Table of Contents:

Final Goals:

The goal of this project is to create an online leaderboard that uploads scores for an “escape room” website (or any applicable website). The scores should be easy to update and retrieve from a Google Sheet. These scores should be ready to display in a web browser or to be manually edited via a Google owner account. Final implementations will work on most modern browsers using commonplace browser mechanics: specifically Localstorage/cookies, using the Jquery library for Javascript; and the Google API.

Background Knowledge:

To follow this tutorial it is recommended that you have are proficient with the following HTML5, Javascript, Google Sheets, Web API’s, and Google Apps Script (GAS)

All code in Google Apps Script (GAS) can be learned through this tutorial and should be fairly easy to decipher if you understand Javascript. The nuances of its functionality will be irrelevant in this tutorial. Before continuing, it may still be helpful to develop a solid foundation of what GAS can do and how it works to reduce confusion. A comprehensive tutorial can be found here.

For all further explanations, the example described and shown will be a 4-part “escape room” puzzle in which there are 4 separate sites for the user to interact with. All the sites are linked in a set order for the player to click through, where the best score is achieved by clicking the least. Each website was created entirely by students in Eric Marintsch’s 12th-grade computer science class. All code described will be on “forked” copies of their websites from CodeSandbox.io in which I have added scripts to their code without editing the original sites (for the most part). The website(s) you use in your own implementation do not need to be in this format. Once you grasp the general setup of the leaderboard system, you may apply this format elsewhere.

If you are interested in a cheat-proof system, this tutorial needs tweaking. The final implementation is extremely easy to bypass by anyone adept at programming. However, I recommend reading this tutorial to see the power of Web API’s and Google Sheets since they are extremely useful and can be made cheat-proof in your own time using O-Auth, O-Auth2, hidden headers, alternatives to localstorage, etc.

Current Implementation:

This explanation will work chronologically as if you are watching a user experience the current implementation:

1. The user will open up the HTML5 escape room site from a link, initializing the score variable. The score variable will be saved using a unique ID tag in the "XXXXXX-score" format in the localstorage. This can be viewed from the Inspector, but hidden to the user.

2. The website will have an added Javascript file that runs a function counts upward by 1 every time the user clicks and updates the localstorage to reflect this. This could be done without local storage, but is easier to access cross-script this way (theoretically both could be used together to prevent manually editing your score/cheating)

3. When the user clicks a link to the next page, the link's URL will have additional "tags" added on by the Javascript beforehand. These tags will save any and all score variables for the next website to see (including cross-domain, unlike localstorage which is only for singular domains). Using tags allows us to edit any pre existing URL without changing the destination.

4. When loading the next page, the same script that initializes the score will now interpret the URL headers. By converting the *string* of the URL to an array of *floats*, we can now update localstorage and continue the process from step 2 and 3. This cycle can be repeated as many times as you like.

5. After the user has completed all relevant pages, they will be taken to a custom page created from a leaderboard template. This template includes an HTML5 Input Form element for the user to enter their 4-character name on the scoreboard. Upon hitting submit, their name will be filtered to avoid any common swear words.

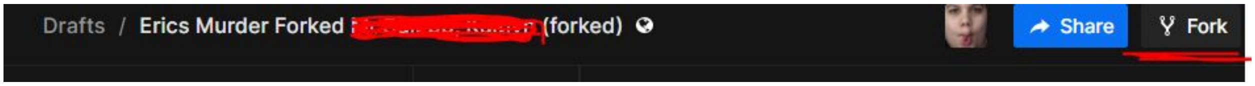
6. Using Jquery's AJAX function (a product of vanilla Javascript's "fetch" function), a Javascript script will access the google API through a GAS script. The GAS script will be called with an object containing the player's name and score. GAS will work directly with your Google Sheet, adding a new row for the player's data on a specified sheet. This step is by far the hardest and presents the most difficulty in perfecting.

7. Javascript is used to holistically generate a nice-looking leaderboard on the page with the top 10 names and scores easily retrieved from the Google Sheet directly (using JQuery). This step can be done however you like, but I chose to use an HTML5 table since I have prior practice with them. There are JS libraries out there for making this easier/nicer if you search online.

Step-by-Step tutorial

Step 1 - Make a copy of the site you will be using

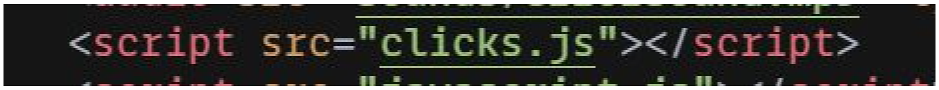
By using CodeSandbox's "fork" feature on a student website, I can own a copy of it.



This is necessary before I am able to make any edits. This will differ if you are using a different program. Whichever format you choose, make sure you can add and link javascript files easily and have access to the HTML for simple editing.

