Documentation for Financial Transactions Web Application

Jason N.

June 1, 2020

Contents

1 Disclaimer

This project is meant solely as a proof of concept to demonstrate how different databases might be used in this context. The project is NOT meant to be used in production. Several security flaws are present, including SQL injection, possible XSS, lack of authentication, etc.

2 Setup

This section is meant to serve as a general guide for setting up integrations used in this project. The detail in this guide is limited as the process will depend heavily on your choices, which I have attempted to outline for you. Many materials are referenced in this guide which contain far more detail, I would strongly suggest reading through these if they apply to your setup.

2.1 Google API

This is required for interacting with the Google Sheets database.

Go to https://console.developers.google.com/ and create a new project if you haven't already done that.

From the library panel, enable the Google Sheets API and the Google Drive API.

From the credentials panel, create an API key.

From the credentials panel, create an OAuth Client ID for a web application. Give it a name, which will appear when users are prompted to give the app permissions. Add the URIs that are expected to use the app. When testing this locally, it can be useful to add http://localhost:5000 or similar. These can always be changed at any moment from the developer console.

In the public/googleApiScript.js file of this repository, remember to change the client id and both instances of the api key to the appropriate values for your project.

2.2 Firebase

Firebase is used to host the web application and two of the databases, as well as storing images for all other databases to reference. All features are available through the same firebase project.

To get started, simply navigate to https://console.firebase.google.com and click "Add Project". Follow the instructions to set the name of the project and decide whether or not you want to make use of analytics.

Once a project has been created, follow the instructions at https://firebase.google.com/docs/web/setup to set up firebase with the front-end application. If you are using the files in this repository, the necessary SDKs are already included, though you'll need to change the firebase config to the appropriate values for your project.

To set up the real-time database, follow the instructions at https://firebase.google.com/docs/database/web/start to create a database and get your real-time database url. If you're using files from this repository, modify the firebase config to use this url instead of the given one.

To set up firestore, follow the instructions at https://firebase.google.com/docs/firestore/quickstart to create a database. If you're using files from this repository, modify the contents of the object passed to the

firebase.initializeApp() method to use appropriate values for your project. This method is called in the public/firebaseScript.js file.

To set up cloud functions, follow the instructions at https://firebase.google.com/docs/functions/get-started. If you're using files from this repository, the files already exist and just need to be deployed.

2.3 MySQL

There are several different implementations of MySQL available. MariaDB was used to create and test this project, which is a fork of MySQL.

If you decide to host the database yourself, you'll need to start the program, log in, and create a database. In this repository, it is named 'mydb', however, this can be changed if desired.

Remember to change the ip address, database name, and credentials in the index.js file of the firebase cloud functions folder.

If you wish to use this repository, the database can be imported using the dump.txt file:

```
mysql -u username -p database_name < dump.txt
```

Otherwise, once the database is created, enter the database using 'use database name;' to enter the database.

To create tables, you can use the following query:

```
CREATE TABLE ‘name‘ ( ‘colname1‘ datatype, ‘colname2‘ datatype, ‘colname3‘ datatype... )
```

Here is a useful website containing various MySQL commands: https://www.mysqltutorial.org/mysql-cheat-sheet.aspx/

3 HTML

3.1 Preamble and head

3.1.1 Preamble

Declares the document as HTML5.

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

3.1.2 meta charset

Specifies that characters in the file are encoded in UTF-8.

```
4 <meta charset = "UTF-8"/>
```

3.1.3 link rel=”stylesheet”

Imports the CSS file.

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

3.1.4 Scripts

Imports the main Javascript file, responsible for the table and UI.

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

Imports the Google API library.

```
8 <script src="https://apis.google.com/js/api.js"></script>
```

Imports other Javascript files, responsible for database management.

```
9  <script src="./googleApiScript.js"></script>
10 <script src="./mysqlScript.js"></script>
11 <script src="./localStorageScript.js"></script>
12 <script src="./imageFirestore.js"></script>
```

3.2 Inputs

Disables autocomplete which remembers past user input by default. `return false` specifies that no `POST` request should be made to the server.

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

3.2.1 Labels

Identifies the purpose of the field to the user, allows the user to select the field by clicking the label. This element is also used by accessibility tools to identify the field.

