Eric Remington Davey

Date: 11/31/2019

CS 441 Software Engineering

Midterm Personal Writeup

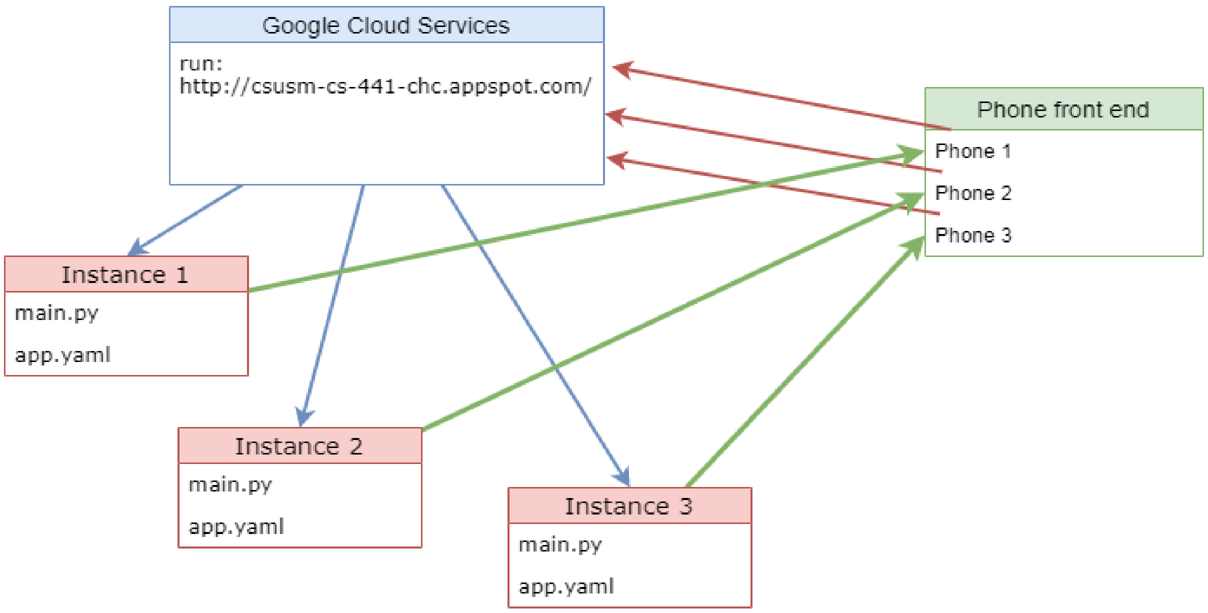
Group: Eric Remington Davey, Hanna Rumpler, Jasmine Doctolero, Michael McDermott, Ace Figueroa, Kaitlyn Vuong, Chris Bertram

When I began this project I had absolutely zero knowledge of how to implement a backend and zero knowledge of how to implement a front end. I had a good grasp of how both parts worked conceptually, as well as a good working knowledge of many programming languages (including designing a database in sequel) and a high-level understanding of how someone 'might' implement the two together. At the outset of the project I expressed very clearly my interest in learning to implement a back end with google cloud services, which I figured the learning and implementing of which would be my primary contribution to the project. This, however, by the end of the project did not end up being the case as I ended up participating in just about every aspect of the project in some way or another. So to start our group spend a great deal of time on researching documentation to figure out exactly how we would implement our app.

