ELEMENTS QUIZ

Documentation

Abstract

This project aims to re-create the Flash Elements Quiz using resources that cut third-party maintenance costs by making it accessible and easy to update regularly, without sacrificing any of its original functionality.

Jingyan He (Cecile)

Rachelyn Collins

Yining Wang (Kristen)

Yitian Wang

**Table of Contents**

[**Introduction** iii](#_Toc502238793)

[**HTML** 1](#_Toc502238794)

[**Included Scripts and Link** 1](#_Toc502238795)

[**Create the Structure of the Quiz** 1](#_Toc502238796)

[**Report Form** 2](#_Toc502238797)

[**JavaScript** 4](#_Toc502238798)

[**Load JSON-encoded Data from The Server** 4](#_Toc502238799)

[**Global** **Variables** 5](#_Toc502238800)

[**Slides Creation** 6](#_Toc502238801)

[**Function for Slide Content Creation** 7](#_Toc502238802)

[**Cover Slide** 7](#_Toc502238803)

[**Section Slide** 11](#_Toc502238804)

[**Question Slide** 12](#_Toc502238805)

[**Results Slide** 15](#_Toc502238806)

[**Pie Chart** 18](#_Toc502238807)

[**Overview** 18](#_Toc502238808)

[**HTML & JavaScript components** 19](#_Toc502238809)

[**JSON** 27](#_Toc502238810)

[**Cover** **Page** 28](#_Toc502238811)

[**Section** **Page** 28](#_Toc502238812)

[**Question** **Page** 29](#_Toc502238813)

[**Results** **Page** 29](#_Toc502238814)

[**Generating Report** 31](#_Toc502238815)

[**Overview** 31](#_Toc502238816)

[**Report Generator Interface and User Input** 33](#_Toc502238817)

[**Query String and Data Passing between Sites** 35](#_Toc502238818)

[**URL Creation** 35](#_Toc502238819)

[**Data Decoding and DOM Update** 36](#_Toc502238820)

[**Decoding function:** 37](#_Toc502238821)

[**Converting query string to an object:** 37](#_Toc502238822)

[**DOM updating function:** 38](#_Toc502238823)

[**Report Data Table and Chart Construction** 39](#_Toc502238824)

[**The title section** 39](#_Toc502238825)

[**The benefit section:** 39](#_Toc502238826)

[**The table section** 40](#_Toc502238827)

[**The Pie Chart Section** 43](#_Toc502238828)

[**Print View and Stylesheet** 44](#_Toc502238829)

[**Application Redesign** 46](#_Toc502238830)

# **Introduction**

The main features for the quiz:

* Question/answer input & backend calculations
* Results page with matched portfolio and pie chart visuals
* PDF download of report with inputs for name and other details

All questions and answers are stored in JSON file. In the JS file, there is a list (array) for storing inputs getting from user. The array will be used for backend calculation then generates recommend investment portfolio.

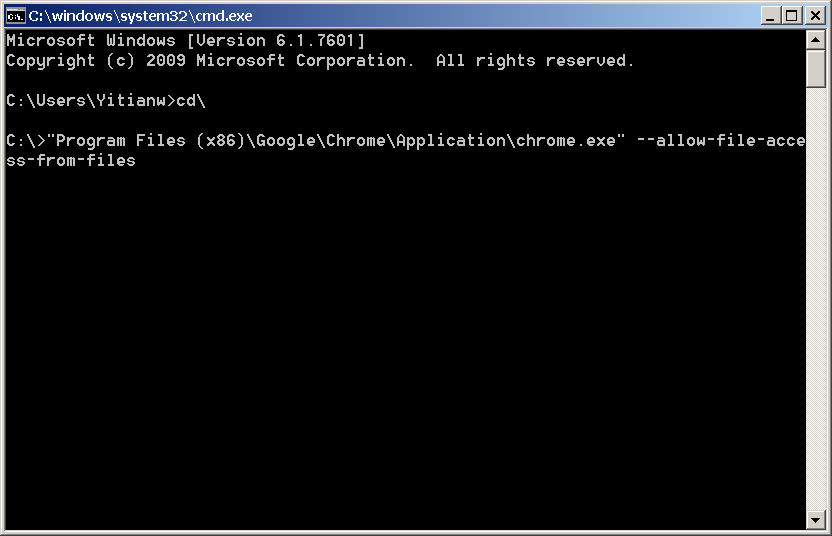
Both the pie chart and report downloading will be explained later.

The main structure of the quiz is a queue of several slides, linked as a list. Currently, there are four types of slides: cover slide, section slide, question slide and results slide. To create a new slide, the type of the slide should be included at first in JSON file. The content must be organized according to existing slides with the same type.

Developing and testing was done almost exclusively in Google Chrome. The quiz appears to work fine on Safari, but not on Internet Explorer or Firefox. This is due to the nature of the JSON file being "blocked" on these browsers; internet explorer accepts XML, however, which is another way the data can be fed into the code.

There may have problems for browser opening Json file. The best option is to open the html file is using Chrome.

1. Close all Chrome tabs
2. Open cmd
3. Find the direction of “chrome.exe” using Explorer
4. Insert the direction of chrome in cmd and open chrome under the condition of “-allow-file-access-from-files”
5. Using the opening window of Chrome



# **HTML**

## **Included Scripts and Link**

<script src=**"https://ajax.googleapis.com/ajax/libs/jquery/3.2.1/jquery.min.js"**></script>

<script src=**"https://code.jquery.com/jquery-1.12.4.min.js"**></script>

<script src=**"https://cdnjs.cloudflare.com/ajax/libs/Chart.js/2.6.0/Chart.bundle.js**</script>

<script type=**"text/javascript"** src=**"function.js"**></script>

<script type=**"text/javascript"** src=**"data.json"**></script>

<link rel=**"stylesheet"** type=**"text/css"** href=**"style.css"**>

## **Create the Structure of the Quiz**

<img id=**"banner"** src=**"banner.png"**>

<div id=**"frame"**>

<ul id=**"queue"**></ul>

</div>

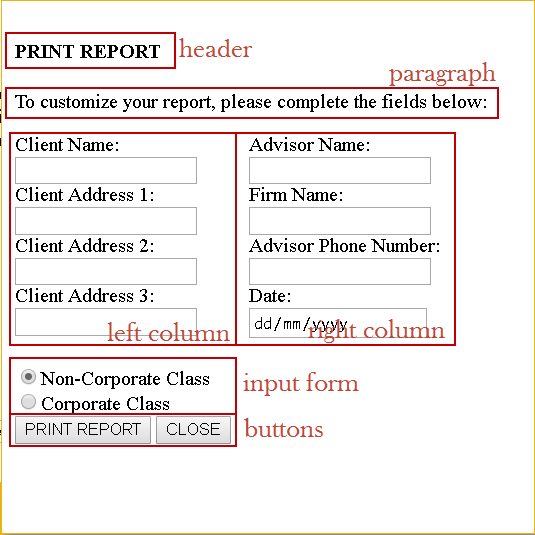
<div id=**"footer"**></div>

## **Report Form**

The report form is not created dynamically (getting data from JSON and creating elements in JS).

The report form takes in basic information from the user, then use that information for full report generation. The display of this window is triggered by clicking the REPORT button located at footer. The REPORT button appears when a portfolio is selected.

* The header and introduction paragraph can be updated through HTML file
* Text fields take users inputs
* Two options of Class
* PRINT REPORT button opens the tab of full report for preview



<!-- create pop-up window for report form -->

<div id=**"report-window"** style=**"display: none"**>

<div id=**"report-title"**>

<h4> **PRINT REPORT**</h4>

<p> **To customize your report, please complete the fields below:**</p>

</div>

<div class=**"report-row"** >

<div class=**"report-column"**>

**Client Name:** <br>

<input id=**"client-name"** type=**"text"**><br>

**Client Address 1:** <br>

<input id=**"address1"** type=**"text"**><br>

**Client Address 2:** <br>

<input id=**"address2"** type=**"text"**><br>

**Client Address 3:** <br>

<input id=**"address3"** type=**"text"**><br>

</div>

<div class=**"report-column"**>

**Advisor Name:** <br>

<input id=**"advisor-name"** type=**"text"**><br>

**Firm Name:** <br>

<input id=**"firm-name"** type=**"text"**><br>

**Advisor Phone Number:** <br>

<input id=**"phone"** type=**"text"**><br>

**Date:** <br>

<input id=**"date"** type=**"date"**><br>

</div>

</div>

<form id=**"classNames"**>

<input type=**"radio"** name=**"className"** value=**"Non-Corporate"** id=**"Non-Corporate"** checked=**"checked"**>**Non-Corporate Class**<br>

<input type=**"radio"** name=**"className"** value=**"Corporate"** id=**"Corporate"**>**Corporate Class** <br>

</form>

<button id=**"print"**>**REPORT**</button>

<button class=**"close"**>**CLOSE**</button>

</div>

# **JavaScript**

The JavaScript contains all the functions used to create this quiz dynamically.

All the data comes from an external JSON file that can be changed at any time. Elements’ styling comes from an external CSS file.

Some of the main features:

* sliding quiz display with dynamic number of slides
* dynamic questions pulled from JSON file
* retrieving the user’s answers
* backend calculations to determine a portfolio match
* dynamic pie chart and fund information
* options to jump directly to portfolios
* pagination
* report generation

All js code has to be in the function “.getJSON” since the js file must access the data in Json then process other functions.

## **Load JSON-encoded Data from The Server**

$**.**getJSON**(**"https://api.myjson.com/bins/gfwb7"**,** **function(**json**)**

Here is the link to the documentation about function “.getjson()” : <http://api.jquery.com/jquery.getjson/>

## **Global** **Variables**

**var** numOfSlides **=** json**.**length**;** // get the amount of slides in the quiz

// (containing coverSlide, questionSlide and resultsSlide)

$**(**"#queue"**).**width**(**numOfSlides**\***800**);** // set the length of slide queue

// each slide has width 800px (set as requirement)

// have to be decided after getting the amount of slides

**var** arr **=** **[];**

**for** **(var** i **=** 0**;** i **<** numOfSlides**;** i**++)** **{** // create an array to store inputs (radio button inputs on questionSlide)

// the length of array is same as number of all slides

// the index of slide is same as the index of item in array

arr**.**push**(**0**);**

**}**

**var** sum**;** // create sum to store the sum of items in array

**var** all\_slides **=** **document.**getElementById**(**"queue"**);** // get the div of slide queue (have created in html)

**var** footer **=** **document.**getElementById**(**"footer"**);** // get the div of footer (have created in html)

// variables used in report form

**var** clientName**,** address1**,** address2**,** address3**,** advisorName**,** firmName**,** phone**,** date**,** userSelectedClass**,** userOption**;**

## **Slides Creation**

// create all slides

**for** **(var** i **=** 0**;** i **<** numOfSlides**;** i**++)** **{**

// create all slides

**var** slide **=** **document.**createElement**(**"li"**);**

slide**.**id **=** "slide" **+** i**;**

all\_slides**.**appendChild**(**slide**);**

**if** **(**json**[**i**].**type **==** "coverPage"**)** **{**

createCoverPageSlide**(**slide**,** i**,** **(**numOfSlides **-** 1**),** footer**);** // the index of last slide

**}** **else** **if** **(**json**[**i**].**type **==** "section"**)** **{**

createSectionSlide**(**slide**,** i**);**

**}** **else** **if** **(**json**[**i**].**type **==** "question"**){**

createQuestionSlide**(**slide**,** i**);**

**}** **else** **if** **(**json**[**i**].**type **==** "results"**){**

createResultsSlide**(**slide**,** i**,** arr**,** sum**);**

**}**

**}**

## **Function for Slide Content Creation**

The whole quiz is divided into four types of pages:

