Documentation for Financial Transactions HTML Page

Jason N.

April 26, 2020

Contents

1 Foreword

Some of the code samples in this document were copied by hand. If there are any discrepencies between code in this document and in the source files, refer to the source files.

This does not apply to the appendix. Code in the appendix was generated directly from the source files.

2 HTML

2.1 Preamble and head

This line declares that the document is an HTML5 document.

```
1 <!DOCTYPE html>
```

`<head>` tags are used to contain meta information about the document.

```
2 <head>
3     <meta charset = "UTF-8"/>
4     <link rel="stylesheet" type="text/css" href="./style.css"/>
5     <script src="./script.js"></script>
6 </head>
```

Within the `head` element:

- The first line defines the character set of the document.
- The second line defines the source of an external CSS document.
- The third line defines the source of an external Javascript document.

2.2 Inputs

The input section of this page is contained within `<article>` tags for the purpose of organisation. This can be used to facilitate styling this part of the page with CSS if desired.

```
10 <article id="inputFields">
```

The `article` element has been assigned a unique id for the purpose of styling. Specifically, this id is used to define padding and overflow. This is described in further detail in section 4.2 of this document.

All input fields and buttons are contained within `<form>` tags. Althought this is not strictly necessary for the purpose of this project, it is useful for organising data and specifying the fields from which data should be submitted.

```
11 <form onsubmit="return false" autocomplete="off">
```

The attribute `onsubmit` is used to define a Javascript function to be executed when pressed. The form expects that `true` is returned when data is successfully submitted. If so, the default behaviour is to clear

the fields and enter the data in the browser URL bar as arguments. To prevent this behaviour, `onsubmit` is set to `return false`.

The attribute `autocomplete` can be used to specify whether user input from a previous session should be used to populate input fields. This attribute also determines whether or not suggestions are displayed when the user enters data. In this case, `autocomplete` has been set to `off` to prevent these actions from occuring. This does not affect the functionality of the program.

The buttons and input fields within the `form` element are contained within `<section>` tags for organisation. This is primarily done to allow elements to be positioned properly by the CSS file.

2.2.1 Common attributes

All `input` elements in this `form` have been assigned a `name` attribute. The `name` attribute is not strictly relevant in this case, but is often used to identify the data when submitting to a database.

All `input` elements have the `required` attribute. Normally this prevents a `form` from being submitted unless all `required` fields contain data. This does not apply to our case as we have disabled the built-in submit function. However, it does still outline missing fields in red.

2.2.2 Labels

Each of the inputs are given a label to specify to a user the type of information which should be entered in the given field. This is done with the `input` element.

```
12 <label for="date">Date:</label><br/>
```

The `for` attribute is used to specify an element which corresponds to this label. This is done by setting the attribute to the id of the other element. Labels allow a user to select an input field by clicking the label rather than the field itself. Labels are also used to facilitate the use of assistive technologies.

2.2.3 Date

The date of a transaction is specified through the use of an `input` element with a `type` attribute of `date`. This can be used to effectively restrict the input to a valid date format and provides an intuitive method for inputting data.

```
11 <section>
12     <label for="date">Date:</label><br/>
13     <input id="date" name="date" type="date" required/>
14 </section>
```

This type of input field is also useful for interpreting dates in Javascript, as it provides methods which return the date in various formats to facilitate displaying and comparing dates.

2.2.4 Text

`input` elements with a `type` attribute of `text` can be used to retrieve a string from a user. This is also the field used for numbers, as these can be easily verified and converted in Javascript.

```
<section>
    <label for="account">Account Number:</label><br/>
    <input id="account" name="account" type="text" placeholder="Account
        ↪ Number" required/>
</section>
```

The advantage of taking numbers from an input field is that it allows for characters such as $ to be included. In the case of this project, users are able to submit Dollar Amounts as purely numberic values, or in a currency format. Currently, the program only accepts dollars as a currency, however, it is possible to allow and store any number of currencies. These characters, of course, have to be filtered out before the number is interpretted and re-inserted before displaying the value.

2.2.5 List

Dropdown lists are created using `<select>` tags containing `option` elements. Each `option` element represents a possible value, the first element is selected by default.

```
<section>
    <label for="type">Transaction Type:</label><br/>
    <select id="type" name="type">
        <option value=""></option>
        <option value="BUY">BUY</option>
        <option value="SELL">SELL</option>
        <option value="DIVIDEND">DIVIDEND</option>
        <option value="INTEREST">INTEREST</option>
        <option value="WITHDRAW">WITHDRAW</option>
        <option value="DEPOSIT">DEPOSIT</option>
    </select>
</section>
```

The `innerHTML` of an `option` element is the text that will be displayed to the user. The `value` attribute of the element is the value that will be read by Javascript. For this project, the `value` and `innerHTML` were made to be identical so that the text in the table would be the same as the text the user had seen in the list.

2.2.6 Buttons

`button` elements are clickable elements which can execute Javascript code specified by an `onclick` attribute. Text within the `innerHTML` of the `button` will be displayed as text within the button, which is useful for communicating the purpose of the button.

```
<section>
    <button id="add" type="submit" onclick="addTransactionButton();">Add
        ↪ Transaction</button>
    <button id="save" type="submit" hidden="true" onclick="saveChanges()
        ↪ ;">Save</button>
    <button id="discard" type="button" hidden="true" onclick="
        ↪ discardChanges();">Discard</button>
</section>
```

In this case, three buttons are present, each set to execute a different Javascript function when clicked.

Two of the three buttons have a `type` attribute of `submit`. This causes each function to trigger the `submit` event along with the Javascript function. However, for this project, this event has been disabled by the `form onsubmit="return false"` attribute. Thus, the only difference is that this causes missing fields to be outlined in red when the button is pressed.

The last button is of `type button`. This element functions exactly the same, except it does not trigger the `submit` event. For this project, this means that missing fields will not be highlighted red, as this is not necessary for the 'Discard Changes' button.

Two of the three buttons also have the `hidden="true"` attribute. This causes the page to render as if these elements did not exist, as these elements are only relevant when editing a row. All three buttons are given unique ids so that `hidden` attributes can be added or removed as needed.

2.3 Table

2.3.1 thead

The header of the table is enclosed in `<thead>` tags. This element includes the first row of the table, denoted by `<tr>` tags, which contains headers for each column.

Every cell in the header is denoted by `<td>` tags. These cells differ from normal cells, such as those in the body of the table, in how they format their contents. Using this element for header cells makes them stand out slightly as well as making it easier to differentiate when styling with CSS.

```
61 <th>
62     <section>
63         Transaction ID
64     </section>
65     <section class="sort">
66         <button type="button" onclick="sortTable(0, true)">^</button>
67         <button type="button" onclick="sortTable(0, false)">V</button>
68     </section>
69 </th>
```

The first 8 header cells are split into two separate `section` elements. This was done to allow for the proper positioning of the header text and the sort buttons. For this reason, the latter `section` element is given the class `sort` to differentiate between the two.

Each of the first 8 header cells contain two buttons for sorting. All sorting buttons call the same function `sort(column, ascending)`, however, they pass different arguments to this function. The first argument is the column number, starting from 0, which allows the Javascript function to determine which column to use when comparing rows. The second argument defines whether data should be sorted in ascending or descending order.

The last header cell contains nothing but text. This column is used to contain the delete and edit buttons created for each row.

```
133 <th>Actions</th>
```

2.3.2 tbody

The table body is enclosed in `<tbody>` tags. This element is meant to be the main container of data in a table.

```
136 <tbody id="tableBody">
```

The table body is important for this project as it is the parent element of all data which will be manipulated. For this reason, it has been given a unique id to reference in Javascript. This was not strictly necessary, as it is also possible to reference this element by its tag name, being the only `tbody` element. Nevertheless, I consider this to be good practice as it is clear which element is being referred to in Javascript and allows for other tables to be added in the future if necessary without breaking the current functionality.

3 Javascript

The following section describes all Javascript functions used in this project. Functions have been grouped according to their purpose, some functions have been omitted for being too similar to other functions.

Each section contains a section in which these are compared to an equivalent function from the Google Sheets project.

3.1 getData()

This function is used to retrieve and format data from the input fields.

```
1  function getData() {
2      var date = document.getElementById("date");
3      var account = document.getElementById("account").value;
4      var type = document.getElementById("type").value;
5      var security = document.getElementById("security").value;
6      var amount = document.getElementById("amount").value;
7      var dAmount = document.getElementById("dAmount").value;
8
9      amount = Number(amount);
10
11     if(dAmount[0] == '$') {
12         dAmount = dAmount.substr(1);
13     }
14     dAmount = Number(dAmount);
15
16     if(validate(date, account, type, security, amount, dAmount)) {
17         var costBasis = calculateCostBasis(amount, dAmount);
18         date = date.value;
19         dAmount = '$' + dAmount.toFixed(2);
20
21         return [ date, account, type, security, amount, dAmount, costBasis
               ↪ ];
22     }
23     else return false;
24 }
```

The function checks whether the data is valid by calling the `validate()` function. If so, data is formatted and sent to the function which called `getData()`. Currently, the caller is either `addTransactionButton()` or `saveChanges()`.

The function first stores the `date` element and the raw values of the other input fields. `date` is treated differently as the element includes useful methods for comparing the date in different formats.

```
2  var date = document.getElementById("date");
3  var account = document.getElementById("account").value;
4  var type = document.getElementById("type").value;
5  var security = document.getElementById("security").value;
6  var amount = document.getElementById("amount").value;
7  var dAmount = document.getElementById("dAmount").value;
```

Next, some of the data is processed. `amount` is converted from a string, as it originated from a text field, to a number. This is done with the built-in `Number()` function, which takes a string as an argument and returns it as a numeric value when possible. If the argument cannot be converted, the function returns `NaN` or 'Not a Number'. We are not concerned with validating that the value can be converted at this stage, as we can simply check if the value is `NaN` during the validation stage, therefore it is safe to convert to a number here.

```
9  amount = Number(amount);
```

A similar conversion is performed on the `dAmount` value. However, before this occurs, we check whether the first character in the string is a dollar sign. If so, we remove the dollar sign by taking a substring of `dAmount` which includes everything including and after the second character. This effectively removes the dollar sign from the string, allowing it to be converted to a numeric value.

```
9  if(dAmount[0] == '$') {
10     dAmount = dAmount.substr(1);
11 }
12 dAmount = Number(dAmount);
```

The function then calls `validate` and passes all the stored variables as arguments to determine whether all the data is valid. If not, the function will return `false` and exit, preventing subsequent steps from occuring.

```
16 if(validate(date, account, type, security, amount, dAmount)) {
17     var costBasis = calculateCostBasis(amount, dAmount);
18     date = date.value;
19     dAmount = '$' + dAmount.toFixed(2);
20
21     return [ date, account, type, security, amount, dAmount, costBasis ];
22 }
23 else return false;
```

If all data is valid, the function calculates and stores the costBasis by calling `calculateCostBasis()` and passing the necessary values. The function also formats the date and dollar amount in the correct formats to be exported to the table. The `toFixed()` method is used to fix the value to 2 decimal places, and a dollar sign is added to the front of the value. Lastly, the function returns a list including all the data to the caller.

Comparison to Google Sheets project

There is no equivalent function in the Google Sheets project. The `getData()` function is required to store input values in memory. Google Apps Script had a built-in function to move or copy cells and did not require most values to be stored like this.

3.2 validate()

The `validate()` function is used to verify that all fields include valid data.

```
26 function validate(date, account, type, security, amount, dAmount) {
27     if(!validateDate(date)) return false;
28     if(!validateAccount(account)) return false;
29     if(!validateType(type)) return false;
30     if(!validateSecurity(security)) return false;
31     if(!validateAmount(amount)) return false;
32     if(!validateDAmount(dAmount)) return false;
33
```

```
34      return true;
35  }
```

The function calls several functions, each of which validates a different input field. If any of the calls returns false, this function returns false. If none of the calls returned false, the function returns true, allowing the caller to proceed.

Comparison to Google Sheets project

The function uses the same method as the Google Sheets project for validating data, by calling different functions which return `true` or `false`. The difference here is that data is taken as arguments and passed to the validating functions, as this data is no longer read from the sheet. The returns of this function have also been standardised such that `false` always indicated an invalid value, this is done mostly for readability.

3.2.1 Check empty

In cases where the only check necessary is that the field is not empty, the function simply compares the value to an empty string. If the value is equal to an empty string, the function prints an alert and returns false, otherwise it returns true.

```
54  function validateAccount(account) {
55      if(account == '') {
56          alert('Error: Missing Account Number');
57          return false;
58      }
59
60      return true;
61  }
```

This is the template used to check account, transaction type, and security, as these are all strings. Although the transaction type field is not a text box, by setting the default empty option to have a value of an empty string, this template still applies.

