Criterion C: Development

The website is made using HTML and CSS, which are the most common languages for website development. The HTML calls Javascript functions on the website which communicates with the server. The server receives data from the database which it can then send back. Communication uses a Javascript AJAX JQuery call in the URL. The data is processed in Javascript on the website and then the data is pushed into the website HTML.

Please note that the code may have been reformatted to be more visible on this document.

HTML & PHP on website

```
<?php
header("Access-Control-Allow-Origin: *");
//Allow from any origin
if (isset($_SERVER['HTTP_ORIGIN'])) { header("Access-Control-Allow-Origin:
{$_SERVER['HTTP_ORIGIN']}"); header('Access-Control-Allow-Credentials:
true'); header('Access-Control-Max-Age: 86400');
// cache for 1 day
}
// Access-Control headers are received during OPTIONS requests
if ($_SERVER['REQUEST_METHOD'] == 'OPTIONS') { if
(isset($_SERVER['HTTP_ACCESS_CONTROL_REQUEST_METHOD']))
header("Access-Control-Allow-Methods: GET, POST, OPTIONS"); if
(isset($_SERVER['HTTP_ACCESS_CONTROL_REQUEST_HEADERS']))
header("Access-Control-Allow-Headers:
{$_SERVER['HTTP_ACCESS_CONTROL_REQUEST_HEADERS']}"); exit(0); }
?>
```

The PHP header sets which browsers and sites can access the website. It also sets the cache to improve loading times.

```
    <head>
        <meta charset="utf-8"><!--Standard character set used in
translating-->
       <meta name="viewport" content="width=device-width,
initial-scale=1"><!--Standard view scale-->
       <title>I Remember When!</title>
       <link rel="icon" href="ASSETS\WebsiteLogo.png"><!--ICON for page-->
       <link rel="stylesheet" href="CSS/style.css?v=1.1">
       <link rel="stylesheet"
href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/4.7.0/css/font-a
wesome.min.css">
       <script src="https://apis.google.com/js/platform.js" async
defer></script>
       <meta name="google-signin-client_id"
content="334986163248-ih52v02opicaasu92620o2tdjrrn3ojm.apps.googleusercont
ent.com">
       <script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.6.0/jquery.min.js"></s
cript>
    </head>
```

The head element holds metadata about the page, such as the tab design, imports stylesheets and links scripts.

```
<div class="wrap">
      <div class="search">
           <input type="text" class="searchTerm" placeholder="Search for
Tags, Dates or Titles" id="searchBar" name="responseTEST"><!--Create the
search bar→
           <button type="submit" class="searchButton"
onclick="makeRequestJQuery(); hide('centered', 'infoText', 'lockScreen');
show('articleContainer', 'articlePublisher')"> <!--Create the search
button that runs javaScript and changes the layout of the site→
                <i class="fa fa-search"></i>
           </button>
      </div>
</div>
```

This is the code for the search bar and search button. Data is inputted in the bar to be pulled when the search button is clicked, calling makeRequestJQuery and rearranging the page. The <i> element is the image.

```
<div id="articleContainer">
      <div id="articleDate"></div>
      <br><br><br></br>
      <div id="articleTitle"></div>
      <div id="article"><br></br>Please be patient. It may take up to 30
seconds to load.</div>
      <div id="notes"></div>
      <div id="articlePublisher"></div>
</div>
```

The article container uses a form of encapsulation to deal with style and easier page rearrangement. These divs have different IDs so they can be filled with different data.

```
<script src="https://code.jquery.com/jquery-3.6.0.min.js"
integrity="sha256-/xUj+3OJU5yExlq6GSYGSHk7tPXikynS7ogEvDej/m4="
crossorigin="anonymous"></script>
<script src="JS/searchScript.js"></script>
```
This imports the scripts for jquery's library from the internet and searchScript from the website.

