Server Call.gs

Backend code for calls from the HTML page JavaScript. Follow the Function hyperlinks to see more information about the function.

|  |  |  |
| --- | --- | --- |
| Function | Return type | Brief description |
| a [Global Variables](#GlobalVariables) | **varied** | Sets the current data range as the [active range](https://developers.google.com/apps-script/reference/spreadsheet/selection.html#getActiveRange()), sets current header range, sets row length, sets column length. |
| [a ColumnID\_(columnValue)](#ColumnID) | **Integer** | Edits the form values for Image and Location. |
| [afetchFiltersWithQuery(currentDB)](#openLaundryApp) | **Object[][]** | Gets all Columns and Values within eligible data for a category type from the previously saved query. |
| [a FetchItems(category, type)](#onOpen) | **Object[][]** | Gets all items with filter from the Spreadsheet Data |
| [a filteritems(formdata)](#include) | **Object[][]** | Filters Query with parameters from form. |
| [a GetCacheQuery\_()](#changeValueOnSubmit) | **Object[][]** | Returns the last [FetchItems()](#FetchItems) results |
| [a getClothing()](#changeValueOnSubmit) | **Array[][]** | Return Clothing types as populated in the second sheet of this document. |
| [a getColors()](#changeValueOnSubmit) | **Array[][]** | Return Color options as populated in the third sheet of this document. |
| a getColumnValue(ID, category) | **String** | Returns a single string for the value of an item within the column *category*. |
| a getFullRow(ID) | **Array[]** | Returns the full row array of a given *ID* |
| a NODuplicates(data) | **Array[]** | Used with [Array.map()](https://www.w3schools.com/jsref/jsref_map.asp) to traverse an array and return unique values. |
| a ServerRemoveFilters() | **Object** | If a filter is applied, this will return the original cached query. |
| a SetCacheQuery\_(data) | **Set Cache** | Sets the current query in cache for faster calls. |
| a updateLaundryItems(value) | **Boolean** | Updates all location fields of the items in the LaundryApplication |
| a updateLocation(id, value) | **Boolean** | Updates the location field of a single *ID* |
|  |  |  |

Global Variables

* I established before how to call for data on a **Google Sheet.** In this code file we are setting the **Google Sheet** we intend to use for all calls at the top of the page.
* Making this link once and outside of a function allows us to use it over and over again in future code on this page.
* The variables I am fetching are the document I intend to use (*ss*), the sheet in that document (*sheet*), all of the rows that have information in them (*rangeData*), the last row and column (*lastRow*& *lastColumn*), the range that has all the items (*searchRange*), the range for the column headers (*header*), and lastly a blank array for saving queries.

Code

Server Calls.gs

var ss = SpreadsheetApp.getActiveSpreadsheet();

var sheet = ss.getSheetByName("Form Responses 1");

var rangeData = sheet.getDataRange();

var lastColumn = rangeData.getLastColumn();

var lastRow = rangeData.getLastRow();

var searchRange = sheet.getRange(2,1, lastRow-1, lastColumn);

var header = sheet.getRange(1,1, 1, lastColumn);

var currentQuery = [];

Parameters

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| SheetName | String | If different, please enter the name of the sheet in which your Form Responses are sent to. |

ColumnID\_( *columnValue*)

* Using the [Global Variable](#GlobalVariables) header, loop through this range until you find the contents that match the *columnValue* that has been passed to the function.
* Return the integer representing that column value.

