DEPARTMENT OF COMPUTER SCIENCE

COLLEGE OF ARTS AND SCIENCES

CSCI 4961/4962 Capstone Deliverable #2

Title of Project: Library Book Finder

Client: Lee Cummings

Supervisor: Dr. Jacob Sukhodolsky

Student(s): Sara Jain, Lorna Xiao

Project Overview

Our project, “Library Book Finder”, will allow students and other individuals to locate a book in the library. It will primarily focus on enabling user-friendly accessibility to shelves that correspond to library books through Google Maps or a text message with a link to Google Maps.

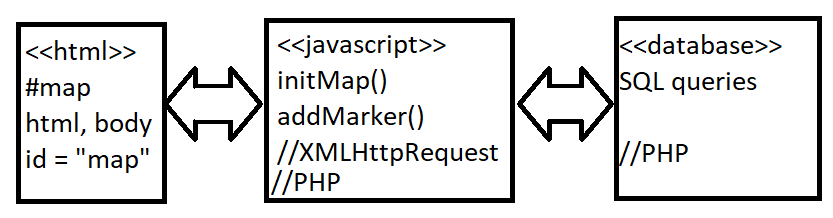
Basic Functionality

When an individual finds the book he or she would like to locate on the library website, there will be a button that prompts two options for users: Send a Text Message or Link to Google Maps. The “Send a Text Message” will allow the user to provide their phone number and get a text with the book’s call number and link to Google Maps. Maps will indicate a given shelf location and the user will be able to navigate through the Maps app to that shelf (with their real time location or predetermined paths). With the call number, they can find the location of the book on the shelf. The other option, “Link to Google Maps” will take the user directly to Google Maps on their current device and show him or her exactly where the shelf (containing the book) is located. The library has these buttons ready or they can be quickly completed. Our project focuses on creating the Google Maps portion and connecting it to the Library site.

Major Components

* Google Maps API
  + HTML/JavaScript/PHP/Google Maps API/Directions API
  + Basic interface on Google Maps, used to create markers
  + The Maps API will receive a call number from Library Catalog Button. It then sends that call number over to DB to query. It will receive coordinates from DB and output a marker on the shelf location of the call number. Then, it directs user to that shelf based on a predetermined path using Directions API. The map will be a link that is sent over to the library site.
* Call Number Database
  + SQL/PHP
  + Holds all call number and shelf location information.
  + The database receives a call number from the client side, queries it for a shelf range. Then, it will send coordinates/floor location to the client side.
* Library Button
  + PHP
  + Holds the user’s desired call number and link to our Google Maps API
  + The button will send the call number to the Maps API and receive a link to Google Maps. The link will include marker and path information.
  + Currently, the library has set up a separate button/link for testing purposes. It will be used to test the database queries.

UML Diagram

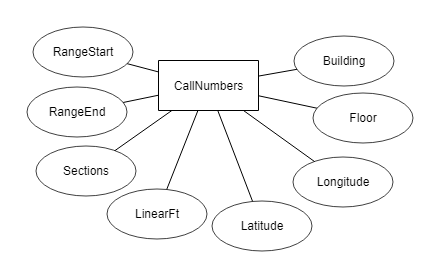


<<html» simply creates a basic webpage and sets the size of the map. We have given an id to our map called “map”.

<<javascript>> holds all of the information to initialize and to interact with the map. It also has the link to the Google Maps API with our API key. We are currently writing the PHP code to send/receive data from the database.

<<database>> holds a table called CallNumbers with all call number ranges and coordinates. We are currently writing the SQL queries and the PHP to get/send information.

ER Diagram



CallNumbers is the table, and each attribute is a column in the database. There are no foreign/primary keys. We only have one table containing all of the shelf information, so there are no relationships with other tables.