JavaScript on website
```
function getURL(){  //Creates URL for story page
  let DEPLOY_ID = "AKfycbzEdX1hyrbE77wRezjqLsuhca4_jfPtHrzCusORFm0d";
  let URL = "https://script.google.com/macros/s/" + DEPLOY_ID + "/dev";
  return URL;
}
```
getURL holds the URL for the google scripts server, allowing jQuery to be called.

```
function makeRequestJQuery(){   //Submit a search for the story
   let queryString = document.getElementById('searchBar').value;
   var url = getURL() + "?searchString=" +
queryString+"<>USER<>"+emailVar;
   console.log("URL "+url);
   // Make an AJAX call to Google Script
   jQuery.ajax({
       crossDomain: true,
       url: url,
       method: "GET",
       dataType: "jsonp",
   })
```
The makeRequestJQuery function takes the value of the search bar and adds it to the URL. It then makes an AJAX call to the google script, adding the search term to the URL where it can be processed by the google scripts server.

```
.done(function( data ) {
       var str=data;
       var lockStatus= str.slice(str.indexOf("<>SECURITY<>")+12,
str.lastIndexOf("<>SECURITY<>"));    //Find lock status
```
The function waits for the request to be sent back meaning that it has the data. The str variable holds all the data the server sent back as a string. lockStatus holds the security status as a

string, splicing a specific part of str. Splicing allows the website to take specific information from str. We could have sent the information separately but this method is much more efficient as requests for data are slow.

```
if(lockStatus=="false"){    //Show security message as they are not
authorised
           hide("articleContainer");
           hide("articleContainer2");
           hide("articleContainer3");
           show("lockScreen");
           if(signInMsg==true){    //If signed in but not authorised
               fillData("lockScreen", "This content is private, if this is
your content please make sure you are signed in with the correct account.
If you still encounter problems, please contact: FAKEEMAIL@EMAIL.COM");
           }else{    //If not signed in
               fillData("lockScreen", "Please sign in to view this
content!");
           }
       }
       if(lockStatus=="true"){
           hide("lockScreen");
           signInMsg=false;    //Hide Please sign in message
       }
//The email has been changed for privacy reasons.
```

The website checks if the security test succeeded, if not, it hides the content and displays a message stating whether the user isn't authorised or hasn't signed in. If the test succeeded, the website hides only the lock screen.

```
 var datePubl= str.slice(str.indexOf("<DP>")+4, str.lastIndexOf("<DP>"));
//Load in first story
       var date= str.slice(str.indexOf("<DA>")+4,
str.lastIndexOf("<DA>"));
       var title= str.slice(str.indexOf("<TI>")+4,
str.lastIndexOf("<TI>"));
       var article= str.slice(str.indexOf("<AR>")+4,
str.lastIndexOf("<AR>"));
       var author= str.slice(str.indexOf("<AU>")+4,
str.lastIndexOf("<AU>"));
       var notes= str.slice(str.indexOf("<NO>")+4,
str.lastIndexOf("<NO>"));
```

```
       fillData("articleTitle", title)
       fillData("article", article)
       fillData("articlePublisher",("-"+author))
       fillData("articleDate", ("Date: "+date+"<br></br>Date Published: 
"+datePubl))
       fillData("notes", notes)
```
str is spliced into its components and the website is filled up with information. For a search, this happens to 3 stories.

```
function makeRequestJQuerySpecial(queryString){ //Search for a story with 
the search prompts
   var url = getURL() + "?searchString=" + queryString;
```
The makeRequestJQuerySpecial function takes an input, the chosen searchPrompt sending it to the server which responds by finding the relevant story for the searchPrompt.

```
viewed_1.addEventListener("click", callViewed_1);

function callViewed_1(){makeRequestJQuerySpecial("<MOST-VIEWED-1>");}
```
This code makes the buttons interactable, calling the makeRequestJQuerySpecial function.

```
function hide(hideThis) {
 var z = document.getElementById(hideThis);
   z.style.display = "none";
}
function show(showThis) {
 var z = document.getElementById(showThis);
   z.style.display = "block";
}

function fillData(element, data) {
 document.getElementById(element).innerHTML = data;
}
```
In order to minimise load times, every element is on the same webpage but they are hidden, shown or filled with information when they are needed.