Code

Server Calls.gs

function ColumnID\_(columnValue) {

var headerValues = header.getValues();

try {

for (var i = 0; i < lastColumn; i++) {

if (headerValues[0][i] == columnValue) {

return i;

break;

}

}

} catch (e) {

Logger.log("Column ID() ERROR:" + e.message);

}  
}

Parameters

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| columnValue | String | The column header title as it appears on the Google Sheet. |

Return

[Integer](https://developers.google.com/apps-script/reference/base/menu) — A numerical value

fetchFiltersWithQuery(*currentDB*)

* A *currentDB* argument is passed to this function.
  + We slice this array so that we are not making edits to the original within this code
* The *currentDB* is a multidimensional array so it must be traversed in two loops:
  + The first loop is traversing the row.
  + The second loop is traversing the columns in that row. The code is taking down the name of each of the column headers that are in each item. (1)
* After saving each of these values it sorts them alphabetically with the previously save column values. Then runs them through the [NoDuplicates](#NoDuplicates)() function to give us an array of just unique column headers.(2)
* Now a new loop is started with the *arrayofValues* representing unique column values.
  + The filters do not need to include ID, Picture, Articles, Timestamp, or other variables that are unique to that item. Therefore if the column is equal to these they are skipped for the filters.(3)
  + Next the column name is added as an entry in an OBJECT.
  + Using the current key, a query is done on *currentDB* to return all the values associated with that key. *keyValues* is sorted to better tally how many of each value is in the database. (4)
* Another loop is done on *keyValues* this loop will count all items consecutively if the value is the same as the previous. If it isn’t then that *value* and *count* will be written to an array and the value can start its tally.
* Once every *value* in a column has been queried that array will be pushed to the object where it will be associated with the correct *column*.
* After every *column* has been queried the function will return an Array to the HTML JavaScript to decrypt and populate a sidebar.

Code

Server Calls.gs

function fetchFiltersWithQuery(currentDB){

var allData = currentDB.slice(0);

var returnFilters = [];

try{

var arrayOfValues = [];

for( var op in allData){

var RowOP = allData[op];

for( var column in RowOP){

arrayOfValues.push(column);

}

}

arrayOfValues.sort();

arrayOfValues= NoDuplicates(arrayOfValues);

for( var x in arrayOfValues){

var key = arrayOfValues[x];

if(key != "ID" && key != "Item Picture" && key != "Articles" && key != "Timestamp" && key != "Default Location" && key != "Updated" && key != "Who"){

var KeyCount ={};

KeyCount["FilterName"] = key;

var keyArray = allData.map(function(itemX){return itemX[key];});

keyArray.sort();

var current = keyArray[0]; var count =0;

for( var y in keyArray){

if(keyArray[y] == current){ count++;}

else{

KeyCount[current] = count;

current = keyArray[y];

count = 1;

}

if(y == keyArray.length-1){

KeyCount[current] = count;

}

}

returnFilters.push(KeyCount);

}

}

return JSON.stringify(returnFilters);

}

catch(e){

Logger.log("fetchFiltersWithQuery() ERROR:" + e.message);

}

}

Parameters

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| currentDB | Object[][] | A database of all the items in the last fetchItems() call. |

Return

JSON.stringify(Object[][]) — an  Object[][] that must be transferred to an HTML page which cannot read complicated object types.

FetchItems(*category, type*)

* This code will be called when a category is selected from the top menu but can be used for any category and value in the respective column.
* We accept two incoming arguments for this function *category*and *value*. The *category*is the column title and the *value*is well...the value in that column.
* We pull all the header and body values from the Range defined in our [global variables.](#GlobalVariables) This allows for the call to be faster as we only pull all the values inside of a function rather than globally.
* Then we search using the *[ColumnID\_()](#ColumnID)*function to figure out which column number to look in for this data.
* Next we will be looping through the data to find the values we want.
  + The data is stored in a multi dimensional array which means we have two fields to loop through.
  + We first loop through the row and determine if the *category*we are looking in has the *value* we are looking for. If the value we are looking for is "All" every row with data in this field will be returned. (1)
  + If the row matches what we need, we take only the columns that have data in them as there are plenty of sizing restrictions that don't apply to every item.
* Notice how we are storing the values we intend to output, in an object item. (2)
  + An object allows us to make key value pairs. Therefore the data that is being stored can be called with the header that it is filed under. Instead of calling for *[ColumnID\_()](#ColumnID)*now we can read the key and determine if it is the field we want to look in.
* After storing the items that meet our criteria, we cache the results using *[SetCacheQuery()](#SetCacheQuery)*[.](#SetCacheQuery) This will allow us to make a quick call back to this information when we wish to filter the results.
* Lastly we return the results with the *JSON.stringfy()*as an object can not be transferred to the HTML page as is.

