FDP MAPS APPLICATION TO-DO LIST

1. Fix controls column so that it also takes 90% of view height. Match height and align button elements to the bottom. The space above the buttons doesn’t matter, can be pixels or percentages. as long as the buttons are aligned to the bottom.
   1. Considering adding rows for each section in the user interface column and setting the space each row takes up. Thereby allocating the space for each piece similar to the way elegant themes DIVI does.
   2. Then try to align last row of the UI column to the maps row. So, align the last row to end at 90% of the view height.
2. Refactor reading csv so that it reads a header line. The first line that is read is simply a header containing the data order which can be parsed.
   1. Consider reading the file on change so that the drop down can be populated when they click on the file as oppose to click submit.
3. Populate select dropdown with header data. Populate data order with header as well.
4. Enter color widget
   1. Min and max color and value limits. Color values are entered as hex values, limits are numerical values. If have time, do type checking for every input field on the maps application. Also, do semantic validation checking on evert input field (i.e max field should be larger than min field).
   2. For html5 colornames, consider creating a html element (no need to append) and using getComputedStyle to return the hex value of the colorname. Perhaps use exception handling to only run the getComputedStyle logic if the hex name fails (meaning an invalid input, possibly a colorname).
   3. Keep an empty html rectangular div. Change background color of div to from min limit color to max limit color.
5. Get rid of the RUN button and refactor the file onchange handle to execute the current RUN’s buttons current code. Place a “Generate Graphs” button in its place instead.
6. Use the area above the last button (currently RUN) to programmatically display error messages associated with user input. Either validate input before trying to use, or catch errors. Validating (and passing something on if not valid) before trying to use and probably an easier idea.
7. Make squeals as beach flag images. When adding markers check if squeal is true, if so, run addFlag(), else run addCircle(). Place member properties on all the markers being pushed so that you can tell if it is a flag or a circle. Check this type property in the Gradient functions to make sure not trying to change the gradient color of a flag.
8. Implement speed and average acceleration
9. Split up JS into multiple files that are each loaded in the index.html
10. Add in D3.js to generate graphs.
11. Download all libraries and link locally before you pass on the application to them. Also make sure that they get their own API Key.