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

3.2.2 Date

The `date` input type is supported by most modern browsers and provides an intuitive UI for selecting dates. It also includes methods for converting or verifying the `Date` object.

```
13 <input id="date" name="date" type="date" placeholder="yyyy-mm-dd"/>
```

3.2.3 Text

The `text` input type allows the user to input a string. For numbers, this string has to be parsed in Javascript.

```
28 <input id="account" name="account" list="accountsList" type="text"
   ↪ placeholder="Account Number"/>
```

3.2.4 List

Lists are created using the `select` element, containing `option` elements. Each option has a `value` which is used in Javascript, and `innerText` which is seen by the user.

```
34 <label for="type">Transaction Type:</label><br/>
35 <select id="type" name="type">
36     <option value=""></option>
37     <option value="BUY">BUY</option>
38     <option value="SELL">SELL</option>
39     <option value="!DIVIDEND">DIVIDEND</option>
40     <option value="!INTEREST">INTEREST</option>
41     <option value="!WITHDRAW">WITHDRAW</option>
42     <option value="!DEPOSIT">DEPOSIT</option>
43 </select>
```

3.2.5 File

Files are uploaded using the `file` input type. The `multiple` attribute allows the user to upload multiple files, which are interpreted as an array of files in Javascript.

```
64     <label id="fileUploadLabel" for="fileUpload">Upload file</label>
65     <input id="fileUpload" name="fileUpload" type="file" onchange="
          ↪ fileUploadChanged();" multiple/>
```

3.2.6 Buttons

Buttons with the `submit` type can be used to check that all required sections are complete and highlight them in red. These buttons can also be used to send a `POST` request to a server if desired. The `onclick` attribute specified the function and parameters that should be executed when pressed.

```
70 <button id="add" type="submit" onclick="addTransactionButton();">Add
   ↪ Transaction</button>
```

3.3 Filters

Filter HTML elements are handled exactly the same as their counterparts in the input section. Some fields have two elements to handle a lower and upper bound, but these are handled solely in Javascript.

3.3.1 Tooltip

The `span` element is a generic container. The `title` attribute will display its value as a tool tip when the element is hovered.

```
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/>
```

3.3.2 Checkbox

The input type `checkbox` provides a toggleable input field which can be evaluated as `true` or `false` with Javascript.

```
153 <label for="filterNa">Filter N/A:</label>
154 <input id="filterNA" name="filterNA" type="checkbox"/>
```

3.4 Options

The options section uses buttons, text inputs, a file input, and drop down menus, which are decribed in the inputs section. The special handling of these elements is done in Javascript.

3.5 Table

3.6 frozenColumns

Cells in columns that are meant to be always visible are marked with a `frozenColumnx` class, where `x` is the column number. CSS is used to keep the column in place when scrolling.

```
<th class="frozenColumn1">
    <section>
        Transaction ID
    </section>
```

3.7 sort buttons

Sorting is done using buttons with an `onclick` attribute that calls a function `sortTable()`. The parameters passed are the column index and a boolean value indicating whether the column should be sorted in ascending or descending order.

```
<section class="sort">
    <button type="button" onclick="sortTable(0, true)">^</button>
    <button type="button" onclick="sortTable(0, false)">v</button>
</section>
```

3.7.1 tbody

The main table body is initially empty. Rows are managed by Javascript and it is marked with a unique id for this purpose.

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

3.8 Firebase scripts

These scripts are taken directly from the firebase documentation. They are required for firebase and its components to function. The `firebase-app.js` script is the main script and is required for all firebase features. The next three scripts are required for collecting analytics data, the realtime database, and firestore, respectively.

The configuration contains API keys and project information required to identify the app. The key is not secret, though it is unique to the project. As it is easily obtained by users of the app, it is strongly recommended to whitelist your domain in the project settings.