Comparison to Google Sheets project

The function to check for an empty field is now done by the same function that check that a field is valid. This was done to better organise the validation process and allow functions to be modified more easily.

The Google Sheets project used the `isBlank()` method of a cell, as no data was stored and passed to it. This project does pass values to the function, therefore the check can be simplified by comparing it to an empty string.

3.2.2 Check NaN

The function to check whether or not a value is a number is identical to the functions that check for only empty values, except for one key difference. In addition to checking if the value is empty, the function checks if the value is `NaN`. This is done using the `isNaN()` function, which takes a value as an argument and returns true of the value is `NaN`.

```
87  if(isNaN(amount)) {
88      alert('Error: Invalid Amount');
89      return false;
```

```
}
```

Comparison to Google Sheets project

This check was not performed in the Google Sheets project, however, if one were to validate a number in the Google Sheets project, one would check that the value was a numeric type, similar to how the date was validated in Google Sheets. In this case, the number can be validated much simpler, by checking whether or not the conversion was successful.

3.2.3 Check date

Currently, a valid date is a date that is not in the future. In order to check this, the function gets the current date by creating a new `Date` object and storing it as a variable. The function also stores the value of the `date` element in a number format which can be easily compared. This is done by storing the `valueAsNumber` property of the element.

```
function validateDate(date) {
    realDate = new Date();
    inputDate = date.valueAsNumber;

    if(!date.checkValidity()) {
        alert('Error: Invalid date');
        return false;
    }

    if(realDate.valueOf() < inputDate) {
        alert('Error: Date is in the future');
        return false;
    }

    return true;
}
```

The first check performed is whether or not the user had inputted a date that exists. If the date field was left empty or incomplete, or the date is non-existant (e.g. November 31) the `date.checkValidity()` function will return `false`. Therefore, we can reuse the statement that checks whether or not a field contains a valid number.

Next, the function must check that the date is not in the future. This is done by simply comparing the dates in number format, with a greater value indicating a later date.

Comparison to Google Sheets project

In Google Sheets, a date was validated by checking that it was of the `[object Date]` type. In this project, we can be confident that the object is of the correct type as it was created by a specific type of input, therefore, it is not necessary to validate the type.

To check that the date was not in the future, the Google Sheets project directly compared two date objects. This was not done for the current project, as we have two different types of data, an element and a `Date` object. In order to compare these, both are converted to the same numeric format.

3.3 generateId()

A unique id generated with the length specified by `idLength` and with characters specified by `characters`.

```
109 function generateId() {
110     var id = '';
111     var idLength = 6;
112
113     var characters = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789';
114     var charactersLength = characters.length;
115
116     var unique = false;
117
118     while(!unique) {
119         for(var i = 0; i < idLength; i++) {
120             id += characters.charAt(Math.floor(Math.random() *
                    ↪ charactersLength));
121         }
122
123         unique = true;
124         for(var i = 0; i < document.getElementsByClassName('idCell').
                ↪ length; i++) {
125             if(document.getElementsByClassName('idCell')[i].innerText ==
                    ↪ id) {
126                 unique = false;
127                 break;
128             }
129         }
130     }
131     return id;
132 }
```

The `unique` variable is used to store whether the generated id is unique. A `while` loop continuously generates and checks ids until a unique one is found.

```
119 for(var i = 0; i < idLength; i++) {
120     id += characters.charAt(Math.floor(Math.random() * charactersLength));
121 }
```

For every character in the id, a random character is chosen from the character set. A number between 0 and the number of possible characters is generated by `Math.random()* charactersLength` as `Math.random()` generates a number $0 \leq n < 1$. This number is converted to an integer by rounding down using the `Math.` ↪ `floor()` function. The result of this integer is used as the index from which to take a character using the `charAt()` method.

```
123 unique = true;
124 for(var i = 0; i < document.getElementsByClassName('idCell').length; i++)
        ↪ {
125     if(document.getElementsByClassName('idCell')[i].innerText == id) {
126         unique = false;
127         break;
128     }
129 }
```

Every generated id is initially assumed to be unique. The function loops through every element with a class of `'idCell'`, comparing the `innerText` of this cell to the generated id. If the values match, the id is not unique and another id must be generated. If no matching id has been found, the id is considered unique and the loop exits, passing the id to the caller.

Comparison to Google Sheets project

The process for generating an id is exactly the same as in the Google Sheets project. The id is compared to the `innerText` of cells exactly as the id was compared to other cells in the first column of the Google Sheets. The two processes are essentially identical, except the process is much faster outside of Google Sheets.

3.4 calculateCostBasis()

The function receives two numberic values as arguments, calculates the quotient of the two, and formats the result as a dollar value.

```
134 function calculateCostBasis(amount, dAmount) {
135     costBasis = '$' + (dAmount / amount).toFixed(2);
136     return costBasis;
137 }
```

The `toFixed()` method is used to fix the value to 2 decimal places, and a dollar sign is added to the front of the value.

Comparison to Google Sheets project

This function is essentially the same as that of the Google Sheets project, except the data is passed to this function, rather than being read from the sheet. Another minor difference is in how data is formatted, as Google Apps Script formatted data by setting a number format, while that format does not exist in pure Javascript and must be set manually.

3.5 addTransaction()

The process of adding a transaction is initiated when the `addTransactionButton()` function is called. This function calls the `getData()` function and stores the result. If the result is not `false`, the function generates a unique id for the transaction and adds this to the front of the `data` array using the `unshift()` method. The function then passes the `data` array to the `addTransaction()` function, which creates and populates the row.

```
156 function addTransactionButton() {
157     var data = getData();
158     if(data) {
159         var id = generateId();
160         data.unshift(id);
161 
162         addTransaction(data);
163     }
164 }
```

The `addTransaction` function gets all necessary data as an array argument. It stores the body of the table in a variable by referencing the element using `document.getElementById('tableBody')`. A new row is

created using the `insertRow()` method of the table body and stored as a variable so that contents can be added. The row is assigned a class of `bodyRow` for reference by other functions or styling.

```
139 function addTransaction(data) {
140     var tableBody = document.getElementById('tableBody');
141     var newRow = tableBody.insertRow(0);
142     newRow.classList += "bodyRow";
143 
144     var actionsContent = "<button type='button' onclick='editRow(this)'>
        ↪ Edit</button> <button type='button' onclick='deleteRow(this)'>
        ↪ Delete</button>";
145     data.push(actionsContent);
146 
147     for(var i = 0; i < data.length; i++) {
148         var newCell = newRow.insertCell(i);
149         newCell.innerHTML = data[i];
150         if(i == 0) {
151             newCell.classList += "idCell";
152         }
153     }
154 }
```

The function creates string containing HTML code for a delete and an edit button and pushes this to the end of the `data` array. The `onclick` functions pass `this` as an argument, which specifies the element that called the function. This is done so that the row to edit or delete can be selected.

```
144 var actionsContent = "<button type='button' onclick='editRow(this)'>Edit</
    ↪ button> <button type='button' onclick='deleteRow(this)'>Delete</
    ↪ button>";
145 data.push(actionsContent);
```

For each item in the array, the function calls the `insertCell()` method of the row to create a new cell. The contents of this cell are defined by the `innerHTML` property, which is set to the corresponding element of the array. For the first element, the cell is also given a special class to identify it as the cell containing the transaction id.

```
148 var newCell = newRow.insertCell(i);
149 newCell.innerHTML = data[i];
150 if(i == 0) {
151     newCell.classList += "idCell";
152 }
```

Comparison to Google Sheets project

The Google Sheets project copied data from the input range to the output range, whereas this function stores the values in memory then writes the new cells. As the table is within a self-contained element, the more complicated process of moving all contents to a separate area is not required. Therefore, the process of adding a row is significantly simpler in HTML and JS.

3.6 deleteRow()

The function takes an element as an argument, this element is supposed to be the button which called the function. The function gets the parent of the parent of the delete button and removes this element. The

first parent of the delete button is the cell, the second parent is the row. By selecting the second parent, we are selecting the row containing the button.

The function uses the `removeChild()` method of the table to remove the specified row element.

```
166 function deleteRow(button) {
167     var row = button.parentElement.parentElement;
168     document.getElementById("tableBody").removeChild(row);
169 
170     if(document.getElementsByClassName('editing').length == 0) {
171         document.getElementById('add').removeAttribute('hidden');
172         document.getElementById('save').setAttribute('hidden', true);
173         document.getElementById('discard').setAttribute('hidden', true);
174     }
175 }
```

It is possible that the delete button is used on the row being edited. If this is the case, the save and discard buttons would remain visible, although there would be no row being edited. To resolve this, the function checks if there are any rows with the class `'editing'`. If so, the function hides the save and discard buttons, and unhides the add button. This is performed whenever the function finds no row being edited, however, it has no effect when the row being edited was not deleted.

Comparison to Google Sheets project

Deleting is far simpler as it is not necessary to transfer the rows to a separate sheet. HTML and JS also allows for a new button to be easily created for every row.

The Google Sheets project did not hide and show buttons depending on whether or not a function was being editted, however, this is done in HTML as the process is much simpler. If one were to do this in Google Sheets, the buttons would have to be indexed before hand so they could be referenced in code.

3.7 editRow()

This element is responsible for highlighting the row being editted and moving the values in the row to the input fields.

```
177 function editRow(button) {
178     if(document.getElementsByClassName('editing').length > 0)
179         document.getElementsByClassName('editing')[0].classList = "bodyRow
              ↪ ";
180 
181     var row = button.parentElement.parentElement;
182     var rowContent = row.getElementsByTagName('td');
183     row.classList = "bodyRow editing";
184 
185     document.getElementById('date').value = rowContent[1].innerText;
186     document.getElementById('account').value = rowContent[2].innerText;
187     document.getElementById('type').value = rowContent[3].innerText;
188     document.getElementById('security').value = rowContent[4].innerText;
189     document.getElementById('amount').value = rowContent[5].innerText;
190     document.getElementById('dAmount').value = rowContent[6].innerText;
191 
192     document.getElementById('add').setAttribute('hidden', true);
193     document.getElementById('save').removeAttribute('hidden');
```

```
194     document.getElementById('discard').removeAttribute('hidden');
195 }
```

First, the function checks whether or not there is a cell already highlighted for editing. This is done by geting a list of all functions with the `'editing'` class. If so, it resets the class to `'bodyRow'`. This is done to ensure that at most, one row is being edited at a time.

```
178 if(document.getElementsByClassName('editing').length > 0)
179     document.getElementsByClassName('editing')[0].classList = "bodyRow";
```

The function uses the same method as the `deleteRow()` function to get the parent row of the button. The function finds the second parent of the calling button and stores this as a variable for reference. The cells in this row are stored in an array by getting all children of the row with the `td` tag. The row is changed to have the classes `bodyRow` and `editing`.

Each input field is then identified using their id and given a value of the `innerText` of the row.

```
181 var row = button.parentElement.parentElement;
182 var rowContent = row.getElementsByTagName('td');
183 row.classList = "bodyRow editing";
184
185 document.getElementById('date').value = rowContent[1].innerText;
186 document.getElementById('account').value = rowContent[2].innerText;
187 document.getElementById('type').value = rowContent[3].innerText;
188 document.getElementById('security').value = rowContent[4].innerText;
189 document.getElementById('amount').value = rowContent[5].innerText;
190 document.getElementById('dAmount').value = rowContent[6].innerText;
```

Lastly, the add button is hidden by assigning it a `hidden` attribute. The save and discard buttons are revealed by removing the `hidden` attributes.

```
192 document.getElementById('add').setAttribute('hidden', true);
193 document.getElementById('save').removeAttribute('hidden');
194 document.getElementById('discard').removeAttribute('hidden');
```

Comparison to Google Sheets project

The process is essentially the same, except that the class is used to mark a row as being editted and for formatting, whereas the Google Sheets project marked this by placing a 1 in the I column.

The process of hiding and revealing buttons is unique to this project, as this is made much simpler by the fact that elements can be referred to using a unique id.

3.8 saveChanges()

The function calls `getData()` to retrieve data from the input fields and stores this as a variable. If `getData ↪ ()` does not return false, the function gets the row being editted by getting the first item with a class of `'editing'`.