Step 2 - Add click tracking to the chosen site

Next I will be adding a script called "clicks.js". This script will run at lunchtime, initializing your score variables. A link to the raw text is here. The script will need to be linked in the HTML as shown below



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

The script has 3 main jobs. Firstly, it initializes the score variable with the "RETRIEVESTARTINGSTATE" function called on launch, which takes data from the URL using the "getUrlVars" function. These 2 effectively read every variable present in the URL. For example:

"https://example-website.com/?s=10"
Where s=10 means that the page will always open with a score of 10 (including refresh)
Unfortunately this is where cheating is VERY easy. The URL can be manually edited and reopened.

You can add additional variables such as a timer by reading more out of the *vars* array and including said variable in the "GOTONEXTPAGEWITHDATA" function.

The second main function is passing all data into the next site in the series. This function will read all final values from the localstorage and convert it into a string. This is added onto whatever url is passed in from the HTML as shown here:

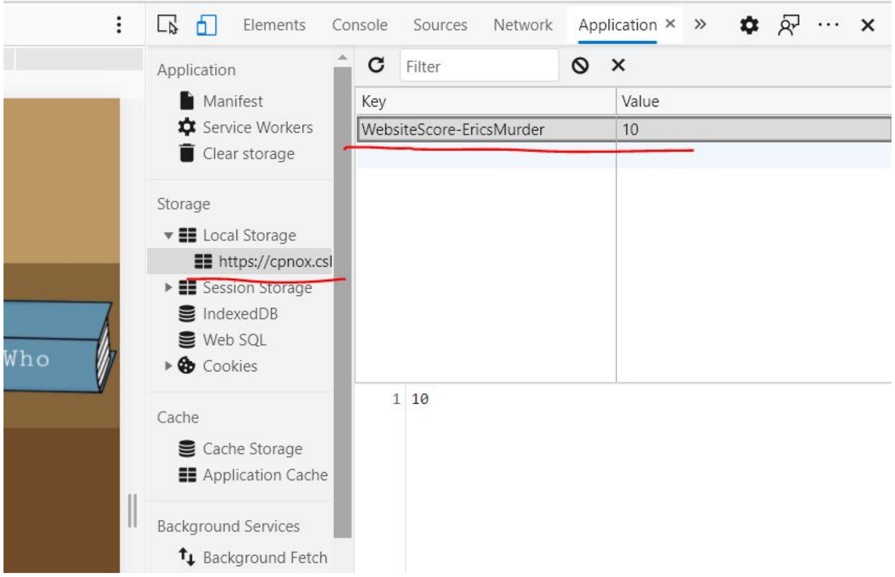
```
onclick="GOTONEXTPAGEWITHDATA('https://example-website.com/');"
```

The third and final feature of "clicks.js" is click tracking. Using the "addClicks" function, the localStorage will be pulled, updated, and rewritten to reflect +1 click. This function is called by the event listener at the start of the script. You can be creative and add all sorts of event listeners to make variations on this.

Step 3 - Make sure the script is well integrated

The most important steps are linking the new clicks.js file in the HTML and replacing the onclick to the next site with our new GOTONEXTSITE function. When editing student sites, the link to the next site may be unusual or hard to find. This will require zombifying their code/making crude edits. This is why we make a copy of their site first. Additionally, change the “websiteScoreID” variable to your own custom ID to avoid variable name overlap with other sites. Edit “firstpageinroom” to reflect whether this page will be opened with or without URL reading (though frankly this is pointless in the current setup, but is a good structure for optional starting parameters).

To make sure that the script is working properly there are 2 steps. Firstly, try adding “/?s=10” to your URL and ensure that the score variable starts as 10 when the page opens. Upon loading in your newly edited URL, open the inspector using F12 or right-clicking, then “Inspect”. In the Application tab on most modern browsers you can go to the localStorage for your site.



Key	Value
WebsiteScore-EricsMurder	10

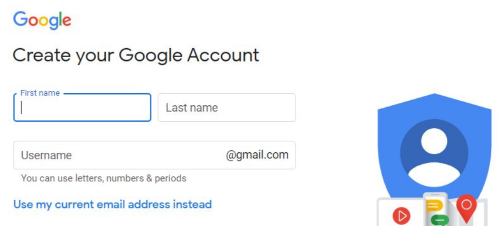
```
1 10
```

Here you can see the Value being stated as 10 as we expect. This example is in Microsoft Edge. The Key will be whatever you named the websiteScoreID variable. Finally, make sure this counts up by increments of 1 whenever you click on your website. This may not work if the event listener was added to the wrong part of the website. For errors, check the Console.