Unlike the other scripts, the firebase script is declared at the bottom, as it requires that the SDKs have loaded first.

```
<!-- 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>
```

```
329
330 <script>
331 // Your web app's Firebase configuration
332 var firebaseConfig = {
333     apiKey: "AIzaSyAmZLFZHDAB9evhvNunxOe5GxXRd_OizmU",
334     authDomain: "financial-transactions-6f065.firebaseapp.com",
335     databaseURL: "https://financial-transactions-6f065.firebaseio.com",
336     projectId: "financial-transactions-6f065",
337     storageBucket: "financial-transactions-6f065.appspot.com",
338     messagingSenderId: "82206982479",
339     appId: "1:82206982479:web:8937bbd1bd4fb6022b053a",
340     measurementId: "G-0564DT8RNQ"
341 };
342 // Initialize Firebase
343 firebase.initializeApp(firebaseConfig);
344 firebase.analytics();
345
346 var database = firebase.database();
347 var firestore = firebase.firestore();
348 </script>
349
350 <script src="./firebaseScript.js"></script>
```

4 Main Javascript

This file handles the UI and general functions required to bridge the front end with the databases.

4.1 formattedStringToNumber()

Removes leading dollar sign if present. Removes all commas. Converts string to a number datatype.

```
1  function formattedStringToNumber(numberAsString) {
2      var number;
3
4      if(numberAsString[0] == '$') {
5          numberAsString = numberAsString.substr(1);
6      }
7
8      number = Number(numberAsString.replace(/,/g, ''));
9
10     return number;
11 }
```

4.2 numberToFormattedString()

Converts number to string datatype. Inserts a comma between every consecutive group of 3 characters.

```
13 function numberToFormattedString(number) {
14     var numberAsString;
15
16     numberAsString = String(number).replace(/\B(?=(\d{3})+(?!\d))/g, ",");
17
18     return numberAsString;
19 }
```

4.3 getData()

Gets values from input fields and performs minor formatting changes. Calls the `validate()` function to have the data verified. If the data is valid, more formatting changes are performed, including adding dollar signs and converting the date to a string. The function returns an array of the data if valid, `false` otherwise.

```
21 function getData() {
22     var date = document.getElementById("date");
23     var account = document.getElementById("account").value;
24     var type = document.getElementById("type").value;
25     var security = document.getElementById("security").value;
26     var amount = document.getElementById("amount").value;
27     var dAmount = document.getElementById("dAmount").value;
28
29     security = security.toUpperCase();
30
31     amount = formattedStringToNumber(amount);
```

```
32
33      dAmount = formattedStringToNumber(dAmount);
34
35      if(validate(date, account, type, security, amount, dAmount)) {
36          var costBasis = '$' + numberToFormattedString(calculateCostBasis(
                ↪ amount, dAmount));
37          date = date.value;
38
39          amount = numberToFormattedString(amount);
40          dAmount = '$' + numberToFormattedString(dAmount.toFixed(2));
41
42          return [ date, account, type, security, amount, dAmount, costBasis
                ↪ ];
43      }
44      else return false;
45  }
```

4.4 validate()

Calls functions to validate all input fields. If any return `false`, the `validate()` function returns `false`. If none of the checks fail, the function returns `true`.

```
47  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  }
```

4.4.1 Check empty

Checks if the input field is an empty string. If so, alerts the user with an error message and returns `false`. Otherwise, returns `true`.

```
80  function validateAccount(account) {
81      if(account == '') {
82          alert('Error: Missing Account Number');
83          return false;
84      }
85
86      return true;
87  }
```

4.4.2 Check NaN

Uses the built-in `isNaN()` function to check that a number is valid.

```
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 }
```

4.4.3 Check date

Gets the current date and stores it in the variable `realDate`. Checks the validity of the date input using the built-in `date.checkValidity()`. Compares the date input to the current date to ensure that the date input is not in the future.

```
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 }
```

4.5 generateId()

Generates an ID of length `idLength` by selecting a random character from the character set using a loop. Checks this ID against all other IDs in the table, if none match, the function returns the ID. If any match, the ID is not unique and the function attempts to generate another until it reaches a unique ID.

```
function generateId() {
    var id = '';
    var idLength = 6;

    var characters = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789';
    var charactersLength = characters.length;

    var unique = false;

    while(!unique) {
        for(var i = 0; i < idLength; i++) {
            id += characters.charAt(Math.floor(Math.random() *
                ↪ charactersLength));
        }

        unique = true;
        for(var i = 0; i < document.getElementsByClassName('idCell').
            ↪ length; i++) {
            if(document.getElementsByClassName('idCell')[i].innerText ==
                ↪ id) {
                unique = false;
                break;
            }
        }
    }
    return id;
}
```