* Cover Page – the very first page of the quiz containing the quiz name and brief introduction
* Section Page – the quiz contains different section. For each section, there is an introduction page
* Question Pages – pages containing question and answer that user is going to take
* Results Page – the page that displays the recommend portfolio based on the questions that user takes

**Cover Slide**

function createCoverPageSlide (slide, i, last\_slide\_index, footer)



// create slide title

**var** title **=** **document.**createElement**(**"h2"**);**

title**.**appendChild**(document.**createTextNode**(**json**[**i**].**title**));**

title**.**className **=** "remove"**;** // to be removed

slide**.**appendChild**(**title**);**

// create intro para

**var** intro **=** **document.**createElement**(**"p"**);**

intro**.**appendChild**(document.**createTextNode**(**json**[**i**].**intro**));**

intro**.**id **=** "cover-slide-para"**;**

intro**.**className **=** "remove"**;** // to be removed

slide**.**appendChild**(**intro**);**

// create button to go to next slide

**var** nextButton **=** **document.**createElement**(**"button"**);**

nextButton**.**appendChild**(document.**createTextNode**(**"Take the questionnaire >>"**));**

nextButton**.**className **=** "next"**;**

nextButton**.**className **+=** " remove"**;** // to be removed

slide**.**appendChild**(**nextButton**);**

// add portfolio drop-down

**var** numOfOptions **=** json**[**i**].**dropDown**.**length**;** // get amount of options

**var** dropDown **=** **document.**createElement**(**"select"**);**

dropDown**.**id **=** "move-dropDown"**;**

dropDown**.**className **=** "drop-down"**;**

slide**.**appendChild**(**dropDown**);**

// add options to drop-down

**for** **(var** j **=** 0**;** j **<** numOfOptions**;** j**++)** **{**

**var** **option** **=** **document.**createElement**(**"option"**);**

**if** **(**j **==** 0**)** **{** // set the first option as default

**option.**disabled **=** "true"**;**

**option.**selected **=** "true"**;**

**}**

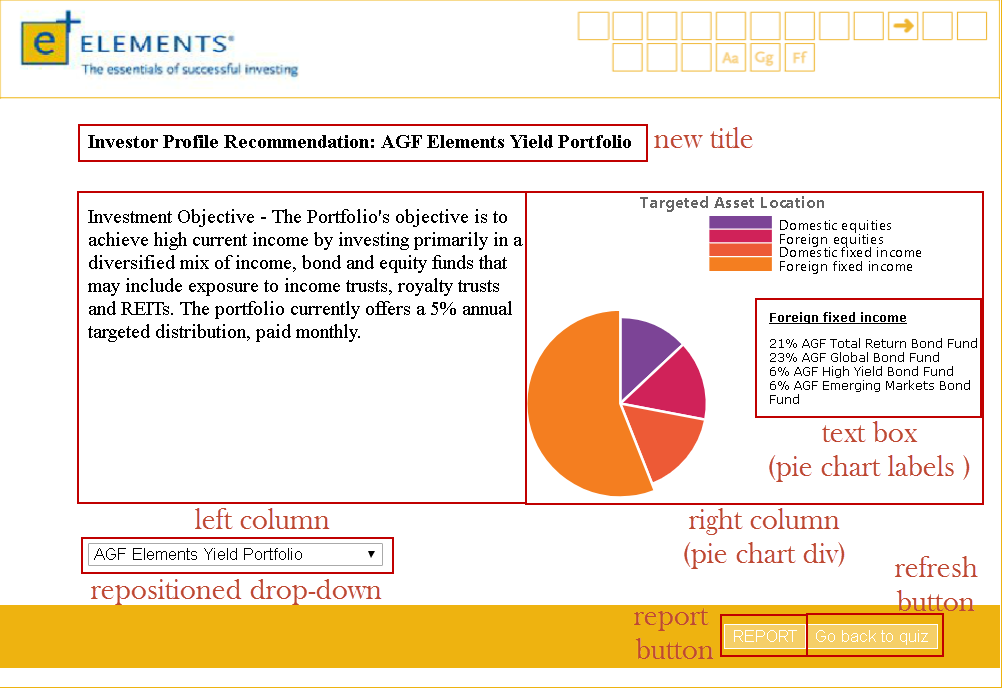
**option.**appendChild**(document.**createTextNode**(**json**[**i**].**dropDown**[**j**].**name**));**

**option.**value **=** json**[**i**].**dropDown**[**j**].**value**;**

dropDown**.**appendChild**(option);**

**}**

When user selects an option from the drop-down, it will process to a results page with the portfolio that user just selected. To create the new slide, elements on the slide are removed (except drop-down is repositioned) then new elements are created on the slide.



// replace elements

**var** flag **=** **false;**

$**(**"#slide" **+** i **+** " .drop-down"**).**change**(function(){**

$**(**"#move-dropDown"**).**addClass**(**"move-dropDown"**);** // reposition the dropDown to the bottom of the slide

$**(**".pagination"**).**hide**();** // hide pagination

$**(**"#report"**).**show**();** // display REPORT button

**if** **(**flag **==** **false)** **{** // remove old elements once

// create new elements once

flag**=true;**

$**(**".remove"**).**remove**();** // remove elements from cover page

// create new elements

//

**var** title **=** **document.**createElement**(**"h4"**);** // create title

title**.**id **=** "results-slide-title"**;**

slide**.**appendChild**(**title**);**

**var** row **=** **document.**createElement**(**"div"**);** // create one row for two columns

row**.**className **=** "results-row"**;**

slide**.**appendChild**(**row**);**

**var** leftColumn **=** **document.**createElement**(**"div"**);** // left column

leftColumn**.**className **=** "results-column"**;**

leftColumn**.**id **=** "leftColumn"**;**

row**.**appendChild**(**leftColumn**);**

**var** rightColumn **=** **document.**createElement**(**"div"**);** // right column

rightColumn**.**className **=** "results-column"**;**

rightColumn**.**id **=** "rightColumn"**;**

row**.**appendChild**(**rightColumn**);**

**var** pieChart **=** **document.**createElement**(**"div"**);** // add pieChart div to right column

pieChart**.**id **=** "canvas-holder"**;**

rightColumn**.**appendChild**(**pieChart**);**

**var** fundBox **=** **document.**createElement**(**"div"**);** // add text box for pieChart labels

fundBox**.**id **=** "fundBox"**;**

rightColumn**.**appendChild**(**fundBox**);**

**var** funds **=** **document.**createElement**(**"p"**);**

funds**.**id **=** "funds"**;**

fundBox**.**appendChild**(**funds**);**

**var** refreshButton **=** **document.**createElement**(**"button"**);** // add REFRESH button for returning back to quiz

refreshButton**.**appendChild**(document.**createTextNode**(**"Go back to quiz"**));**

refreshButton**.**id **=** "refresh"**;**

footer**.**appendChild**(**refreshButton**);**

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

**window.location.**reload**();**

**});**

**}**

// get piechart info from the last slide (Results slide)

i **=** last\_slide\_index**;**

$**(this).**find**(**"option:selected"**).**each**(function(){**

**var** optionValue **=** $**(this).**attr**(**"value"**);** // get value from dropDown option then create pieChart

userOption **=** optionValue**;**

**var** arg **=** eval**(**"json[i]."**+** optionValue**);** // get the type of portfolio

// remove exitsing pieChart and iframe to avoid duplication

$**(**"iframe"**).**remove**();**

$**(**"canvas"**).**remove**();**

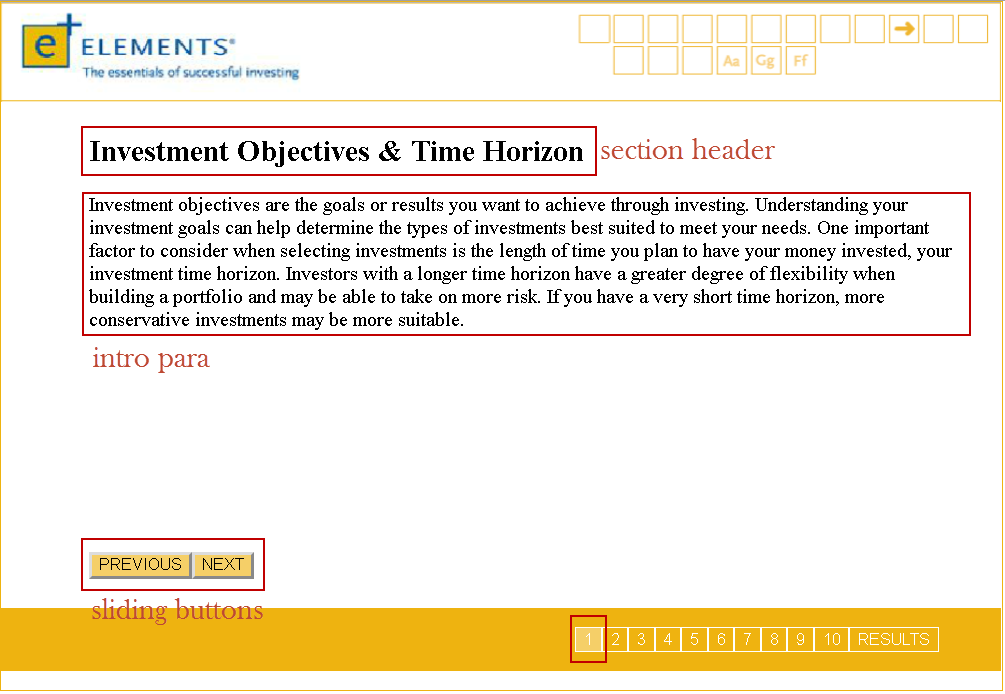
createPieChart**(**slide**,** i**,** arg**);** // call function to create pie-chart

**});**

**});**

### **Section Slide**

**function** createSectionSlide**(**slide**,** i**)**



// create section header

**var** title **=** **document.**createElement**(**"h2"**);**

title**.**appendChild**(document.**createTextNode**(**json**[**i**].**title**))**

slide**.**appendChild**(**title**);**

// create section intro paragraph

**var** intro **=** **document.**createElement**(**"p"**);**

intro**.**appendChild**(document.**createTextNode**(**json**[**i**].**intro**));**

intro**.**className **=** "section-intro"**;**

slide**.**appendChild**(**intro**);**

// create Previous and Next buttons to control sliding

**var** previouButton **=** **document.**createElement**(**"button"**);**

previouButton**.**appendChild**(document.**createTextNode**(**"PREVIOUS"**)section );**

previouButton**.**className **=** "previous"**;**

slide**.**appendChild**(**previouButton**);**

**var** nextButton **=** **document.**createElement**(**"button"**);**

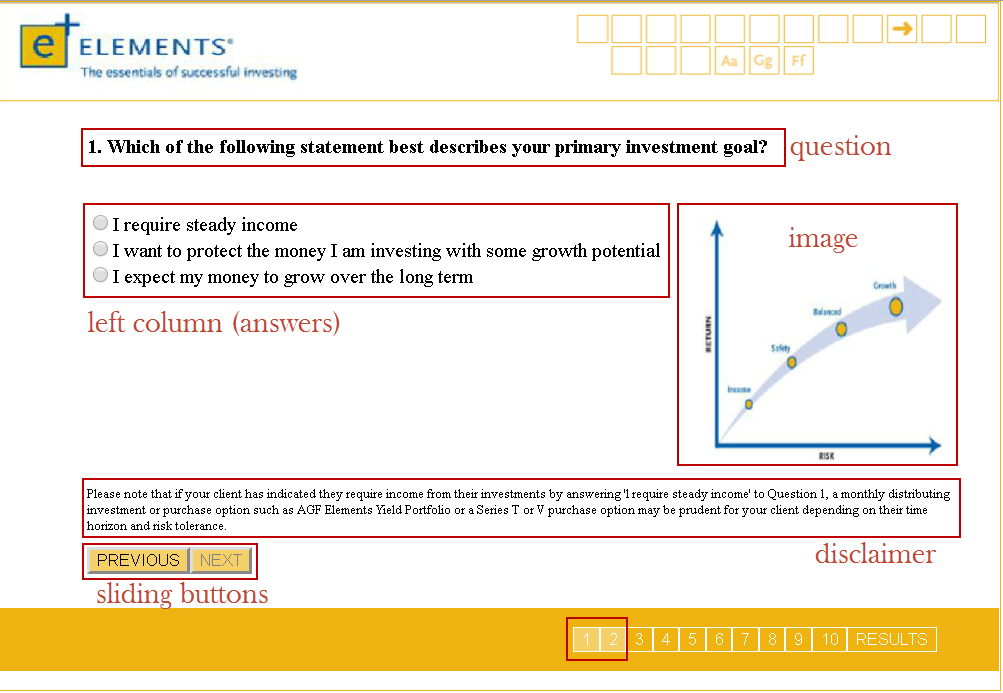
nextButton**.**appendChild**(document.**createTextNode**(**"NEXT"**));**

