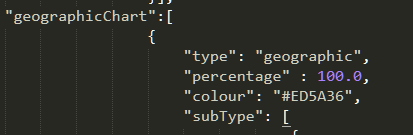
Fund object with “superscript” defined: Fund object without “superscript”:

The third table has a little different style, so I used an if statement to check:

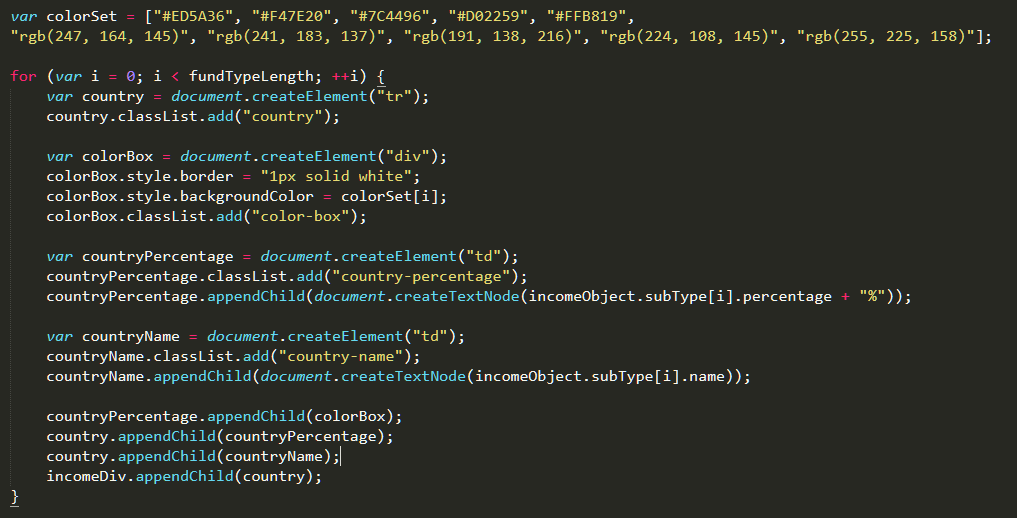


 this case is for non-geo table (first and second table), and else case is for geo table (third table)

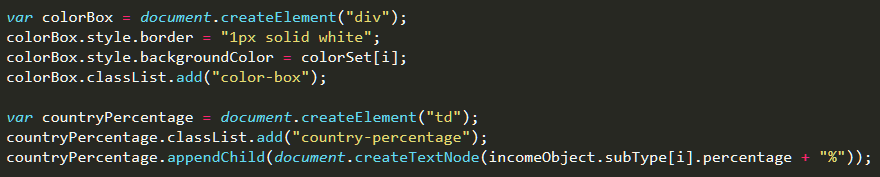


JSON object (only third table has type “geographic” (geographicChart[0].type == “geographic”)).

For “geographicChart”, the building function has a slightly difference:

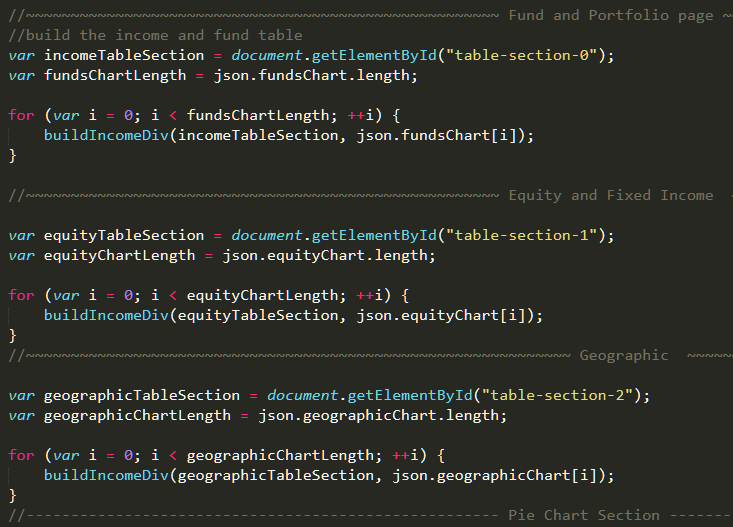


It has an array of color code string and a block of code to create the color box on the left of percentage number:



When the box div is created, it will be assigned a background color from the color code array. The maximum number of colors is ten.

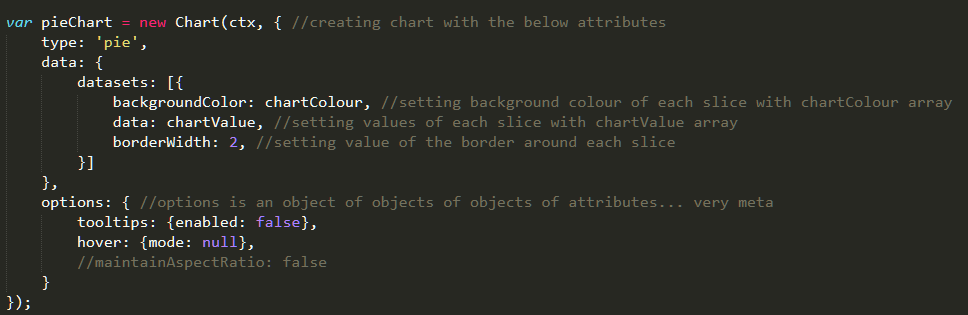
In the main function, an array of incomeObject will be looped through, and pass every element into “buildIncomeDiv()”. Running time O(n^2).