4.6 calculateCostBasis()

Divides the dollar amount by the amound and sets precision to 2 decimal places.

```
function calculateCostBasis(amount, dAmount) {
    costBasis = (dAmount / amount).toFixed(2);
    return costBasis;
}
```

4.7 addTransaction()

Takes an array as the argument, meant to contain all data necessary to create a row. Constructs remaining cells in the row, such as the actions column, and formats the files column. Adds the new row to the table and adds cells, modifying classes where necessary.

```
function addTransaction(data) {
    var staging = data;
    var tableBody = document.getElementById('tableBody');
    var newRow = tableBody.insertRow(0);
    newRow.classList += "bodyRow";

```

```
    var actionsContent = "<button type='button' onclick='editRow(this)'>
        ↪ Edit</button><button type='button' onclick='deleteRow(this)'>
        ↪ Delete</button>";
    var fileContent = '<table><tbody>';
    if(data.length > 8) {
        for(let i = 0; i < data[8].length; i++) {
            fileContent += "<tr><td><a onclick='downloadFile(`" + data[8][
                ↪ i][0] + "`);' href='javascript:void(0);'>" + data[8][i
                ↪ ][1] + "</a></td>";
            fileContent += "<td><button type='button' onclick='
                ↪ removeFileFromTable(`" + data[8][i][0] + "`, this);'>-</
                ↪ button></td></tr>";
        }
    }
    fileContent += '</tbody></table><input type="file" onchange="addFile(
        ↪ this, 0);" multiple/>';
    staging[8] = fileContent;
    staging[9] = actionsContent;

    var calculateCostBasis = true;
    if(data[3][0] == '!') {
        calculateCostBasis = false;
        data[3] = data[3].substr(1);
    }

    for(var i = 0; i < 10; i++) {
        var newCell = newRow.insertCell(i);
        var idShowing = (document.getElementById('toggleId').innerText ==
            ↪ "Hide Transaction ID");

        newCell.innerHTML = data[i];

        if(i == 0) {
            if(idShowing)
                newCell.classList = "idCell frozenColumn1";
            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";
        }
        */
```

```
217
218         if(i == 7 && !calculateCostBasis) {
219             newCell.innerHTML = "N/A";
220         }
221     }
222 }
```

4.8 fileIdGenerator()

Creates and arbitrary, random file ID, large enough that it is extremely unlikely to generate two identical IDs. This is done as it is less feasible to check existing databases for matching IDs.

```
224 function fileIdGenerator() {
225     return Math.floor(Math.random() *
        ↪ 10000000000000000000000000000000000000000).toString(36);
226 }
```

4.9 addTransactionButton()

Gets data by calling the `getData()` function. Adds the transaction ID to the `data` array. Creates a new empty array for storing file data, calls the `uploadFile()` function to continue the process.

```
228 function addTransactionButton() {
229     var data = getData();
230     if(data) {
231         var id = generateId();
232         data.unshift(id);
233
234         fileList = new Array()
235         uploadFile(data, addTransactionWithFileName, fileList, 0);
236         clearInput(false);
237     }
238 }
```

4.10 addTransactionWithFileName()

Takes the row data as an argument and calls `addTransaction()` to finalise the process. Logs the data in the Javascript console. Calls the `loadDataLists()` function to update the list of securities and accounts lists.

```
240 function addTransactionWithFileName(data) {
241     addTransaction(data);
242     console.log(data);
243     loadDataLists();
244 }
```

4.11 deleteRow()

Gets the row of the delete button. Removes files from the local database which were referenced from this row (using the unique file ID). Removes the row. Resets the buttons in the input section if necessary (i.e. if the deleted row was being edited, the editing actions must be hidden and the add transaction button must be restored).

```
246 function deleteRow(button) {
247     var row = button.parentElement.parentElement;
248 
249     var fileRows = row.getElementsByTagName('td')[8].getElementsByTagName
        ↪ ('table')[0].getElementsByTagName('tr');
250     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 }
```

4.12 editRow()

The function checks if there is already a row with the `editing` class. If so, it removes this class from the old row.