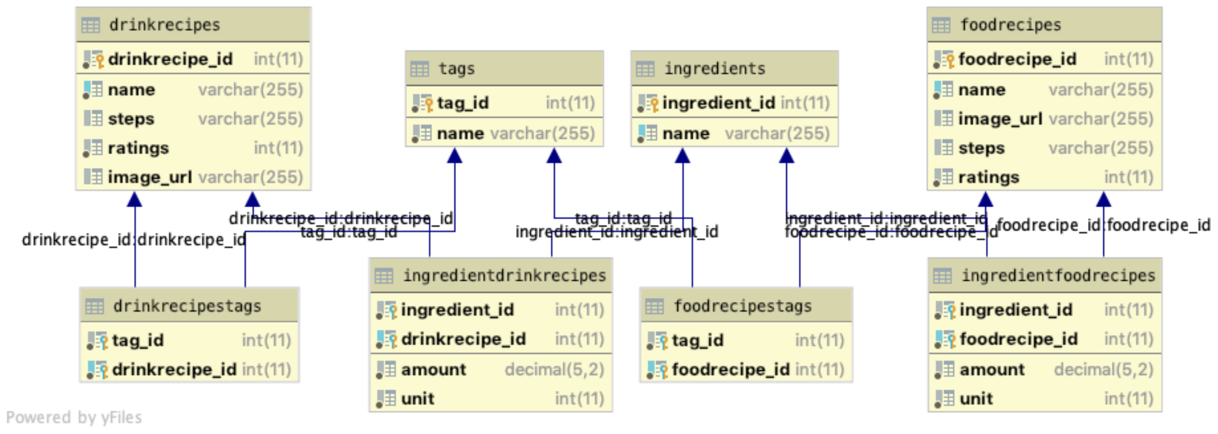
During the first 5 weeks of class I spent a good deal of time reading through documentation and learning about how google cloud services worked, and more specifically google-app engine. We figured that the first thing we should focus on was how the front end and back might need to communicate with each other and decided on using url-routes to implement that, so google app-engine was a rational choice. App-engine works by creating a unique instance for each user (which runs a script (in our case python file) and an app.yaml file) that connects to the Url, taking the line-ending route after the main url as an 'argument' which is passed to the main.py.

Figure 1 is shown below.

Figure 1



Once deciding on using Google app-engine as our source of communication between the back end and the front end we had to look into how we might implement a database, while researching I came upon technologies such as Django framework and Mongo DB. After consulting with the group we decided to implement our database in SQL as many of us had experience with SQL from our database management class. Conveniently, google cloud services provides a remote deployment option for an SQL database called 'Cloud-SQL', so we decided to go with that as our database host. Design of the SQL database is show below, this UML was made by Michael McDermott in our group.



The majority of the work I contributed to the backend other than research (and helping structure the initial ERD for the database) was figuring out how use a python script deployed on google app-engine to query cloud-SQL. Documentation and sample code was provided by Apache, links are provided here https://github.com/GoogleCloudPlatform/python-docs-samples/tree/master/appengine/standard_python37 Provided sample code from Apache for connecting to cloud-SQL via app-engine using Python.

And https://www.youtube.com/watch?v=w01H2CXp_7Y although it's using PHP it still explains how to set up your app.yaml to connect to Cloud SQL.

This concludes my contribution to the back-end of the project, as for the front-end…

Due to other life circumstances and a miscalculation on our part as to when the project was due, I decided to jump on the front-end team in the last few weeks of the semester to help get a working front end out and ready for presentation time. Once getting up to speed on the technology we were using for our front-end (HTML, css and javascript, deployed by Cordova). I helped to develop a few use-cases throughout the semester, however one which I developed while working on the front end is the following.

Name: Eric Remington Davey

Identifier Drink/Food Randomizer

Description Front end randomizes an ordered list of image url's and their names.

Goal To provide a list of randomized image url's and their names to be accessed at a later time by the app by retrieving them from local storage.

Preconditions

1. Upon successfully querying the back end for an ordered list of image url's and their associated names.

Assumptions (*Optional, List all assumptions that have been made)*

1. The App itself has loaded properly.
2. The list of names and image url's have been retrieved successfully from the back-end.

3. The JSON array is formatted properly to give the size of the JSON array and has the necessary components such as image-url's and drink/food names.

Frequency Only occurs once throughout runtime of the App.

