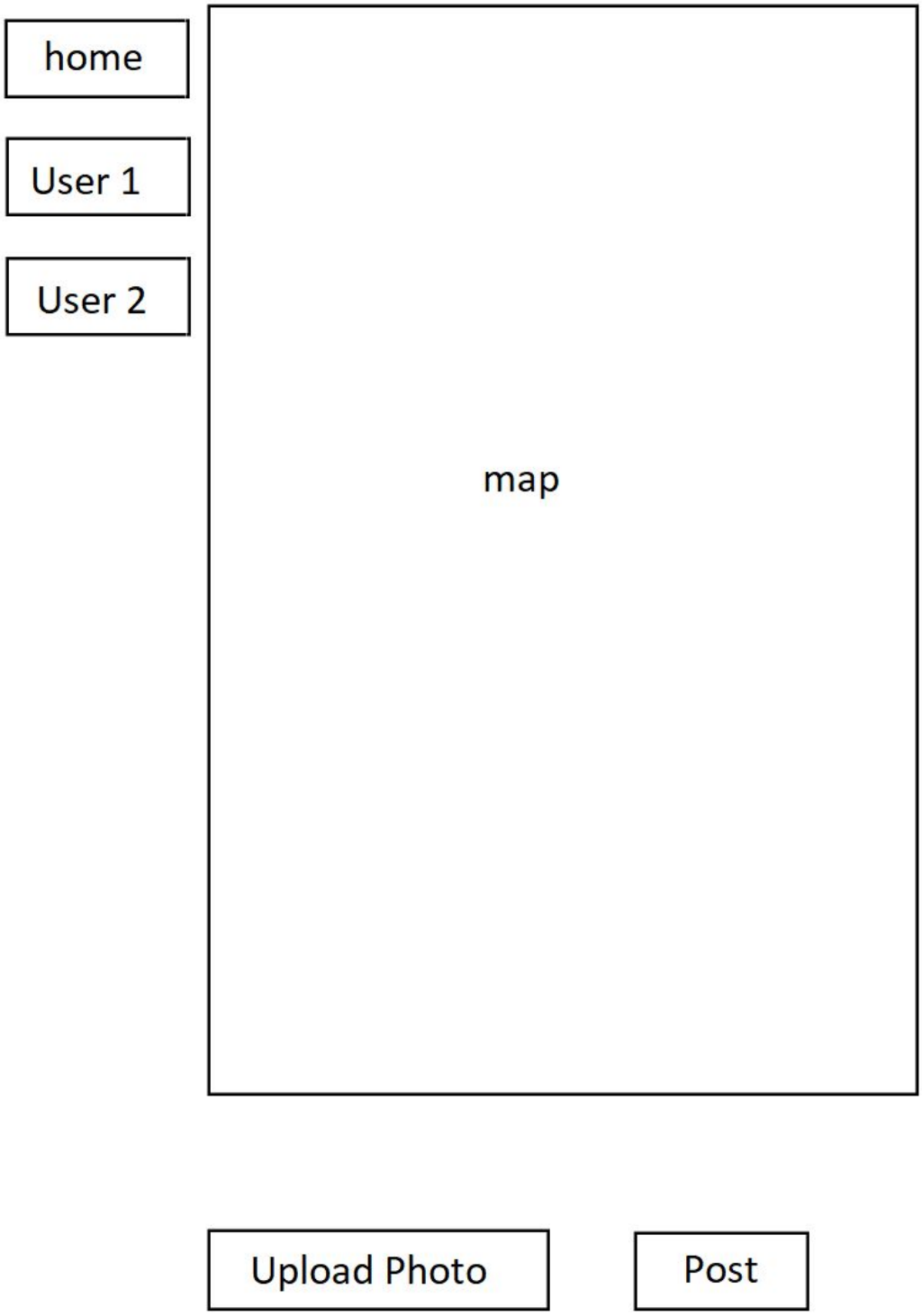
home
User 1
User 2
map
Upload Photo
Post

Demo

- I put the demo code on the github, it should work if you just open the html file in the browser
- You have to add a working api key `AIzaSyATCXX-X0rDI84Y44XIASkNMrzkmibbVzI`

Assumptions

- We forget about being able to remove markers in the prototype
- Just posting images and viewing images at flags, that's it
- Maybe we forget about layers as well for now (everyone's flags can be seen and there isn't much distinction between them)

Google map api dependency

Load the google map api dependency in the html file.

```
<script

src="https://maps.googleapis.com/maps/api/js?key=AIzaSyATCXX-X0rDI84Y44XIASkNMrzkmibbVzI&callback=initMap&libraries=&v=weekly"
      defer
    ></script>
```

Note: notice the callback=initMap portion of the url, this specifies which function initializes the map in our javascript code.

REST API call when the user enters this fragment

1. **getMapGroupInfo()**

Explanation: We need to grab all the info about the MapGroup to populate the front-end. Origin, radius, all images bundled with the user who posted them.