The function then sets the current row to have the `editing` class. All the input fields are set to use values from the row.

In the case of type, the function first tries to apply the display value of the type. If this fails, the function tries to add an exclamation mark to the front to account for the indication of types without cost basis.

The add transaction button is hidden and set to a regular button so that the enter key no longer activates it. The save and discard buttons are unhidden, and save is set to the submit type button.

The file upload is cleared and the content of the file upload label is set depending on the number of files from the row.

```
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 }
```

4.13 saveChanges()

The function attempts to get data from the input section by calling `getData()`, which also validates this data.

The row to modify is determined by checking for the `editing` class. An array is created from the cells in this row.

If the first character of the type is and exclamation mark, this character is removed before being inserted into the row, and the cost basis is set to N/A.

Each element in the data array is moved into the appropriate cell in the row.

The add transaction button is restored, and the other buttons are hidden.

If the file upload was changed (i.e. a file was uploaded or the element was cleared), the `uploadFile()` function is called to get the new files if they exist. If no changes were made, the file input is cleared.

The date in the input is reset and all other fields are cleared. Lastly, the function loads existing accounts and securities for the autofill feature.

```
301 function saveChanges() {
```

```
302     data = getData();
303     if(data) {
304         rowToEdit = document.getElementsByClassName('editing')[0];
305         cellsToEdit = rowToEdit.getElementsByTagName('td');
306 
307         if(data[2][0] == '!') {
308             data[2] = data[2].substr(1);
309             data[6] = "N/A";
310         }
311 
312         for(var i = 0; i < data.length; i++) {
313             cellsToEdit[i + 1].innerHTML = data[i];
314         }
315         rowToEdit.classList = "bodyRow";
316 
317         document.getElementById('add').removeAttribute('hidden');
318         document.getElementById('save').setAttribute('hidden', true);
319         document.getElementById('discard').setAttribute('hidden', true);
320 
321         document.getElementById('add').setAttribute('type','submit');
322         document.getElementById('save').setAttribute('type','button');
323 
324         if(fileEditted) {
325             uploadFile([cellsToEdit[8]], updateExistingFileName, new Array
                ↪ (), 0);
326         }
327         else {
328             removeFileUpload();
329         }
330 
331         resetDate();
332         clearInput(true);
333         loadDataLists();
334     }
335 }
```

4.14 discardChanges()

The `editing` class is removed from the row currently being edited.

The add transaction button is restored, and the other buttons are hidden.

The date is reset, input fields are cleared, and any files in the file upload are removed.

```
337 function discardChanges() {
338     document.getElementsByClassName('editing')[0].classList = "bodyRow";
339 
340     document.getElementById('add').removeAttribute('hidden');
341     document.getElementById('save').setAttribute('hidden', true);
342     document.getElementById('discard').setAttribute('hidden', true);
343 
344     document.getElementById('add').setAttribute('type','submit');
345     document.getElementById('save').setAttribute('type','button');
```

```
346
347     resetDate();
348     clearInput(true);
349     removeFileUpload();
350 }
```

4.15 sortTable()

```
352 function sortTable(column, ascending) {
353     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 }
```

4.16 resetDate()

```
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 }
```

4.17 validateFilters()

```
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 }
```

4.17.1 always true

```
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 }
```

4.17.2 date range

```
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 }
```

4.17.3 amount range

```
function validateAmountRange(min, max) {
    if(isNaN(Number(min))) {
        alert('Error: Min Amount is NaN');
        return false;
    }

    if(isNaN(Number(max))) {
        alert('Error: Max Amount is NaN');
        return false;
    }

    if(Number(min) > Number(max) && min != '' && max != '') {
        alert('Error: Invalid Amount Range');
        return false;
    }

    return true;
}
```

4.17.4 generic ranges

```
function validateFilterId(id) {
    return true;
}
```