Basic Course

1. Use case begins when the app is started and successfully loads the JSON array from the back-end.
2. An Array which is the size of the amount of results returned in the JSON array is created, and numbers from 0 to the length of the array are randomly populated.
3. The array of elements is used as a naming scheme and to designate the index of the array, i.e. drink name and image from index 1 from JSON is assigned a random number ex. "7" so drink name 1 and image 1 becomes drink name7 and image7.
4. Drink and images are placed into local storage under their new name tokens.
5. Use case ends when the app is done loading.

Alternate Course A: None

Post conditions

HTML document is ready and all onDocument ready javascript functions have finished executing.

Actors Both back-end and Front-end

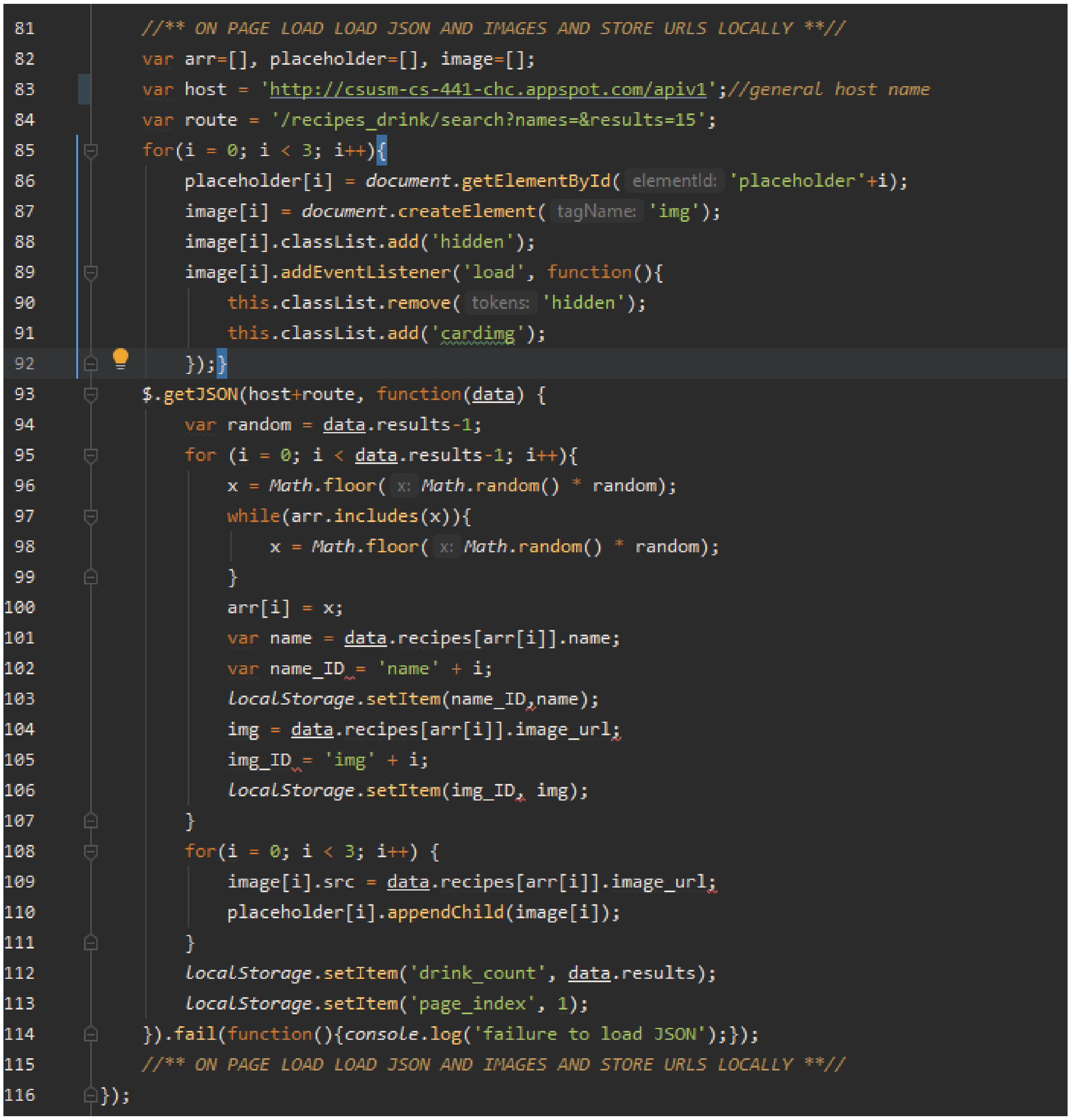
Included Use Cases (*Optional, List of use cases that this use case includes*)