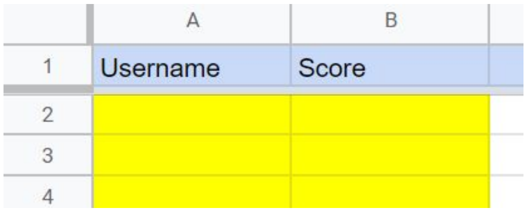
What is amazing about the all-in-one script “clicks.js” is that it only takes adding this 1 file. By repeating this process on all 4 of my escape room sites, they will each accumulate score variables sequentially, saving at each new page. This means that any site refreshes will revert to the initial score when the page was loaded up the first time. This is inherent to the system and requires reworking to change, though is possible to work around. Make sure that all links between pages link to **your own copied sites containing clicks.js**. If not ALL links are to websites containing clicks.js, the chain between sites will be severed and clicks will not accumulate at all.

Step 4 - Setup Google Sheet as database

To act as a database with all of our scores, we will be creating a Google Sheet that can score lots of information for free. It is also highly adaptable and customizable. Firstly, create a Google Suite account. If you already have a Gmail account this will work, but I recommend creating a custom account for the new sheet. This way, it is easier to keep track of authorization and security if you choose to go that route. This part is pretty self-explanatory.

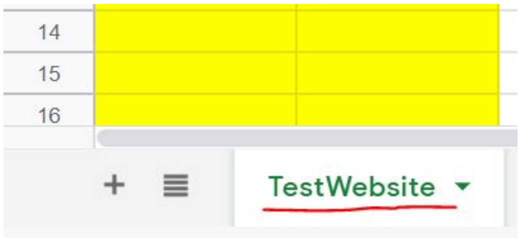

Google
Create your Google Account
First name
Last name
Username
@gmail.com
You can use letters, numbers & periods
Use my current email address instead

Open up Google Sheets and click a template under "Start a new Spreadsheet". Open it. Set the spreadsheet visibility to Public. At the top of your spreadsheet, set the first 2 columns at the top to a **memorable name (for later step)**. In my example I use "Username" and "Score":



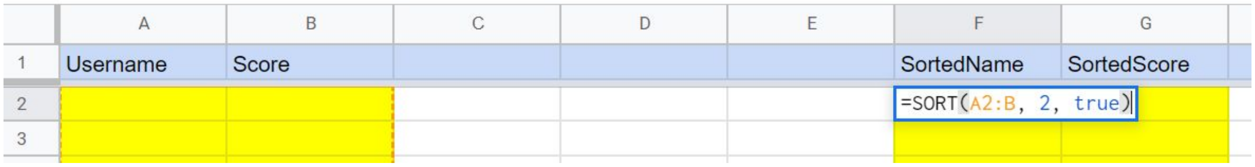
	A	B
1	Username	Score
2		
3		
4		

(colors optional)



14		
15		
16		

The name of the active sheet at the bottom is important. You will need to input this information into your Javascript sheets.js (Step 6) for everything to work together correctly. Finally, duplicate the two columns named "username" and "score".

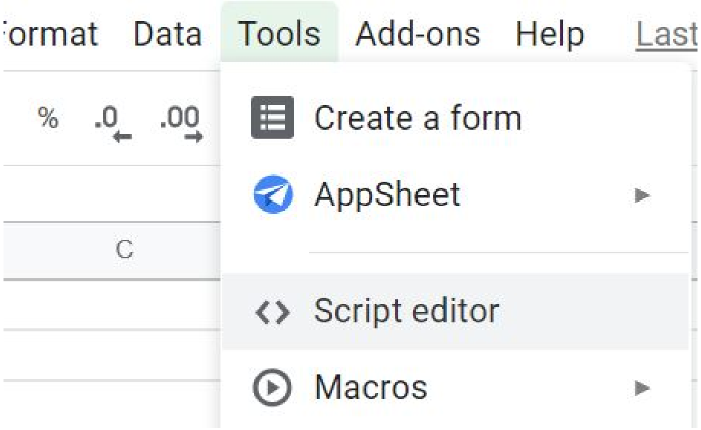


	A	B	C	D	E	F	G
1	Username	Score				SortedName	SortedScore
2						=SORT(A2:B, 2, true)	
3							

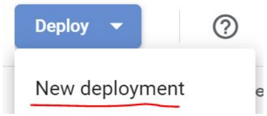
These columns will be sorted copied of the first 2, both unedited by the script. Change the names of these columns to avoid duplicates. The unsorted column will be written to and the sorted column will be read from. This sort function will put the lowest scores at the top of the leaderboard. You can change this by editing the =SORT function accordingly.