Google Maps API Code

//can create an array of markers

var markers = [

{

coords:{lat:38.637111,lng:-90.234447},

content:'<h1>Range: PA 6137.A-PA 8034.Z</h1>'

}

];

//add markers to the map

function addMarker(props){

var marker = new google.maps.Marker({

position:props.coords,

map:map

});

}

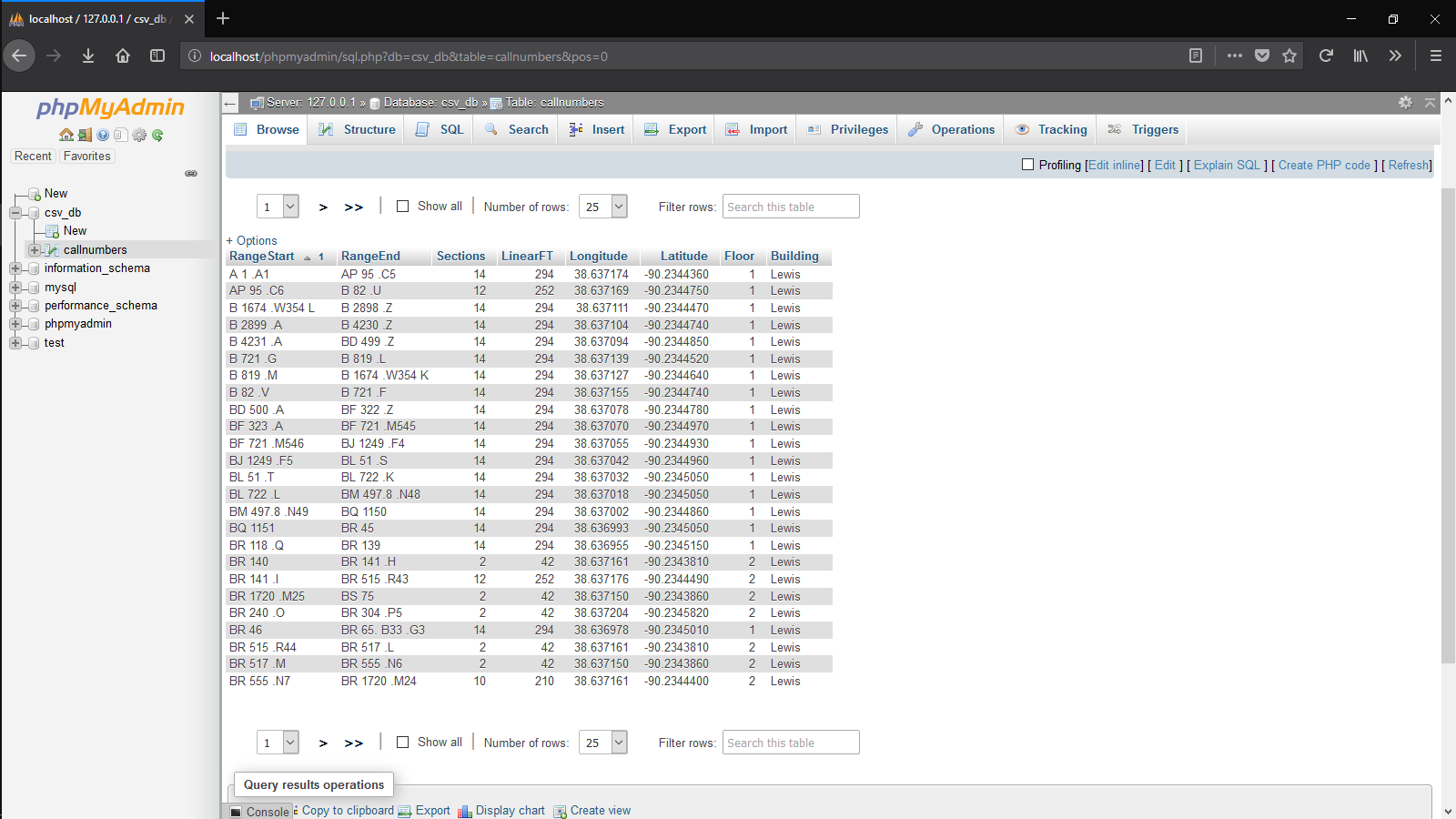
//connects to Google Maps API and displays map

<script async defer src="https://maps.googleapis.com/maps/api/js?key=AIzaSyDKm79Pba4nX72e6z6bfXcdBGRpZs\_A1KY&callback=initMap">