The function stores the cells of this row as a an array by getting all child elements with `'td'` tags. Every element in the `data` array is written to a cell in the row, starting from the second cell so that the id is not replaced. The function then sets the class to `bodyRow`, removing the `editing` class.

```
function saveChanges() {
    data = getData();
    if(data) {
        rowToEdit = document.getElementsByClassName('editing')[0];
        cellsToEdit = rowToEdit.getElementsByTagName('td');

        for(var i = 0; i < data.length; i++) {
            cellsToEdit[i + 1].innerHTML = data[i];
        }
        rowToEdit.classList = "bodyRow";
    }

    document.getElementById('add').removeAttribute('hidden');
    document.getElementById('save').setAttribute('hidden', true);
    document.getElementById('discard').setAttribute('hidden', true);
}
```

Lastly, the function reveals the 'add transaction' button and hides the 'save' and 'discard' buttons.

Comparison to Google Sheets project

The function differs in how data is transfered. Instead of using a built-in `copyTo()` method, each cell is individually given data.

This project hides and reveals buttons based on whether or not a row is being edited, as HTML and Javascript makes this process much simpler. This step was not taken in the Google Sheets project.

3.9 discardChanges()

This function is responsible for unmarking the editing row and changing the state of buttons.

Firstly, the element with a class of `'editing'` is reset to only have the `'bodyRow'` class.

Then the 'add transaction' button is revealed, and the 'save' and 'discard' buttons are hidden.

```
function discardChanges() {
    document.getElementsByClassName('editing')[0].classList = "bodyRow";

    document.getElementById('add').removeAttribute('hidden');
    document.getElementById('save').setAttribute('hidden', true);
    document.getElementById('discard').setAttribute('hidden', true);
}
```

Comparison to Google Sheets project

The Google Sheets project cleared formatting and removed the edit indicator by clearing the entire I column and reseting formatting for all rows. With Javascript, the process is much simpler as it is easy to select the specific row to reset by referencing its class.

The Google Sheets project also cleared all input fields when changes were discarded. This can be done by setting all input fields to an empty string, but I felt that this was unnecessary and that entering similar data would be easier without this feature.

3.10 sortTable()

This function is responsible for sorting each column. The function takes two arguments: the column number, and a true or false value to determine whether the data should be sorted in ascending or descending order.

This function utilises a simple sorting algorithm known as 'bubble sort' or 'sinking sort'. The function steps through every element in the function, comparing it to an adjacent element, and swapping when necessary. This is done until a loop occurs in which no swaps were made. Although the algorithm is considered inefficient compared to other available algorithms, it is far easier to implement and is more than sufficient for this project.

```
222 function sortTable(column, ascending) {
223     var rows = document.getElementsByClassName('bodyRow');
224 
225     var sorting = true;
226     while(sorting) {
227         sorting = false;
228         for(var i = 0; i < (rows.length - 1); i++) {
229             rowA = rows[i].getElementsByTagName('td')[column];
230             rowB = rows[i + 1].getElementsByTagName('td')[column];
231 
232             var swap = false;
233 
234             if(ascending && rowA.innerHTML.toLowerCase() > rowB.innerHTML.
                    ↪ toLowerCase()) swap = true;
235             else if(!ascending && rowA.innerHTML.toLowerCase() < rowB.
                    ↪ innerHTML.toLowerCase()) swap = true;
236 
237             if(swap) {
238                 sorting = true;
239                 document.getElementById('tableBody').insertBefore(rows[i +
                        ↪  1], rows[i]);
240             }
241         }
242     }
243 }
```

First, all the rows are stored in a list of elements for reference.

This is done by taking a list of all elements with the `'bodyRow'` class. It would also be possible to do this without the class, as all body rows are contained within `<tbody>` tags. One could get a list of all rows that are children of the `tbody` element with `document.getElementsByTagName('tbody')[0].getElementsByTagName ↪ ('tr');`.

However, I believe that the approach above makes the function less readable, and therefore, I have gone with the approach using a separate class.

```
223 var rows = document.getElementsByClassName('bodyRow');
```

Next, the main loop of the function begins. A variable `sorting` is given a value of `true` so that the function is allowed to start. With every loop, `sorting` is set to `false`. The value changes during the loop if any rows have been swapped, otherwise, the rows are considered to be in the correct order and the function terminates as the `while` loop ends.

Every loop, the function steps through every row excluding the last row. It stores the cell of the current row and the row below in the specified column.

```
var sorting = true;
while(sorting) {
    sorting = false;
    for(var i = 0; i < (rows.length - 1); i++) {
        rowA = rows[i].getElementsByTagName('td')[column];
        rowB = rows[i + 1].getElementsByTagName('td')[column];
```

A variable `swap` is initially set to false. All text is converted to lowercase for comparison using the `toLowerCase()` method. The values in the cells defined by `rowA` and `rowB` are compared using a `>` or `<` operator depending on whether `ascending` is `true` or `false`. If the values are determined to be in the wrong order, `swap` is set to `true`.

If the rows are required to be swapped, the `sorting` variable is set to true so that the `while` loop does not terminate. The `insertBefore()` method can be used to change the order of the children of an element. This method needs to be called by the parent element, hence why the `tableBody` element is chosen.

The method takes two elements of arguments, and places the first element above the second element. `rows[i]` is always above `rows[i + 1]`, therefore to swap them, we list `rows[i + 1]` as the first argument.

```
swap = false;

if(ascending && rowA.innerHTML.toLowerCase() > rowB.innerHTML.toLowerCase
    ↪ ()) swap = true;
else if(!ascending && rowA.innerHTML.toLowerCase() < rowB.innerHTML.
    ↪ toLowerCase()) swap = true;

if(swap) {
    sorting = true;
    document.getElementById('tableBody').insertBefore(rows[i + 1], rows[i
        ↪ ]);
}
```

Comparison to Google Sheets project

In Google Sheets, when buttons are assigned to a function, the arguments are not defined. For this reason, it was necessary to have a different button for ascending and descending orders. In this project, we can define both the column and whether it should sort in ascending or descending by passing different arguments. Therefore, only one function is required to sort any column.

This function is somewhat more complex than the Google Sheets equivalent. For the Google Sheets project, there existed a built-in function to sort a column, whereas this had to be written from scratch for HTML and Javascript.

4 CSS

4.1 Vertical Scrolling Table

The table id refers to the `article` that contains the table, rather than the table itself. This article was given a max height of 80 visual heights, or approximately 80% of the screen height. `overflow: auto;` specifies that, if necessary, a scroll bar should be present, this is true for both horizontal and vertical scrolling.

```
29 #table {
30     max-height: 80vh;
31     overflow: auto;
32 }
```

The last three properties below are important for keeping the header in place while scrolling.

```
39 th {
40     min-width: 200px;
41     width: 10%;
42     position: sticky;
43     background: white;
44     top: 0;
45 }
```

- `position: sticky;` is used to keep the object in place when scrolling.
- `background: white;` is used to give the element a non-transparent background, so that data cannot be seen through the header.
- `top: 0;` is used to specify that the element should remain at the top of its parent element with no offset.

4.2 Horizontal Scrolling on Overflow

The inputFields id is used to identify the `article` element that contains the `form`. The important property here is the `overflow-x: auto;` line, which specifies that, if the child element is wider than this element, a horizontal scroll bar should be present.

The `form` element, which is a child of the `article` is given a minimum width to ensure that the scroll bar is created rather than reducing the width of the element.

```
 5 #inputFields {
 6     padding: 10px 0;
 7     overflow-x: auto;
 8 }
 9
10 form {
11     min-width: 1900px;
12 }
```

4.3 Miscellaneous

4.3.1 Sort buttons

The header cells contain two `sections`, one of which has a special class. Both sections are given a `margin` and `padding` of 0 to minimise wasted space. Both sections are also set to `display: inline-block` to specify that they should be arranged horizontally.

Both sections are given a `width` of 80% of the parent element. However, this is overruled for the element with a class of `sort`, which is assigned a `width` of 10% to ensure that both elements fit horizontally in the parent.

To further reduce wasted space, the `border` and `padding` of the buttons are set to 0. The button's `display` ↪ property is set to `block` as otherwise, it would inherit the `inline-block` property from its parent and attempt to arrange horizontally. Lastly, the buttons are set to take up the entire width of the parent element.

```
47 th > section {
48     width: 80%;
49     display: inline-block;
50     padding: 0;
51     margin: 0;
52 }
53
54 .sort {
55     width: 10%;
56 }
57
58 .sort > button {
59     padding: 0;
60     border: 0;
61     display: block;
62     width: 100%;
63 }
```

4.3.2 Editing highlight

The current row selected for highlighting is specified using a class. As such, it is possible to give this row unique styling. For example, currently the row is given a yellow background.

```
65 .editing {
66     background-color: yellow;
67 }
```

4.3.3 Table borders

Table borders do not render properly with a scrolling body and fixed header. To resolve this, borders are rendered using the `box-shadow` property.

Two box shadows are defined, one extends outwards from the right and bottom by one pixel in each direction. The other is given the `inset` property, so that it extends inwards from the left and top. This creates a full

border with a width of 1 pixel in each direction. As these borders are offset from the element, adjacent elements will overlap borders, preventing borders from combining into extra thick borders.

```
#table,
table,
td,
th {
    box-shadow: 1px 1px black, inset 1px 1px black;
}
```

A HTML Source Code

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset = "UTF-8"/>
        <link rel="stylesheet" type="text/css" href="./style.css"/>

        <script src="./script.js"></script>
        <script src="https://apis.google.com/js/api.js"></script>
        <script src="./googleApiScript.js"></script>
        <script src="./mysqlScript.js"></script>
        <script src="./localStorageScript.js"></script>
        <script src="./imageFirestore.js"></script>
    </head>
    <body>
        <article id="inputFields">
            <section class="sectionHead">
                <h1>Input:</h1>
                <button class="toggleSection" type="button" onclick="
                    ↪ toggleSection(this)">Hide</button>
            </section>
            <form onsubmit="return false" autocomplete="off">
                <section>
                    <label for="date">Date:</label><br/>
                    <input id="date" name="date" type="date" placeholder="
                        ↪ yyyy-mm-dd"/>
                </section>

                <section>
                    <label for="account">Account Number:</label><br/>
                    <input id="account" name="account" list="accountsList"
                        ↪  type="text" placeholder="Account Number"/>
                    <datalist id="accountsList">
                    </datalist>
                </section>

                <section>
                    <label for="type">Transaction Type:</label><br/>
                    <select id="type" name="type">
                        <option value=""></option>
                        <option value="BUY">BUY</option>
                        <option value="SELL">SELL</option>
                        <option value="!DIVIDEND">DIVIDEND</option>
                        <option value="!INTEREST">INTEREST</option>
                        <option value="!WITHDRAW">WITHDRAW</option>
                        <option value="!DEPOSIT">DEPOSIT</option>
                    </select>
                </section>

                <section>
                    <label for="security">Security:</label><br/>
                    <input id="security" name="security" list="
                        ↪ securitiesList" type="text" placeholder="
```

```
                    ↪ Security"/>
49                  <datalist id="securitiesList">
50                  </datalist>
51              </section>
52
53              <section>
54                  <label for="amount">Amount:</label><br/>
55                  <input id="amount" name="amount" type="text"
                        ↪ placeholder="Unit Amount"/>
56              </section>
57
58              <section>
59                  <label for="dAmount">$ Amount:</label><br/>
60                  <input id="dAmount" name="dAmount" type="text"
                        ↪ placeholder="$ Amount"/>
61              </section>
62
63              <section>
64                  <label id="fileUploadLabel" for="fileUpload">Upload
                        ↪ file</label>
65                  <input id="fileUpload" name="fileUpload" type="file"
                        ↪ onchange="fileUploadChanged();" multiple/>
66                  <button id="removeFile" type="button" onclick="
                        ↪ removeFileUpload();">X</button>
67              </section>
68
69              <section>
70                  <button id="add" type="submit" onclick="
                        ↪ addTransactionButton();">Add Transaction</button
                        ↪ >
71                  <button id="save" type="button" hidden="true" onclick
                        ↪ ="saveChanges();">Save</button>
72                  <button id="discard" type="button" hidden="true"
                        ↪ onclick="discardChanges();">Discard</button>
73              </section>
74          </form>
75      </article>
76
77      <article id="filter">
78          <section class="sectionHead">
79              <h1>Filters:</h1>
80              <button class="toggleSection" type="button" onclick="
                    ↪ toggleSection(this)">Hide</button>
81          </section>
82          <form onsubmit="return false" autocomplete="off">
83              <section>
84                  <label for="filterId">Transaction ID:</label><br/>
85                  <input id="filterId" class="filterField" name="
                        ↪ filterId" type="text" placeholder="Transaction
                        ↪ ID"/>
86              </section>
87
88              <section>
89                  <label for="startDate">From:</label><br/>
```