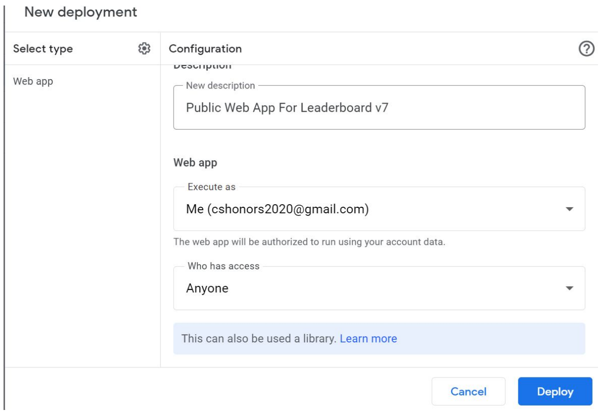
Step 5 - Create a GAS script for Google API

To connect our sheet to our modified site, we need to use an API (which I am new at). API's are very difficult to work with, but allow a website to communicate with our google sheet. They act as the middle-men of the web, ensuring that neither end of the communication can harm the other. Accessing an API from a website is difficult since websites themselves can't act as servers. I recommend doing extended research if you are unfamiliar with how API's work. Luckily, Google has an extremely simple workaround which we can access for free. We can access the Google API using a custom script in GAS. First, create a script on **the same account as your google sheet was created**. From your new google sheet, navigate to this menu at the top of the screen:



You should be redirected to a new GAS script called Code.gs. Name this whatever you want. This file is inextricably linked to your google sheet. Inside this script you will use this code. The script's functions DoGet and DoPost essentially work as Reading and Writing calls. The DoPost function will take in "e" as a variable containing the "username" and the "score" (or whatever you replaced these with) from a javascript file in the next step. The custom function handleResponse works with the google sheet by inserting a row in the proper sheet on the proper columns. Now, we need to make this script accessible anywhere on the web. In the top right, click "Deploy"





Description

New description: Public Web App For Leaderboard v7

Web app

Execute as: Me (cshonors2020@gmail.com)

The web app will be authorized to run using your account data.

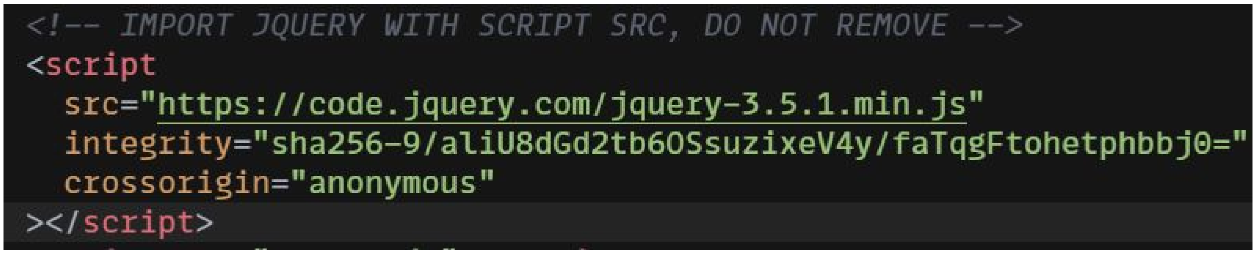
Who has access: Anyone

This can also be used a library. Learn more

Make sure that "Execute As" is set to the account that owns the Google Sheet, and that "Anyone" can call this script anonymously. Save the "Web App URL" for the next step.

Step 6 - Creating the leaderboard website display

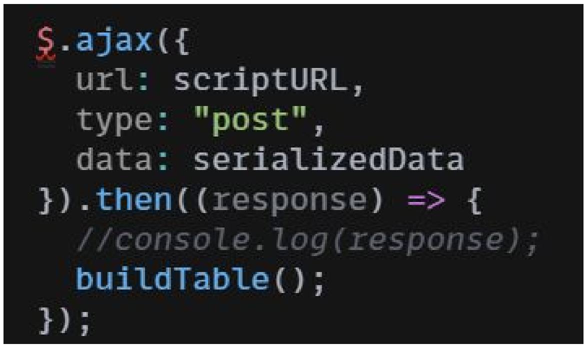
The last step is to connect the GAS script and escape room sites together. So, using our GOTONEXTPAGE function in clicks.js, we will link to a site similar to this template. This template must contain clicks.js, but also a new javascript file called Sheets.js. The code can be found here. Be very careful with your variable names here since they correspond to variable names in both the GAS script, the Google Sheet, and your Clicks.js. This requires double-checking to ensure. Make sure that you are using the JQuery library for Javascript. This allows for the $.Ajax functions to run. The 2 Ajax functions will read and write by passing information into the GAS script. To make sure you have JQuery active, link to the CDN in your HTML as shown here (if you're not doing this another way):



```
<!-- IMPORT JQUERY WITH SCRIPT SRC, DO NOT REMOVE -->
<script
  src="https://code.jquery.com/jquery-3.5.1.min.js"
  integrity="sha256-9/aliU8dGd2tb6OSsuzixeV4y/faTqgFtohetphbbj0="
  crossorigin="anonymous"
></script>
```

Integrity and crossorigin should be optional.