</script>



Database

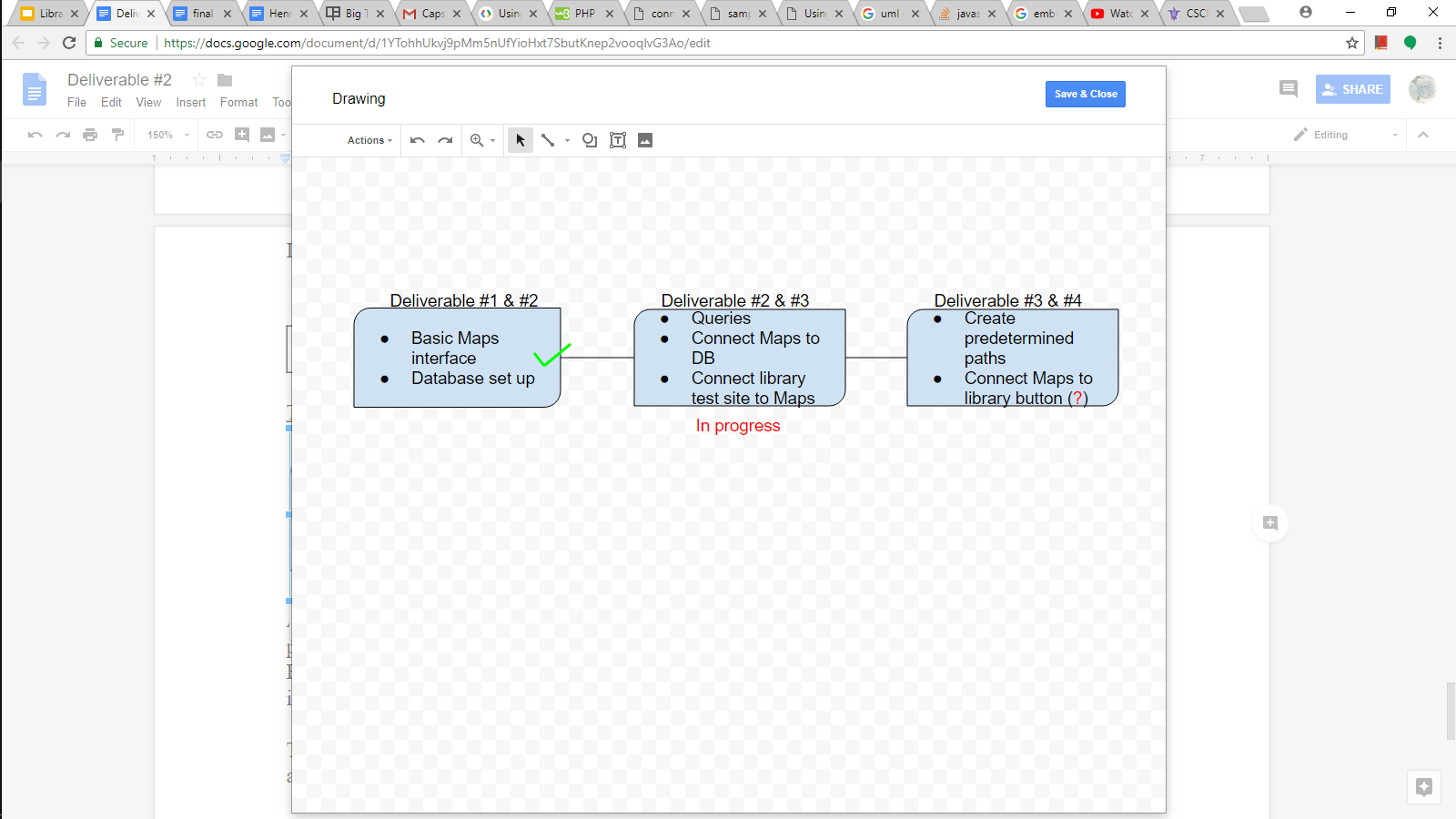


Formal Requirements

The following list shows the necessary requirements to create a successful program that meets the goals of the project.

