COM554 Technical Log Report

Week One

I began my RIA by firstly creating wireframe diagrams of what I wanted it to look like. This helped me to improve upon my requirements too. I based the design on a website of a similar topic https://bitcoin.org/en/. The content of each tab of the RIA is in <section> tags. To create this I used the below CSS which created a rounded border around each section and gave it padding so the section wouldn't start or end at the edge of the window. Line 58:

```
section {    padding-top: 5px;    padding-bottom: 30px;    border: 2px solid
lightgrey;    border-radius: 5px;    background-color: white;    }
```

In the <body> tag I added max-width and margin properties so the content of the RIA is displayed in the middle with white space on either side. Line 4:

```
body {    max-width: 1000px;    margin: auto; }
```

I then added contrasting background colours to <section>(white) and <body> (`#F7F7F7)` to give more focus to the content. I also imported a font set from Google fonts. Line 1:

```
@import url('https://fonts.googleapis.com/css?family=Ubuntu');
```

I plan to add more styling to my RIA after every feature I implement.

Week Two

Tabs

The first requirement I implemented was tabs. I added the css class="tab" to each <a> link in the <nav> element. I used a jQuery .on("click ") event method to detect when an element with the .tab class had been clicked. Line 5:

```
        $(".tab").on("click", function () {
```

I added the css class "active" to the tab clicked and removed the css class "active" from the other tabs. To remove "active" from all links except the class clicked, I used the .siblings() function which allows me to retrieve all of the siblings of this element in the DOM tree. Using .addClass() and .removeClass() in this allowed me to meet requirement 1. Line 6:

```
        $(this).addClass("active").siblings().removeClass("active");
```

Each <section> has an id with the same value as the corresponding tab link contents and has the class “tabcontent”. To remove the current displayed content I used the .hide() method on the tabcontent class which hides all displayed elements of the class. The next step was to display the content of the tab clicked. I used $(this).html() to get the contents of the link clicked and assigned it to var tab. The .show() method was run on “#”+tab to display the desired section. Line 9:

```
        var tab = $(this).html();  $("#" + tab).show();
```

To make the home section display upon load of the RIA and the other sections remain hidden I added the display property of “none” to the .section css (line 59). I then added an id “Home” to the home <section> and for the css of “#Home” I added the display property “block”.

Menu

The next feature I implemented was a drop down menu containing tab links. It appears when the mouse is hovered over a nav link. A css class “dropdown” was added to a <div> that contains the dropdown menu code. When the mouse hovers over the “dropdown“ element, the .hover() event method is triggered. It triggers both mouseenter and mouseleave events. Starting at line 22:

```
   $(".dropdown").hover(
        function (event) {     $("#MenuDropdown").toggle(200);        },
        function (event) {     $("#MenuDropdown").toggle(200);        })
```

The .toggle() method changes the css display value of #MenuDropdown to “block” when the mouse hovers over the element or to “none” when the mouse is no longer hovered over it. The parameter “200” in .toggle() is used to create a fade in/out animation of a 200 milliseconds duration.

Week Three

Accordion

To create the accordion I added a <button class=”accordion”> followed by a <div class=”accordionpanel”> to my HTML for each accordion part. A .click() event method was used to handle a click on the button. I used the nextElementSibling property on the button (this) which returns the element immediately following it in the DOM tree. I then used the .toggle() method to change the display of the element either block or none. The parameter 200 in this method allows for it to fade in/out in 200 milliseconds. Line 33:

```
        $(".accordion").click(function () {
        var panel = this.nextElementSibling;
        $(panel).toggle(200);   })
```

HTML5 Video

The next feature I focused on adding was a HTML5 video. I created a blank div with the id "VideoBox" in my HTML. I then used the .append() method on this id. This method inserts the content from the parameter as the last child in the element.

```
$('#VideoBox').append(
    '<video id="EthereumVideo" width="900" controls>' +
   ' <source src="ethereum.mp4" type="video/mp4"> Your browser does not support
   the video tag.</video>')
```

Week Four

Hide/Show/Toggle Buttons

I added buttons that allowed for control of the video player. A hide/show button which either hides or shows the video player and a small/large button which makes the size of the video player small or large. Both buttons were controlled using a .click() event method. To hide the video I used the .toggle() method on the id of the video element. It also has the parameter of 400 which allows for a fade in/out transition of 400 milliseconds.

To change the size of the video player I used the .css() method to get the width value of the video and if it was large (900px) then the .css() method was used again but with another parameter to change the width to small (500px). If the width property wasn't large then the if statement fell to an else statement which used .css() to change the property to large. The initial width was already set when the video element was appended to the HTML. Lines 116:

```
$("#EthereumVideo").css("width", "500px")
```

After adding these video features I realised the video continued playing when it was hidden and when the user navigated to a different tab. To prevent this I used the .paused() method into the tab click and hide/show click event methods. Line 10 and 109:

```
$('#EthereumVideo')[0].pause()
```

Tooltip

After this I added a tooltip to the RIA. It is displayed when the mouse is over the video element and disappears when it leaves the video element. I initially tried to use the hover method to handle the mouseenter and mouseleave events but the tooltip would not follow the mouse. I discovered this was because the mouseenter event is only called once when the mouse enters the video element. To get around this I instead used the mousemove and mouseout event methods on the video element. The mousemove event is activated every time the mouse moves within the video element.

