SheetsApi.js

Description

This file encapsulates the functions initializing the google client, calling the google sheets api, and some helper methods for parsing the response of api calls.

How to use it

In order to use it successfully, the google api.js file should be imported before the SheetsApi.js and your own js file should be imported after that. Here is an example.

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

When using the SheetsApi to login, the login user should have the permission to read the target spreadsheet.

Constructor for SheetsApi Object

SheetsApi (spreadSheetId, ApiKey, ClientId)

SpreadSheetId is the target spreadsheet id. This can be found in the url of the spreadsheet. The following shows the structure of the URL and where spreadSheetId can be found:

```
https://docs.google.com/spreadsheets/d/spreadsheetId/edit#gid=sheetId
```

The ApiKey and ClientId should be configured in the developer console of google. Instructions can be found on the following websites. The OAuth 2.0 client ID should be set for Web application when creating.

1. If not already done, enable the Google Sheets API and check the quota for your project at https://console.developers.google.com/apis/api/sheets

2. Get access keys for your application. See https://developers.google.com/api-client-library/javascript/start/start-js#get-access-keys-for-your-application

3. For additional information on authentication, see https://developers.google.com/sheets/api/quickstart/js#step_2_set_up_the_sample

After successfully created the OAuth 2.0 client ID, please get into the client ID by clicking its name and add the “http://localhost:8000” into the authorized JavaScript origins.

Functions

Function name	Return type	Parameters	Description
updateSignInStatus(isSignedIn)	void	isSignedIn: a boolean value to show if there is a user has signed in.	The console will log “Ready to make api call” when a user signs in and “Need log in.” when a user signs out
initClient()	void	N/A	This is the function initializing the client of gapi. It is a private function and should not be called.
handleClientLoad()	void	N/A	This is the function to load the gapi. Should be called after created the SheetsApi instance.
handleSignInClick(event)	void	event (can be ignored)	This is the function handle the user’s sign in operation.
handleSignOutClick(event)	void	event (can be ignored)	This is the function handle the user’s sign out operation.
getSpreadsheetInfo()	Promise	N/A	This function will return a promise for getting the information of the spreadsheet.
parseSpreadsheetInfo(response)	Object	Response: the response from getSpreadsheetInfo()	This function will return an object containing the title of the spreadsheet and the sheets information in the spreadsheet
getSheet(inputRange)	Promise	inputRange: a string value specifying the range of the sheet required	This function returns a promise for getting the target sheet
parseSheetValues(response)	String[][]	response: the response from getSheet(inputRange)	This function parses the response from getSheet(inputRange) and returns a 2D array of values in the response
filterByKeyword(values, keyword, columnIndex)	String[][]	values: the input 2D array. keyword: filter keyword. columnIndex: the specific index of column to be filtered, if less than	This function returns the 2D array after filtering the input array by the keyword.

		0, then any column includes the keyword will add the row to the result.	
update(inputRange, inputValues)	Promise	inputRange: the target cells that are to be updated. inputValues: a 2D array of the values.	This function returns a promise for update cells.
parseUpdate(response)	int	response: the response from update(inputRange, inputValues)	This function parses the response from update(inputRange, inputValues) and returns the number of cells updated.
batchAdd(inputRange, inputValues)	Promise	inputRange: for this function normally use only the sheet name. inputValues: a 2D array of the values to be added.	This function returns a promise for batch adding the values to a sheet.
parseBatchAdd(response)	int	response: the response from batchAdd(inputRange, inputValues)	This function parses the response from batchAdd(inputRange, inputValues) and returns the number of rows updated.
parseErrorMessage(reason)	String	reason: the error response from a promise	This function parses the error response and logs the error message to the console.error and returns the error message.
getTableHeaders(sheetName)	Promise	sheetName: the target sheet name	This function will return a promise for getting the headers of the target sheet
parseTableHeaders(response)	String[]	response: the response from getTableHeaders	This function takes the response of getTableHeaders and return an array of headers
getNotationFromColName(headers, colName)	String	headers: the array of headers. colName: the target column name	This function returns the `A1` notation of corresponding column name
getCharFromNum(num)	String	num: the number or index. Starting from 0	This function takes the number and return a corresponding capital char

selectFromTableWhereConditions (header, resturnCols, sheetName)	Promise	headers: an array of the headers of the target sheet. returnCols: an array of the names of columns need to return, passing “*” will return all columns. sheetName: the target sheet name	This function does things like the sql sentence `Select returnCols from sheetName where conditions`
parseSelect(response, returnCols, conditions, returnType)	String[][] or object[]	response: the response from selectFromTableWhereConditions. returnCols: the same as it is in select… conditions: an array of conditions. Each condition is an object with format: {header:"the name of a header", value:"the value to check for"}. returnType: 0 for an array of objects, 1 for a 2D array	This function takes the response from the selectFromTableWhereConditions and return the wanted type of values
insertIntoTableCoValues(headers, sheetName, toInsert)	Promise	headers: an array of the headers of the target sheet. sheetName: the target sheet name. toInsert: This is an array of objects with format [{header1:"value1", header2:"value2"}, {...},...,{...}]	This function does things like the sql sentence ` insert into sheetName values(toInsert)`
parseInsert(response)	int	response: the response from insert…	This function takes the response of insert and returns the updated row number
batchUpdateTable(sheetValues, sheetName, colVal, conditions)	Promise	sheetValues: the whole set of values of the target sheet,	This function does things like thesql sentence `update

		including the headers. sheetName: the target sheet name. colVal: an object of value to be updated with format {header: “value”}. conditions: an array of conditions. Each condition is an object with format: {header:"the name of a header", value:"the value to check for"}.	sheetName set colVal where conditions`
parseBatchUpdate(response)	int	response: the response of batchUpdateTable	This function takes the response of batchUpdateTable and return the updated row number