nextButton**.**className **=** "next"**;**

slide**.**appendChild**(**nextButton**);**

**Question Slide**

**function** createQuestionSlide**(**slide**,** i**)**



// create question

**var** question **=** **document.**createElement**(**"h4"**);**

question**.**appendChild**(document.**createTextNode**(**json**[**i**].**question**));**

question**.**id **=** "question"**;**

slide**.**appendChild**(**question**);**

// create one row for two columns

**var** row **=** **document.**createElement**(**"div"**);**

row**.**className **=** "row"**;**

slide**.**appendChild**(**row**);**

// create left column for answer list

**var** leftColumn **=** **document.**createElement**(**"div"**);**

leftColumn**.**className **=** "question-column"**;**

leftColumn**.**className **+=** " question-left"**;**

row**.**appendChild**(**leftColumn**);**

// create right column for img

**var** rightColumn **=** **document.**createElement**(**"div"**);**

rightColumn**.**className **=** "question-column"**;**

rightColumn**.**className **+=** " question-right"**;**

row**.**appendChild**(**rightColumn**);**

// create Form for answer list

**var** formElement **=** **document.**createElement**(**"form"**);**

formElement**.**id **=** "form" **+** i**;**

leftColumn**.**appendChild**(**formElement**);**

// add Answers to the slide

**var** numOfAnswers **=** json**[**i**].**answers**.**length**;**

**for** **(var** j **=** 0**;** j **<** numOfAnswers**;** j**++)** **{**

**function** addRadioButton**(text,** value**){**

**var** label **=** **document.**createElement**(**"label"**);**

**var** **radio** **=** **document.**createElement**(**"input"**);**

**radio.**className **=** "radio-button"**;**

**radio.**setAttribute**(**"type"**,** "radio"**);**

**radio.**setAttribute**(**"name"**,** "radioButton"**);**

**radio.**setAttribute**(**"value"**,** value**);**

**radio.**id **=** i**;**

label**.**appendChild**(radio);**

label**.**appendChild**(document.**createTextNode**(text));** // add text to radio buttons

formElement**.**appendChild**(**label**);**

**}**

addRadioButton**(**json**[**i**].**answers**[**j**].text,** json**[**i**].**answers**[**j**].**value**);** // add value to radio buttons

**}**

// add disclaimer

**if** **(**json**[**i**].**disclaimer **!=** **null){**

**var** disclaimer **=** **document.**createElement**(**"p"**);**

disclaimer**.**className **=** "disclaimer"**;**

slide**.**appendChild**(**disclaimer**);**

disclaimer**.**appendChild**(document.**createTextNode**(**json**[**i**].**disclaimer**));**

**}**

// add img to right column

**if** **(**json**[**i**].**img **!=** **null)** **{**

**var** img **=** **document.**createElement**(**"IMG"**);**

img**.**className **=** "img"**;**

img**.**setAttribute**(**"src"**,** json**[**i**].**img**);**

img**.**setAttribute**(**"alt"**,** "chart display error"**);**

rightColumn**.**appendChild**(**img**);**

**}**

// create two buttons to control sliding

//

**var** previouButton **=** **document.**createElement**(**"button"**);**

previouButton**.**appendChild**(document.**createTextNode**(**"PREVIOUS"**));**

previouButton**.**className **=** "previous"**;**

slide**.**appendChild**(**previouButton**);**

**var** nextButton **=** **document.**createElement**(**"button"**);**

nextButton**.**appendChild**(document.**createTextNode**(**"NEXT"**));**

nextButton**.**className **=** "next"**;**

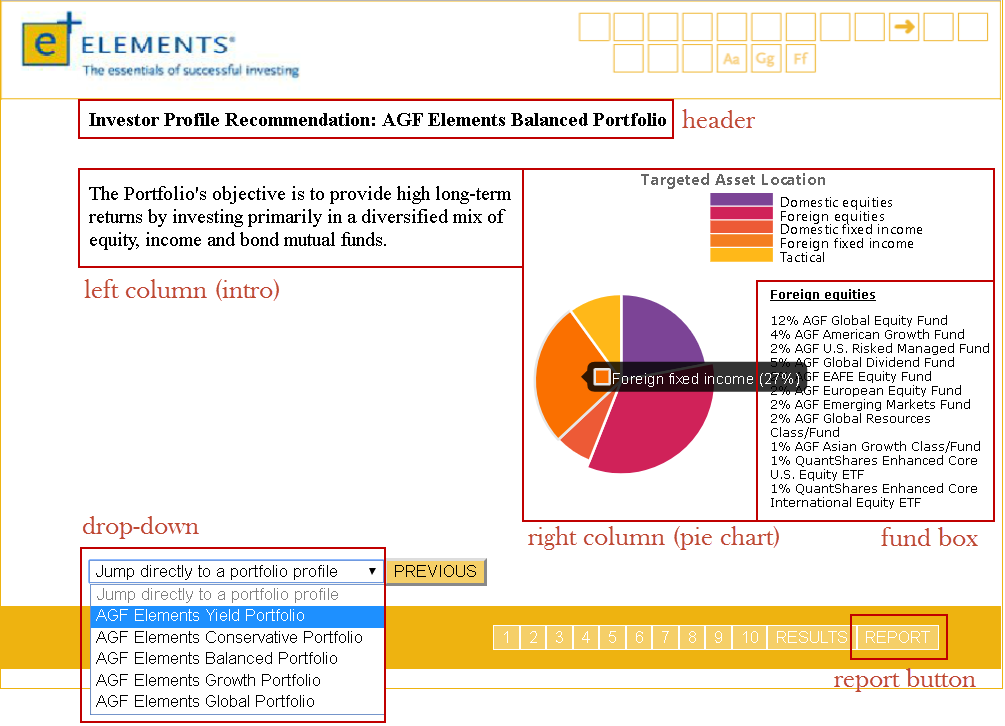
nextButton**.**id **=** **(**"next" **+** i**);**

nextButton**.**disabled **=** **true;**

slide**.**appendChild**(**nextButton**);**

### **Results Slide**

**function** createResultsSlide**(**slide**,** i**,** arr**,** sum**)**



// slide title

**var** title **=** **document.**createElement**(**"h4"**);**

title**.**id **=** "results-slide-title"**;**

slide**.**appendChild**(**title**);**

**var** row **=** **document.**createElement**(**"div"**);**

row**.**className **=** "results-row"**;**

slide**.**appendChild**(**row**);**

**var** leftColumn **=** **document.**createElement**(**"div"**);**

leftColumn**.**className **=** "results-column"**;**

leftColumn**.**id **=** "leftColumn"**;**

row**.**appendChild**(**leftColumn**);**

**var** rightColumn **=** **document.**createElement**(**"div"**);**

rightColumn**.**className **=** "results-column"**;**

rightColumn**.**id **=** "rightColumn"**;**

row**.**appendChild**(**rightColumn**);**

**var** pieChart **=** **document.**createElement**(**"div"**);**

pieChart**.**id **=** "canvas-holder"**;**

rightColumn**.**appendChild**(**pieChart**);**

**var** fundBox **=** **document.**createElement**(**"div"**);**

fundBox**.**id **=** "fundBox"**;**

rightColumn**.**appendChild**(**fundBox**);**

**var** funds **=** **document.**createElement**(**"p"**);**

funds**.**id **=** "funds"**;**

fundBox**.**appendChild**(**funds**);**

// add portfolio drop-down

**var** numOfOptions **=** json**[**i**].**dropDown**.**length**;** // get amount of options

**var** dropDown **=** **document.**createElement**(**"select"**);**

dropDown**.**className **=** "drop-down"**;**

// add options to drop-down

**for** **(var** j **=** 0**;** j **<** numOfOptions**;** j**++)** **{**

**var** **option** **=** **document.**createElement**(**"option"**);**

**if** **(**j **==** 0**)** **{**

**option.**disabled **=** "true"**;**

**option.**selected **=** "true"**;**

**}**

**option.**appendChild**(document.**createTextNode**(**json**[**i**].**dropDown**[**j**].**name**));**

**option.**value **=** json**[**i**].**dropDown**[**j**].**value**;**

dropDown**.**appendChild**(option);**

**}**

slide**.**appendChild**(**dropDown**);**

// create only previous button

**var** previouButton **=** **document.**createElement**(**"button"**);**

previouButton**.**appendChild**(document.**createTextNode**(**"PREVIOUS"**));**

previouButton**.**className **=** "previous"**;**

slide**.**appendChild**(**previouButton**);**

// create pop-up window for report form

**var** modal **=** **document.**createElement**(**"div"**);**

modal**.**id **=** "myModal"**;**

slide**.**appendChild**(**modal**);**

// append the pre-build report window (from html file) to pop-up window

**var** reportContent **=** **document.**getElementById**(**"report-window"**);**

modal**.**appendChild**(**reportContent**);**

$**(**"#next" **+** **(**i **-** 1**)).**click**(function(){** // when user process to last slide (results slide)

sum **=** 0**;**

**for** **(var** j **=** 0**;** j **<** arr**.**length**;** j**++)** **{** // calculate the sum getting from user

sum **=** sum **+** **parseInt(**arr**[**j**]);**

**}**

// remove existing pieChart and iframe to avoid duplication

$**(**"iframe"**).**remove**();**

$**(**"canvas"**).**remove**();**

**if** **(**sum **<** 18**)** **{**

createPieChart**(**slide**,** i**,** json**[**i**].**yield**);**

userOption **=** "yield"**;**

**}** **else** **if** **(**sum **>=** 18 **&&** sum **<=** 30**)** **{**

createPieChart**(**slide**,** i**,** json**[**i**].**conservative**);**

userOption **=** "conservative"**;**

**}** **else** **if** **(**sum **>=** 31 **&&** sum **<=** 43**)** **{**

createPieChart**(**slide**,** i**,** json**[**i**].**balanced**);**

userOption **=** "balanced"**;**

**}** **else** **if** **(**sum **>=** 44 **&&** sum **<=** 55**)** **{**

createPieChart**(**slide**,** i**,** json**[**i**].**growth**);**

userOption **=** "growth"**;**

**}** **else** **if** **(**sum **>** 55**)** **{**

createPieChart**(**slide**,** i**,** json**[**i**].**global**);**

userOption **=** "global"**;**

**}**

$**(**"#report"**).**show**();** // display REPORT button

**});**

// change content on RESULTS slide by changing dropdown options

$**(**"#slide" **+** i **+**" .drop-down"**).**change**(function(){**

$**(this).**find**(**"option:selected"**).**each**(function(){**

**var** optionValue **=** $**(this).**attr**(**"value"**);**

userOption **=** optionValue**;**

**var** arg **=** eval**(**"json[i]."**+** optionValue**);**

// remove exitsing pieChart and iframe to avoid duplication

$**(**"iframe"**).**remove**();**

$**(**"canvas"**).**remove**();**

createPieChart**(**slide**,** i**,** arg**);**

**});**

**});**

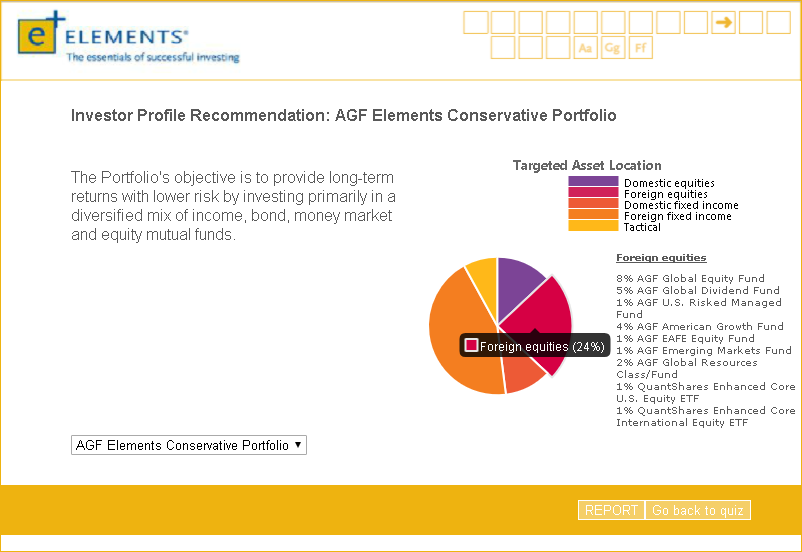
## **Pie Chart**

### **Overview**

Along with the questionnaire results that are displayed on the last slide, a pie chart is generated to deliver visual information about the targeted asset locations for each type of Elements portfolio.