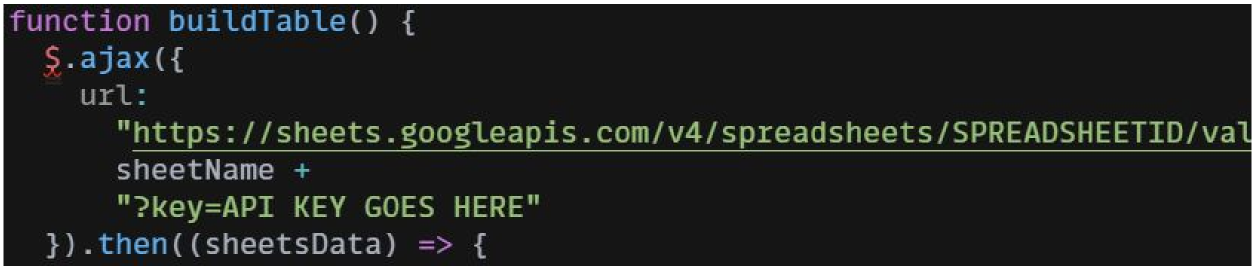
So what does sheet.js accomplish? Firstly, the submitForm function will upload the username and score to the GAS for sheet writing. It checks to make sure that you are only submitting once and that you have typed a name into the arbitrary input form. It concatenates all your given data into a playerData object. The names here represent the column headers on your sheet. The data is **converted to json for security purposes**. JSON and JSONP are uniquely privileged for data transmission on the web. The Post (write) request to the GAS, written in Jquery, is shown below:



```
$.ajax({
  url: scriptURL,
  type: "post",
  data: serializedData
}).then((response) => {
  //console.log(response);
  buildTable();
});
```

The URL is specified at the top of your script as the link to your GAS Web App (from previous step). "Post" means to "add 1 piece of data", where the data is the serialized player data in JSON. The ".Then" function waits real-time for the response to return as finished before it then builds the HTML table on the page with buildTable. If you don't want to use the Jquery Ajax syntax, the same thing can be accomplished using the simple, vanilla fetch() function.

The next step is reading the data from the google sheet. This is the easiest step, requires no fancy security bypassing, and can be done many ways.

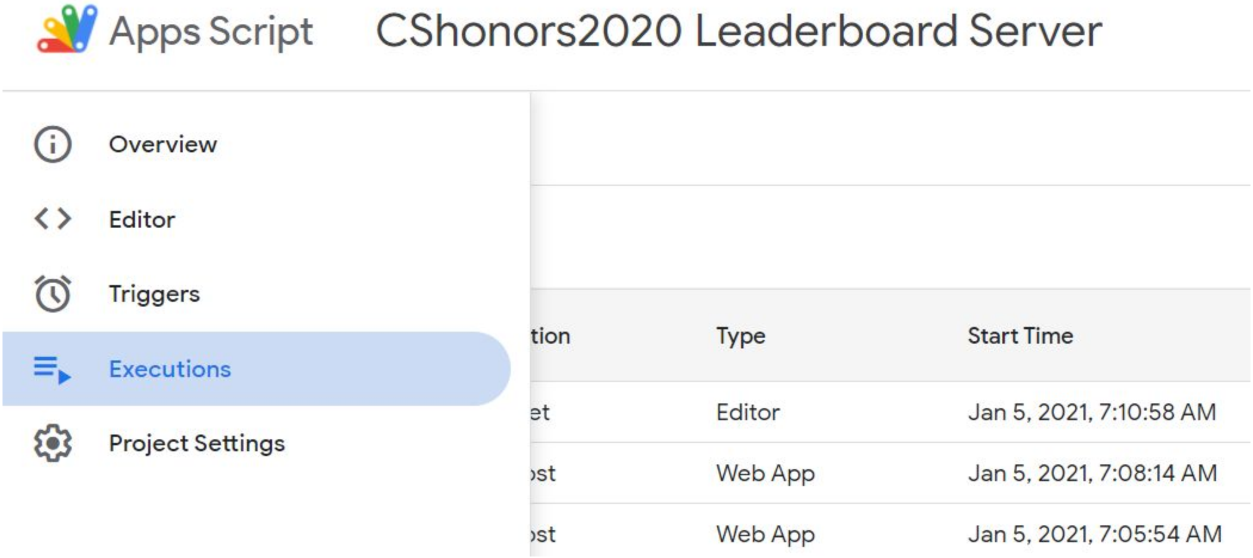


```
function buildTable() {
  $.ajax({
    url:
      "https://sheets.googleapis.com/v4/spreadsheets/SPREADSHEETID/val
      sheetName +
      "?key=API KEY GOES HERE"
  }).then((sheetsData) => {
```

To get the SPREADSHEETID, go to your google sheet. It will be the long string of numbers in the URL between /d/ and /edit. The API key is a user-specific code that allows you to read all the information. This is to ensure that you will get the data passed back from this request. You can generate your own key here. I am not sure if this key is necessary. You can also receive this data by using a similar request to the GAS web app (doGet will be called to return data). The following code is all generating a table for the webpage with the retrieved data. This is all styled with CSS to make a fairly presentable leaderboard. In retrospect, I believe using something other than an HTML table would have been more efficient and customizable. I recommend a dynamic flex-box layout or using a Javascript library to do the generation.