The Promise can be used as following:

```
let sa = new SheetsApi("example", "example", "example");
sa.handleClientLoad();

function loadData() {
    sa.getSheet("student_info").then(response => {
        let result = sa.parseSheetValues(response);
        //codes of whatever you want to do with the result of response
    }, reason => {
        let message = sa.parseErrorMessage(reason);
        //codes of whatever you want to do with the error message
    });
}
```

The inputRange is in A1 notation. Valid ranges are:

- `Sheet1!A1:B2` refers to the first two cells in the top two rows of Sheet1.
- `Sheet1!A:A` refers to all the cells in the first column of Sheet1.
- `Sheet1!1:2` refers to the all the cells in the first two rows of Sheet1.
- `Sheet1!A5:A` refers to all the cells of the first column of Sheet 1, from row 5 onward.

- `A1:B2` refers to the first two cells in the top two rows of the first visible sheet.
- `Sheet1` refers to all the cells in Sheet1.

  If the sheet name has spaces or starts with a bracket, surround the sheet name with single quotes (`'`), e.g. `'Sheet One'!A1:B2`. For simplicity, it is safe to always surround the sheet name with single quotes.

Example

```
<!DOCTYPE html>
<html>
<head>

<title>Personal Librarian</title>
<style>
      table, th, td {
            border: solid 1px black;
            border-collapse: collapse;
      }

      th, td {
            padding: 4px 8px;
      }
</style>

</head>
<body>
<p><a href="https://docs.google.com/spreadsheets/d/1n2w0s1lqSZ4kHX3zeNYT-
UNRPEr1aextWaG_bsJisn8/edit?usp=sharing" target="_blank">Spreadsheet share
link</a></p>
<button id="signin-button" onclick="ts.handleSignInClick()">Sign in</button>
<button id="signout-button" onclick="ts.handleSignOutClick()">Sign out</button>
<button onclick="loadData()">Load Data</button>
<button onclick="testBatchAdd()">Batch Add</button>
<br>
<input type="text" id="update-range" placeholder="range">
<input type="text" id="update-value" placeholder="new value">
<button onclick="testUpdate()">Update</button>
<span>Range Example: student_info!A3:A3</span>
<table id="data"></table>


<script src="https://code.jquery.com/jquery-3.3.1.min.js"></script>
<script src="https://apis.google.com/js/api.js"></script>
```

```
<script src="SheetsApi.js"></script>
<script>
	let ts = new SheetsApi("1n2w0s1lqSZ4kHX3zeNYT-UNRPEr1aextWaG_bsJisn8",
"AIzaSyDeampVGzzd8NvBiUtEsNVmNkAQU1TZ17I", "21358841826-
edt9rotek8r1rbivt91nabpn2sc2g6ts.apps.googleusercontent.com");
	ts.handleClientLoad();

	function loadData() {
	ts.getSheet("student_info").then(response => {
			let values = ts.parseSheetValues(response);
			let columns = values[0].length;
			let rows = values.length;
			let temp = "";
			let tempValue = "";
			for (let i = 0; i < rows; i++) {
				temp += "<tr>";
				for (let j = 0; j < columns; j++) {
					tempValue =
values[i][j]===undefined?"":values[i][j];
					temp += (i===0?"<th":"<td") + " id='" +
getCharFromNum(j) + (i+1) + "'>" + tempValue + (i===0?"</th>":"</td>");
				}
				temp += "</tr>";
			}
			$("#data").html(temp);
		}, reason => {
			console.log(ts.parseErrorMessage(reason));
		});
	}

	function testBatchAdd() {
		let values = [
			["111","hua","zeru","zhua@upei.ca","computer
science","someone","somecode","4th year","blabla","","","","",""],
			["222","hua","zeru","zhua@upei.ca","computer
science","someone","somecode","4th year","blabla"],
			["333","hua","zeru","zhua@upei.ca","computer
science","someone","somecode","4th year","blabla","","","","",""],
			["444","hua","zeru","zhua@upei.ca","computer
science","someone","somecode","4th year","blabla","","","","",""]
		];
		ts.batchAdd("student_info",values).then(function(response) {
			let result = ts.parseBatchAdd(response);
			if (result === values.length) {
				console.log("Success");
```

```
                        loadData();
                  } else {
                        console.log("Fail");
                  }
            }, function(reason) {
                  console.log(ts.parseErrorMessage(reason));
            });
      }

      function testUpdate() {
            let range = $("#update-range").val();
            let values = [[$("#update-value").val()]];
            ts.update(range, values).then(function(response) {
                  console.log(response);
                  let result = ts.parseUpdate(response);
                  if (result > 0) {
                        console.log("Success");
                        loadData();
                  }
            }, function(reason) {
                  console.log(ts.parseErrorMessage(reason));
            });
      }

      function getCharFromNum(num) {
            return String.fromCharCode('A'.charCodeAt(0) + num);
      }
</script>
</body>
</html>
```