Code

Server Calls.gs

function fetchItems(category, type){

var allData = searchRange.getValues();

var headerData = header.getValues();

var column\_indx= ColumnID\_(category);

var returnItems = [];

try{

for(var i =0; i<lastRow-1; i++){

if(allData[i][column\_indx] == type || type=="All"){

var obj = {};

for(var col=0; col<lastColumn; col++){

if(allData[i][col] != ""){

var header\_name = headerData[0][col];

obj[header\_name]=allData[i][col];

}

}

returnItems.push(obj);

}

}

SetCacheQuery\_(returnItems);

currentQuery = returnItems.slice(0);

return JSON.stringify(returnItems);

}

catch(e){

Logger.log("fetchItems() ERROR:" + e.message);

}

}

Parameters

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| category | String | The Column Header to look for the *type* in. |
| type | String | The value to be queried for. |

Return

JSON.stringify(Object[][]) — an  Object[][] that must be transferred to an HTML page which cannot read complicated object types.

filterItems(*formdata*)

* This code will be called when a sidebar filter form is submitted.
* The parameters passed to this function is a string of form values relating to *columns* and *values*.
* The code uses the previously cache query from [GetCacheQuery()](#GetCacheQuery) to speed up the filtering as these items are the ones already displayed in the interface.
* Using the *map()* and [*NoDupllicates()*](#NoDuplicates) function, the *formdata* variable is consolidated into an array of unique column names. If the user chooses more than one option in a filter they will be returned as two different filters. This condenses so items that only fit into one of these colors will still be displayed. (1)
* Next the code will be looping through the data to filter the values called for.
  + If the item has the same value in the column that is being filtered it receives a point (2).
  + After looping through all the columns, if this item has the same amount of points filters the item passes and is added to the return Object. (3)

Code

Server Calls.gs

function filterItems(formdata){

var Qry = JSON.parse(GetCacheQuery\_());

var returnItems = [];

try{

var Categories = formdata.map(function(col){return col.name;});

Categories= NoDuplicates(Categories);

for(var i in Qry){

var item = Qry[i];

var itemPasses = 0;

for(var j in formdata){

var column = formdata[j].name;

if(item[column] == formdata[j].value){

itemPasses++;

}

if(itemPasses == Categories.length){

Logger.log("This Item Passes: " +item.ID);

returnItems.push(item);

}

}

}

return JSON.stringify(returnItems);

}

catch(e){

Logger.log("filterItems() ERROR:" + e.message);

}  
}

Parameters

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| formdata | Object | The data from the form submission |

Return

JSON.stringify(Object[][]) — an  Object[][] that must be transferred to an HTML page which cannot read complicated object types.

GetCacheQuery\_()

A hidden function to return the  [Cache](https://developers.google.com/apps-script/reference/html/html-output.html) object of the previously ran database query.

Code

Server Calls.gs

function GetCacheQuery\_(){

var cache = CacheService.getScriptCache();

var cached = cache.get("currentQuery");

if (cached != null) {

return cached;

}  
}

Return

Object[][]) — an  Object[][] that has been cached on the server

getClothing()

Fetches an [Object[][]](https://developers.google.com/apps-script/reference/html/html-output.html) from the file containing Article Clothing options.

Code

Server Calls.gs

function getClothing(){

Logger.log("getClothing() start");

var sheet2 = ss.getSheetByName("Article Types");

var rangeData2 = sheet2.getDataRange();

var lastRow2 = rangeData2.getLastRow();

var Clothing = sheet2.getRange(1,1, lastRow2).getValues();

return Clothing;  
}

Parameters

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| sheetName | String | The name of the sheet holding article types |

Return

Object[][] —

GetColors()

Fetches an [Object[][]](https://developers.google.com/apps-script/reference/html/html-output.html) from the file containing Color Clothing options.

