EECE 430L | Lab #1: Web Frontend I

Introduction

This lab will guide you through building a basic website that will allow a user to input LBP-USD exchange transactions and will display the average exchange rate accordingly.

There will be two rates maintained, one for selling USD in exchange for LBP, and one for selling LBP in exchange for USD. Accordingly, the user will specify what type of transaction they have completed when entering a transaction, and the exchange rate for both types of transactions will be listed to the user.

The website will adopt a simple logic, it will display the final exchange rate as simply the average of each transaction of each type. Each transaction will be weighed accordingly. If no data is present (for one of the two transaction types), it will indicate to the user that the exchange rate is not available yet to display.

The website will be built with HTML, CSS, and JS, it is recommended to use VSCode as a code editor and Google Chrome to view the website. The code will be maintained using Git. Before you begin, create a GitHub account using your AUB email and with a username that makes your name obvious. We will adopt the basic model of simply making sequential commits on the main branch.

Throughout the lab, everytime a HTML element, CSS property, or JS construct is used that you are not familiar with, take the time to look it up on MDN before moving forward.

Phase 1. Displaying Current Exchange Rate

1. Begin by following the link to the GitHub assignment. Once you accept the assignment, there will be an empty repository automatically created for you to work within. Obtain the URL of the repository, and then create a directory anywhere on your machine that will house your project. Within that directory in the command line execute:

```
git clone URL_OF_YOUR_REPOSITORY
```

2. Create a file within this directory named `index.html`. This file will effectively contain your website. Open this file in your code editor.
3. Add the following content into your file. The `<html>` tags indicate that their contents are an html document. The `<head>` section will contain metadata about the document. The `<body>` section will contain the content of the document.

```
<html>
<head>

</head>

<body>
```

```
</body>
</html>
```

4. Open the `index.html` file in your browser. You will naturally see an empty document/website. Now we will add basic metadata that will indicate the title of the website. Within the `<head>` section, add a `<title>` element, and as its contents type in what you would like to be the title of this project. Save the file, and reload the page in the browser in order to load the most recently saved version of the website. You should see your title in the tab of the page. Ex:

```
...
<head>
  <title>LBP Exchange Tracker</title>
</head>
...
```

5. As a first step we will add the page elements that we will use to display the current exchange rates. We will add the section that allows the user to input transactions later. Within the `<body>` tag, add two `<div>` sections. The first one will contain the page header, and the second will contain the elements that display today's exchange rate.
6. Within the first `<div>` add a `<h1>` header. As its contents, add your desired header for the page.
7. Within the second `<div>`, add the following code block. Note that we can add an ID to any HTML element, however we will add them only to the two span elements below because we are going to use the IDs to reference them later.

```
<h2>Today's Exchange Rate</h2>
<p>LBP to USD Exchange Rate</p>
<h3>Buy USD: <span id="buy-usd-rate">Not available yet</span></h3>
<h3>Sell USD: <span id="sell-usd-rate">Not available yet</span></h3>
```

8. Looking at the webpage now, you should see the *structure* of the elements needed to display the current exchange rate. To enhance this page, we will now *style* it using CSS. Create a file within your project folder named `style.css`. Within your HTML file, you will need to make a reference to this CSS file, to specify that the contents of the CSS should apply to your `index.html`. Do so by adding the following element in the `<head>` tag.

```
  <link rel="stylesheet" href="style.css">
```

9. Within the CSS file, we will add the following ruleset that will set the background color and font-family to be used for the whole website, by applying the two properties to any HTML element of type 'body'. Feel free to add any other values you would like for the color and font.

```
```

```
body {
  background-color: #e9ebf0;
  font-family: Helvetica, Arial;
}
```

10. Now to add styling to the page header, we will create a CSS *class* with a couple styling properties. Additionally we will add a color property to all `<h1>` elements that exist within the 'header' class element. Add the following rulesets to the CSS file:

```
.header {
  background-color: #0093d5;
  padding: 10px;
}

.header h1 {
  color: #fff;
}
```

11. In order to have the above class apply to our page header, attribute the above class to the `<div>` element in your HTML file that contains the page header. You can do so by modifying the `<div>` element to look like so:

```
<div class="header">
    <h1>YOUR PAGE HEADER</h1>
</div>
```

12. Now instead of having the page content be left-aligned, we can have it be centered in the page with a wrapper. To do so, add to the second `<div>` element that contains the page contents the class attribute `wrapper`. After doing so, we will define the following rulesets in the CSS file to achieve this styling:

```
.wrapper {
  margin-top: 50px;
  background-color: #fff;
  border-radius: 15px;
  margin-left: auto;
  margin-right: auto;
  padding: 30px;
  width: 50%;
}

.wrapper p {
  font-style: italic;
  color: #666;
}
```

13. We will now create a git commit with your work until this point. Use `git add .` to stage the newly added files for commit. Then execute `git commit -m "MESSAGE_DESCRIBING_THIS_PHASE"`.