Pie Chart features:

* Dynamic (number of slices, legend items, chart title, colors)
* On-hover animation (colour of slice darkens)
* Tooltip displays name of legend item + its % on hover
* Slices explode/implode upon being clicked
* Slices display associated funds list upon being clicked



Browser support:

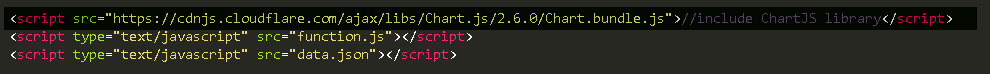
* Chrome 50+
* Firefox 45+
* Internet Explorer 11
* Edge 14+
* Safari 9+

Browser support for the canvas element is available in all modern & major mobile browsers.

The pie chart is created using Chart.js, a JavaScript framework for developers to create different types of graphs. [Official Chart.js Documentation](http://www.chartjs.org/docs/latest/)

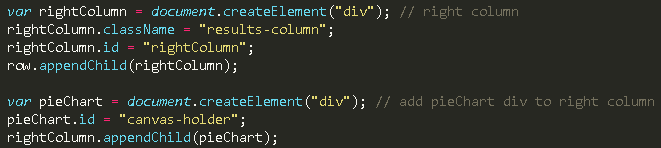
### **HTML & JavaScript components**

**1. To use Chart.js, the library must be referenced in the HTML as such:**



This line indicates to the browser running the script that it will be using features included in the chart.js library. The version used in this code is v2.6.0.

**2. Create the HTML canvas container (div) *within the JavaScript*:**



Due to the way the quiz layout is set up (a large banner where a section slides into view), the canvas container cannot be declared directly in the HTML. Using the lines of code above, a div with id “canvas-holder” is created. Since it will be in the right column of the results page, the div is appended to parent id “rightColumn”.

**3. Previously created canvas/iframe must be removed because another chart is loaded in its place:**



This code must go before the pie chart creation function is called so that any existing canvas and iframe is removed. If this is not done, any pie chart that is created over an existing one will glitch out and display data from previous charts.

**4. “Create Pie Chart” function:**



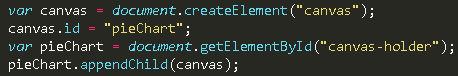
There are many instances in which a pie chart needs to be generated throughout the code; implementing a function, “createPieChart”, allows it to be easily called whenever necessary. The rest of the code regarding the pie chart goes within the two curly brackets to indicate that they make up the function. In order to perform its function, 3 parameters need to be passed in: “slide”, “i”, “portfolio”.

**“slide”** – the blank slide that the pie chart is generated on; portion that slides in/out of view

**“i”** – the numerical index of the slide on which the pie chart is generated

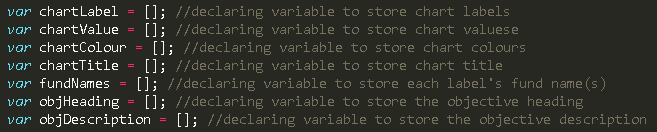
**“portfolio”** – the investment portfolio (e.g. yield, conservative, balanced, growth, global) that the investor has been matched with

**5. Create the canvas HTML node *within the JavaScript*:**

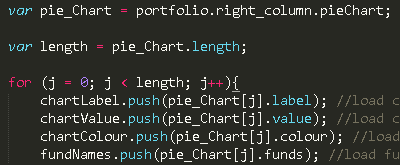


The code above creates a <canvas> node with id “pieChart” within the HTML, which is where the pie chart is rendered. It is also appended to the “canvas-holder” div so that it can be styled on the page.

**6. Declare variables that will be used to store the dynamic data from the JSON files:**



**7. Store the data from the JSON files in the corresponding variables:**



The variable, “pie\_Chart”, stores the array of objects assigned to the object key “pieChart” from the JSON file. The variable, “length”, stores the *number of indexes* this array contains. In context, this number refers to the number of slices the pie chart will have.

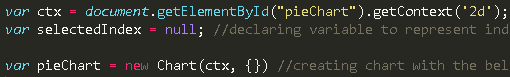
The “.push” function loads the preceding variable with data from within the brackets that follow. Arrays can also be loaded into a variable using “.push” when the index is incrementally looped (instead of replacing existing variable data, it is added onto the end and stored as the value of the next index).

Pseudocode-esque description of the ‘for loop’:

* For every array element in pieChart, push its “label” value into variable “chartLabel” until there chartLabel is an array loaded with one of these values in each increasing index   
  i.e. chartLabel = [ “label1”, “label2”, “label3”, “label4” ]
* For every array element in pieChart, push its “value” value into variable “chartValue” until there chartValue is an array loaded with one of these values in each increasing index   
  i.e. chartValue = [ value1, value2, value3, value4 ]
* Etc.

The idea is that value1 corresponds with “label1”, value2 with “label2”, and so on.

**8. Declare variables to build the pie chart:**

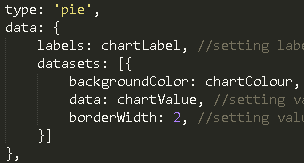


The first line in the above code is a method that returns a drawing context on the canvas. Here, the parameter value “2d” leads to the creation of an object representing a two-dimensional rendering context.

“selectedIndex” is declared as a global (in the scope of the createPieChart function) variable to be used later in slice-specific functions.

The parameters involved in the pie chart creation include the context (ctx) and an object containing chart characteristics (type, data, options).

**9. Specify the pie type and data characteristics:**

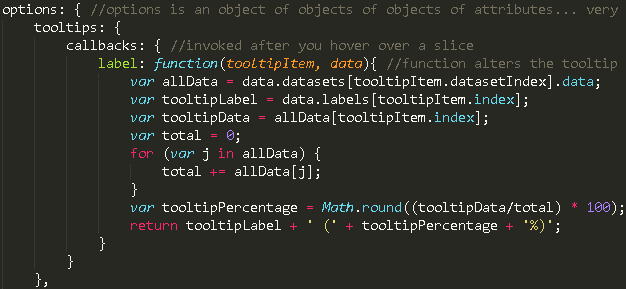


The “type” key dictates what type of chart is created (pie, bar, etc.). The “data” consists of an object of labels and datasets, which are set to the variables that were loaded earlier with arrays of values.

This part of the code is responsible for generating (one example):

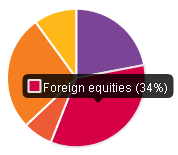


**10. Set tooltips configuration under “options”:**

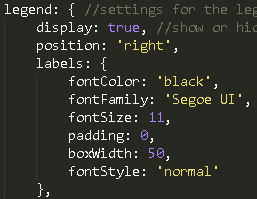


Tooltips appear when pie slices are hovered over. Here, a function has been set so that the tooltips specifically display certain information in the following format:

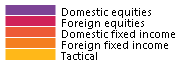
Label (X%) 🡪 The function calculates the percentage of the pie slice and displays it along with the label.



**11. Set legend configuration under “options”:**



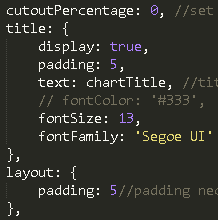
The legend characteristics are set in this section of code; “boxWidth” decides the number of pixels wide that the coloured blocks in the legend will be:





The above line of code is a do-nothing function that is also a legend object. It overrides the Chart JS library’s default legend setting which allows the user to click on a legend item to remove it from the pie chart; the code disables this feature by forcing no events when a legend item is clicked.

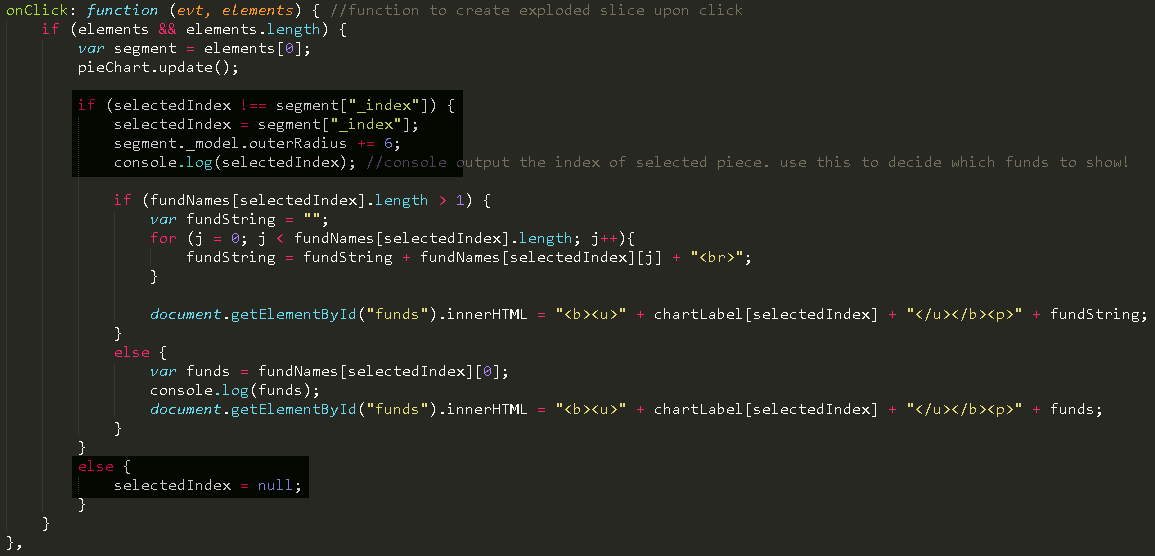
**12. Set basic cut out, title, and layout configurations:**



Cut-out percentage value sets the radius of a circle to be cut out from the center of the chart, creating a donut chart.

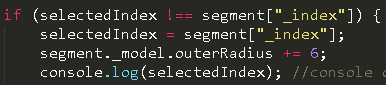
The title stylings are set within title: { }, with the title text being the chartTitle data retrieved from the JSON file.

**13. Create a function that explodes a slice and displays its fund list when it is clicked:**



An onClick: function() { } is created for this purpose. The code within will be split up and explained in the following sections:

**13-a. Create section of function that explodes a slice when it is clicked:**



The variable, ‘segment’, holds the numerical value of each slice that the ChartJS library’s elements[0] array index fed in (from the small section of code above it).