GoogleScript on server

```
var ss =
SpreadsheetApp.openById("1BaL53vnfuDmTZHE2HEGs1PaN7XyqfPu0JdUjkw5wXu0");
//This links the database to the server
```

ss is set as the spreadsheet so that it can be referred to later more easily.

```
function doGet(e){      //This gets the searchString through the URL
  Logger.log(e);
  const searchString = e.parameter.searchString;
//This extracts the searchString from the URL

  var JSONResponse = respondToRequest(searchString, e.parameter);
  Logger.log(JSONResponse);
  return ContentService
    .createTextOutput(e.parameter.callback + "(" + JSONResponse + ")")
    .setMimeType(ContentService.MimeType.JAVASCRIPT);
//This converts the code to JSONP to skip through the same-origin-policy of
google
}

function doPost(e){     //This returns the information to the website
  const searchString = e.parameter.searchString;
  var JSONResponse = respondToRequest(searchString, e.parameter);
  return ContentService.createTextOutput(JSONResponse)
.setMimeType(ContentService.MimeType.JSON);
}
```

doGet and doPost functions allow the googleScripts to communicate with the website. It takes the searchString from the end of the URL to be used as a parameter. They are partly predefined by google scripts to interact with the URL.
MimeType and JSON tell the server the format of the data. The MimeType converts the data to JSONP which allows it to be passed through the same-origin google policy which blocks requests if they are in JSON.

```
function respondToRequest(searchString, parameter){    //Main search function
  if (DEBUG){ console.info(searchString + ":" + parameter );}
  var originalSearchString=searchString;
  searchString=getSearchString(searchString);
```

respondToRequest which will process the searchString and return stories. getSearchString is called to remove the user information so that we are left with the search term.

```
 var statement;
  var statementDef=false;  //This variable switches to true when the statement
is filled to ensure that the value isn't overwritten
//The following code checks if a quick search button was chosen
  if (searchString=="<FREQUENT-SEARCHES-1>"){
    statement=ss.getSheetByName("Data").getRange("Y26").getValue();
    statementDef=true;
    return prematureReturn(statement);
```

If the user selected a searchPrompt on the website, we run prematureReturn which skips the search, instead sending the preloaded story corresponding to the searchPrompt. These are loaded at the end of each search after the data has been returned to improve efficiency. statementDef is used to ensure that the story isn't overwritten later while the spreadsheet can still be updated.

```
function findDate(searchString){ //Checks for a year
  searchString = searchString.replace(/[^0-9]/g, "#"); //REGEX equation to
replace all non digits.
  var array=searchString;
  var arrayA=array.split("#");

  var matches = 0;
  var values = [];
  var s=0
  while(s<arrayA.length){
    if(arrayA[s].match(/\d{4}/)){
      matches=matches+1;
      values.push(arrayA[s]);
    }
    s=s+1;
  }
  return(values);
}
```

findDate checks if there is a date in the searchString. It uses RegEx to keep only digits. Each set of digits is put into an array and checked for four consecutive digits. If so, we assume it is a year, put it into the values array and return values.

RegEx allows us to search for specific patterns.

```
function checkForMonth(searchString){     //Function to check if the user
searched for a month
  var shortMon=["jan", "feb", "mar", "apr", "may", "jun", "jul", "aug", "sep",
"oct", "nov", "dec"];
  var longMon=["january", "february", "march", "april", "may", "june", "july",
"august", "september", "october", "november", "december"];
  var search=searchString.toLowerCase();
  var month="none";
  for(var i=0; i<12;i++){
    if((search.indexOf(shortMon[i])!=-1) || (search.indexOf(longMon[i])!=-1)){
//Checks for different ways of writing months
      month=shortMon[i];
    }
  }
  return capitalizeFirstLetter(month);
}
```