Step 7 - Error testing

This is a precarious setup. If you are finding errors in this step, go through previous steps to make sure that everything is in working order. To see a working version of the leaderboard, see this link. Due to the score in the URL, making a submission will have no effect. The google sheet being accessed in my example is visible here. If the console is returning an error involving "CORS" (cross origin resource sharing) or "ERROR 401,402,403,404" then it is an issue with the web app call (the ajax functions). If you are getting nothing new in your google sheet, make sure that the GAS script is actually being called properly.



tion	Type	Start Time
et	Editor	Jan 5, 2021, 7:10:58 AM
ost	Web App	Jan 5, 2021, 7:08:14 AM
ost	Web App	Jan 5, 2021, 7:05:54 AM

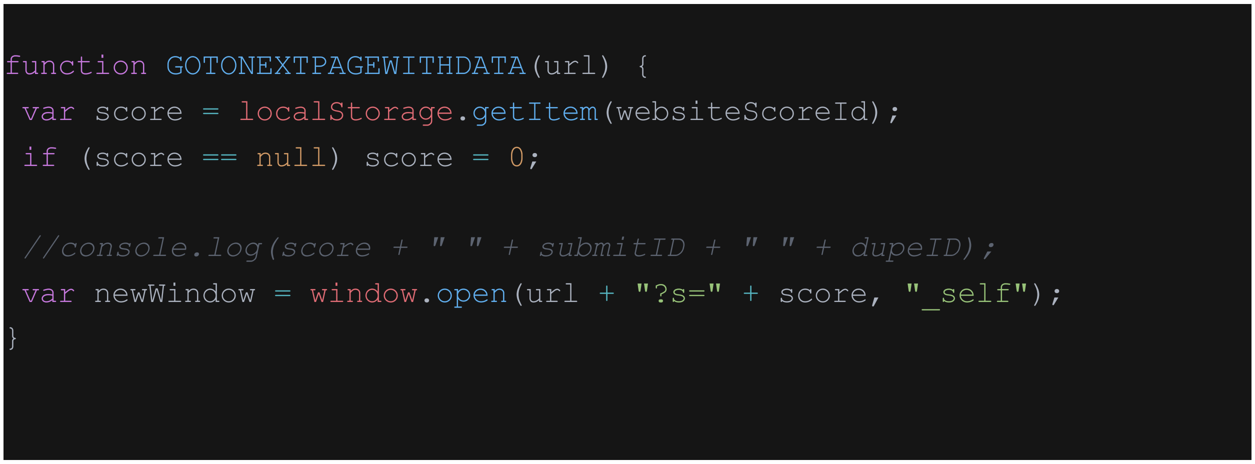
In the Executions tab, you can actually see the live updates from any website calling your script. This can help you debug greatly.

Conclusion:

Congratulations! You now have the structure of a somewhat simple leaderboard system! Hopefully you have learned about many different facets of full-stack web development throughout this journey. I am still learning myself, so take everything in this tutorial with a grain of salt. I recommend innovating on this design since it lacks any form of cheat prevention. It could also be distilled down further for a minimalist setup. Overall, I feel that my final product is robust and adaptable enough that it is a great entry into understanding API's and advanced web design.

Code sample 1 clicks.js ------

```
var websiteScoreId = "WebsiteScore-EricsMurder";
var firstPageInRooms = true;

document.addEventListener("click", addClick);
window.self.name = Math.random() * 100;

RETRIEVESTARTINGSTATE();
if (firstPageInRooms) {
 localStorage.setItem(websiteScoreId, 0);
}

function getUrlVars() {
 var vars = {};
 var parts = window.location.href.replace(/[?&]+([^=&]+)=([^&]*)/gi,
function (
   m,
   key,
   value
 ) {
   vars[key] = value;
 });
 return vars;
}

function RETRIEVESTARTINGSTATE() {
 //determined by GOTOURLWITHDATA func
 let score = getUrlVars()["s"];
 if (score == null) score = 0;
 localStorage.setItem(websiteScoreId, score);
}

function addClick() {
 let prevClicks = localStorage.getItem(websiteScoreId);
 if (prevClicks == null) prevClicks = 0;
 prevClicks = parseInt(prevClicks);
 localStorage.setItem(websiteScoreId, prevClicks + 1);
}
```