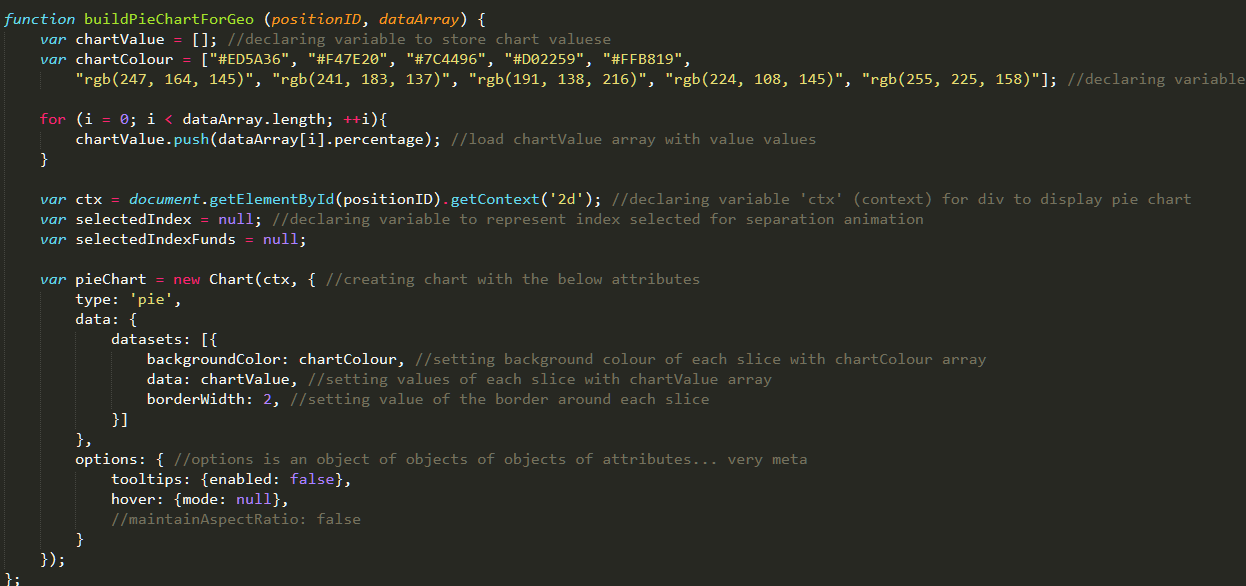
**1.4.4 The Pie Chart Section**

The pie chart is similar to the pie chart embeded in quiz page, but this one is static.

So in the option attribute, so the tooltips and hover is disabled.



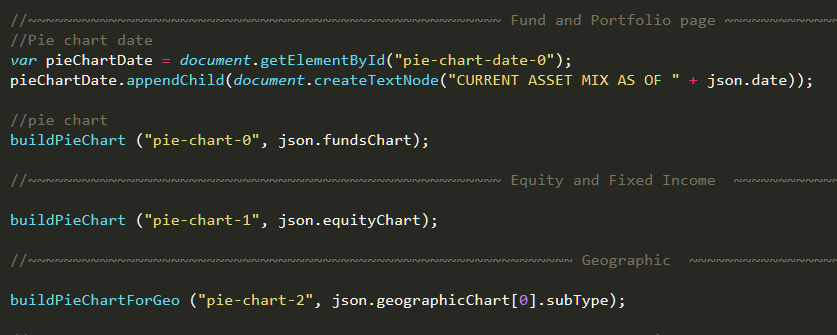
For geographic pie chart, the color set is different. Thus, there is an array of color code string in geo pie chart function:



The rest is the same.

Improvement suggestion: try to set some “if statement” to check which color set is in use. Place the two color set into one single function, and reduce the duplicated code.

In the main function block, first data section calls “buildPieChart()” and the third one invoke “buildPieChartForGeo()”:



**Section 1.5 Print View and Stylesheet**

The report page has two stylesheets. One is for the screen view, and one is for the print view.



In each stylesheet, there is a “media wrapper”, which is used to tell the browser to apply the correct stylesheet.

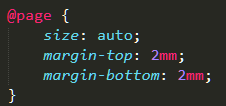
 and .

Each style has a slight differences.

In the print-style.css, there is a class selector called “.page-break”. This one is used to set the page breaker in the print view:

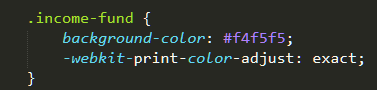


This is also a adjustion for the print view:



This code will hide the URL and html title.

By default, the background color will bot be printed, but we can add "-webkit-print-color-adjust: exact;” to the class selector. Then, the background color can be printed. Note that this is only suported in Chrome.



On the top right of the page, there is a print button. It is juat invoke “window.print()”