checkForMonth checks if there is a month in the searchString. It looks for the long and short spelling ignoring capitalisation. If it finds months, it returns them.

```
    var isDate;
    var keyword=[];                                                  //THE
KEYWORD THAT WILL BE LOOKED FOR ON THE SHEETS
    Logger.log("Start month test "+checkForMonth(searchString));
    if(checkForMonth(searchString)!="None"){           //IS THERE A MONTH
      isDate=true;
      keyword.push(checkForMonth(searchString));
      Logger.log("Keyword "+keyword);
      if((findDate(searchString).length)!=0){
//IS THERE A MONTH+YEAR
        Logger.log("MY");
        isDate=true;
        var arrayAA=[];
        arrayAA=findDate(searchString);
        for(var k=0;k<arrayAA.length;k++){
          keyword.push(arrayAA[k]);
        }
      }
    }else if((findDate(searchString).length)!=0){
//IS THERE A YEAR
      Logger.log("Y");
      isDate=true;
      var arrayBB=[];
      arrayBB=findDate(searchString);
      for(var k=0;k<arrayBB.length;k++){
        keyword.push(arrayBB[k]);
      }
    }else{
//NO DATE
      Logger.log("Has no date");
      isDate=false;
      var keywordConversion = searchString;
      keyword=keywordConversion.split(" ");                        //CREATE
SEPERATE ARRAY SLOTS FOR EACH WORD
    }
  }
```

This code checks for a date. It checks for months, calling checkForMonth and results to the keyword array. It then checks for a year, adding that to the keyword if found. If the boolean isDate is true, the server checks for stories with those months or years, else, if there is no date, it moves on to point calculation. This starts by splitting searchString at its spaces, putting each word into another element.

```
  var counter=0;
  var cellCount=[];
  var points=[];
  var mar="";
  Logger.log("Points Logic Commenced. isDate: "+isDate);
  if(isDate==false){
//CHECKING WORDS
    while(counter<ss.getSheetByName("Data").getRange("P11").getValue()){
//LOOP THROUGH STORIES
```

```
      mar="L"+(2+counter);
//CURRENT CELL
      cellCount[counter]=mar;
//NAME OF CELL
      points[counter]=keywordTest(mar, keyword);
//ADD POINTS
      counter++;
      Logger.log(points);
    }Logger.log("isdate is false. mar="+mar+" counter="+counter+"
cellCount="+cellCount);
  }else{
//CHECKING DATES... IS IT THE SAME THING AS ABOVE
    while(counter<ss.getSheetByName("Data").getRange("P11").getValue()){
//LOOP THROUGH STORIES
      mar="M"+(2+counter);
//CURRENT CELL
      cellCount[counter]=mar;
//NAME OF CELL
      points[counter]=keywordTest(mar, keyword);
//ADD POINTS
      counter++;
      Logger.log(points);
    }Logger.log("isdate is true. mar="+mar+" counter="+counter+"
cellCount="+cellCount);
  }
```

Points are assigned to stories for matches to the keyword array. This starts by looping through the stories. mar keeps track of the cell address. We create arrays called cellCount and points to track cells and the points received; Scripts doesn't support two dimensional arrays. For each cell, we run keywordTest which returns the amount of points. We check either dates or everything else, depending on isDate.

```
function keywordTest(mar, keyword){
  var splitVar1=ss.getSheetByName("Data").getRange(mar).getValue();
  var data = splitVar1.split(" ");
  Logger.log("In the keyword test, this is the data being tested: "+data);
  var counter=0;
  var points=0;
  for(var i=0;i<keyword.length;i++){
    if(data[0].toLowerCase()==keyword[i].toLowerCase()){     //FIRST SET
      points=points+3;
    }
  }
  counter=1;                                                  //SKIP FIRST SET

  while(counter<data.length){                           //LOOP THE STORY WORDS
    for(var j=0;j<keyword.length;j++){                   //LOOP THE SHEET WORDS
      if(data[counter].toLowerCase()==keyword[j].toLowerCase()){
        points=points+1;
      }
    }
    counter=counter+1;
  }
  return points;
}
```

With a split keyword, we first check if the first word in the cell, the title is the same. If so, we give the story 3 points. Then, it checks if any other words are the same. We use a while loop with counter to loop through the cell and an if loop with j to loop through the keyword. If we get a match here, we add 1 point. Note that we set both to lowercase to ensure we don't lose matches due to capitalisation. We then return the total points. It uses a nested loop to go through all the data for each keyword.

```
  var highestCell;
  var highestPoints=-1;
  var highestPoint;
  for(var pointsLoop=0;pointsLoop<points.length;pointsLoop++){
    if(points[pointsLoop]>highestPoints){
      highestCell=cellCount[pointsLoop];
//highestCell IS CELL NAME OF COMPILED WITH HIGHEST POINTS.
      highestPoints=points[pointsLoop];
      highestPoint=pointsLoop;
    }
  }
```