Code

Server Calls.gs

function getColors(){

Logger.log("getColors() start");

var sheet3 = ss.getSheetByName("Colors");

var rangeData3 = sheet3.getDataRange();

var lastRow3 = rangeData3.getLastRow();

var Clothing = sheet3.getRange(1,1, lastRow3).getValues();

return Clothing;  
}

Parameters

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| sheetName | String | The name of the sheet holding the color variants. |

Return

Object[][] —

getColumnValue(*ID, category)*

* Using the [globalVariable](#GlobalVariables) *searchRange* the code pulls the values for the range of cells containing data.
* The parameter *category* is the column label for which the user is requesting information about.
* Using the [*ColumnID*](#ColumnID)*()* function, an integer marking the column location in an array is returned. (1)
* Every row of the *searchRange* is now looped and the first column being the ID field is checked to see if it equals the parameter *ID*.
* If it is the same ID, the value in that row and column as returned by [*ColumnID*](#ColumnID)*()* is returned to the HTML JavaScript. (2)

Server Calls.gs

function getColumnValue(ID, category){

var allData = searchRange.getValues();

var column\_indx= ColumnID\_(category);

try{

for(var i =0; i<lastRow; i++){

if(allData[i][0] == ID){

return allData[i][column\_indx];

}

}

}

catch(e){

Logger.log("getColumnValue() ERROR:" + e.message);

}  
}

Parameters

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| category | String | The Column Header to look for the *type* in. |
| ID | Integer | The unique identifier for this item. |

Return

String-

getFullRow(*ID*)

* Using the [globalVariable](#GlobalVariables) *searchRange* the code pulls the values for the range of cells containing data.
* The parameter *ID*  is the unique identifier for an items data row.
* Every row of the *searchRange* is looped and the first column being the ID field is checked to see if it equals the parameter *ID*.(1)
* If it is the same ID, the entire row array is returned to the HTML JavaScript. (2)

Code

Server Calls.gs

function getFullRow(ID){

var allData = searchRange.getValues();

try{

for(var i =0; i<lastRow; i++){

if(allData[i][0] == ID){

return allData[i];

}

}

}

catch(e){

Logger.log("returnIDvalues() ERROR:" + e.message);

}  
}

Parameters

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| ID | Integer | The unique identifier for the item you wish to get all information from. |

Return

Array[]

NoDuplicates(*data*)

* *NoDuplicates*() takes an array and filters all the items by checking if the item it is currently looping through is in the array at an index before the current one.
* If there is an instance of this value, the value will not be returned a second time.

Code

Server Calls.gs

function NoDuplicates(data){

return data.filter(function(item, index){

return data.indexOf(item) >= index;

});  
}

Parameters

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| data | Object | An Object with key value pairs. |

Return

Array[] — an  Array[] of the values within a Object[][] row

ServerRemoveFilters()

A function available to HTML JavaScript to call and return the  [Cache](https://developers.google.com/apps-script/reference/html/html-output.html) object of the previously ran database query.

Code

Server Calls.gs

function ServerRemoveFilters(){

return GetCacheQuery\_();  
}

Return

None

SetCacheQuery\_(*data*)

A hidden function to set the  [Cache](https://developers.google.com/apps-script/reference/html/html-output.html) object of the previously ran database query.