1. All following use cases that require usage of navigating the displayed images.
2. Use cases that involve clicking on and searching for drinks displayed on screen.
3. Any use cases that involve retrieving data from local storage on the phone.

Notes

This use case is incredibly important to the navigation of the app as all images and their associated names are now storage under local storage under this naming scheme.

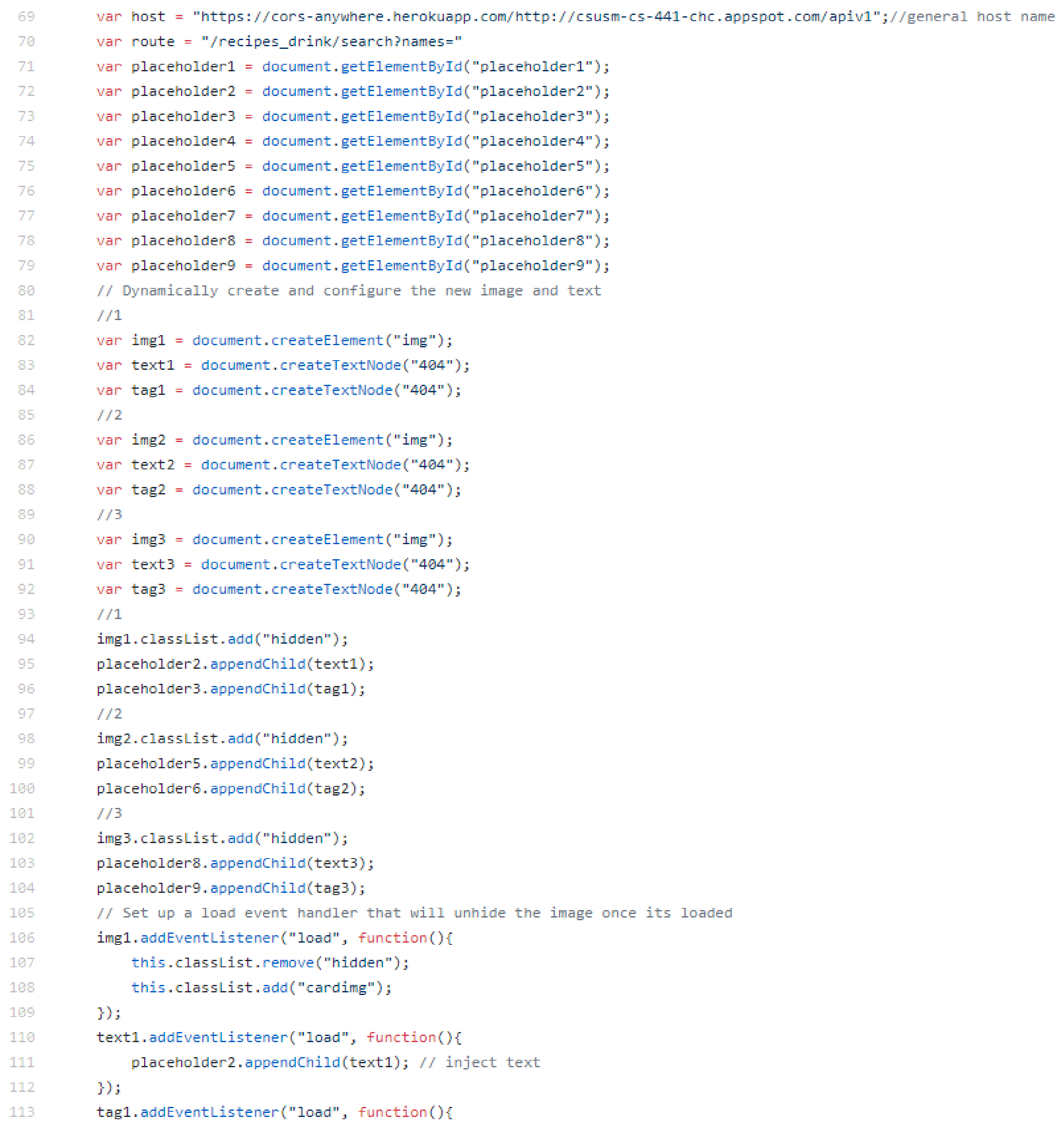
Implemented code for use case after refactoring:



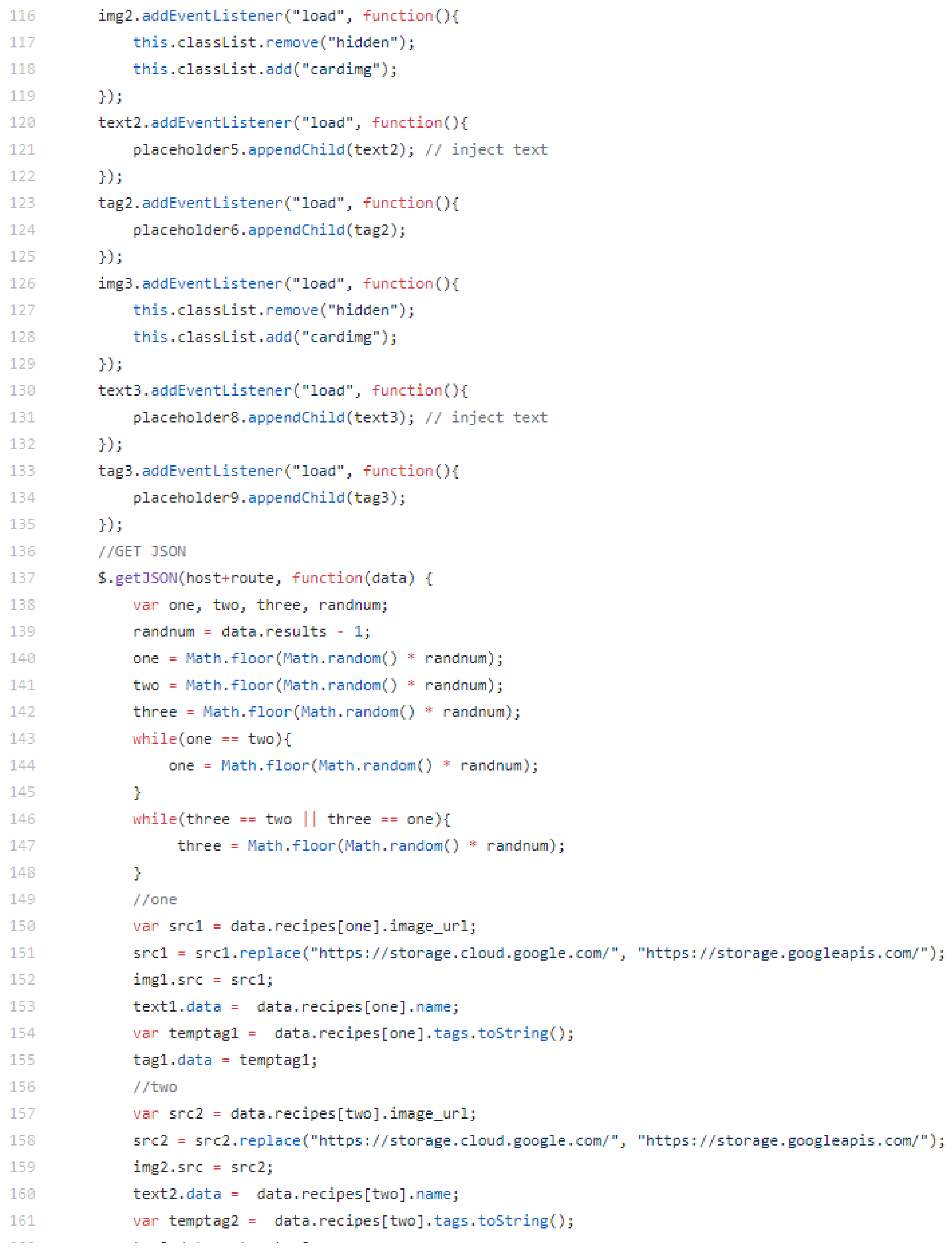
```
    //** ON PAGE LOAD LOAD JSON AND IMAGES AND STORE URLS LOCALLY **//
    var arr=[], placeholder=[], image=[];
    var host = 'http://csusm-cs-441-chc.appspot.com/apiv1';//general host name
    var route = '/recipes_drink/search?names=&results=15';
    for(i = 0; i < 3; i++){
        placeholder[i] = document.getElementById( elementId: 'placeholder'+i);
        image[i] = document.createElement( tagName: 'img');
        image[i].classList.add('hidden');
        image[i].addEventListener('load', function(){
            this.classList.remove( tokens: 'hidden');
            this.classList.add('cardimg');
        });}
    $.getJSON(host+route, function(data) {
        var random = data.results-1;
        for (i = 0; i < data.results-1; i++){
            x = Math.floor( x: Math.random() * random);
            while(arr.includes(x)){
                x = Math.floor( x: Math.random() * random);
            }
            arr[i] = x;
            var name = data.recipes[arr[i]].name;
            var name_ID = 'name' + i;
            localStorage.setItem(name_ID,name);
            img = data.recipes[arr[i]].image_url;
            img_ID = 'img' + i;
            localStorage.setItem(img_ID, img);
        }
        for(i = 0; i < 3; i++) {
            image[i].src = data.recipes[arr[i]].image_url;
            placeholder[i].appendChild(image[i]);
        }
        localStorage.setItem('drink_count', data.results);
        localStorage.setItem('page_index', 1);
    }).fail(function(){console.log('failure to load JSON');});
    //** ON PAGE LOAD LOAD JSON AND IMAGES AND STORE URLS LOCALLY **//
});
```