This finds the highest scoring cell stored in highestCell. To ensure a cell is chosen, highestPoints is set to -1. We repeat this process three times for the top three stories, removing the highest each time.

```
  var startOfRow=1;
  //This code finds out when the row, which becomes storyRow starts so that
two character x-Coords don't break it.
  var secondCoordIsChar = (/[a-zA-Z]/).test(highestCell.substring(1,2))
  Logger.log("secondCoordIsChar="+secondCoordIsChar);
  if(secondCoordIsChar==true){
    startOfRow=2
  }

  var storyRow=highestCell.substring(startOfRow,(highestCell.length));

  ss.getSheetByName("Data").getRange("W21").setValue(ss.getSheetByName("Data")
.getRange("C"+storyRow).getValue());   //AUTHOR

  var cellValue=ss.getSheetByName("Data").getRange("Y21").getValue();

  statement=cellValue+"<>SECOND<>"+secondCellValue+"<>THIRD<>"+thirdCellValue;
  }
```

The row of highestCell is found and its elements are loaded into the chamber, specific cells on the google sheet. The above code shows only the author for legibility, for other information it is loaded into different cells using the same process. This process is used for all three stories. It is concatenated into a string on the google sheet which is concatenated into ‘cellValue’s and then into statement.

```
  var myJSON =
JSON.stringify(statement+"<>SECURITY<>"+securityTest(getEmail(originalSearchSt
ring))+"<>SECURITY<>");

  return myJSON;
```

A security check is added to the end of statement and it is returned, ending the respondToRequest function. The returned value is used in doPost, sent back to the website to be processed.

```
function getEmail(searchString){    //Function to gextract the user's email
from the searchstring

  return
searchString.slice(searchString.indexOf("<>USER<>")+8,searchString.length);
}
```
getEmail takes the original searchString before the user was removed for the search engine. It takes the user's email that was sent by the website.

```
function securityTest(email){       //Function to check if the user is
authorised to access the stories
  var runs=ss.getSheetByName("Data").getRange("AG5").getValue();
  var flag=false;
  for(var i=0;i<runs;i++){
    Logger.log("SEC-CHECK-01: Runs-"+runs);
    Logger.log("SEC-CHECK-02: Email-"+email+"
Data-"+ss.getSheetByName("Data").getRange("AI"+(i+1)).getValue());

if(JSON.stringify(ss.getSheetByName("Data").getRange("AI"+(i+1)).getValue())==
JSON.stringify(email)){
      flag=true;
    }
    Logger.log("SEC-CHECK-03: Flag-"+flag);
  }return flag;
}
```
securityTest takes the email and sets runs to the amount of users. It loops through the registered users and checks if the email is one of theirs. If it is, we return true. The user cannot enter someone else's email as they need to sign in to their google account for their email to be attached.

<u>CSS on website</u>

```
html, body{
   background: #00a896;
   background: linear-gradient(to bottom, #028090 0%, #00a896 90%);
   height:100%;
   width: device-width;
   background-repeat: no-repeat, repeat;
   background-attachment: fixed;
   background-size: cover;
   margin:0;
   padding:0;
}
```
CSS gives the website its design. We set values for properties. Body is the main page of the website, with the background, size and margin.

Google Sheets

```
={'Form Responses'!A2:A}
```
The second sheet replicates the first because when the first gets updated it disrupts the formatting.

```
=INDEX(AF:AF, MATCH(99^99,AF:AF, 1)); =IF(ISBLANK(Users!A3),"",AH2+1)

=INDEX( FILTER(D:D, NOT(ISBLANK( D:D ))), ROWS(FILTER(D:D,
NOT(ISBLANK(D:D)))) )
```
The index functions check cells content. This first takes the bottom value of a growing list. The second takes the bottom-most text.

```
="<TI>"&S21&"<TI><DA>"&T21&"<DA><DP>"&U21&"<DP><AR>"&V21&"<AR><AU>"&W21&"<
AU><NO>"&X21&"<NO>"
```
Data from selected stories is placed in the chamber and concatenated into the statement that can be attached to the URL and sent to the website. We add extra characters in between to allow the string to be separated by the website.