Code

Server Calls.gs

function SetCacheQuery\_(data) {

var cache = CacheService.getScriptCache();

cache.put("currentQuery", JSON.stringify(data), 1800); // cache for 30 minutes

}

Parameters

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| data | Object[][] | The recent filterItems query, that will be saved for easier filtering and restoring. |

Return

None

updateLaundryItems(*value*)

* The parameters passed to this function are a *value* corresponding to the new location of all the items with a current Location of “Laundry”.
* Using the [*ColumnID*](#ColumnID)*()* function, an integer marking the column location of each of the fields that will be updated are defined.(1)
* A logic test is performed to verify the current status of this item is “Laundry” and that the current user accessing this HTML interface is the original owner. (2)
* If both of these tests are passed, the new information is passed to the proper location within the rows array (3).
  + If the *value* initially entered is “Default” the default location field is queried to provide the right entry in the current location column.
* Update the database by setting the values of the entire row to the new array of values (4).

Code

Server Calls.gs

function updateLaundryItems(value){

try{

var loc\_indx= ColumnID\_("Where are you keeping this article of clothing?");

var ID\_indx= ColumnID\_("ID")+1;

var default\_loc= ColumnID\_("Default Location");

var user\_indx= ColumnID\_("Who");

var updated\_indx= ColumnID\_("Updated");

var IDs = sheet.getRange(1,ID\_indx, lastRow, lastColumn).getValues();

for (var row in IDs){

var newLocation;

if(IDs[row][loc\_indx] == "Laundry" && Session.getActiveUser().getEmail() != ""){

if(value == "Default"){newLocation=IDs[row][default\_loc];}

else{newLocation = value;}

IDs[row][loc\_indx] = newLocation;

IDs[row][user\_indx] = Session.getEffectiveUser().getEmail();

IDs[row][updated\_indx] = Utilities.formatDate(new Date(), "GMT-4", "MM/dd/yyyy HH:mm");

var sheetR = parseInt(row,10)+1;

var outputNew= [];

outputNew.push(IDs[row]);

var updated = sheet.getRange(sheetR,1, 1, lastColumn).setValues(outputNew);

}

}

return true;

}

catch(e){

Logger.log("updateLaundryItems() ERROR:" + e.message);

return false;

}  
}

Parameters

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| value | String | The new location to set all Laundry items to. |

Return

Boolean— A value representing whether this function was a success or not.

updateLocation(*id, value*)

* The parameters passed to this function are a *value* corresponding to the new location of the item with *id*.
* Using the [*ColumnID*](#ColumnID)*()* function, an integer marking the column location of each of the fields that will be updated are defined.(1)
* A logic test is performed to verify the current ID is the same as the variable *id* passed to this function and that the current user accessing this HTML interface is the original owner. (2)
* If both of these tests are passed, the new information is passed to the proper location within the rows array (3).
  + If the *value* initially entered is “Default” the default location field is queried to provide the right entry in the current location column.
* Update the database by setting the values of the entire row to the new array of values (4).

Code

Server Calls.gs

function updateLocation(id, value){

try{

var loc\_indx= ColumnID\_("Where are you keeping this article of clothing?");

var ID\_indx= ColumnID\_("ID")+1;

var user\_indx= ColumnID\_("Who");

var updated\_indx= ColumnID\_("Updated");

var IDs = sheet.getRange(1,ID\_indx, lastRow, 1).getValues();

for (var row in IDs){

if(IDs[row][0] == id && Session.getActiveUser().getEmail() != ""){

var sheetR = parseInt(row,10)+1;

var cell = sheet.getRange(sheetR,1, 1, lastColumn).getValues();

cell[0][loc\_indx] = value;

cell[0][user\_indx] = Session.getEffectiveUser().getEmail();

cell[0][updated\_indx] = Utilities.formatDate(new Date(), "GMT-4", "MM/dd/yyyy HH:mm");

var updated = sheet.getRange(sheetR,1, 1, lastColumn).setValues(cell);

return true;

}

}

}

catch(e){

Logger.log("updateLocation() ERROR:" + e.message);

return false;

}  
}

Parameters

|  |  |  |
| --- | --- | --- |
| Name | Type | Description |
| id | Integer | The unique identifier for a single object to be updated |
| value | String | The new location variable to be set. |

Return

Boolean— A value representing whether this function was a success or not.