```
90                      <input id="startDate" class="filterField" name="
                           ↪ startDate" type="date" placeholder="yyyy-mm-dd
                           ↪ "/>
91
92                      <br/>
93
94                      <label for="endDate">To:</label><br/>
95                      <input id="endDate" class="filterField" name="endDate"
                           ↪  type="date" placeholder="yyyy-mm-dd"/>
96                  </section>
97
98                  <section>
99                      <label for="filterAccount">Account Number:</label><br
                           ↪ />
100                     <input id="filterAccount" class="filterField" name="
                           ↪ filterAccount" type="text" placeholder="Account
                           ↪ Number"/>
101                 </section>
102
103                 <section>
104                     <label for="type">Transaction Type:</label><br/>
105                     <select id="filterType" class="filterField" name="
                           ↪ filterType">
106                         <option value=""></option>
107                         <option value="BUY">BUY</option>
108                         <option value="SELL">SELL</option>
109                         <option value="DIVIDEND">DIVIDEND</option>
110                         <option value="INTEREST">INTEREST</option>
111                         <option value="WITHDRAW">WITHDRAW</option>
112                         <option value="DEPOSIT">DEPOSIT</option>
113                     </select>
114                 </section>
115
116                 <section>
117                     <label for="filterSecurity">Security:</label>
118                     <span title="Enter search terms here. Separate search
                           ↪ terms with && or || for AND and OR statements,
                           ↪ respectively. Exclusive filters are marked by a
                           ↪ leading !. Use || to filter by multiple
                           ↪ securities (e.g. SPY || TLT) and && to exclude
                           ↪ multiple securities (e.g. !SPY && !TLT).">?</
                           ↪ span><br/>
119                     <input id="filterSecurity" class="filterField" name="
                           ↪ filterSecurity" type="text" placeholder="
                           ↪ Security"/>
120                 </section>
121
122                 <section>
123                     <label for="lowAmount">Min Amount:</label><br/>
124                     <input id="lowAmount" class="filterField" name="
                           ↪ lowAmount" type="text" placeholder="Min Amount
                           ↪ "/>
125
126                     <br/>
```

```
127
128                     <label for="highAmount">Max Amount:</label><br/>
129                     <input id="highAmount" class="filterField" name="
                            ↪ highAmount" type="text" placeholder="Max Amount
                            ↪ "/>
130                 </section>
131
132                 <section>
133                     <label for="lowDAmount">Min $ Amount:</label><br/>
134                     <input id="lowDAmount" class="filterField" name="
                            ↪ lowDAmount" type="text" placeholder="Min $
                            ↪ Amount"/>
135
136                     <br/>
137
138                     <label for="highDAmount">Max $ Amount:</label><br/>
139                     <input id="highDAmount" class="filterField" name="
                            ↪ highDAmount" type="text" placeholder="Max $
                            ↪ Amount"/>
140                 </section>
141
142                 <section>
143                     <label for="lowCostBasis">Min Cost Basis:</label><br/>
144                     <input id="lowCostBasis" class="filterField" name="
                            ↪ lowCostBasis" type="text" placeholder="Min Cost
                            ↪ Basis"/>
145
146                     <br/>
147
148                     <label for="highCostBasis">Max Cost Basis:</label><br
                            ↪ />
149                     <input id="highCostBasis" class="filterField" name="
                            ↪ highCostBasis" type="text" placeholder="Max Cost
                            ↪  Basis"/>
150
151                     <br/>
152
153                     <label for="filterNa">Filter N/A:</label>
154                     <input id="filterNA" name="filterNA" type="checkbox"/>
155                 </section>
156
157                 <section>
158                     <button type="submit" onclick="applyFilter()">Apply</
                            ↪ button>
159                     <button type="button" onclick="clearFilter()">Clear</
                            ↪ button>
160                 </section>
161             </form>
162         </article>
163
164         <article id="options">
165             <section class="sectionHead">
166                 <h1>Options:</h1>
```

```
167                 <button class="toggleSection" type="button" onclick="
                        ↪ toggleSection(this)">Hide</button>
168             </section>
169             <form onsubmit="return false" autocomplete="off">
170                 <section>
171                     <button id="toggleId" type="button" onclick="toggleID
                            ↪ ()">Hide Transaction ID</button>
172                 </section>
173
174                 <section>
175                     <label for="fileInput">Import Transaction Type CSV
                            ↪ file</label>
176                     <input id="fileInput" name="fileInput" type="file"
                            ↪ onchange="readFile(this)"/>
177                 </section>
178
179                 <section id="transactionTypesConfig">
180                         <label for="typesArray">Transaction types (comma-
                            ↪ separated):</label>
181                         <input id="typesArray" name="typesArray" type="
                            ↪ text" placeholder="Transaction types"/>
182                         <button id="applyTypesButton" type="button"
                            ↪ onclick="applyTypes()">Apply</button>
183                         <button id="editTypesButton" type="button" onclick
                            ↪ ="editTypes()">Edit</button>
184                         <button type="button" onclick="saveFile()">Save</
                            ↪ button>
185                 </section>
186
187                 <section id="googleSheetButtons">
188                     <button type="button" onclick="writeGoogleSheetDB()">
                            ↪ Write to Sheets</button>
189                     <button type="button" onclick="readGoogleSheetDB()">
                            ↪ Read from Sheets</button>
190                 </section>
191
192                 <section>
193                     <section>
194                         <select id="sheet" onchange="getNewSheetData()">
195                             <option value="1ROHpaAIUw-JHX8SrzvkEPCG1qgI-
                                    ↪ siJ9oucY6g5e4Co">default</option>
196                         </select>
197                     </section>
198
199                     <section>
200                         <select id="tab" onchange="getNewTabData()">
201                         </select>
202                     </section>
203                 </section>
204
205                 <section id="logoutSection">
206                     <button type="button" onclick="loadSheetData()">Reload
                            ↪  Sheets</button>
```

```
207                     <button type="button" onclick="auth2.disconnect()">Log
                            ↪  out</button>
208                 </section>
209
210                 <section>
211                     <button type="button" onclick="writeToFirebase()">
                            ↪ Write to Firebase</button>
212                     <button type="button" onclick="readFromFirebase()">
                            ↪ Read from Firebase</button>
213                 </section>
214
215                 <section>
216                     <button type="button" onclick="writeToFirestore()">
                            ↪ Write to Firestore</button>
217                     <button type="button" onclick="readFromFirestore()">
                            ↪ Read from Firestore</button>
218                 </section>
219
220                 <section>
221                     <button type="button" onclick="writeToMySQL()">Write
                            ↪ to MySQL</button>
222                     <button type="button" onclick="readFromMySQL()">Read
                            ↪ from MySQL</button>
223                 </section>
224             </form>
225         </article>
226
227         <article>
228         <h1>Table:</h1>
229         <article id="table">
230             <table>
231                 <thead>
232                     <tr>
233                         <th class="frozenColumn1">
234                             <section>
235                                 Transaction ID
236                             </section>
237                             <section class="sort">
238                                 <button type="button" onclick="sortTable
                                        ↪ (0, true)">^</button>
239                                 <button type="button" onclick="sortTable
                                        ↪ (0, false)">v</button>
240                             </section>
241                         </th>
242                         <th class="frozenColumn2">
243                             <section>
244                                 Date
245                             </section>
246                             <section class="sort">
247                                 <button type="button" onclick="sortTable
                                        ↪ (1, true)">^</button>
248                                 <button type="button" onclick="sortTable
                                        ↪ (1, false)">v</button>
249                             </section>
```

```
                </th>
                <th>
                    <section>
                        Account Number
                    </section>
                    <section class="sort">
                        <button type="button" onclick="sortTable
                            ↪ (2, true)">^</button>
                        <button type="button" onclick="sortTable
                            ↪ (2, false)">v</button>
                    </section>
                </th>
                <th>
                    <section>
                        Transaction Type
                    </section>
                    <section class="sort">
                        <button type="button" onclick="sortTable
                            ↪ (3, true)">^</button>
                        <button type="button" onclick="sortTable
                            ↪ (3, false)">v</button>
                    </section>
                </th>
                <th>
                    <section>
                        Security
                    </section>
                    <section class="sort">
                        <button type="button" onclick="sortTable
                            ↪ (4, true)">^</button>
                        <button type="button" onclick="sortTable
                            ↪ (4, false)">v</button>
                    </section>
                </th>
                <th>
                    <section>
                        Amount
                    </section>
                    <section class="sort">
                        <button type="button" onclick="sortTable
                            ↪ (5, true)">^</button>
                        <button type="button" onclick="sortTable
                            ↪ (5, false)">v</button>
                    </section>
                </th>
                <th>
                    <section>
                        $ Amount
                    </section>
                    <section class="sort">
                        <button type="button" onclick="sortTable
                            ↪ (6, true)">^</button>
                        <button type="button" onclick="sortTable
                            ↪ (6, false)">v</button>
```

```
                        </section>
                    </th>
                    <th>
                        <section>
                            Cost Basis
                        </section>
                        <section class="sort">
                            <button type="button" onclick="sortTable
                                ↪ (7, true)">^</button>
                            <button type="button" onclick="sortTable
                                ↪ (7, false)">v</button>
                        </section>
                    </th>
                    <th>
                        <section>
                            File
                        </section>
                        <section class="sort">
                            <button type="button" onclick="sortTable
                                ↪ (8, true)">^</button>
                            <button type="button" onclick="sortTable
                                ↪ (8, false)">v</button>
                        </section>
                    </th>
                    <th>Actions</th>
                </tr>
            </thead>
            <tbody id="tableBody">
            </tbody>
        </table>
    </article>
    </article>

    <!-- The core Firebase JS SDK is always required and must be
        ↪ listed first -->
    <script src="https://www.gstatic.com/firebasejs/7.14.2/firebase-
        ↪ app.js"></script>

    <script src="https://www.gstatic.com/firebasejs/7.14.2/firebase-
        ↪ analytics.js"></script>
    <script src="https://www.gstatic.com/firebasejs/7.14.2/firebase-
        ↪ database.js"></script>
    <script src="https://www.gstatic.com/firebasejs/7.14.3/firebase-
        ↪ firestore.js"></script>

    <script>
    // Your web app's Firebase configuration
    var firebaseConfig = {
        apiKey: "AIzaSyAmZLFZHDAB9evhvNunxOe5GxXRd_OizmU",
        authDomain: "financial-transactions-6f065.firebaseapp.com",
        databaseURL: "https://financial-transactions-6f065.firebaseio.
            ↪ com",
        projectId: "financial-transactions-6f065",
        storageBucket: "financial-transactions-6f065.appspot.com",
```

```
            messagingSenderId: "82206982479",
            appId: "1:82206982479:web:8937bbd1bd4fb6022b053a",
            measurementId: "G-0564DT8RNQ"
        };
        // Initialize Firebase
        firebase.initializeApp(firebaseConfig);
        firebase.analytics();

        var database = firebase.database();
        var firestore = firebase.firestore();
        </script>

        <script src="./firebaseScript.js"></script>
    </body>
</html>
```

B Javascript Source Code

B.1 Main Script

```
function formattedStringToNumber(numberAsString) {
    var number;

    if(numberAsString[0] == '$') {
        numberAsString = numberAsString.substr(1);
    }

    number = Number(numberAsString.replace(/,/g, ''));

    return number;
}

function numberToFormattedString(number) {
    var numberAsString;

    numberAsString = String(number).replace(/\B(?=(\d{3})+(?!\d))/g, ",");

    return numberAsString;
}

function getData() {
    var date = document.getElementById("date");
    var account = document.getElementById("account").value;
    var type = document.getElementById("type").value;
    var security = document.getElementById("security").value;
    var amount = document.getElementById("amount").value;
    var dAmount = document.getElementById("dAmount").value;

    security = security.toUpperCase();

    amount = formattedStringToNumber(amount);

    dAmount = formattedStringToNumber(dAmount);

    if(validate(date, account, type, security, amount, dAmount)) {
        var costBasis = '$' + numberToFormattedString(calculateCostBasis(
            ↪ amount, dAmount));
        date = date.value;

        amount = numberToFormattedString(amount);
        dAmount = '$' + numberToFormattedString(dAmount.toFixed(2));

        return [ date, account, type, security, amount, dAmount, costBasis
            ↪ ];
    }
    else return false;
}

function validate(date, account, type, security, amount, dAmount) {
```

```
48      if(!validateDate(date)) return false;
49      if(!validateAccount(account)) return false;
50      if(!validateType(type)) return false;
51      if(!validateSecurity(security)) return false;
52      if(!validateAmount(amount)) return false;
53      if(!validateDAmount(dAmount)) return false;
54
55      return true;
56  }
57
58  function validateDate(date) {
59      realDate = new Date();
60      inputDate = date.valueAsNumber;
61
62      if(date.value == '') {
63          alert('Error: Missing date');
64          return false;
65      }
66
67      if(!date.checkValidity()) {
68          alert('Error: Invalid date');
69          return false;
70      }
71
72      if(realDate.valueOf() < inputDate) {
73          alert('Error: Date is in the future');
74          return false;
75      }
76
77      return true;
78  }
79
80  function validateAccount(account) {
81      if(account == '') {
82          alert('Error: Missing Account Number');
83          return false;
84      }
85
86      return true;
87  }
88
89  function validateType(type) {
90      if(type == '') {
91          alert('Error: Missing Transaction Type');
92          return false;
93      }
94
95      return true;
96  }
97
98  function validateSecurity(security) {
99      if(security == '') {
100         alert('Error: Missing Security');
101         return false;
```