This code is executed on the document ready section, so it is done before the app is finished loading. The images are received from the back end and randomized and placed into local storage, the first three images are immediately displayed onscreen in the first three index cards, and navigation hereafter uses this naming scheme to retrieve image url's and their names from local storage.

This section of code before refactoring:



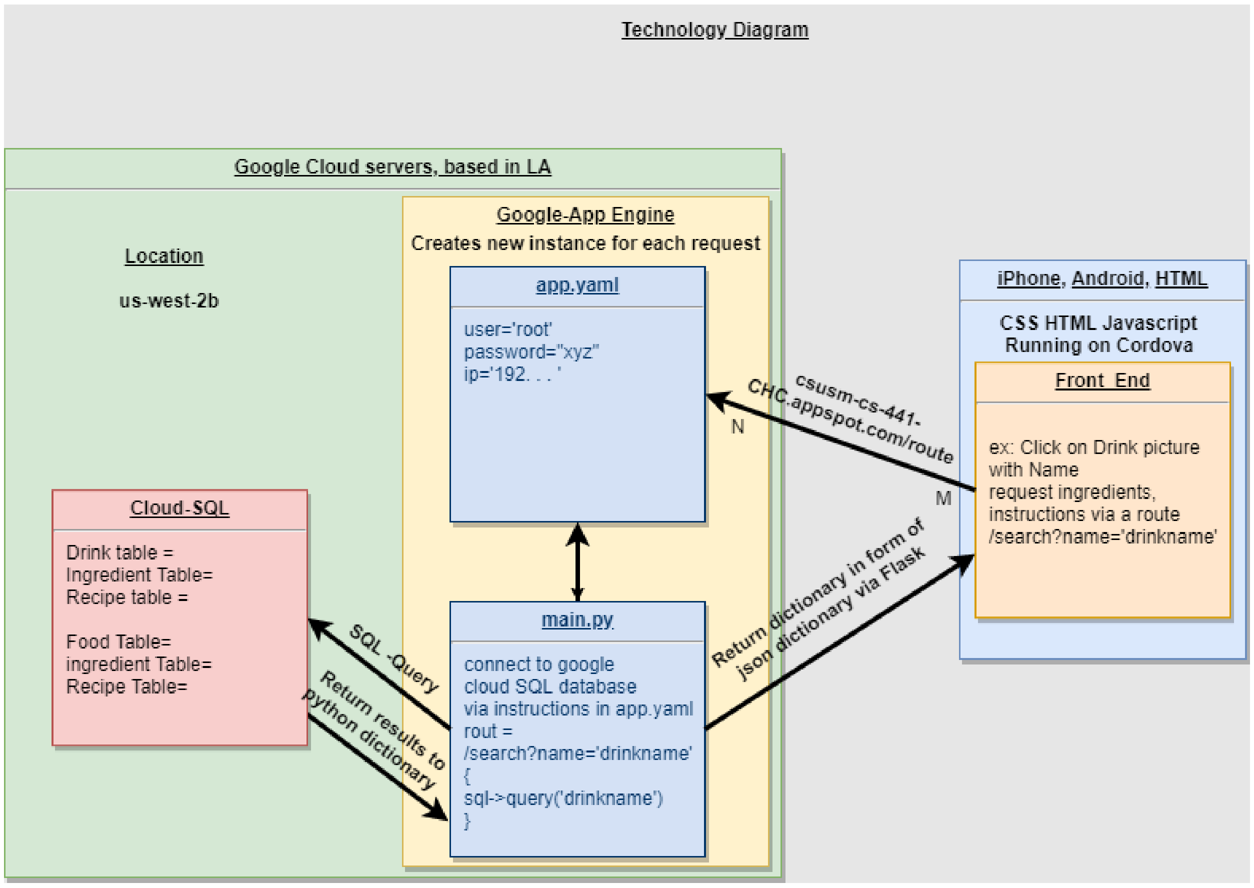
```
var host = "https://cors-anywhere.herokuapp.com/http://csusm-cs-441-chc.appspot.com/apiv1";//general host name
var route = "/recipes_drink/search?names="
var placeholder1 = document.getElementById("placeholder1");
var placeholder2 = document.getElementById("placeholder2");
var placeholder3 = document.getElementById("placeholder3");
var placeholder4 = document.getElementById("placeholder4");
var placeholder5 = document.getElementById("placeholder5");
var placeholder6 = document.getElementById("placeholder6");
var placeholder7 = document.getElementById("placeholder7");
var placeholder8 = document.getElementById("placeholder8");
var placeholder9 = document.getElementById("placeholder9");
// Dynamically create and configure the new image and text
//1
var img1 = document.createElement("img");
var text1 = document.createTextNode("404");
var tag1 = document.createTextNode("404");
//2
var img2 = document.createElement("img");
var text2 = document.createTextNode("404");
var tag2 = document.createTextNode("404");
//3
var img3 = document.createElement("img");
var text3 = document.createTextNode("404");
var tag3 = document.createTextNode("404");
//1
img1.classList.add("hidden");
placeholder2.appendChild(text1);
placeholder3.appendChild(tag1);
//2
img2.classList.add("hidden");
placeholder5.appendChild(text2);
placeholder6.appendChild(tag2);
//3
img3.classList.add("hidden");
placeholder8.appendChild(text3);
placeholder9.appendChild(tag3);
// Set up a load event handler that will unhide the image once its loaded
img1.addEventListener("load", function(){