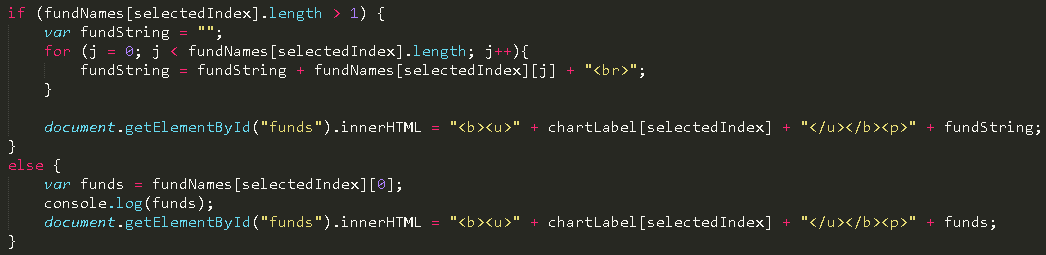
When the slice is clicked, its numerical value is assigned to selectedIndex, and its outer radius property is increased.

\*Only one slice can be exploded at a time. If another one is clicked, the previously exploded one, implodes (well, un-explodes).



If a currently exploded slice is clicked again, this piece of code makes it so that it un-explodes.

**13-b. Create section of function that displays fund list for clicked slice:**



This section is nested within the “if” statement above (not under the else, however). Some fund lists contain many funds, while some only contain one, so the ones that contain 2+ are converted to a string and outputted with line breaks between each fund name.

# **JSON**

The JSON (JavaScript Object Notation) file is the file that feeds data into the JavaScript so that elements can be built. It allows the project to be dynamic, meaning “changing”, as none of the data is hard-coded into the original source code. Quiz elements like questions, answers, answer weightings, pie chart data, and even quiz slides, are all created with objects from the JSON file.

The JSON file is meant to be easy for humans to read and write, which is why it was used in this project. The goal was to make updating and maintaining the quiz simple and intuitive enough so that someone without coding experience could do it.

Each slide is an item in the array of slides. Slides have to be in the order as how they are going to display.

Note:

* Each element must have a “name” followed by colon and a “value”. Both of them have to be in quotation marks
* In JSON, items in array has to be separated by comma; however, the last item should NOT end with comma
* Items in array can have several attributes but they have to be grouped by curly brackets. All items have to be grouped by square brackets
* Calling variables from JS is using the way of “parent.child”

## **Cover** **Page**

**{**

"type" **:** "coverPage"**,**

"title"**:** "Find your Element - INVESTOR QUESTIONNAIRE"**,**

"intro"**:** "This questionnaire is designed to help you choose the right investment to meet your needs by matching your unique situation with the profile of one of the five AGF Elements portfolios. Your tolerance for risk, your time horizon and your expected rate of return over the life of the investment all play a part in finding the right Elements portfolio for you."**,**

"dropDown" **:** **[**

**{**"name" **:** "Jump directly to a portfolio profile"**,** "value" **:** ""**},**

**{**"name" **:** "AGF Elements Yield Portfolio"**,** "value" **:** "yield"**},**

**{**"name" **:** "AGF Elements Conservative Portfolio"**,** "value" **:** "conservative"**},**

**{**"name" **:** "AGF Elements Balanced Portfolio"**,** "value" **:** "balanced"**},**

**{**"name" **:** "AGF Elements Growth Portfolio"**,** "value" **:** "growth"**},**

**{**"name" **:** "AGF Elements Global Portfolio"**,** "value" **:** "global"**}**

**]**

**},**

**Section** **Page**

**{**

"type" **:** "section"**,**

"title" **:** "Investment Objectives & Time Horizon"**,**

"intro" **:** "Investment objectives are the goals or results you want to achieve through investing. Understanding your investment goals can help determine the types of investments best suited to meet your needs. One important factor to consider when selecting investments is the length of time you plan to have your money invested, your investment time horizon. Investors with a longer time horizon have a greater degree of flexibility when building a portfolio and may be able to take on more risk. If you have a very short time horizon, more conservative investments may be more suitable."

**},**

## **Question** **Page**

**{**

"type" **:** "question"**,**

"question" **:** "1. Which of the following statement best describes your primary investment goal?"**,**

"answers" **:** **[**

**{**"text" **:** "I require steady income"**,** "value" **:** "0"**},**

**{**"text" **:** "I want to protect the money I am investing with some growth potential"**,** "value" **:** "4"**},**

**{**"text" **:** "I expect my money to grow over the long term"**,** "value" **:** "8"**}**

**],**

"disclaimer" **:** "Please note that if your client has indicated they require income from their investments by answering 'l require steady income' to Question 1, a monthly distributing investment or purchase option such as AGF Elements Yield Portfolio or a Series T or V purchase option may be prudent for your client depending on their time horizon and risk tolerance."**,**

"img" **:** ".\/q1.png"

**},**

## **Results** **Page**

"type" **:** "results"**,**

"yield" **:**

**{**

"title" **:** "Investor Profile Recommendation: AGF Elements Yield Portfolio"**,**

"left\_column" **:**

**{**

"intro" **:** "Investment Objective - The Portfolio's objective is to achieve high current income by investing primarily in a diversified mix of income, bond and equity funds that may include exposure to income trusts, royalty trusts and REITs. The portfolio currently offers a 5% annual targeted distribution, paid monthly."

**},**

"right\_column" **:**

**{**

"subtitle" **:** "Targeted Asset Location"**,**

"pieChart"**:** **[**

**{**

"label"**:** "Domestic equities"**,**

"value"**:** 13**,**

"colour"**:** "#7C4496"**,**

"funds"**:** **[**

"13% AGF Dividend Income Fund"

**]**

**},**

**{**

"label"**:** "Foreign equities"**,**

"value"**:** 15**,**

"colour"**:** "#D02259"**,**

"funds"**:** **[**

"15% AGF Global Dividend Fund"

**]**

**},**

**{**

"label": "Domestic fixed income",

"value": 16,

"colour": "#ED5A36",

"funds": [

"16% AGF Fixed Income Plus Fund"

]

},

{

"label": "Foreign fixed income",

"value": 56,

"colour": "#F47E20",

"funds": [

"21% AGF Total Return Bond Fund",

"23% AGF Global Bond Fund",

"6% AGF High Yield Bond Fund",

"6% AGF Emerging Markets Bond Fund"

]

}

]

}

},

# **Generating Report**

## **Overview**

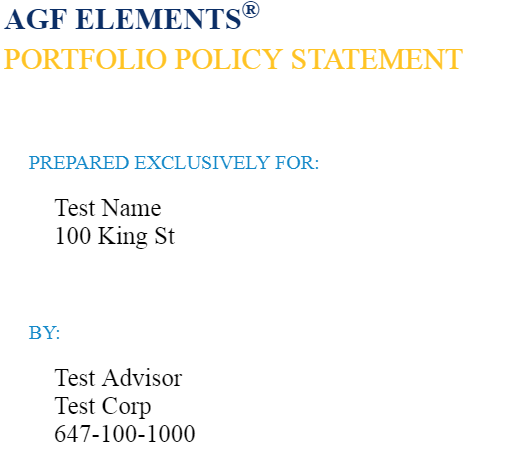
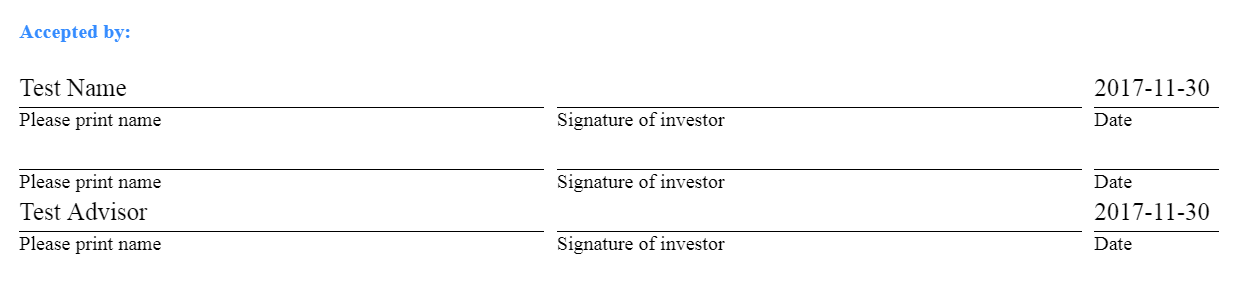
The report generating feature is used to save a PDF version of the portfolio report to user’s computer.

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.



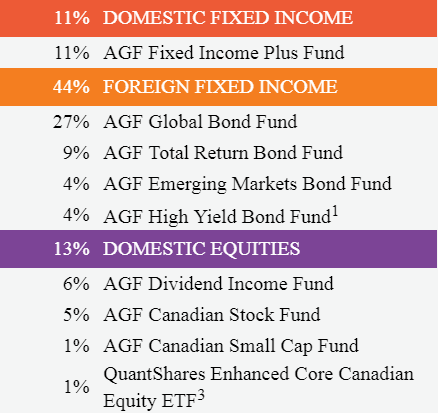
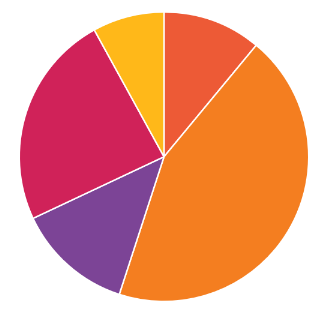
When user clicks on the “REPORT” button, a new webpage will open, and display the report:

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:

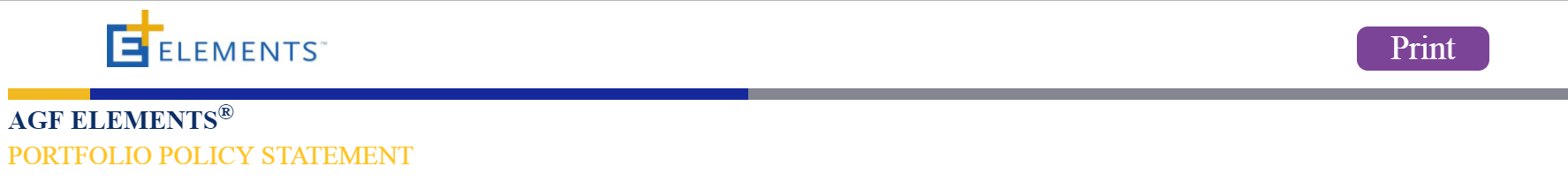
*“report-conservative.html?form/client-name=Test%20Name&form/address1=100%20King%20St&form/address2=&form/address3=&form/advisor-name=Test%20Advisor&form/firm-name=Test%20Corp&form/phone=647-100-1000&form/date=2017-11-30&form/class-name=Non-Corporate”*

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).

## **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.



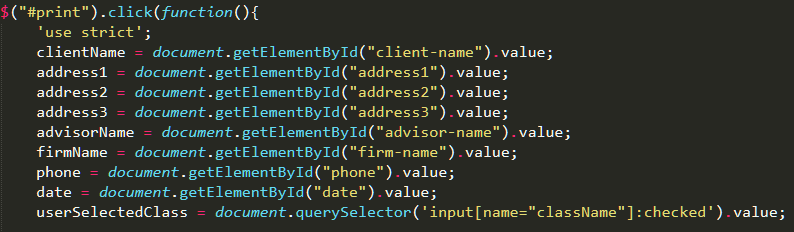
By default, all the option is unselected, so a new attribute “checked” is used to make the first radio selected.

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.

## **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.

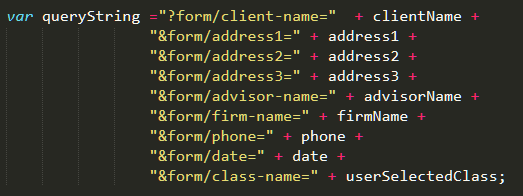
Query string is the part of URL which containing data that does not belong to hierarchical path structure.