In the mousemove event method it firstly removes any existing tooltip using the below code (line 125). The .remove() method removes the entire tooltip element and everything inside it. This prevents multiple tooltips from being appended to the page. Line 136:

```
$("body #ToolTip").remove();
```

I then used the .append() to add the tooltip to the body of the page. The .css() method is then used to change top and left properties to the location of the mouse. An event object was passed through the event method which contains information about the event including the X and Y coordinates of the mouse. Line 128:

```
$("#ToolTip")
    .css("top", (evt.pageY) + "px")
    .css("left", (evt.pageX) + "px");
```

When the mouseout event method triggered on the video element it simply removes the tooltip using the .remove() method. Line 136:

```
$("body #ToolTip").remove();
```

Week Five

Automatic Construction of a List, Select and Table Element

To implement the automatic construction of a list and select element I used fairly similar code. A .getJSON() method is called on the page load which loads a local json file. For example line 70:

```
$.getJSON("json/howtobuy.json", function (data) {
```

A for statement is then used to iterate through the JSON data and .append() is used to add each value to a <ul> element as a <li> element. Line 72:

```
for (var k in data) {
    $("#EthFacts").append("<li>" + data[k] + "</li>");       }
```

There is a also another list which is automatically populated using .getJSON(). This list is an ordered list. It works the same as above. See line 72:

```
$("#HowToBuy").append("<li><p>" + k + ": " + data[k] + "</p></li>");
```

For populating a select box the code was very similar. A JSON is retrieved but this time the key is added instead of appending the value. It is appended to a select element as an option element.

```
$("#FAQSelect").append("<option value='"+k+"'>" + k + "</option>");
```

When an option is changed the .change() event method is triggered. In this method the value of the selected option is used to retrieve corresponding JSON data and display it. I used a css selector ':selected' to get the selected option element and the .val() method to return the value of the element. Line 87:

```
option = $('#FAQSelect option:selected').val()
```

The previous content of the div with the id "FAQSection" is emptied using the .empty() method. An if statement is then used to determine if the value is not the default value of "Select..". If not then a .get() request returns a local JSON file. The value of the selected option is used as a key to return a value with multiple key/value pairs. A for statement is used to append this to the div with the id FAQSection. Line 90:

```
$.getJSON("json/ethereumfaq.json", function (data) {
    for (var k in data[option]) {
      $("#FAQSection").append("<hr> <h3>" + k + "</h3><p> " +
      data[option][k] + "<\p>  ");
    }
})
```

While working with the .get() method I decided to add a feature that displayed the live price of Ethereum and updated it at an interval of five seconds. Line 40:

```
setInterval(function () {
        retrievePriceInfo()
}, 5000);
```

During each interval it also adds the price and the current date and time to a table. To do this it uses .get() to perform an external API request to cryptocompare.com which returns a JSON. The html() method is used to insert price data from the JSON into a <p> element with the id "EthereumPrice". A check is then performed on the table that limits the amount of rows to 10 by deleting the last row if it is longer than 10. Line 51:

```
$("table#EthPriceTable tr:nth-child(10)").remove();
```

The Date.Now() method is used to get the current date and time which is inserted along with the price into the first row of the table. LIne 56:

```
$("#EthPriceTable tr:first").after("<tr><td>" + now + "</td><td>£" + data['GBP'] +
"</td><td>$" + data['USD'] + "</td></tr>");
```

Week Six

Animation

When the user clicks the game tab the game section title is animated using the .animate() method. It makes the font three times bigger and the color changes slightly by adding some opacity. This animation takes 1.5 seconds to complete. Line 12:

```
if (tab == "Game") {
     $("#GameTitle").animate({
          opacity: 0.9,
          fontSize: "3em"
     }, 1500);
```

Game

The final feature that I added to the website was a game. It used the HTML5 canvas for drawing. The logic for this is stored in game.js. The aim of the game is to catch falling coins in a wallet which moves along the X axis and follows the mouse. A set interval method is used to create a loop which controls the game. When the mouse enters the canvas the game begins. The following logic occurs during every iteration.

- Canvas is cleared. Background color, player score text and the wallet image are drawn on canvas.
- The Y position of the coin is updated. This Y position increases a larger amount every time the player scores. The coin image is then drawn on canvas.
- A check is performed to see if the coin is colliding with the wallet. If it is then one point is added to the player score, the coin Y position is reset to the top of the canvas and the X position is chosen randomly.
- A check is performed to see if the coin Y position is below the bottom of the canvas. If it is the coin is reset to the top.
- If the player score is equal to 25 then the game restarts.

I used jQuery to retrieve the canvas element and image elements. It was also used to handle a mousemove event over the canvas.

Summary

Requirement	**Filename**	**Relevant Line of Code**
1. Use of jQuery selectors to add and remove classes	scripts.js	6
2. Adding elements directly into the DOM	scripts.js	56, 64, 72, 80, 92, 100, 126
3. Adding HTML5 elements	scripts.js	100-103
4. Adding css	scripts.js	9, 116, 118, 129, 130
5. Tabs	scripts.js	5-18
6 Accordions	scripts.js	33-36
7 Hide/Show/Fade/Toggle capabilities	scripts.js	Fade/Toggle: 24 ,27, 35, 107-110 Hide: 7 Show: 9
8 Menus (not simply select elements)	scripts.js	22-29
9 Event Handling	scripts.js	5, 22, 33, 86, 107, 114, 124, 135
	game.js	22
10 Tooltips	scripts.js	124-136
11 Basic animation effect	scripts.js	13-16
12 Automatic construction and dynamic population of: (a)list (b)table (c)select element	scripts.js	List: 62-66, 70-74 Table: 47-57 Select: 78-82
Additional functionality		
Game	game.js	Entire file