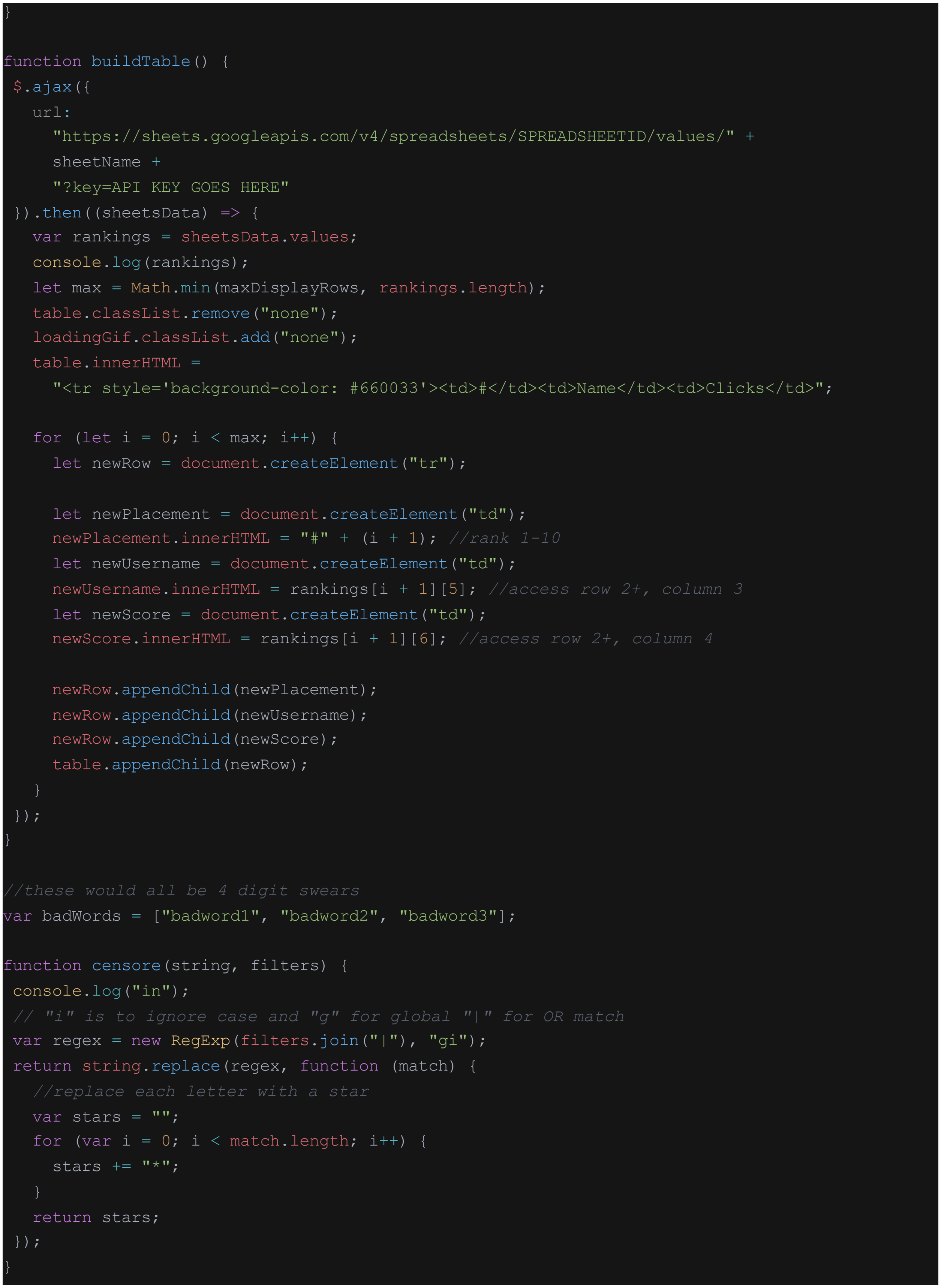
```
function GOTONEXTPAGEWITHDATA(url) {
 var score = localStorage.getItem(websiteScoreId);
 if (score == null) score = 0;

 //console.log(score + " " + submitID + " " + dupeID);
 var newWindow = window.open(url + "?s=" + score, "_self");
}
```