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)

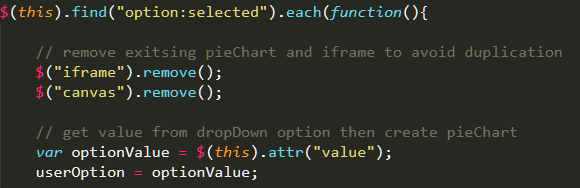
Query string is start with “?”, following with parameters’ names and values. Each parameter is separated by “&”.

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

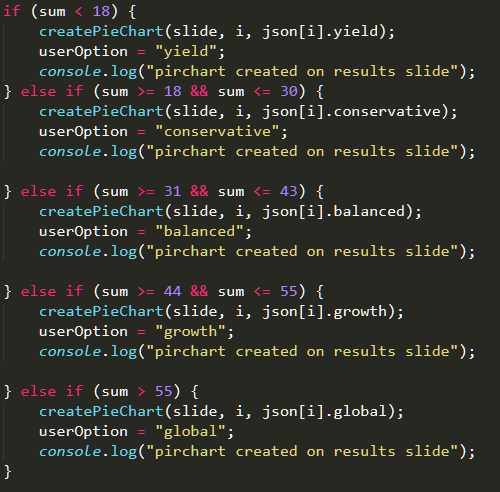


(main/function.js)

“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:

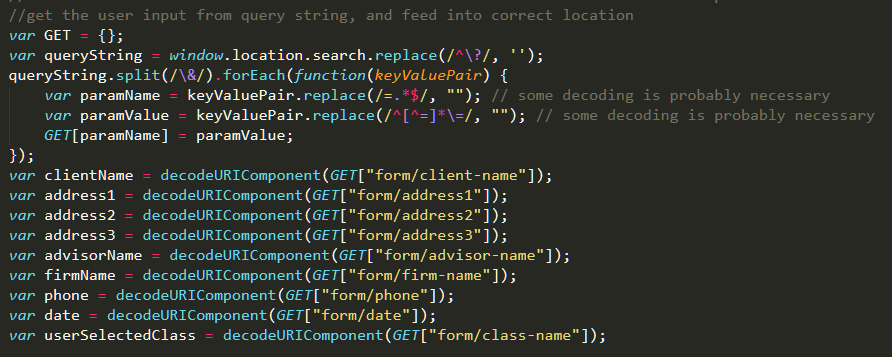
 (main/function.js)

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.

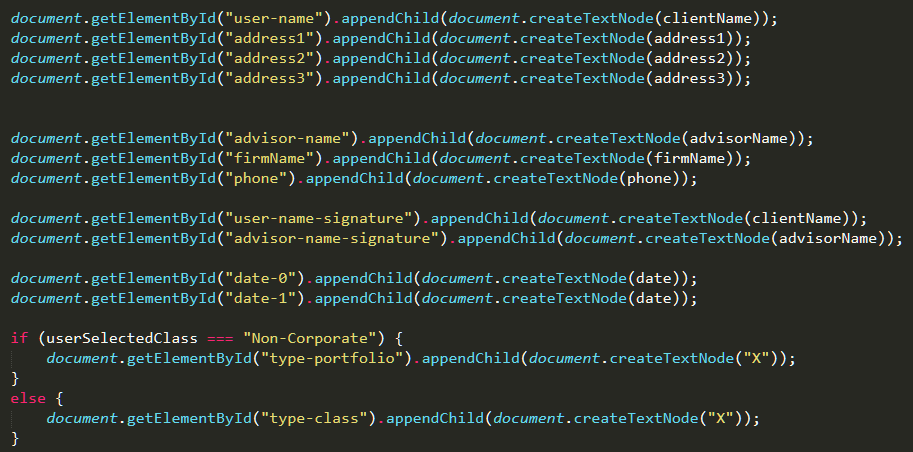
### **Converting query string to an object:**

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)

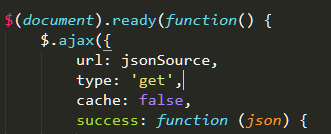
This part is simply use JavaScript DOM function to put data into the correct location.

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()).

## **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.

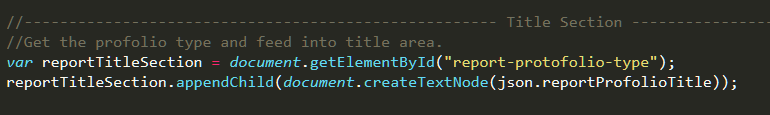
All data is stored in the JSON file. When the report DOM is ready, Ajax request is sent to get JSON.



### **The title section**



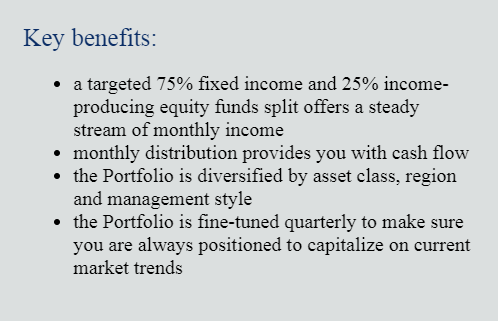
Is feed by



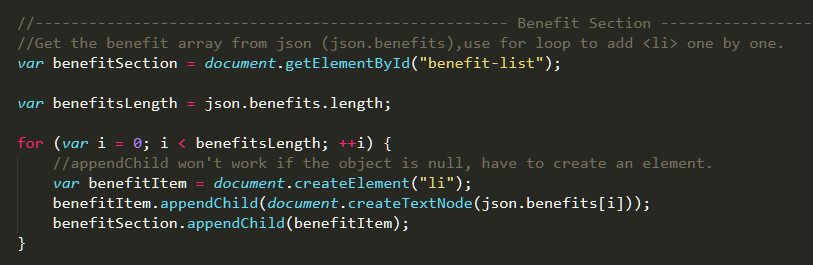
Here is the JSON:



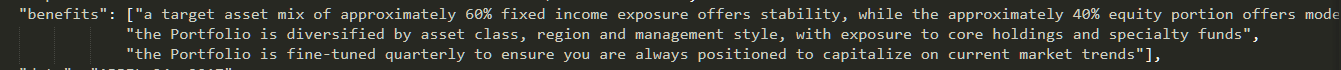
### **The benefit section:**



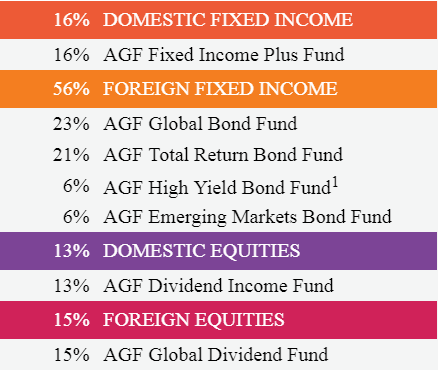
Is controlled by this piece of code:



All the benefits is stored in an array structure:



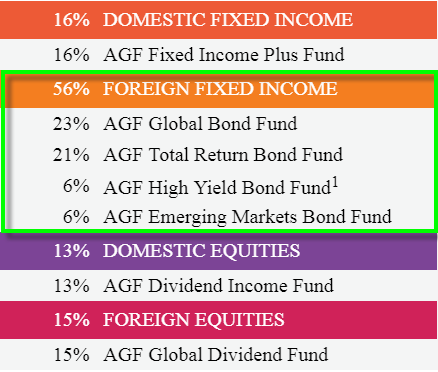
### **The table section**

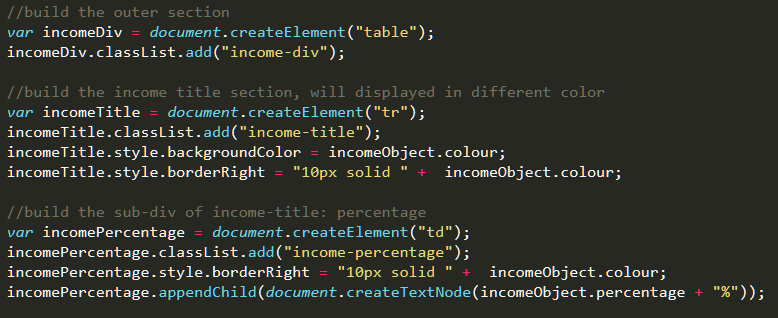


the table section is built by a helper function:

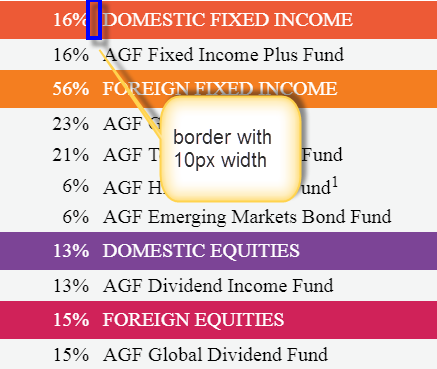


This function will consume the element node (divToBeAppend), and an object called incomeObject. Note that this function only build one “income title” with its sub fund:

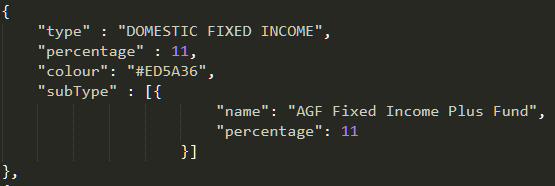
 this big “table” actually contains four tables.



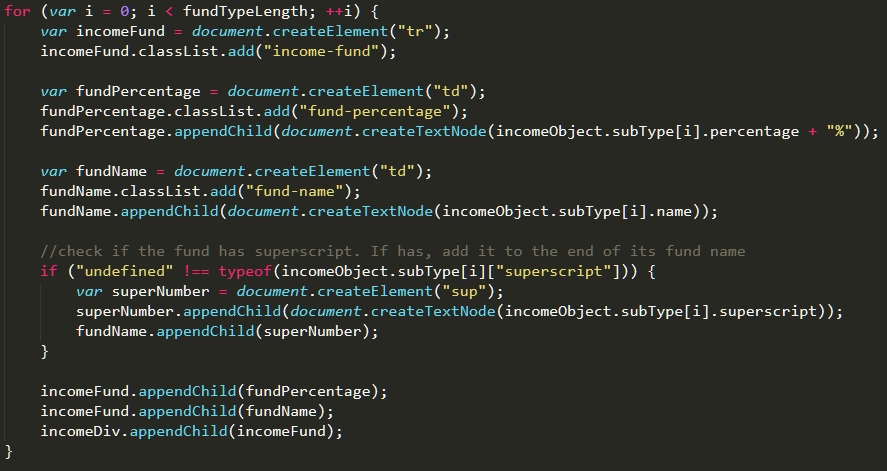
This section is used to construct the table. Table head background color is from JSON. “border-right” is added in order to separete percentage number and income/fund name.