|  |  |  |
| --- | --- | --- |
| Requirement | Description | Progress |
| 1A | Overlay on top of the bird’s eye view of the library floor plans on Google Maps | Completed |
| 1B | Overlay must be clickable (interactive) so that the user can see the call number ranges on each shelf  Unnecessary from a user’s perspective. Only need a marker and a path. | Removed |
| 1C | Overlay must permit zoom in/out feature | Completed |
| 1D | Overlay must function for each of the floors with shelves | In Progress |
| 2 | The program will be written in Javascript using the Google Maps API | In Progress |
| 3A | For every call number range (shelf) in the Excel sheet, longitudes and latitudes of the shelf must be included in two separate columns | Completed |
| 3B | Coordinates used as longitude and latitude are matched with the proper shelves as to be used to create markers on Google Maps | Completed |
| 4 | Excel sheet converted into a database that allows the clients to modify and update library information content (call number ranges, latitude and longitude incase they move the shelves) | Completed |
| 5 | Javascript program should use the database to find (query) the range of call numbers (shelf) based on an inputted call number | In Progress |
| 6A | When a user requests book info, the call number must be extracted from the button on the library site through a link | On Hold |
| 6B | The call number is then parsed through the database to check which call range number it belongs to | In Progress |
| 6C | The corresponding coordinates for that range (shelf) is outputted and sent to Google Maps to pinpoint location. | In Progress |
| 7A | A marker must be placed at that location | Completed |
| 7B | User should be able to navigate to reach destination. | Not started |
| 8A | Navigation will either consist of real-time movements of user to shelf or predetermined paths a user can take | Not started |
| 8B | For predetermined paths, user inputs current floor and destination floor | Not started |
| 9A | Send all database information over to the library | Not started |
| 9B | Send our Maps API over to library and makes sure it outputs a link to Google Maps | Not started |

Timeline



After each step, we test the major components (depending on which) to make sure it works properly and continue onto the next step.

For example, we tested between each stage as we wrote our javascript code: marker, infoWindow, multiple markers.

There is a (?) next to *Connect Maps to library button* because we might not have permissions to access that code or portion of the library site.

Issues/Potential Problems

* Change in Google Maps API pricing and billing policy (goes into effect June 11, 2018)
  + Up to $200 free usage, requires a billing account
* Client may need to create their Google Maps API account and generate a key for their use
  + Currently uses our personal Gmail accounts
* We may not have access to the official button on the library site
  + We will have to send the database and our Maps code over to the clients

Current Task:

So we will continue working on the PHP and SQL for the database and connection during deliverable #3. A few google searches makes the PHP quite simple and easy to do. We were able to contact Dr. Sukhodolsky as well for his PHP notes and advice. The task at hand is actually sending information from the javascript side to the database. The Google Maps API tutorials have some help and sample code for making connections, which I will use to complete this. However, if there is a more efficient way, then I will implement that. The tutorials use PHP and XML, whereas other tutorials elsewhere online say you only need PHP.

Sample of the code from the Google Maps API tutorial:

This is javascript code that returns the information from an XML file. The PHP that pulls information from the database returns this XML file. I want to find a way to directly pull information from PHP rather than through an XML file since the data will be very small.

downloadUrl('https://storage.googleapis.com/mapsdevsite/json/mapmarkers2.xml', function(data) { // Change this depending on the name of your PHP or XML file

var xml = data.responseXML;

var markers = xml.documentElement.getElementsByTagName('marker');

Array.prototype.forEach.call(markers, function(markerElem) {

var id = markerElem.getAttribute('id');

var name = markerElem.getAttribute('name');

var address = markerElem.getAttribute('address');

var type = markerElem.getAttribute('type');

var point = new google.maps.LatLng(

parseFloat(markerElem.getAttribute('lat')),

parseFloat(markerElem.getAttribute('lng')));

var infowincontent = document.createElement('div');

var strong = document.createElement('strong');

strong.textContent = name

infowincontent.appendChild(strong);

infowincontent.appendChild(document.createElement('br'));

var text = document.createElement('text');

text.textContent = address

infowincontent.appendChild(text);

var icon = customLabel[type] || {};

var marker = new google.maps.Marker({

map: map,

position: point,

label: icon.label

});

marker.addListener('click', function() {

infoWindow.setContent(infowincontent);

infoWindow.open(map, marker);

}); }); });}

function downloadUrl(url, callback) {

var request = window.ActiveXObject ?

new ActiveXObject('Microsoft.XMLHTTP') :

new XMLHttpRequest;

request.onreadystatechange = function() {

if (request.readyState == 4) {

request.onreadystatechange = doNothing;

callback(request, request.status);

} };

request.open('GET', url, true);

request.send(null);}