4.18 stringFilter()

```
function stringFilter(filtertext, tableitem) {
    filters = filtertext.split(" && ");

    for(var i = 0; i < filters.length; i++) {
        filterORs = filters[i].split(" || ");
        var meetsCriteria = false;

        for(var ii = 0; ii < filterORs.length; ii++) {
            if(filterORs[ii][0] == "!" && !tableitem.toUpperCase().
               ↪ includes(filterORs[ii].toUpperCase().substr(1)))
               ↪ meetsCriteria = true;
            if(filterORs[ii][0] != "!" && tableitem.toUpperCase().includes
               ↪ (filterORs[ii].toUpperCase())) meetsCriteria = true;
        }

        if(!meetsCriteria) return false;
    }

    return true;
}
```

4.19 applyFilter()

```
512 function applyFilter() {
513     unfilterAll();
514 
515     rows = document.getElementsByClassName('bodyRow');
516 
517     filterId = document.getElementById('filterId').value;
518     startDate = document.getElementById('startDate');
519     endDate = document.getElementById('endDate');
520     filterAccount = document.getElementById('filterAccount').value;
521     filterType = document.getElementById('filterType').value;
522     filterSecurity = document.getElementById('filterSecurity').value;
523     lowAmount = document.getElementById('lowAmount').value;
524     highAmount = document.getElementById('highAmount').value;
525     lowDAmount = document.getElementById('lowDAmount').value;
526     highDAmount = document.getElementById('highDAmount').value;
527     lowCostBasis = document.getElementById('lowCostBasis').value;
528     highCostBasis = document.getElementById('highCostBasis').value;
529 
530     if(lowDAmount[0] == '$') lowDAmount = lowDAmount.substr(1);
531     if(highDAmount[0] == '$') highAmount = highDAmount.substr(1);
532     if(lowCostBasis[0] == '$') lowCostBasis = lowCostBasis.substr(1);
533     if(highCostBasis[0] == '$') highCostBasis = highCostBasis.substr(1);
534 
535     if(validateFilters(filterId, startDate, endDate, filterAccount,
        ↪ filterType, filterSecurity, lowAmount, highAmount, lowDAmount,
        ↪ highDAmount, lowCostBasis, highCostBasis)) {
536         for(var i = 0; i < rows.length; i++) {
537             cells = rows[i].getElementsByTagName('td');
538             var hide = false;
539 
540             if(filterId != '' && !stringFilter(filterId,cells[0].innerText
                ↪ ))
541                 hide = true;
542 
543             if(startDate.value != '' && startDate.value > cells[1].
                ↪ innerText)
544                 hide = true;
545 
546             if(endDate.value != '' && endDate.value < cells[1].innerText)
547                 hide = true;
548 
549             if(filterAccount != '' && !stringFilter(filterAccount, cells
                ↪ [2].innerText))
550                 hide = true;
551 
552             if(filterType != '' && filterType != cells[3].innerText)
553                 hide = true;
554 
555             if(filterSecurity != '' && !stringFilter(filterSecurity, cells
                ↪ [4].innerText))
556                 hide = true;
557 
```

```
            if(lowAmount != '' && Number(lowAmount) >
               ↪ formattedStringToNumber(cells[5].innerText))
                hide = true;

            if(highAmount != '' && Number(highAmount) <
               ↪ formattedStringToNumber(cells[5].innerText))
                hide = true;

            if(lowDAmount != '' && Number(lowDAmount) >
               ↪ formattedStringToNumber(cells[6].innerText.substr(1)))
                hide = true;

            if(highDAmount != '' && Number(highDAmount) <
               ↪ formattedStringToNumber(cells[6].innerText.substr(1)))
                hide = true;

            if(lowCostBasis != '' && Number(lowCostBasis) >
               ↪ formattedStringToNumber(cells[7].innerText.substr(1)))
                hide = true;

            if(highCostBasis != '' && Number(highCostBasis) <
               ↪ formattedStringToNumber(cells[7].innerText.substr(1)))
                hide = true;

            if(filterNA.checked && cells[7].innerText == "N/A")
                hide = true;

            if(hide)
                rows[i].setAttribute('hidden', true);
        }
    }
}
```

4.20 clearFilter()

```
function clearFilter() {
    unfilterAll();

    var fields = document.getElementsByClassName('filterField');

    for(var i = 0; i < fields.length; i++) {
        fields[i].value = '';
    }

    document.getElementById('filterNA').checked = false;
}
```