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

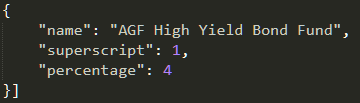
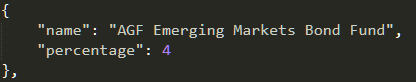
 an example of incomeObject

Those code is used to build the fund row:

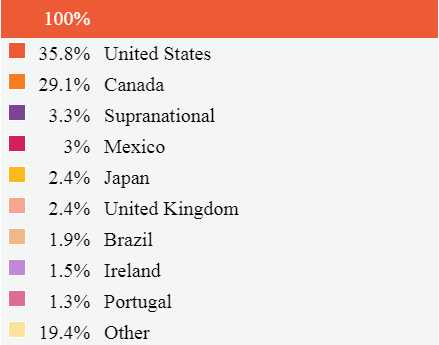


Some fund name may contain superscript, so an if statement is here to check the fund object. If property “superscript” is defined, then append its value to the end of fund name:

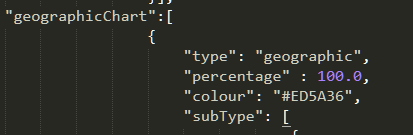
Fund object with “superscript” defined: Fund object without “superscript”:

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

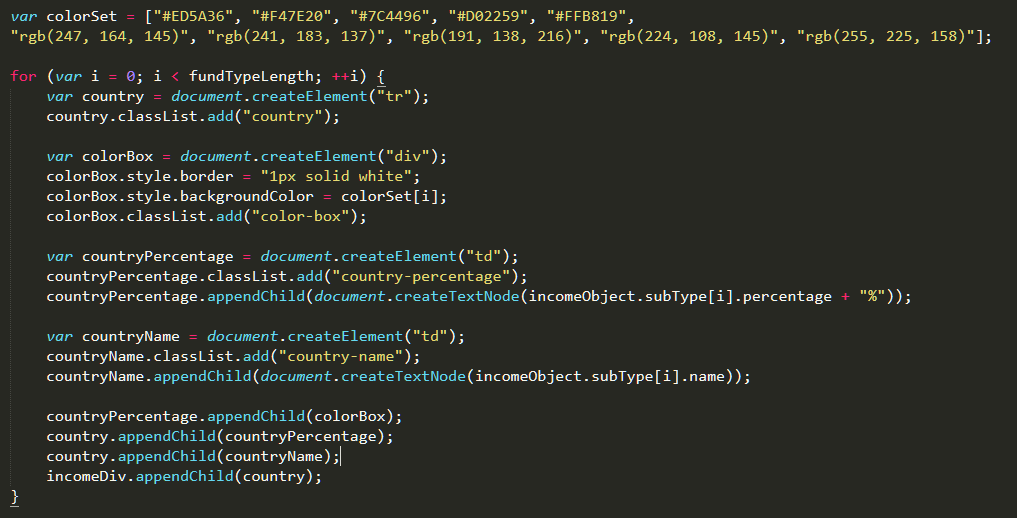


 this case is for non-geo table (first and second table), and else case is for geo table (third table)

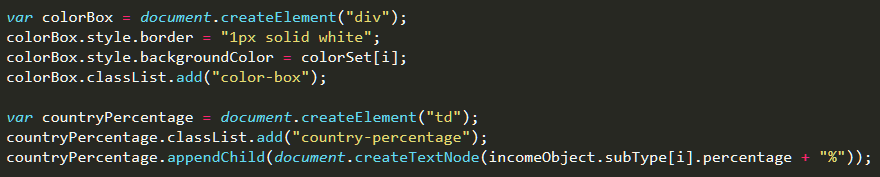


JSON object (only third table has type “geographic” (geographicChart[0].type == “geographic”)).

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

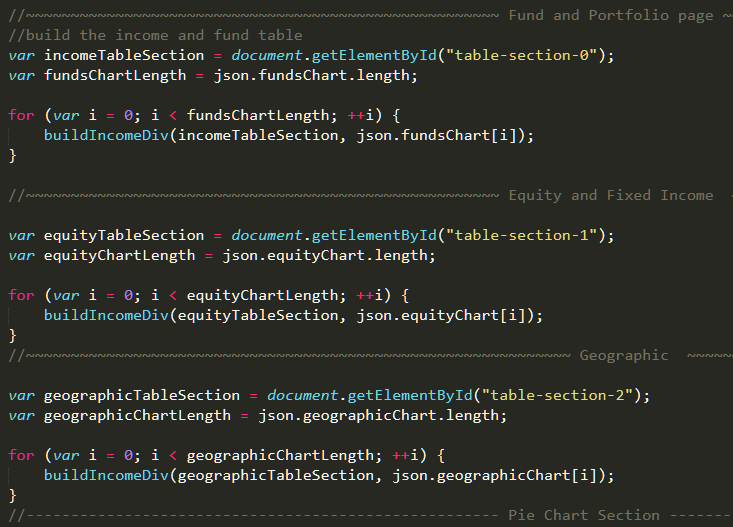


It has an array of color code string and a block of code to create the color box on the left of percentage number:



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.

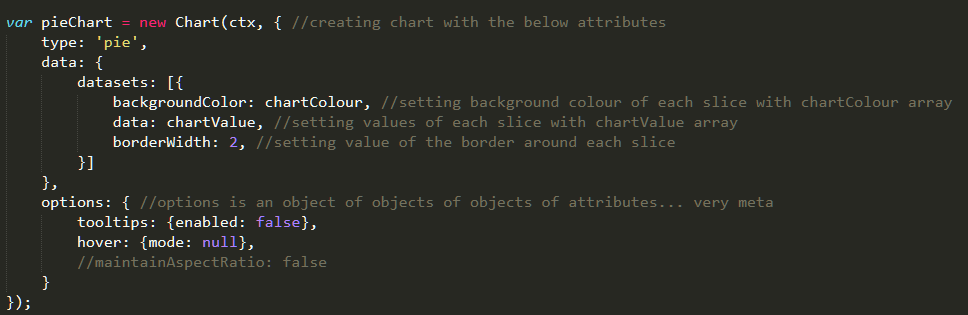
In the main function, an array of incomeObject will be looped through, and pass every element into “buildIncomeDiv()”. Running time O(n^2).



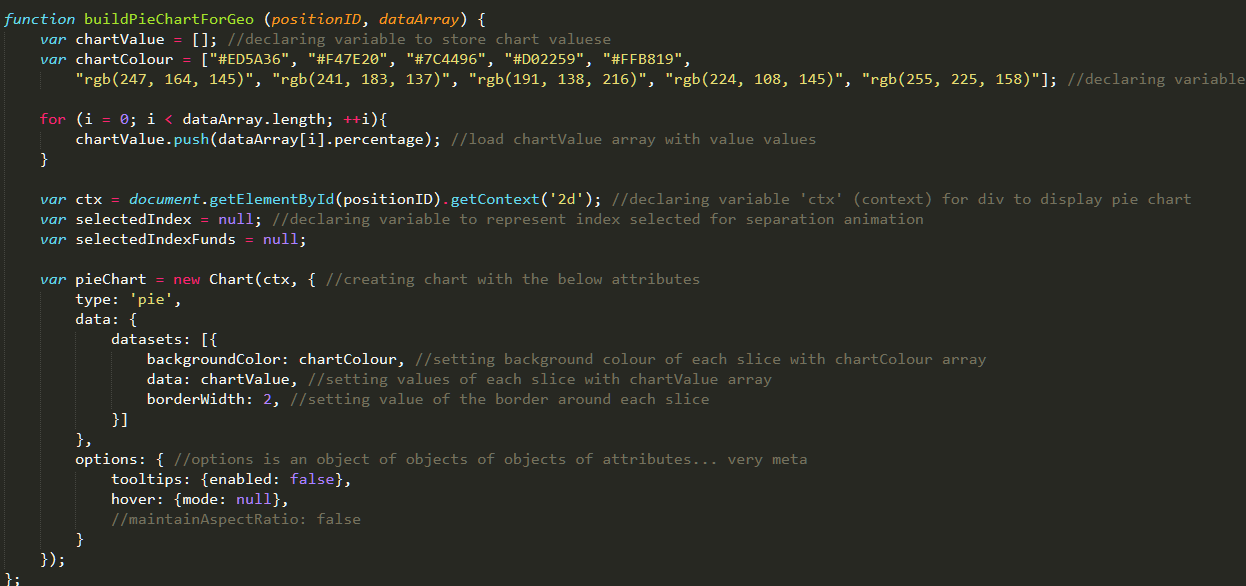
### **The Pie Chart Section**

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.



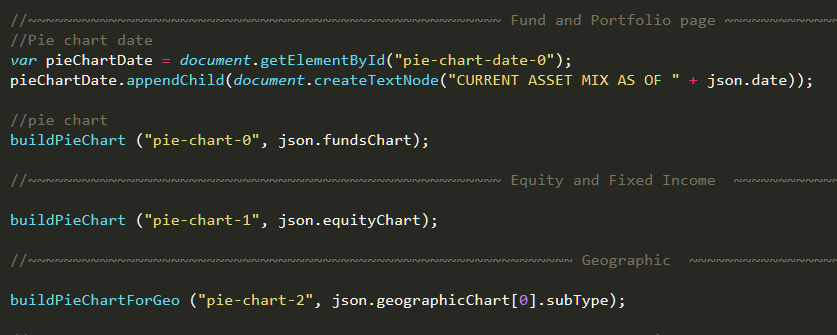
For geographic pie chart, the color set is different. Thus, there is an array of color code string in geo pie chart function:



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()”:



## **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.

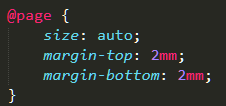
 and .

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:

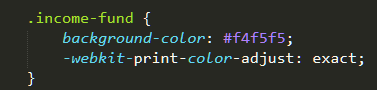


This is also a adjustion for the print view:



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.

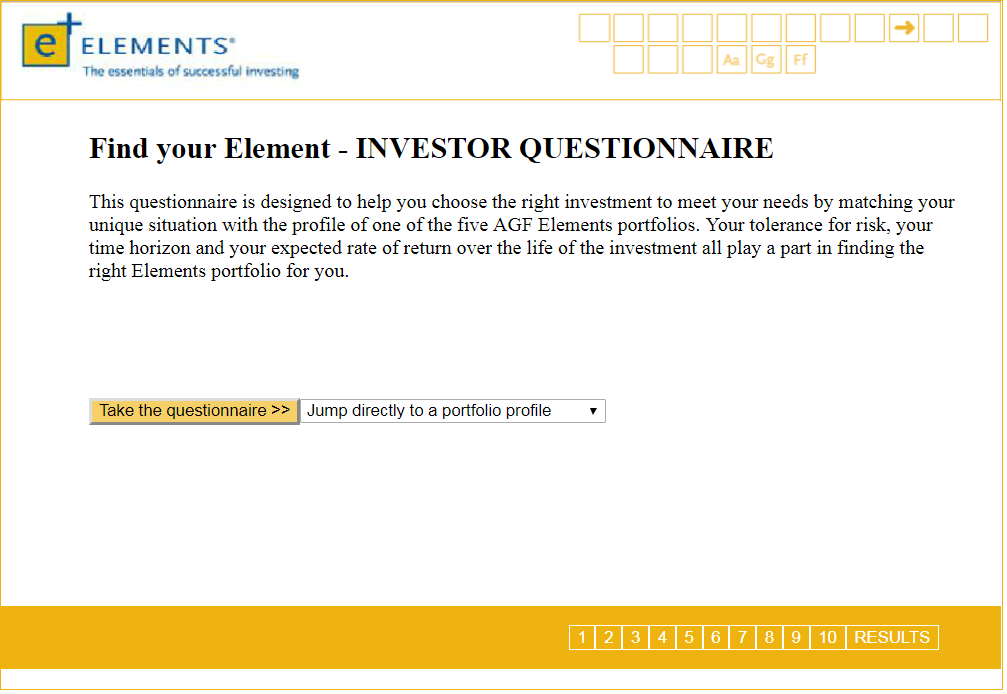
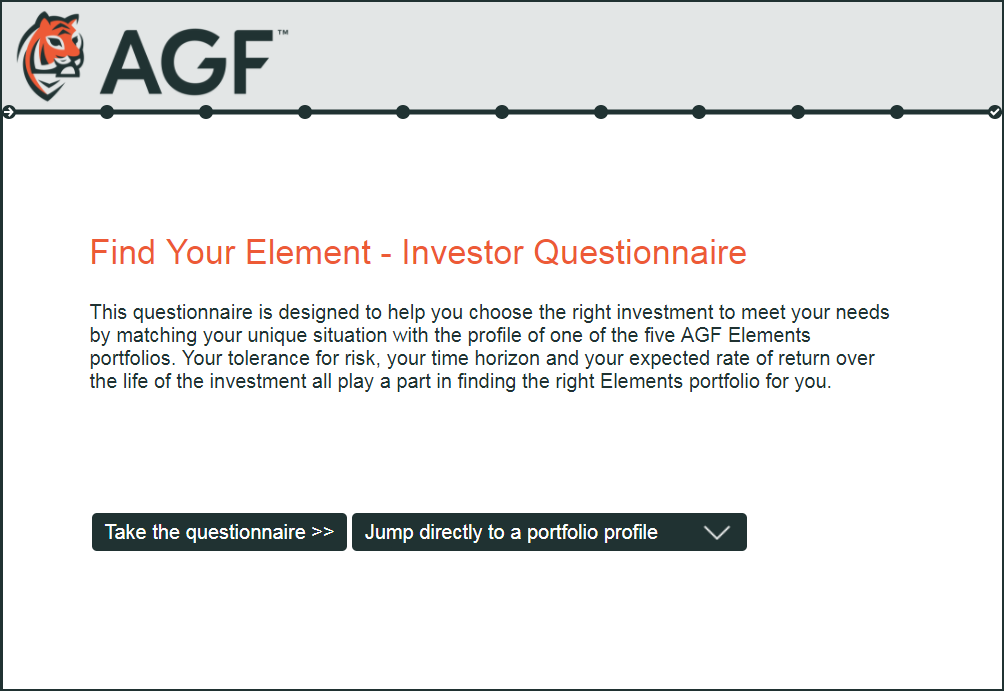