```
102     }
103
104     return true;
105 }
106
107 function validateAmount(amount) {
108     if(amount == '') {
109         alert('Error: Missing Amount');
110         return false;
111     }
112
113     if(isNaN(amount)) {
114         alert('Error: Invalid Amount');
115         return false;
116     }
117
118     return true;
119 }
120
121 function validateDAmount(dAmount) {
122     if(dAmount == '') {
123         alert('Error: Missing $ Amount');
124         return false;
125     }
126
127     if(isNaN(dAmount)) {
128         alert('Error: Invalid $ Amount');
129         return false;
130     }
131
132     return true;
133 }
134
135 function generateId() {
136     var id = '';
137     var idLength = 6;
138
139     var characters = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789';
140     var charactersLength = characters.length;
141
142     var unique = false;
143
144     while(!unique) {
145         for(var i = 0; i < idLength; i++) {
146             id += characters.charAt(Math.floor(Math.random() *
                    ↪ charactersLength));
147         }
148
149         unique = true;
150         for(var i = 0; i < document.getElementsByClassName('idCell').
                ↪ length; i++) {
151             if(document.getElementsByClassName('idCell')[i].innerText ==
                    ↪ id) {
152                 unique = false;
```

```
153                 break;
154             }
155         }
156     }
157     return id;
158 }
159 
160 function calculateCostBasis(amount, dAmount) {
161     costBasis = (dAmount / amount).toFixed(2);
162     return costBasis;
163 }
164 
165 function addTransaction(data) {
166     var staging = data;
167     var tableBody = document.getElementById('tableBody');
168     var newRow = tableBody.insertRow(0);
169     newRow.classList += "bodyRow";
170 
171     var actionsContent = "<button type='button' onclick='editRow(this)'>
        ↪ Edit</button><button type='button' onclick='deleteRow(this)'>
        ↪ Delete</button>";
172     var fileContent = '<table><tbody>';
173     if(data.length > 8) {
174         for(let i = 0; i < data[8].length; i++) {
175             fileContent += "<tr><td><a onclick='downloadFile(`" + data[8][
                ↪ i][0] + "`);' href='javascript:void(0);'>" + data[8][i
                ↪ ][1] + "</a></td>";
176             fileContent += "<td><button type='button' onclick='
                ↪ removeFileFromTable(`" + data[8][i][0] + "`, this);'>-</
                ↪ button></td></tr>";
177         }
178     }
179     fileContent += '</tbody></table><input type="file" onchange="addFile(
        ↪ this, 0);" multiple/>';
180     staging[8] = fileContent;
181     staging[9] = actionsContent;
182 
183     var calculateCostBasis = true;
184     if(data[3][0] == '!') {
185         calculateCostBasis = false;
186         data[3] = data[3].substr(1);
187     }
188 
189     for(var i = 0; i < 10; i++) {
190         var newCell = newRow.insertCell(i);
191         var idShowing = (document.getElementById('toggleId').innerText ==
            ↪ "Hide Transaction ID");
192 
193         newCell.innerHTML = data[i];
194 
195         if(i == 0) {
196             if(idShowing)
197                 newCell.classList = "idCell frozenColumn1";
198             else {
```

```
                newCell.classList = "idCell";
                newCell.setAttribute("hidden", true);
            }
        }
        else if(i == 1) {
            if(idShowing)
                newCell.classList = "frozenColumn2";
            else
                newCell.classList = "frozenColumn1";
        }
        /*
        else if(i == 2) {
            if(idShowing)
                newCell.classList = "frozenColumn3";
            else
                newCell.classList = "frozenColumn2";
        }
        */

        if(i == 7 && !calculateCostBasis) {
            newCell.innerHTML = "N/A";
        }
    }
}

function fileIdGenerator() {
    return Math.floor(Math.random() *
        ↪ 10000000000000000000000000000000000000000).toString(36);
}

function addTransactionButton() {
    var data = getData();
    if(data) {
        var id = generateId();
        data.unshift(id);

        fileList = new Array()
        uploadFile(data, addTransactionWithFileName, fileList, 0);
        clearInput(false);
    }
}

function addTransactionWithFileName(data, fileName) {
    addTransaction(data);
    console.log(data);
    loadDataLists();
}

function deleteRow(button) {
    var row = button.parentElement.parentElement;

    var fileRows = row.getElementsByTagName('td')[8].getElementsByTagName
        ↪ ('table')[0].getElementsByTagName('tr');
    for(let i = 0; i < fileRows.length; i++) {
```

```
251         let fileId = fileRows[i].getElementsByTagName('a')[0].getAttribute
                ↪ ('onclick').split('`')[1];
252         deleteFileFromIndexedDB(fileId);
253     }
254 
255     document.getElementById("tableBody").removeChild(row);
256 
257     if(document.getElementsByClassName('editing').length == 0) {
258         document.getElementById('add').removeAttribute('hidden');
259         document.getElementById('save').setAttribute('hidden', true);
260         document.getElementById('discard').setAttribute('hidden', true);
261 
262         document.getElementById('add').setAttribute('type','submit');
263         document.getElementById('save').setAttribute('type','button');
264     }
265     loadDataLists();
266 }
267 
268 function editRow(button) {
269     if(document.getElementsByClassName('editing').length > 0)
270         document.getElementsByClassName('editing')[0].classList = "bodyRow
                ↪ ";
271 
272     var row = button.parentElement.parentElement;
273     var rowContent = row.getElementsByTagName('td');
274     row.classList = "bodyRow editing";
275 
276     document.getElementById('date').value = rowContent[1].innerText;
277     document.getElementById('account').value = rowContent[2].innerText;
278 
279     document.getElementById('type').value = rowContent[3].innerText;
280     if(document.getElementById('type').value == '') document.
            ↪ getElementById('type').value = '!' + rowContent[3].innerText;
281 
282     document.getElementById('security').value = rowContent[4].innerText;
283     document.getElementById('amount').value = rowContent[5].innerText;
284     document.getElementById('dAmount').value = rowContent[6].innerText;
285 
286     document.getElementById('add').setAttribute('hidden', true);
287     document.getElementById('save').removeAttribute('hidden');
288     document.getElementById('discard').removeAttribute('hidden');
289 
290     document.getElementById('add').setAttribute('type','button');
291     document.getElementById('save').setAttribute('type','submit');
292 
293     removeFileUpload();
294     uploadLabel = document.getElementById('fileUploadLabel');
295     if(rowContent[8].getElementsByTagName('tr').length > 0) {
296         uploadLabel.innerHTML = String(rowContent[8].getElementsByTagName
                ↪ ('tr').length) + " file(s)";
297     }
298     fileEditted = false;
299 }
300 
```

```
function saveChanges() {
    data = getData();
    if(data) {
        rowToEdit = document.getElementsByClassName('editing')[0];
        cellsToEdit = rowToEdit.getElementsByTagName('td');

        if(data[2][0] == '!') {
            data[2] = data[2].substr(1);
            data[6] = "N/A";
        }

        for(var i = 0; i < data.length; i++) {
            cellsToEdit[i + 1].innerHTML = data[i];
        }
        rowToEdit.classList = "bodyRow";

        document.getElementById('add').removeAttribute('hidden');
        document.getElementById('save').setAttribute('hidden', true);
        document.getElementById('discard').setAttribute('hidden', true);

        document.getElementById('add').setAttribute('type','submit');
        document.getElementById('save').setAttribute('type','button');

        if(fileEditted) {
            uploadFile([cellsToEdit[8]], updateExistingFileName, new Array
                ↪ (), 0);
        }
        else {
            removeFileUpload();
        }

        resetDate();
        clearInput(true);
        loadDataLists();
    }
}

function discardChanges() {
    document.getElementsByClassName('editing')[0].classList = "bodyRow";

    document.getElementById('add').removeAttribute('hidden');
    document.getElementById('save').setAttribute('hidden', true);
    document.getElementById('discard').setAttribute('hidden', true);

    document.getElementById('add').setAttribute('type','submit');
    document.getElementById('save').setAttribute('type','button');

    resetDate();
    clearInput(true);
    removeFileUpload();
}

function sortTable(column, ascending) {
    var rows = document.getElementsByClassName('bodyRow');
```

```
354
355     var sorting = true;
356     while(sorting) {
357         sorting = false;
358         for(var i = 0; i < (rows.length - 1); i++) {
359             rowA = rows[i].getElementsByTagName('td')[column];
360             rowB = rows[i + 1].getElementsByTagName('td')[column];
361
362             var swap = false;
363
364             if(ascending && rowA.innerHTML.toLowerCase() > rowB.innerHTML.
                    ↪ toLowerCase()) swap = true;
365             else if(!ascending && rowA.innerHTML.toLowerCase() < rowB.
                    ↪ innerHTML.toLowerCase()) swap = true;
366
367             if(swap) {
368                 sorting = true;
369                 document.getElementById('tableBody').insertBefore(rows[i +
                        ↪  1], rows[i]);
370             }
371         }
372     }
373 }
374
375 function resetDate() {
376     const today = new Date();
377     const year = new Intl.DateTimeFormat('en', { year: 'numeric' }).format
            ↪ (today);
378     const month = new Intl.DateTimeFormat('en', { month: '2-digit' }).
            ↪ format(today);
379     const day = new Intl.DateTimeFormat('en', { day: '2-digit' }).format(
            ↪ today);
380
381     document.getElementById('date').value = `${year}-${month}-${day}`;
382 }
383
384 function clearInput(clearAccount) {
385     if(clearAccount)
386         document.getElementById('account').value = '';
387
388     document.getElementById('type').value = '';
389     document.getElementById('security').value = '';
390     document.getElementById('amount').value = '';
391     document.getElementById('dAmount').value = '';
392 }
393
394 function validateFilters(filterId, startDate, endDate, filterAccount,
        ↪ filterType, filterSecurity, minAmount, maxAmount, minDAmount,
        ↪ maxDAmount, minCostBasis, maxCostBasis) {
395     if(!validateFilterId(filterId)) return false;
396     if(!validateDateRange(startDate, endDate)) return false;
397     if(!validateFilterAccount(filterAccount)) return false;
398     if(!validateFilterSecurity(filterSecurity)) return false;
399     if(!validateAmountRange(minAmount, maxAmount)) return false;
```

```
400     if(!validateDAmountRange(minDAmount, maxDAmount)) return false;
401     if(!validateCostBasisRange(minCostBasis, maxCostBasis)) return false;
402 
403     return true;
404 }
405 
406 function validateFilterId(id) {
407     return true;
408 }
409 
410 function validateDateRange(start, end) {
411     if(!start.checkValidity()) {
412         alert('Error: Invalid Start Date');
413         return false;
414     }
415 
416     if(!end.checkValidity()) {
417         alert('Error: Invalid End Date');
418         return false;
419     }
420 
421     if(start.valueAsNumber > end.valueAsNumber) {
422         alert('Error: Invalid Date Range');
423         return false;
424     }
425 
426     return true;
427 }
428 
429 function validateFilterAccount(account) {
430     return true;
431 }
432 
433 function validateFilterSecurity(security) {
434     return true;
435 }
436 
437 function validateAmountRange(min, max) {
438     if(isNaN(Number(min))) {
439         alert('Error: Min Amount is NaN');
440         return false;
441     }
442 
443     if(isNaN(Number(max))) {
444         alert('Error: Max Amount is NaN');
445         return false;
446     }
447 
448     if(Number(min) > Number(max) && min != '' && max != '') {
449         if(min != '') alert(min + '2');
450         if(max != '') alert(max + '1');
451         alert('Error: Invalid Amount Range');
452         return false;
453     }
```