4.21 unfilterAll()

```
function unfilterAll() {
```

```
598     rows = document.getElementsByClassName('bodyRow');
599
600     for(var i = 0; i < rows.length; i++) {
601         rows[i].removeAttribute('hidden');
602     }
603 }
```

4.22 toggleID()

```
605 function toggleID() {
606     var button = document.getElementById('toggleId');
607     var rows = document.getElementsByTagName('tr');
608     var cells = rows[0].getElementsByTagName('th');
609
610     if(button.innerText == "Hide Transaction ID") {
611         button.innerText = "Show Transaction ID";
612
613         cells[0].setAttribute('hidden', true);
614         cells[0].classList = "";
615         cells[1].classList = "frozenColumn1";
616         //cells[2].classList = "frozenColumn2";
617         for(var i = 1; i < rows.length; i++) {
618             cells = rows[i].getElementsByTagName('td');
619
620             cells[0].setAttribute('hidden', true);
621             cells[0].classList = "idCell";
622             cells[1].classList = "frozenColumn1";
623             //cells[2].classList = "frozenColumn2";
624         }
625     }
626     else {
627         button.innerText = "Hide Transaction ID";
628
629         cells[0].removeAttribute('hidden');
630         cells[0].classList = "frozenColumn1";
631         cells[1].classList = "frozenColumn2";
632         //cells[2].classList = "frozenColumn3";
633         for(var i = 1; i < rows.length; i++) {
634             cells = rows[i].getElementsByTagName('td');
635
636             cells[0].removeAttribute('hidden');
637             cells[0].classList = "idCell frozenColumn1";
638             cells[1].classList = "frozenColumn2";
639             //cells[2].classList = "frozenColumn3";
640         }
641     }
642 }
```

4.23 readFile()

```
function readFile(fileIn){
    if(fileIn.files && fileIn.files[0]) {
        var reader = new FileReader();
        reader.onload = function (e) {
            var output = e.target.result;
            document.getElementById('typesArray').value = output;
        };
        reader.readAsText(fileIn.files[0]);
    }
}
```

4.24 saveFile()

```
function saveFile() {
  var element = document.createElement('a');
  element.setAttribute('href', 'data:text/plain;charset=utf-8,' +
     ↪ encodeURIComponent(document.getElementById('typesArray').value));
  element.setAttribute('download', 'transaction-types.csv');

  element.style.display = 'none';
  document.body.appendChild(element);

  element.click();

  document.body.removeChild(element);
}
```

4.25 applyTypes()

```
function applyTypes() {
    var typesArray = document.getElementById('typesArray').value.split
       ↪ (',');
    setTransactionTypesList(typesArray);
}
```

4.26 editTypes()

```
function editTypes() {
    document.getElementById('typesArray').value = readCurrentTypes().join
       ↪ (',');
}
```

4.27 setTransactionTypes()

```
function setTransactionTypesList(typesArray) {
    var type = document.getElementById('type');
    var filterType = document.getElementById('filterType');

    type.innerHTML = '<option value=""></option>';
    filterType.innerHTML = '<option value=""></option>';

    for(var i = 0; i < typesArray.length; i++) {
        var typeAsText = typesArray[i];
        if(typesArray[i][0] == '!') typeAsText = typesArray[i].substr(1);

        type.innerHTML += '<option value="' + typesArray[i] + '">' +
            ↪ typeAsText + '</option>';
        filterType.innerHTML += '<option value="' + typeAsText + '">' +
            ↪ typeAsText + '</option>';
    }
}
```

4.28 toggleSection()

```
function toggleSection(button) {
    var form = button.parentElement.parentElement.getElementsByTagName('
        ↪ form')[0];

    if(button.innerText == "Hide") {
        form.setAttribute("hidden",true);
        button.innerText = "Show";
    }
    else {
        form.removeAttribute("hidden");
        button.innerText = "Hide";
    }
}
```

4.29 loadDataLists()

```
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;
```

```
717
718         if(!accounts.includes(tableAccount)) accounts.push(tableAccount);
719         if(!securities.includes(tableSecurity)) securities.push(
                ↪ tableSecurity);
720     }
721
722     accountsList.innerHTML = '';
723     securitiesList.innerHTML = '';
724
725     for(var i = 0; i < accounts.length; i++) {
726         accountsList.innerHTML += '<option value="' + accounts[i] + '"/>';
727     }
728
729     for(var i = 0; i < securities.length; i++) {
730         securitiesList.innerHTML += '<option value="' + securities[i] +
                ↪ '"/>';
731     }
732 }
```