Code Sample 2 GAS script------

```
//  1. Enter sheet name where data is to be written below
        var SHEET_NAME = "Website1";

//  2. Run > setup
//
//  3. Publish > Deploy as web app
//    - enter Project Version name and click 'Save New Version'
//    - set security level and enable service (most likely execute as 'me' and access 'anyone, even anonymously)
//
//  4. Copy the 'Current web app URL' and post this in your form/script action
//
//  5. Insert column names on your destination sheet matching the parameter names of the data you are passing in (exactly matching case)

var SCRIPT_PROP = PropertiesService.getScriptProperties(); // new property service

// If you don't want to expose either GET or POST methods you can comment out the appropriate function
function doGet(e){
  return handleResponse(e);
}

function doPost(e){
    return handleResponse(e);
}

function handleResponse(e) {
  //SpreadsheetApp.getActiveSpreadsheet().getActiveSheet().appendRow(["bob", 2032]);

  // shortly after my original solution Google announced the LockService[1]
  // this prevents concurrent access overwritting data
  // [1] http://googleappsdeveloper.blogspot.co.uk/2011/10/concurrency-and-google-apps-script.html
  // we want a public lock, one that locks for all invocations
  var lock = LockService.getPublicLock();
  lock.waitLock(30000);  // wait 30 seconds before conceding defeat.

   // If you are passing JSON in the body of the request uncomment this block
  var jsonString = e.postData.getDataAsString();
  e.parameter = JSON.parse(jsonString);
  //GmailApp.sendEmail("CShonors2020@gmail.com", "welcome!", "the deal is: " + e);

  try {
    // next set where we write the data - you could write to multiple/alternate destinations
    var doc = SpreadsheetApp.openById(SCRIPT_PROP.getProperty("key"));
    var sheet = doc.getSheetByName(e.parameter["SheetName"]);

    // we'll assume header is in row 1 but you can override with header_row in GET/POST data
    var headRow = e.parameter.header_row || 1;
    var headers = sheet.getRange(1, 1, 1, sheet.getLastColumn()).getValues()[0];
    var nextRow = sheet.getLastRow()+1; // get next row
    var row = [];
    // loop through the header columns
    for (i in headers){
      if (headers[i] == "Timestamp"){ // special case if you include a 'Timestamp' column
        row.push(new Date());
      } else { // else use header name to get data
        row.push(e.parameter[headers[i]]);
        //row.push(headers[i]);
        Logger.log(headers[i]);
      }
    }
    // more efficient to set values as [][] array than individually
    sheet.getRange(nextRow, 1, 1, row.length).setValues([row]);
    // return json success results
    return ContentService
          .createTextOutput(JSON.stringify({"result":"success", "row": nextRow}))
          .setMimeType(ContentService.MimeType.JSON);
  } catch(e){
    // if error return this
    return ContentService
          .createTextOutput(JSON.stringify({"result":"error", "error": e}))
          .setMimeType(ContentService.MimeType.JSON);
  } finally { //release lock
    lock.releaseLock();
  }
}

function setup() {
    var doc = SpreadsheetApp.getActiveSpreadsheet();
    SCRIPT_PROP.setProperty("key", doc.getId());
}
```

Code Sample 3 Sheets.js-----

```
var sheetName = "SHEETNAME-GOES-HERE";
var websiteScoreId = "WebsiteScore-WEBSITE_TITLE";
var scriptURL = "YOUR-URL-GOES-HERE";

var canSubmit = true; //sets 1 submission max

var maxDisplayRows = 10;
var table = document.getElementById("leaderboard");
var loadingGif = document.getElementById("loading-gif");

buildTable();

function submitForm() {
 var name = document.getElementById("username").value;
 var score = localStorage.getItem(websiteScoreId);
 //console.log("name: " + name);
 if (name == undefined || name == "") {
   alert("You didn't put a name!");
   return;
 }
 if (!canSubmit) {
   alert("You already submitted your score.");
   return;
 }
 alert(
   "you have submitted your score!😎 Give it a second to update. If your score is off the
charts, it won't change the screen."
 );
  canSubmit = false;
 document.getElementById("entername").style.display = "none";

 name = censore(name, badWords);
 var playerData = {
   //edit the sheetname to change which leaderboard to write to
   SheetName: sheetName,
   //make sure the key for these properties are your column headers
   Username: name,
   Score: score
 };
 var serializedData = JSON.stringify(playerData);
 //console.log(serializedData);

 $.ajax({
   url: scriptURL,
   type: "post",
   data: serializedData
 }).then((response) => {
   //console.log(response);
   buildTable();
 });
```



```
}

function buildTable() {
 $.ajax({
   url:
     "https://sheets.googleapis.com/v4/spreadsheets/SPREADSHEETID/values/" +
     sheetName +
     "?key=API KEY GOES HERE"
 }).then((sheetsData) => {
   var rankings = sheetsData.values;
   console.log(rankings);
   let max = Math.min(maxDisplayRows, rankings.length);
   table.classList.remove("none");
   loadingGif.classList.add("none");
   table.innerHTML =
     "<tr style='background-color: #660033'><td>#</td><td>Name</td><td>Clicks</td>";

   for (let i = 0; i < max; i++) {
     let newRow = document.createElement("tr");

     let newPlacement = document.createElement("td");
     newPlacement.innerHTML = "#" + (i + 1); //rank 1-10
     let newUsername = document.createElement("td");
     newUsername.innerHTML = rankings[i + 1][5]; //access row 2+, column 3
     let newScore = document.createElement("td");
     newScore.innerHTML = rankings[i + 1][6]; //access row 2+, column 4

     newRow.appendChild(newPlacement);
     newRow.appendChild(newUsername);
     newRow.appendChild(newScore);
     table.appendChild(newRow);
   }
 });
}

//these would all be 4 digit swears
var badWords = ["badword1", "badword2", "badword3"];

function censore(string, filters) {
 console.log("in");
 // "i" is to ignore case and "g" for global "|" for OR match
 var regex = new RegExp(filters.join("|"), "gi");
 return string.replace(regex, function (match) {
   //replace each letter with a star
   var stars = "";
   for (var i = 0; i < match.length; i++) {
     stars += "*";
   }
   return stars;
 });
}
```