```
454
455     return true;
456 }
457
458 function validateDAmountRange(min, max) {
459     if(isNaN(Number(min))) {
460         alert('Error: Min $ Amount is NaN');
461         return false;
462     }
463
464     if(isNaN(Number(max))) {
465         alert('Error: Max $ Amount is NaN');
466         return false;
467     }
468
469     if(Number(min) > Number(max) && min != '' && max != '') {
470         alert('Error: Invalid $ Amount Range');
471         return false;
472     }
473
474     return true;
475 }
476
477 function validateCostBasisRange(min, max) {
478     if(isNaN(Number(min))) {
479         alert('Error: Min Cost Basis is NaN');
480         return false;
481     }
482
483     if(isNaN(Number(max))) {
484         alert('Error: Max Cost Basis is NaN');
485         return false;
486     }
487
488     if(Number(min) > Number(max) && min != '' && max != '') {
489         alert('Error: Invalid Cost Basis Range');
490         return false;
491     }
492
493     return true;
494 }
495
496 function stringFilter(filtertext, tableitem) {
497     filters = filtertext.split(" && ");
498
499     for(var i = 0; i < filters.length; i++) {
500         filterORs = filters[i].split(" || ");
501         var meetsCriteria = false;
502
503         for(var ii = 0; ii < filterORs.length; ii++) {
504             if(filterORs[ii][0] == "!" && !tableitem.toUpperCase().
                ↪ includes(filterORs[ii].toUpperCase().substr(1)))
                ↪ meetsCriteria = true;
```

```
505             if(filterORs[ii][0] != "!" && tableitem.toUpperCase().includes
                    ↪ (filterORs[ii].toUpperCase())) meetsCriteria = true;
506         }
507 
508         if(!meetsCriteria) return false;
509     }
510 
511     return true;
512 }
513 
514 function applyFilter() {
515     unfilterAll();
516 
517     rows = document.getElementsByClassName('bodyRow');
518 
519     filterId = document.getElementById('filterId').value;
520     startDate = document.getElementById('startDate');
521     endDate = document.getElementById('endDate');
522     filterAccount = document.getElementById('filterAccount').value;
523     filterType = document.getElementById('filterType').value;
524     filterSecurity = document.getElementById('filterSecurity').value;
525     lowAmount = document.getElementById('lowAmount').value;
526     highAmount = document.getElementById('highAmount').value;
527     lowDAmount = document.getElementById('lowDAmount').value;
528     highDAmount = document.getElementById('highDAmount').value;
529     lowCostBasis = document.getElementById('lowCostBasis').value;
530     highCostBasis = document.getElementById('highCostBasis').value;
531 
532     if(lowDAmount[0] == '$') lowDAmount = lowDAmount.substr(1);
533     if(highDAmount[0] == '$') highAmount = highDAmount.substr(1);
534     if(lowCostBasis[0] == '$') lowCostBasis = lowCostBasis.substr(1);
535     if(highCostBasis[0] == '$') highCostBasis = highCostBasis.substr(1);
536 
537     if(validateFilters(filterId, startDate, endDate, filterAccount,
            ↪ filterType, filterSecurity, lowAmount, highAmount, lowDAmount,
            ↪ highDAmount, lowCostBasis, highCostBasis)) {
538         for(var i = 0; i < rows.length; i++) {
539             cells = rows[i].getElementsByTagName('td');
540             var hide = false;
541 
542             if(filterId != '' && !stringFilter(filterId,cells[0].innerText
                    ↪ ))
543                 hide = true;
544 
545             if(startDate.value != '' && startDate.value > cells[1].
                    ↪ innerText)
546                 hide = true;
547 
548             if(endDate.value != '' && endDate.value < cells[1].innerText)
549                 hide = true;
550 
551             if(filterAccount != '' && !stringFilter(filterAccount, cells
                    ↪ [2].innerText))
552                 hide = true;
```

```
553
554             if(filterType != '' && filterType != cells[3].innerText)
555                 hide = true;
556
557             if(filterSecurity != '' && !stringFilter(filterSecurity, cells
                    ↪ [4].innerText))
558                 hide = true;
559
560             if(lowAmount != '' && Number(lowAmount) >
                    ↪ formattedStringToNumber(cells[5].innerText))
561                 hide = true;
562
563             if(highAmount != '' && Number(highAmount) <
                    ↪ formattedStringToNumber(cells[5].innerText))
564                 hide = true;
565
566             if(lowDAmount != '' && Number(lowDAmount) >
                    ↪ formattedStringToNumber(cells[6].innerText.substr(1)))
567                 hide = true;
568
569             if(highDAmount != '' && Number(highDAmount) <
                    ↪ formattedStringToNumber(cells[6].innerText.substr(1)))
570                 hide = true;
571
572             if(lowCostBasis != '' && Number(lowCostBasis) >
                    ↪ formattedStringToNumber(cells[7].innerText.substr(1)))
573                 hide = true;
574
575             if(highCostBasis != '' && Number(highCostBasis) <
                    ↪ formattedStringToNumber(cells[7].innerText.substr(1)))
576                 hide = true;
577
578             if(filterNA.checked && cells[7].innerText == "N/A")
579                 hide = true;
580
581             if(hide)
582                 rows[i].setAttribute('hidden', true);
583         }
584     }
585 }
586
587 function clearFilter() {
588     unfilterAll();
589
590     var fields = document.getElementsByClassName('filterField');
591
592     for(var i = 0; i < fields.length; i++) {
593         fields[i].value = '';
594     }
595
596     document.getElementById('filterNA').checked = false;
597 }
598
599 function unfilterAll() {
```

```
600     rows = document.getElementsByClassName('bodyRow');
601
602     for(var i = 0; i < rows.length; i++) {
603         rows[i].removeAttribute('hidden');
604     }
605 }
606
607 function toggleID() {
608     var button = document.getElementById('toggleId');
609     var rows = document.getElementsByTagName('tr');
610     var cells = rows[0].getElementsByTagName('th');
611
612     if(button.innerText == "Hide Transaction ID") {
613         button.innerText = "Show Transaction ID";
614
615         cells[0].setAttribute('hidden', true);
616         cells[0].classList = "";
617         cells[1].classList = "frozenColumn1";
618         //cells[2].classList = "frozenColumn2";
619         for(var i = 1; i < rows.length; i++) {
620             cells = rows[i].getElementsByTagName('td');
621
622             cells[0].setAttribute('hidden', true);
623             cells[0].classList = "idCell";
624             cells[1].classList = "frozenColumn1";
625             //cells[2].classList = "frozenColumn2";
626         }
627     }
628     else {
629         button.innerText = "Hide Transaction ID";
630
631         cells[0].removeAttribute('hidden');
632         cells[0].classList = "frozenColumn1";
633         cells[1].classList = "frozenColumn2";
634         //cells[2].classList = "frozenColumn3";
635         for(var i = 1; i < rows.length; i++) {
636             cells = rows[i].getElementsByTagName('td');
637
638             cells[0].removeAttribute('hidden');
639             cells[0].classList = "idCell frozenColumn1";
640             cells[1].classList = "frozenColumn2";
641             //cells[2].classList = "frozenColumn3";
642         }
643     }
644 }
645
646 function readFile(fileIn){
647     if(fileIn.files && fileIn.files[0]) {
648         var reader = new FileReader();
649         reader.onload = function (e) {
650             var output = e.target.result;
651             document.getElementById('typesArray').value = output;
652         };
653         reader.readAsText(fileIn.files[0]);
```

```
654     }
655 }
656
657 function saveFile() {
658   var element = document.createElement('a');
659   element.setAttribute('href', 'data:text/plain;charset=utf-8,' +
        ↪ encodeURIComponent(document.getElementById('typesArray').value));
660   element.setAttribute('download', 'transaction-types.csv');
661
662   element.style.display = 'none';
663   document.body.appendChild(element);
664
665   element.click();
666
667   document.body.removeChild(element);
668 }
669
670 function applyTypes() {
671     var typesArray = document.getElementById('typesArray').value.split
        ↪ (',');
672     setTransactionTypesList(typesArray);
673 }
674
675 function editTypes() {
676     document.getElementById('typesArray').value = readCurrentTypes().join
        ↪ (',');
677 }
678
679 function setTransactionTypesList(typesArray) {
680     var type = document.getElementById('type');
681     var filterType = document.getElementById('filterType');
682
683     type.innerHTML = '<option value=""></option>';
684     filterType.innerHTML = '<option value=""></option>';
685
686     for(var i = 0; i < typesArray.length; i++) {
687         var typeAsText = typesArray[i];
688         if(typesArray[i][0] == '!') typeAsText = typesArray[i].substr(1);
689
690         type.innerHTML += '<option value="' + typesArray[i] + '">' +
            ↪ typeAsText + '</option>';
691         filterType.innerHTML += '<option value="' + typeAsText + '">' +
            ↪ typeAsText + '</option>';
692     }
693 }
694
695 function toggleSection(button) {
696     var form = button.parentElement.parentElement.getElementsByTagName('
        ↪ form')[0];
697
698     if(button.innerText == "Hide") {
699         form.setAttribute("hidden",true);
700         button.innerText = "Show";
701     }
```

```
    else {
        form.removeAttribute("hidden");
        button.innerText = "Hide";
    }
}

function loadDataLists() {
    var accountsList = document.getElementById("accountsList");
    var securitiesList = document.getElementById("securitiesList");
    var rows = document.getElementsByClassName("bodyRow");

    var accounts = [];
    var securities = [];

    for(var i = 0; i < rows.length; i++) {
        var tableAccount = rows[i].getElementsByTagName("td")[2].innerText
            ↪ ;
        var tableSecurity = rows[i].getElementsByTagName("td")[4].
            ↪ innerText;

        if(!accounts.includes(tableAccount)) accounts.push(tableAccount);
        if(!securities.includes(tableSecurity)) securities.push(
            ↪ tableSecurity);
    }

    accountsList.innerHTML = '';
    securitiesList.innerHTML = '';

    for(var i = 0; i < accounts.length; i++) {
        accountsList.innerHTML += '<option value="' + accounts[i] + '"/>';
    }

    for(var i = 0; i < securities.length; i++) {
        securitiesList.innerHTML += '<option value="' + securities[i] +
            ↪ '"/>';
    }
}

window.onload = function() {
    resetDate();
    initDb();
}
```

B.2 Google API Script

```
var auth2;

var spreadsheetId = "1R0HpaAIUw-JHX8SrzvkEPCG1qgI-siJ9oucY6g5e4Co";
var sheetId = "Sheet1";
var sheetIdNum = 0;

function loadSheetData() {
    if(auth2.isSignedIn.get())
    {
        getAllUserSheets();
    }
    else {
        authenticate()
            .then(function() {
                if(auth2.isSignedIn.get()) getAllUserSheets();
            });
    }
}

function getNewSheetData() {
    spreadsheetId = document.getElementById('sheet').value;
    if(auth2.isSignedIn.get())
    {
        getTabsOfSheet();
    }
    else {
        authenticate()
            .then(function() {
                if(auth2.isSignedIn.get()) getTabsOfSheet();
            });
    }
}

function populateSheetSelector(arrayOfSheets) {
    document.getElementById('sheet').innerHTML = '<option value="1
       ↪ R0HpaAIUw-JHX8SrzvkEPCG1qgI-siJ9oucY6g5e4Co">default</option>';

    for(var i = 0; i < arrayOfSheets.length; i++) {
        document.getElementById('sheet').innerHTML += '<option value="' +
           ↪ arrayOfSheets[i].id + '">' + arrayOfSheets[i].name + '</
           ↪ option>';
    }
}

function getNewTabData() {
    data = document.getElementById('tab').value.split(/,(.+)/);
    sheetIdNum = data[0];
    sheetId = data[1];
}

function populateTabSelector(arrayOfTabs) {
    document.getElementById('tab').innerHTML = '';
```