4.30 readCurrentTypes()

```
734 function readCurrentTypes() {
735     var types = document.getElementById('type').getElementsByTagName('
            ↪ option');
736     var currentTypes = [];
737
738     for(var i = 1; i < types.length; i++) {
739         currentTypes.push(types[i].value);
740     }
741
742     return currentTypes;
743 }
```

4.31 tableToArrays()

```
745 function tableToArrays() {
746     var rows = document.getElementsByClassName('bodyRow');
747     var data = new Array();
748     data.push(["Transaction Id", "Date", "Account Number", "Transaction
            ↪ Type", "Security", "Amount", "$ Amount", "Cost Basis", "Files"])
            ↪ ;
749
750     for(var i = 0; i < rows.length; i++) {
751         var cells = rows[i].getElementsByTagName('td');
752         var cellData = new Array();
753
754         for(var j = 0; j < 8; j++) {
755             cellData.push(cells[j].innerText);
756         }
757         cellData.push(getFileNamesIds(cells[8]));
```

```
758             data.push(cellData);
759         }
760 
761         console.log(data);
762         return data;
763 }
```

4.32 arraysToTable()

```
765 function arraysToTable(dataArr) {
766         while(document.getElementsByClassName('bodyRow').length > 0) {
767             document.getElementById("tableBody").removeChild(document.
                    ↪ getElementsByClassName('bodyRow')[0]);
768         }
769 
770         document.getElementById('add').removeAttribute('hidden');
771         document.getElementById('save').setAttribute('hidden', true);
772         document.getElementById('discard').setAttribute('hidden', true);
773 
774         document.getElementById('add').setAttribute('type', 'submit');
775         document.getElementById('save').setAttribute('type', 'button');
776 
777         while(dataArr.length > 0) {
778             let data = dataArr[dataArr.length - 1];
779             let files = parseFileNamesIds(data[8]);
780             data.pop();
781             if(files.length > 0) {
782                 data.push(files);
783             }
784             addTransaction(data);
785             dataArr.pop();
786         }
787         loadDataLists();
788 }
```

4.33 window.onload = function()

```
790 window.onload = function() {
791         resetDate();
792         initDb();
793 }
```

5 firebaseScript.js

5.1 clearFirebase()

5.2 writeToFirebase()

5.3 readFromFirebase()

5.4 clearFirestore()

5.5 writeToFirestore()

5.6 readFromFirestore()

6 googleApiScript.js

6.1 Global Variables

6.2 loadSheetData()

6.3 getNewSheetData()

6.4 populateSheetSelector()

6.5 getNewTabData()

6.6 populateTabSelector()

6.7 getAllUserSheets()

6.8 getTabsOfSheet()

6.9 authenticate()

6.10 loadClientSheets()

6.11 loadClient()

6.12 readGoogleSheetDB()

6.13 readGoogleTypes()

6.14 writeGoogleSheetDB()

6.15 setGoogleRows()

6.16 clearGoogleRow()

6.17 writeGoogleDB()

6.18 gapi.load()

7 imageFirestore.js

7.1 writeImagesToFirestore()

7.2 readImagesFromFirestore()

7.3 getFileNamesIds()

7.4 parseFileNamesIds()

7.5 clearIndexedDb()

8 localStorageScript.js

8.1 Global Variables

8.2 initDb()

8.3 fileUploadChanged()

8.4 uploadFile()

8.5 addFile()

8.6 updateExistingFileName()

8.7 removeFileFromTable()

8.8 deleteFileFromIndexedDB()

8.9 removeFileUpload()

8.10 downloadFile()

8.11 window.onbeforeunload = function()

9 mysqlScript.js

9.1 writeToMySQL()

9.2 readFromMySQL()

10 CSS

10.1 Vertical Scrolling Table

10.2 Horizontal Scrolling on Overflow

10.3 Miscellaneous

10.3.1 Sort buttons

10.3.2 Editing highlight

10.3.3 Table borders