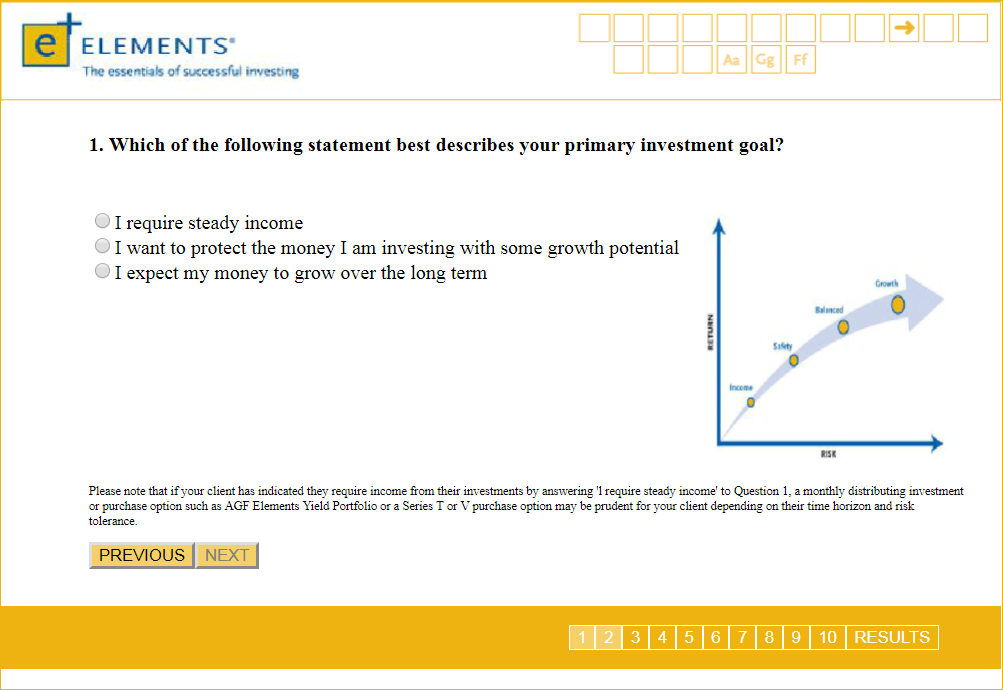
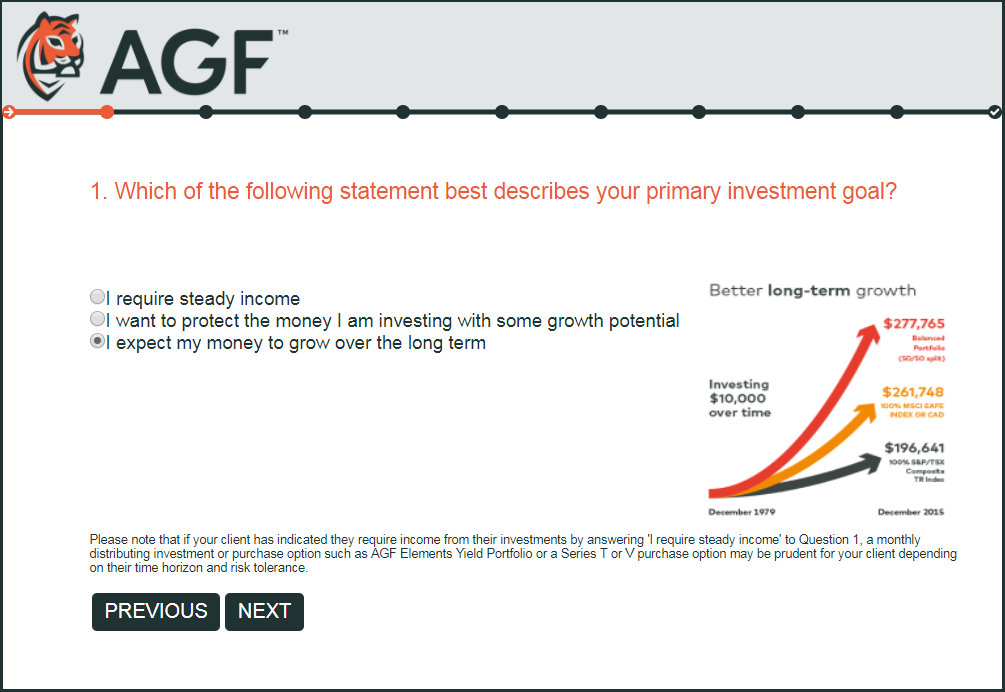
On the top right of the page, there is a print button. It is juat invoke “window.print()”



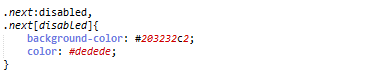
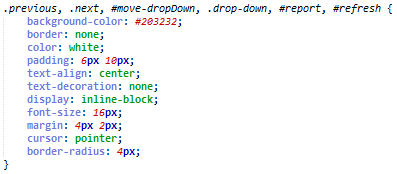
# **Application Redesign**

The elements quiz was redesigned to match the AGF.com branding through a series of CSS and HTML changes. While the layout of each slide was left fairly similar (with minor margin and padding tweaks), some major changes included the progress bar and button styling. The majority of the redesign changes are made before the “end of most redesign changes” comment in “stylesheet.css”.

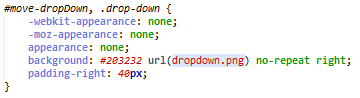
All of the buttons throughout the slides were styled using the following straightforward CSS:



Button styling

Disabled button styling (before an answer has been selected)

The changes to the “Jump directly to a portfolio profile” dropdown are slightly more complicated. In addition to the changes that affect all buttons shown above, the default dropdown arrow was removed using the ‘appearance’ property (-webkit for Chrome and Safari, -moz for Firefox… more can be added as needed for other browsers). The new arrow that is put in is actually a background image (dropdown.png). The background is set to a default colour (#203232 is the same colour as the background image) and the file location of the image is added. It is set to not repeat and be right justified to work with the padding and ensure that the arrow stays on the right of the button regardless of the length of text inside.



Remove default arrow

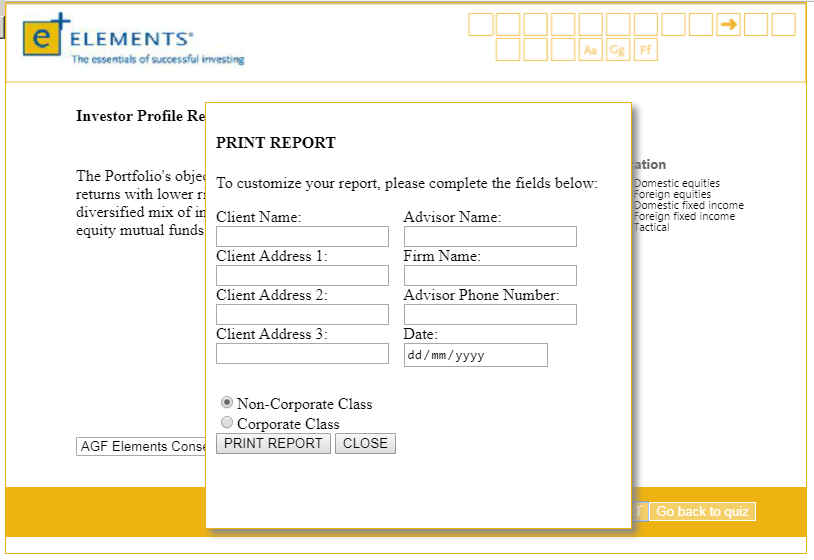
Add custom background image

Leave space for background’s arrow

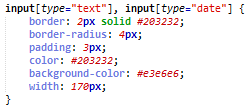
Background png image



The report modal window was also styled to match the redesigned quiz. Some layout and font changes were made, as well as many changes to the styling of the input fields.

Button styling consisted of changing the border and border radius of the text and date input fields, to make them dark gray and slightly rounded. Padding was also added between the filler text and the border of the input fields. A background color was added, and the width was set so that the date input would be just as wide as the other input fields.



Modal text input styling

The labels above the input boxes were removed and replaced with placeholder text in the HTML, as shown below:



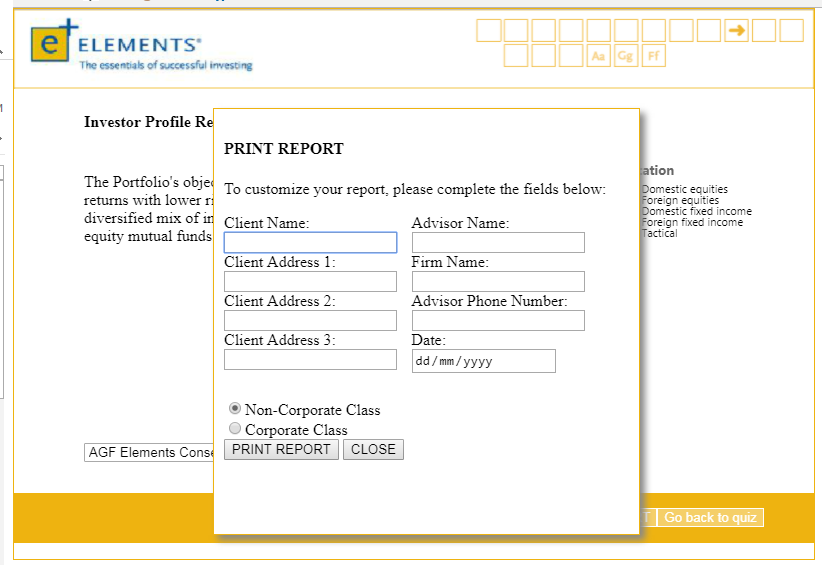
The default ‘glow’ around each text input field was removed with the outline property, and an orange border when the user clicks into the field to edit the text was added.



Removing the default blue ‘glow’ on an input field



Adding the orange border

Final result when focusing on field

The progress bar at the bottom of the original design was removed and replaced by one just under the quiz header. This header requires three files: progress-kiaod.css, progress-style.css and progress-style.scss. Most of the CSS changes to the progress bar were through progress-style.css.

The small arrow and checkmark icons which were added on the first and last steps of the progress bar are SVGs within each step (each circle) of the progress bar. As of the time this documentation was written, the progress bar is not dynamic; it does not adjust with the number of slides in the quiz. This can be changed later on by taking the number of slides from the JSON and following the logic of the old-style pagination (separate documentation).