```
50
51      for(var i = 0; i < arrayOfTabs.length; i++) {
52          document.getElementById('tab').innerHTML += '<option value="' +
                ↪ arrayOfTabs[i].properties.sheetId + ',' + arrayOfTabs[i].
                ↪ properties.title + '">' + arrayOfTabs[i].properties.title +
                ↪ '</option>';
53      }
54  }
55
56  function getAllUserSheets() {
57      return gapi.client.drive.files.list({
58          "pageSize": 1000,
59          "orderBy": "name",
60          "q": "mimeType = 'application/vnd.google-apps.spreadsheet'",
61      })
62          .then(function(response) {
63              populateSheetSelector(JSON.parse(response.body).files);
64              getNewSheetData();
65              console.log("Response", response);
66          },
67          function(err) { console.error("Execute error", err); });
68  }
69
70  function getTabsOfSheet() {
71      return gapi.client.sheets.spreadsheets.get({
72        "spreadsheetId": spreadsheetId,
73        "includeGridData": false
74      })
75          .then(function(response) {
76              populateTabSelector(JSON.parse(response.body).sheets);
77              getNewTabData();
78              console.log("Response", response);
79          },
80          function(err) { console.error("Execute error", err); });
81  }
82
83  function tableToArrays() {
84      var rows = document.getElementsByClassName('bodyRow');
85      var data = new Array();
86      data.push(["Transaction Id", "Date", "Account Number", "Transaction
            ↪ Type", "Security", "Amount", "$ Amount", "Cost Basis", "Files"])
            ↪ ;
87
88      for(var i = 0; i < rows.length; i++) {
89          var cells = rows[i].getElementsByTagName('td');
90          var cellData = new Array();
91
92          for(var j = 0; j < 8; j++) {
93              cellData.push(cells[j].innerText);
94          }
95          cellData.push(getFileNamesIds(cells[8]));
96          data.push(cellData);
97      }
98
```

```
99      console.log(data);
100     return data;
101 }
102
103 function arraysToTable(dataArr) {
104     while(document.getElementsByClassName('bodyRow').length > 0) {
105         document.getElementById("tableBody").removeChild(document.
                ↪ getElementsByClassName('bodyRow')[0]);
106     }
107
108     document.getElementById('add').removeAttribute('hidden');
109     document.getElementById('save').setAttribute('hidden', true);
110     document.getElementById('discard').setAttribute('hidden', true);
111
112     document.getElementById('add').setAttribute('type', 'submit');
113     document.getElementById('save').setAttribute('type', 'button');
114
115     while(dataArr.length > 0) {
116         let data = dataArr[dataArr.length - 1];
117         let files = parseFileNamesIds(data[8]);
118         data.pop();
119         if(files.length > 0) {
120             data.push(files);
121         }
122         addTransaction(data);
123         dataArr.pop();
124     }
125     loadDataLists();
126 }
127
128 function authenticate() {
129 return gapi.auth2.getAuthInstance()
130     .signIn({scope: "https://www.googleapis.com/auth/drive"})
131     .then(function() { console.log("Sign-in successful"); },
132         function(err) { console.error("Error signing in", err); });
133 }
134
135 function loadClientSheets() {
136 gapi.client.setApiKey("AIzaSyDC6JNuMW78Q-gWsp0PFEaTsICYjHWymAo");
137 return gapi.client.load("https://content.googleapis.com/discovery/v1/apis/
    ↪ sheets/v4/rest")
138     .then(function() { console.log("GAPI client loaded for API");
        ↪ loadSheetData(); },
139         function(err) { console.error("Error loading GAPI client for API",
            ↪  err); });
140 }
141
142 function loadClient() {
143 gapi.client.setApiKey("AIzaSyDC6JNuMW78Q-gWsp0PFEaTsICYjHWymAo");
144 return gapi.client.load("https://content.googleapis.com/discovery/v1/apis/
    ↪ drive/v3/rest")
145     .then(function() { console.log("GAPI client loaded for API");
        ↪ loadClientSheets(); },
```

```
146         function(err) { console.error("Error loading GAPI client for API",
                ↪  err); });
147 }
148
149 function readGoogleSheetDB() {
150     clearIndexedDb("sheets");
151     return gapi.client.sheets.spreadsheets.values.get({
152         "spreadsheetId": spreadsheetId,
153         "range": sheetId + "!A2:I214748354"
154     })
155         .then(function(response) {
156             console.log("Response", response);
157
158             dataArr = [];
159             if(JSON.parse(response.body).values != undefined) {
160                 dataArr = JSON.parse(response.body).values;
161             }
162             arraysToTable(dataArr);
163
164             readGoogleTypes();
165         },
166         function(err) { console.error("Execute error", err); });
167 }
168
169 function readGoogleTypes() {
170     return gapi.client.sheets.spreadsheets.values.get({
171         "spreadsheetId": spreadsheetId,
172         "range": sheetId + "!J1:J214748354",
173         "majorDimension": "COLUMNS"
174
175     })
176         .then(function(response) {
177             console.log("Response", response);
178             var typesArr = JSON.parse(response.body).values[0];
179             setTransactionTypesList(typesArr);
180         },
181         function(err) { console.error("Execute error", err); });
182 }
183
184 function readCurrentTypes() {
185     var types = document.getElementById('type').getElementsByTagName('
            ↪ option');
186     var currentTypes = [];
187
188     for(var i = 1; i < types.length; i++) {
189         currentTypes.push(types[i].value);
190     }
191
192     return currentTypes;
193 }
194
195 function writeGoogleSheetDB() {
196     if(auth2.isSignedIn.get())
197     {
```

```
198         setGoogleRows()
199     }
200     else {
201         authenticate()
202             .then(function() {
203                 if(auth2.isSignedIn.get()) setGoogleRows();
204             });
205     }
206 }
207
208 function setGoogleRows() {
209     return gapi.client.sheets.spreadsheets.batchUpdate({
210         "spreadsheetId": spreadsheetId,
211         "resource": {
212         "requests": [
213             {
214             "updateSheetProperties": {
215                 "properties": {
216                     "gridProperties": {
217                         "columnCount": 10,
218                         "rowCount": 1
219                 },
220                     "sheetId": sheetIdNum
221                 },
222                 "fields": "gridProperties"
223             }
224             }
225         ]
226         }
227     })
228         .then(function(response) {
229             console.log("Response", response);
230
231             clearGoogleRow();
232         },
233         function(err) { console.error("Execute error", err); });
234 }
235
236 function clearGoogleRow() {
237     return gapi.client.sheets.spreadsheets.values.clear({
238     "spreadsheetId": spreadsheetId,
239     "range": sheetId + "!A1:J1",
240     "resource": {}
241     })
242         .then(function(response) {
243             console.log("Response", response);
244
245             writeGoogleDB();
246         },
247         function(err) { console.error("Execute error", err); });
248 }
249
250 function writeGoogleDB() {
251     writeImagesToFirestore("sheets");
```

```
    return gapi.client.sheets.spreadsheets.values.batchUpdate({
        "spreadsheetId": spreadsheetId,
        "resource": {
            "data": [
            {
                "range": sheetId + "!A1",
                "values": tableToArrays(),
                "majorDimension": "ROWS"
            },
            {
                "range": sheetId + "!J1",
                "values": [readCurrentTypes()],
                "majorDimension": "COLUMNS"
            }
            ],
            "valueInputOption": "RAW"
        }
        })
        .then(function(response) {
            console.log("Response", response);
        },
        function(err) { console.error("Execute error", err); });
}

gapi.load("client:auth2", function() {
    auth2 = gapi.auth2.init({client_id: "217251662395-9
       ↪ pu2qa1hubgrblav1nhvnfaascd6povv.apps.googleusercontent.com"});
    loadClient();
});
```

B.3 Firebase Script

```
function clearFirebase() {
    firebase.database().ref('Data').remove();
}

function writeToFirebase() {
    writeImagesToFirestore("firebase");
    clearFirebase();

    var data = tableToArrays();
    var typesArr = readCurrentTypes();

    for(var i = 1; i < data.length; i++) {
        firebase.database().ref('Data/' + String(i - 1)).set({
            id: data[i][0],
            date: data[i][1],
            account: data[i][2],
            type: data[i][3],
            security: data[i][4],
            amount: data[i][5],
            dAmount: data[i][6],
            costBasis: data[i][7],
            files: data[i][8]
        });
    }
    for(var i = 0; i < typesArr.length; i++) {
        firebase.database().ref('Types/' + String(i)).set({
            value: typesArr[i]
        });
    }
}

function readFromFirebase() {
    clearIndexedDb("firebase");
    return firebase.database().ref('/').once('value').then(function(
        ↪ snapshot) {

        while(document.getElementsByClassName('bodyRow').length > 0) {
            document.getElementById("tableBody").removeChild(document.
                ↪ getElementsByClassName('bodyRow')[0]);
        }

        data = snapshot.val().Data;
        console.log(data);
        for(var i = data.length - 1; i >= 0; i--) {
            let staged = [data[i].id, data[i].date, data[i].account, data[
                ↪ i].type, data[i].security, data[i].amount, data[i].
                ↪ dAmount, data[i].costBasis];
            if(parseFileNamesIds(data[i].files).length > 0) {
                staged.push(parseFileNamesIds(data[i].files));
            }
            console.log(staged);
            addTransaction(staged);
```

```
        }

        types = snapshot.val().Types;
        var typesArr = [];
        for(var i = 0; i < types.length; i++) {
            typesArr.push(types[i].value);
        }
        setTransactionTypesList(typesArr);
        loadDataLists();
    });
}

function clearFirestore() {
    firestore.collection("Data").get().then((querySnapshot) => {
        querySnapshot.forEach((doc) => {
            firestore.collection("Data").doc(doc.id).delete();
        });
    }).then(function() {
        firestore.collection("Types").get().then((querySnapshot) => {
            querySnapshot.forEach((doc) => {
                firestore.collection("Types").doc(doc.id).delete();
            });
        })
    }).then(function() { return 0 });
}

function writeToFirestore() {
    writeImagesToFirestore("firestore");
    firestore.collection("Data").get().then((querySnapshot) => {
        querySnapshot.forEach((doc) => {
            firestore.collection("Data").doc(doc.id).delete();
        });
    }).then(function() {
        firestore.collection("Types").get().then((querySnapshot) => {
            querySnapshot.forEach((doc) => {
                firestore.collection("Types").doc(doc.id).delete();
            });
        }).then(function() {
            var data = tableToArrays();
            var typesArr = readCurrentTypes();

            for(var i = 1; i < data.length; i++) {
                firestore.collection("Data").add({
                    id: data[i][0],
                    date: data[i][1],
                    account: data[i][2],
                    type: data[i][3],
                    security: data[i][4],
                    amount: data[i][5],
                    dAmount: data[i][6],
                    costBasis: data[i][7],
                    files: data[i][8],
                    index: i - 1
                })
```

```
                .then(function(docRef) {
                    console.log("Document written with ID: ", docRef.id);
                })
                .catch(function(error) {
                    console.error("Error adding document: ", error);
                });
            }
            for(var i = 0; i < typesArr.length; i++) {
                firestore.collection("Types").add({
                    value: typesArr[i],
                    index: i
                })
                .then(function(docRef) {
                    console.log("Document written with ID: ", docRef.id);
                })
                .catch(function(error) {
                    console.error("Error adding document: ", error);
                });
            }
        })
    });
}

function readFromFirestore() {
    clearIndexedDb("firestore");
    firestore.collection("Data").get().then((querySnapshot) => {
        var data = new Array();

        querySnapshot.forEach((doc) => {
            data[doc.data().index] = doc.data();
            console.log(data);
        });

        while(document.getElementsByClassName('bodyRow').length > 0) {
            document.getElementById("tableBody").removeChild(document.
                ↪ getElementsByClassName('bodyRow')[0]);
        }

        for(let i = data.length - 1; i >= 0; i--) {
            let staged = [data[i].id, data[i].date, data[i].account, data[
                ↪ i].type, data[i].security, data[i].amount, data[i].
                ↪ dAmount, data[i].costBasis];
            if(parseFileNamesIds(data[i].files).length > 0) {
                staged.push(parseFileNamesIds(data[i].files));
            }
            console.log(staged);
            addTransaction(staged);
        }
        loadDataLists();
    });

    firestore.collection("Types").get().then((querySnapshot) => {
        var typesArr = [];

```

```
154             querySnapshot.forEach((doc) => {
155                 typesArr[doc.data().index] = doc.data().value;
156             });
157 
158             setTransactionTypesList(typesArr);
159         });
160 }
```

B.4 MySQL Script