Phase 2. Accepting New Transactions from User

14. We will now add a section to allow a user to record a transaction of theirs. We will have to collect from the user the USD amount, the LBP amount, and the type of the transaction (LBP for USD, or USD for LBP). Within the main content `<div>`, add the following elements to create a break, add a header for the section, and a form which will contain the user input.

```
...
<hr />

<h2>Record a recent transaction</h2>
<form name="transaction-entry">

</form>
...
```

15. Add the following code block within the form element to have an `<input>` to collect the LBP amount of the transaction, and to have a label associated with this input. Note that the `for` attribute of the label associates it with an input element (by the ID of the input element). Note also that the `type` attribute of the input is set according to the type of expected input.

```
<div class="amount-input">
  <label for="lbp-amount">LBP Amount</label>
  <input id="lbp-amount" type="number" />
</div>
```

16. Add now the necessary elements to collect the USD amount of the transaction (similar to above). Set the ID of the input to be `usd-amount`.
17. Add the following CSS rulesets to style the elements we've just added.

```
.wrapper label {
  display: block;
}

.wrapper input {
  width: 600px;
  padding: 3px 6px;
}

.amount-input {
```

```
  margin-bottom: 10px;
}
```

18. Now that we can collect the transaction quantities, we still need to allow the user to specify the transaction type. Add the following `<select>` element after the other form items, and add the following CSS ruleset to style it.

```
<select id="transaction-type">
  <option value="usd-to-lbp">USD to LBP</option>
  <option value="lbp-to-usd">LBP to USD</option>
</select>
```

```
.wrapper select {
  display: block;
}
```

19. Finally, we need to allow the user to submit their transaction. Add the following `<button>` at the end of the form, with its corresponding styling. For the styling, note the *pseudo selector* which applies when the button is in the hover *state*.

```
<button id="add-button" class="button" type="button">Add</button>
```

```
.button {
  margin: 10px 2px;
  padding: 5px 20px;
  border-radius: 8px;
  background-color: #0093d5;
  border: none;
  color: #fff;
  text-align: center;
  display: inline-block;
  font-size: small;
}

.button:hover {
  cursor: pointer;
  opacity: 0.75;
}
```

20. To provide some spacing between the sections, add the following CSS ruleset:

```

```

```
.wrapper hr {
  margin: 40px auto;
}
```

21. Make your browser full screen, and then resize the window gradually to the smallest size possible. You will realize that the website is not rendered ideally for smaller screens, especially with the styling we have added to the input fields. We will make some adjustments to respond appropriately to different screen sizes, and layout the elements in a different way depending on the size of the screen. This is called *responsiveness*. Remove the `width` property from the `.wrapper` and `.wrapper input` rulesets, because we will define these values as a function of the screen size using the following rulesets. These rulesets make use of *media queries*.

```
@media only screen and (min-width: 1400px) {
  .wrapper {
    width: 50%;
  }

  .wrapper input {
    width: 600px;
  }
}

@media only screen and (max-width: 1399px) and (min-width: 750px) {
  .wrapper {
    width: 75%;
  }

  .wrapper input {
    width: 400px;
  }
}
```

22. We will now create a git commit with your work until this point. Use `git add .` to stage the newly added files for commit. Then execute `git commit -m "MESSAGE_DESCRIBING_THIS_PHASE"`.

Phase 3. Adjusting According to User Input

23. We will now use JavaScript to provide the ability to dynamically read the new transactions being inputted by the user and use them to update our exchange rate averages. First create a file named `script.js` in your project folder. In order to indicate that the main HTML file needs to load this script, add the following tag in the HTML file before the closing `<body>` tag.

```
<script src="script.js"></script>
```

24. Add the following contents to the JS file, which will serve as the structure for the file. The `document.getElementByID` method can be used to get a reference to any HTML element within the website (DOM). Note that the `addEventListener` method call is used to specify that the click of the add button event should execute the `addItem` function. Two arrays are defined to hold the rates of all the user entered transactions. `addItem` will add the current transaction to the list of existing transactions. It will then execute `updateRates` which will read from the current set of transactions and calculate the average and update the displayed values.

```
var addButton = document.getElementById("add-button");
addButton.addEventListener("click", addItem);


var sellUsdTransactions = [];
var buyUsdTransactions = [];


function addItem() {


 updateRates();
}


function updateRates() {

}
```

25. Write similar declarations as `var addButton = document.getElementById("add-button");` above to get references to all the elements that you will need to dynamically modify or read from.
26. Reread the introduction for a reminder of the logic being adopted for the exchange rate calculation.
27. Complete the `addItem` function to read the values of the USD and LBP transaction amounts, then compute the LBP to USD ratio, and depending on the transaction type, add it to the appropriate array. To access the value of an input element, just access the `value` field of the element reference. Finally, clear the input fields to allow the user to enter another transaction before `updateRates()` is called.
28. Within the `updateRates` method, calculate the average exchange rate for each type of transaction (USD in exchange for LBP and the reverse), given at least one transaction exists, then update the `<span>` elements that display each type of exchange rate. Use `SPAN_ELEMENT_REFERENCE.innerHTML` to set the updated value. Remember to display the appropriate message when no data exists for a transaction type. Also, round the exchange rate value to two decimal places.
29. Give your website a test and make sure you are good to go.
30. We will now create a git commit with your work until this point. Use `git add .` to stage the newly added files for commit. Then execute `git commit -m "MESSAGE_DESCRIBING_THIS_PHASE"`. After doing so, we will create a 'git tag' to mark the current code state as your completed release of this project. Do so by executing `git tag v1`. We will push all the local commits and tags to the remote repository. We will do so with the following two commands: `git push origin main`, and

`git push origin v1`. The tag v1 will mark your release that will be assessed. To make sure that your changes were successfully pushed, you can check in the GitHub interface in your browser.