    this.classList.remove("hidden");
    this.classList.add("cardimg");
});
text1.addEventListener("load", function(){
    placeholder2.appendChild(text1); // inject text
});
tag1.addEventListener("load", function(){
```



```
    img2.addEventListener("load", function(){
        this.classList.remove("hidden");
        this.classList.add("cardimg");
    });
    text2.addEventListener("load", function(){
        placeholder5.appendChild(text2); // inject text
    });
    tag2.addEventListener("load", function(){
        placeholder6.appendChild(tag2);
    });
    img3.addEventListener("load", function(){
        this.classList.remove("hidden");
        this.classList.add("cardimg");
    });
    text3.addEventListener("load", function(){
        placeholder8.appendChild(text3); // inject text
    });
    tag3.addEventListener("load", function(){
        placeholder9.appendChild(tag3);
    });
    //GET JSON
    $.getJSON(host+route, function(data) {
        var one, two, three, randnum;
        randnum = data.results - 1;
        one = Math.floor(Math.random() * randnum);
        two = Math.floor(Math.random() * randnum);
        three = Math.floor(Math.random() * randnum);
        while(one == two){
            one = Math.floor(Math.random() * randnum);
        }
        while(three == two || three == one){
             three = Math.floor(Math.random() * randnum);
        }
        //one
        var src1 = data.recipes[one].image_url;
        src1 = src1.replace("https://storage.cloud.google.com/", "https://storage.googleapis.com/");
        img1.src = src1;
        text1.data =  data.recipes[one].name;
        var temptag1 =  data.recipes[one].tags.toString();
        tag1.data = temptag1;
        //two
        var src2 = data.recipes[two].image_url;
        src2 = src2.replace("https://storage.cloud.google.com/", "https://storage.googleapis.com/");
        img2.src = src2;
        text2.data =  data.recipes[two].name;
        var temptag2 =  data.recipes[two].tags.toString();
```