```
function writeToMySQL() {
    writeImagesToFirestore("mysql");
    var data = tableToArrays();
    var types = readCurrentTypes();

    fetch('http://localhost:5000/api', {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json',
        },
        body: JSON.stringify([data, types]),
    });
}

function readFromMySQL() {
    clearIndexedDb("mysql");
    fetch('http://localhost:5000/api')
        .then(response => {
            return response.json()
        })
        .then(fullresponse => {
            console.log(fullresponse);

            while(document.getElementsByClassName('bodyRow').length > 0) {
                document.getElementById("tableBody").removeChild(document.
                    ↪ getElementsByClassName('bodyRow')[0]);
            }

            var data = fullresponse[0];
            for(var i = data.length - 1; i >= 0; i--) {
                let staged = [data[i].id, data[i].date, data[i].account,
                    ↪ data[i].type, data[i].security, data[i].amount, data
                    ↪ [i].dAmount, data[i].costBasis];
                if(parseFileNamesIds(data[i].files).length > 0) {
                    staged.push(parseFileNamesIds(data[i].files));
                }
                addTransaction(staged);
            }
            loadDataLists();

            var types = fullresponse[1];
            var typesArr = [];
            for(var i = 0; i < types.length; i++) {
                typesArr.push(types[i].typename);
            }
            setTransactionTypesList(typesArr);
        })
}
```

B.5 Local Storage Script

```
let db;
let dbVersion = 1;
let dbReady = false;

var fileEditted = false;

function initDb() {
    let reset = indexedDB.deleteDatabase('FileStorage');
    reset.onsuccess = function(a) {
        let request = indexedDB.open('FileStorage', dbVersion);

        request.onerror = function(e) {
            console.error('Unable to open database.');
        }

        request.onsuccess = function(e) {
            db = e.target.result;
            console.log('db opened');
        }

        request.onupgradeneeded = function(e) {
            let db = e.target.result;
            db.createObjectStore('files', {keyPath:'id', autoIncrement:
                ↪ false});
            dbReady = true;
        }
    }
}

function fileUploadChanged() {
    fileIn = document.getElementById('fileUpload');
    if(fileIn.files && fileIn.files[0]) {
        document.getElementById('fileUploadLabel').innerHTML = String(
            ↪ fileIn.files.length) + " file(s)";
    }
    fileEditted = true;
    console.log("updated file upload");
}

function uploadFile(data, cb, fileList, index) {
    fileId = fileIdGenerator();
    fileIn = document.getElementById('fileUpload');
    if(fileIn.files && fileIn.files[index]) {
        var reader = new FileReader();
        reader.onload = function (e) {
            console.log(e.target.result);

            let bits = btoa(e.target.result);
            let ob = {
                id: fileId,
                type: fileIn.files[index].type,
                name: fileIn.files[index].name,
```

```
51                  data: bits
52              };
53
54              let trans = db.transaction(['files'], 'readwrite');
55              let addReq = trans.objectStore('files').put(ob);
56
57              addReq.onerror = function(e) {
58                  console.log('error storing data');
59                  console.error(e);
60              }
61
62              trans.oncomplete = function(e) {
63                  console.log('data stored');
64                  fileList.push([fileId, fileIn.files[index].name]);
65                  index++;
66                  uploadFile(data, cb, fileList, index);
67              }
68          };
69          reader.readAsBinaryString(fileIn.files[index])
70      }
71      else {
72          removeFileUpload();
73          data.push(fileList);
74          cb(data);
75      }
76  }
77
78  function addFile(fileIn, index) {
79      fileId = fileIdGenerator();
80      if(fileIn.files && fileIn.files[index]) {
81          var reader = new FileReader();
82          reader.onload = function (e) {
83              console.log(e.target.result);
84
85              let bits = btoa(e.target.result);
86              let ob = {
87                  id: fileId,
88                  type: fileIn.files[index].type,
89                  name: fileIn.files[index].name,
90                  data: bits
91              };
92
93              let trans = db.transaction(['files'], 'readwrite');
94              let addReq = trans.objectStore('files').put(ob);
95
96              addReq.onerror = function(e) {
97                  console.log('error storing data');
98                  console.error(e);
99              }
100
101             trans.oncomplete = function(e) {
102                 console.log('data stored');
103                 let table = fileIn.parentElement.getElementsByTagName('
                        ↪ tbody')[0];
```

```
104                 let newRowContent = "<tr><td><a onclick='downloadFile(`" +
                        ↪  fileId + "`);' href='javascript:void(0);'>" +
                        ↪ fileIn.files[index].name + "</a></td>";
105                 newRowContent += "<td><button type='button' onclick='
                        ↪ removeFileFromTable(`" + fileId + "`, this);'>-</
                        ↪ button></td></tr>";
106                 table.innerHTML += newRowContent;
107                 index++;
108                 addFile(fileIn, index);
109             }
110         };
111         reader.readAsBinaryString(fileIn.files[index])
112     }
113     else {
114         fileIn.value = null;
115     }
116 }
117
118 function updateExistingFileName(data) {
119     for(let i = 0; i < data[0].getElementsByTagName('tr').length; i++) {
120         let fileId = data[0].getElementsByTagName('tr')[i].
                ↪ getElementsByTagName('a')[0].getAttribute('onclick').split
                ↪ ('`')[1];
121         deleteFileFromIndexedDB(fileId);
122     }
123
124     var fileContent = '<table><tbody>';
125     if(data.length > 1) {
126         for(let i = 0; i < data[1].length; i++) {
127             fileContent += "<tr><td><a onclick='downloadFile(`" + data[1][
                    ↪ i][0] + "`);' href='javascript:void(0);'>" + data[1][i
                    ↪ ][1] + "</a></td>";
128             fileContent += "<td><button type='button' onclick='
                    ↪ removeFileFromTable(`" + data[1][i][0] + "`, this);'>-</
                    ↪ button></td></tr>";
129         }
130     }
131     fileContent += '</tbody></table><input type="file" onchange="addFile(
            ↪ this, 0);" multiple/>';
132     data[0].innerHTML = fileContent;
133 }
134
135 function removeFileFromTable(fileId, cell) {
136     var row = cell.parentElement.parentElement;
137
138     if(confirm("Delete " + row.getElementsByTagName('a')[0].innerText +
            ↪ "?")) {
139         row.parentElement.removeChild(row);
140         deleteFileFromIndexedDB(fileId);
141     }
142 }
143
144 function deleteFileFromIndexedDB(fileId) {
145     let trans = db.transaction(['files'], 'readwrite');
```

```
146     let addReq = trans.objectStore('files').delete(fileId);
147 }
148 
149 function removeFileUpload() {
150     document.getElementById('fileUpload').value = null;
151     document.getElementById('fileUploadLabel').innerHTML = "Upload file";
152     fileEditted = true;
153 }
154 
155 function downloadFile(fileId) {
156     console.log('downloading');
157     var trans = db.transaction(['files'], 'readonly');
158     var dlReq = trans.objectStore('files').get(fileId);
159 
160     dlReq.onerror = function(e) {
161         console.log('error reading data');
162         console.error(e);
163     };
164 
165     dlReq.onsuccess = function(e) {
166         console.log('data read');
167         console.log(dlReq.result);
168         var element = document.createElement('a');
169         element.setAttribute('href', 'data:' + dlReq.result.type + ';
                ↪ base64,' + dlReq.result.data);
170         element.setAttribute('download', dlReq.result.name);
171 
172         element.style.display = 'none';
173         document.body.appendChild(element);
174 
175         element.click();
176 
177         document.body.removeChild(element);
178     };
179 }
180 
181 window.onbeforeunload = function(){
182     indexedDB.deleteDatabase('FileStorage');
183 }
```

B.6 Firestore Images Script

```
function writeImagesToFirestore(database) {
    firestore.collection("Images:" + database).get().then((querySnapshot)
       ↪ => {
        querySnapshot.forEach((doc) => {
            firestore.collection("Images:" + database).doc(doc.id).delete
               ↪ ();
        });
    }).then(function() {
        var trans = db.transaction(['files'], 'readonly');
        var dlReq = trans.objectStore('files').getAll();

        dlReq.onerror = function(e) {
            console.log('error reading data');
            console.error(e);
        };

        dlReq.onsuccess = function(e) {
            console.log(dlReq.result);

            for(let i = 0; i < dlReq.result.length; i++) {
                firestore.collection("Images:" + database).add({
                    id: dlReq.result[i].id,
                    type: dlReq.result[i].type,
                    name: dlReq.result[i].name,
                    data: dlReq.result[i].data,
                })
                .then(function(docRef) {
                    console.log("Image written with ID: ", docRef.id);
                })
                .catch(function(error) {
                    console.log("Error adding image: ", error);
                });
            }
        };
    });
}

function readImagesFromFirestore(database) {
    firestore.collection("Images:" + database).get().then((querySnapshot)
       ↪ => {
        let trans = db.transaction(['files'], 'readwrite');

        trans.oncomplete = function(e) {
            console.log('data stored');
        }

        querySnapshot.forEach((doc) => {
            console.log("Writing ", doc.data().name);

            let ob = {
                id: doc.data().id,
                type: doc.data().type,
```

```
                name: doc.data().name,
                data: doc.data().data
            };

            let addReq = trans.objectStore('files').put(ob);
        });
    });
}

function getFileNamesIds(cell) {
    var links = cell.getElementsByTagName('a');
    var result = '';

    for(let i = 0; i < links.length; i++) {
        result += links[i].innerText + '/' + links[i].getAttribute('
            ↪ onclick').split('`')[1] + '/';
    }

    return result;
}

function parseFileNamesIds(string) {
    var result = new Array();

    if(string != '') {
        let strarr = string.split('/');
        for(let i = 0; i < strarr.length - 1; i += 2) {
            result.push([strarr[i + 1], strarr[i + 0]]);
        }
    }

    return result;
}

function clearIndexedDb(database) {
    console.log("db reset");
    let trans = db.transaction(['files'], 'readwrite');
    var clearReq = trans.objectStore('files').clear();

    trans.oncomplete = function(e) {
        readImagesFromFirestore(database);
    }
}
```

C CSS Source Code

```
body {
    font-size: 14px;
}

input,
select {
    min-width: 100px;
    width: 80%;
}

button {
    width: 40%;
}

span {
    border: solid;
    border-width: 1px;
}

#fileUpload {
    display: none;
}

#removeFile {
    width: 16px;
    padding: 0;
    border: 0;
}

.sectionHead {
}

    .toggleSection {
        width: 100px;
    }

    .sectionHead > * {
        display: inline-block;
    }

#table {
    max-height: 50vh;
    overflow: auto;
}

#inputFields,
#filter {
    padding: 10px 0;
    overflow-x: auto;
}

#inputFields > form,
```

```
#filter > form {
    min-width: 1800px;
}

    #inputFields > form > section {
        width: 14%;
        display: inline-block;
    }

    #filter > form > section {
        width: 10%;
        display: inline-block;
    }

        #filter > form > section > button {
            width: 100%;
        }

#filterNA {
    display: inline-block;
    width: 50%;
    min-width: 16px;
}

#options > form > section {
    display: inline-block;
    width: 300px;
    margin: 15px;
}

    #toggleId,
    #fileInput {
        width: 80%;
    }

    #transactionTypesConfig {
        width: 500px !important;
    }

    #transactionTypesConfig > button {
        display: inline-block;
        width: 30%;
        padding: 0;
    }

    #typesArray {
        width: 40%;
    }

    #options > form > #googleSheetButtons {
        width: 200px;
    }

    #logoutSection#logoutSection {
```

```
        width: 100px;
    }

    #googleSheetButtons > button,
    #logoutSection > button {
        width: 100%;
        padding: 0;
        margin: 0;
    }

table {
    width: 100%;
    margin: auto;
    border-collapse: collapse;
}
    th {
        min-width: 200px;
        width: 10%;
        position: sticky;
        top: 0;
    }

    th > section {
        width: 80%;
        display: inline-block;
        padding: 0;
        margin: 0;
    }

    th,
    td {
        z-index: 0;
        background: white;
    }

    .frozenColumn1,
    .frozenColumn2,
    .frozenColumn3 {
        position: sticky;
        padding: 0;
        min-width: 200px;
    }

    td.frozenColumn1,
    td.frozenColumn2,
    td.frozenColumn3 {
        z-index: 1;
    }

    th.frozenColumn1,
    th.frozenColumn2,
    th.frozenColumn3 {
        z-index: 2;
    }
```

```
161
162     .frozenColumn1 {
163         left: 0;
164     }
165
166     .frozenColumn2 {
167         left: 200px;
168     }
169
170     .frozenColumn3 {
171         left: 400px;
172     }
173
174     .sort {
175         width: 10%;
176     }
177
178     .sort > button {
179         padding: 0;
180         border: 0;
181         display: block;
182         width: 100%;
183     }
184
185     .editing > td {
186         background: yellow;
187     }
188
189     #table,
190     table,
191     td,
192     th {
193         box-shadow: 1px 1px black, inset 1px 1px black;
194     }
```