Assumption: User has the MapGroup ID implicitly in a cookie or something

a. Called by API gateway

b. Communicates with MapGroup service via REST calls to retrieve all media images, their associated coordinates, and who posted them

c. Communicates with the Map service via REST calls to retrieve the **origin** and **radius** of the map

Note: The MapGroup service could communicate with the Map service instead of the API gateway doing it directly, whichever we decide

Information returned

- Radius and origin of the map
- Members of the group
- All images and their associated posters

Example using the retrieved Origin and Radius

- Below is a function that initializes the map in the front-end inside index.js
- Notice the “center: **ORIGIN**” parameter, this is required and sets the center of the map.
- Notice the “ltLngBounds: getBounds()” parameter, get bounds takes the **ORIGIN (lat, long) and radius** as the third parameter. This is a function we defined. This restricts the area of the mapgroup. The lat / long is part of the **ORIGIN** value but in this example is hard coded in
- This code when run will populate the giant box labeled “map” on the first page with a google map

```
function initMap() {

map = new
google.maps.Map(document.getElementById("map"), {
    center: ORIGIN,
    restriction: {
      latLngBounds:
getBounds(44.541355555,-77.372986, 10),
      strictBounds: false,
```

```
    },
    disableDefaultUI: true,
    zoom: 15,
    styles: myStyles,
  });
}
```

Example of how to use the retrieved images to create all the markers on the map

This code does the following,
1. Extracts the longitude and latitude from the image
2. Creates a marker object (flag on the map)
3. Creates an info window (a window that pops up with the picture)
4. Sets an onClick listener function on the marker object so that the information view opens when you click it
5. Associates the **image** as the content to be displayed in the information view

```
window.addMarker = function(img) {
latValue = getLat(img);
longValue = getLong(img);
var mark = new google.maps.Marker( {
position: {lat: latValue, lng: longValue},
map,
title: "Test",
});
```

```
const iw = new google.maps.InfoWindow({
   content: img,
 });

 mark.addListener("click", () => {
   iw.open(map, mark);
 });

mark.setMap(map);

}
```

So, we could loop through all the images we retrieve and call a function similar to this.

The functions getLong(img) and getLat(img)

I used this dependency / library to extract the exif data.

```
<script
src="https://cdn.jsdelivr.net/npm/exif-js"></script>
```

This is the function that grabs and converts the image that a user uploads to the file/input field in the current demo I posted on github. It pushes the image into an array of

images.

```
window.setImage = function() {

  const file =
document.querySelector('input[type=file]').files[0];
  const reader = new FileReader();
  reader.addEventListener("load", function () {
    // convert image file to base64 string
    //img.src = reader.result;
    var nImage = new Image();
    nImage.src = reader.result;
    imgList.push(nImage);
  }, false);
  if (file) {
    reader.readAsDataURL(file);
  }

}
```

With the image set in the array, the following function extracts all the meta data from the image, and assigns it to a global variable I called **imgData**

```
function setExif() {

if(imgCounter > 0) {
imgData.exifdata = null;
```

```
}
EXIF.getData(imgList[imgCounter], function() {
imgData = imgList[imgCounter].exifdata;
//  imgData = img.exifdata;
  console.log(imgData);
});

}
```

Now with the **imgData** variable set, we can call this next function which extracts certain parts of information from **imgData** and calls another function which converts it finally into a **longitude value** and a **latitude value** that we use when we make a **marker object**.

- At the end of the function I set the latitude and longitude to two global variables (testLat and testLong) that are used by the AddMarker() function, just for testing
- The function also calls a subroutine (ConvertDMSToDD that I copied off the internet) to convert the meta data

```
function setCoords() {
  var latDegree = imgData.GPSLatitude[0].numerator;
  var latMinute = imgData.GPSLatitude[1].numerator;
  var latSecond = imgData.GPSLatitude[2];
  var latDirection = imgData.GPSLatitudeRef;

  var lonDegree = imgData.GPSLongitude[0].numerator;
  var lonMinute = imgData.GPSLongitude[1].numerator;
```

```
  var lonSecond = imgData.GPSLongitude[2];
  var lonDirection = imgData.GPSLongitudeRef;

  testLat = ConvertDMSToDD(latDegree, latMinute,
latSecond, latDirection);
  testLong = ConvertDMSToDD(lonDegree, lonMinute,
lonSecond, lonDirection);

  console.log("LAT: " + latFinal);
    console.log("LONG: " + lonFinal);
}
```

Conclusion / TODO:

1. We need the REST API GET call that retrieves the origin for the MapGroup.

2. We need the REST API GET call that retrieves the radius for the MapGroup.

3. We need the REST API GET call that retrieves all [image, poster] or just images associated with the MapGroup.

- Once we have these calls, I think it will be easy to use that data to populate the front-end mapgroup fragment

4. We may need a REST API GET call that retrieves the different users in the MapGroup. On the front-end we could just use this to put a text box with their name near the map, we could skip the

layer functionality for now.

5. We need the REST API PUT/POST call that sends a new image to the MapGroup/Map services.

- This call could just be embedded in the function that makes a new marker object.

6. Need a basic front-end structure for this fragment