As you can see, this use case benefitted quite a bit from refactoring.

Finally, I developed a working diagram to explain the technologies that ended up implementing for the entirety of our project, it's shown below.


Technology Diagram
Google Cloud servers, based in LA
Location
us-west-2b
Google-App Engine
Creates new instance for each request
app.yaml
user='root'
password="xyz"
ip='192. . . '
main.py
connect to google cloud SQL database via instructions in app.yaml
rout = /search?name='drinkname'
{
sql->query('drinkname')
}
Cloud-SQL
Drink table =
Ingredient Table=
Recipe table =
Food Table=
ingredient Table=
Recipe Table=
SQL -Query
Return results to python dictionary
iPhone, Android, HTML
CSS HTML Javascript
Running on Cordova
Front_End
ex: Click on Drink picture with Name
request ingredients, instructions via a route
/search?name='drinkname'
csusm-cs-441-CHC.appspot.com/route
N
M
Return dictionary in form of json dictionary via Flask

Conclusion:

Throughout the semester our team encountered many difficulties whether it be scheduling, non-class related issues, or confusion and miscommunication, however, having the team on a goal oriented style of teamwork such as Agile really helped to overcome and improve a lot of our weaknesses, communication got better, coordination got better and scheduling of our goals became more clear the more meetings we had. We started the semester with lofty goals and we fell short on quite a few of them, but we managed to hit our main goals of implementation by the end and we now have a list of use-cases of which can be implemented in the future. If I were to implement something next it would be to allow the user to retrieve information from the list of returned items from a search and to help the back-end implement a search-by ingredients route to our database.