Web User-Interface Development History

<insert update here>

Update : August 2018

Overview:

webUI development was significantly adapted by CAM2 API database that was released around mid 2018. webUI team intended to develop a system that automatically updated WebUI Google Fusion Table based on change log in the CAM2 API but due to consideration under subject#4 the manual method was implemented instead. Other than that, interface improvement was done based on updating libraries and frameworks used by the web application and preparing the visualization interfaces for running computer vision module. More details of the development was written on this document.

API Camera Database & Google Maps Display:

Initial Problem:

When the project was first inherited this summer, the website had a google maps page that displayed cameras from the older database (database that was not on MongoDB and the original version of the CAM2 API database). It did not check for updates to the database and was not able to display live camera footage. A lot had changed since this was last updated, as the research team now has an accessible API.

Accomplishment:

- Updated google Fusion table and Google spreadsheet with cameras from CAM2 API database (MongoDB).
- Re-write GoogleAppsScript for sync process between Google spreadsheet and Google Fusion Table due to execution time limit being reached.
- Implemented Modal dialog box to display cameras with its metadata.
- Transitioned from Bootstrap 3 to Bootstrap 4
- Fixed Travis Configuration to automate github pages deployment

- Built a script to fetch all metadata from CAM2 API database to local spreadsheet and upload it to an existing Googlesheet file.
- Added collaborators and sponsors page
- Improved website dynamic feature by adding databases to elements on the web.
- Added Application form (including additional Google Form) on the web.
- Added Developer section on the web.
-

Long-Term Goal:

The following Goals was established during this period :

- Display real time metrics of the cameras current location including local time and weather
- Running Computer Vision Module to Analyze image (and videos) in real time while providing metrics output on the Modula dialog box.

Encountered issues/Decision Making

The following were the Issues we encountered that we had to solve:

Subject #1:Google Fusion Tables Vs. Django Models Vs. PostgreSQL

Problem: A decision had to be made as to how we should configure our database from information pulled from the API.

Explanation: it was stated in the webUI documentation, using Google Services will improved the performance of rendering google maps markers significantly at a large scale.

Subject #2: Google Fusion Table and GoogleSheet

Problem: Updating a google fusion tables is a little complicated because you can't directly update a fusion table without a google account. Therefore the only to update the camera fusion table was by uploading a csv file to google drive and from there converting it into a fusion table.

Explanation: Google Fusion Table was built to be integrated with GoogleSheet. Thus, the way to update Fusion Table is by updating a GoogleSheet. This way is also efficient because there would be no need to write a script for handling Google Oauth.

Subject #3: ChromeDriver for Travis CI

Problem: When testing the application using `python manage.py test app`, local test will run on multithreading (at least on most computers) while Travis will run it asyncrohonously due to spare resources. During trial and error, it was found that chromedriver perform faster on Travis than geckodriver (Firefox).
Using geckodriver, some test would receive `webdriverexception` or `broken pipe error`. One of possible cause would be caused by loading/closing browser by the webdriver

Solution: use chromedriver at all time. If you would like to use geckodriver, then use it locally. Expect inconsistent errors when you have geckodriver on TravisCI.

Subject #4: webUI fusion table automation vs. manual updates
After the release of CAM2 API, the webUI team decided to develop a process of updating Fusion Table with any changes occur in the API database. 2 considerations were made:

1. Automation with CRON scheduler
2. Manual update with single command line

1. Automation with CRON scheduler
There were 3 methods of implementation:

- Using Heroku CRON add-ons
  - Pros: straightforward implementation
  - Cons: required additional cost, potentially miss work schedule
- Configuring CRON Manually by code
  - Pros: reduce the risk of missing work schedule
  - Cons: required additional cost, extra work needs to be done
- Configuring CRON on the HELPS Machine
  - Pros: no extra cost required
  - Cons: needs continuous run on HELPS machine, HELPS maintenance might cause extra problems.

2. Manual Update with signle command line
To have a script that interface the API client and populate Googlesheet with the cameras metadata from the CAM2 API.

The team decided to implement manual update until there is a need for automation, mainly due to the infrequent updates from the CAM2 API.

Major Bug Fixing - Common Bugs in Web Development

Issue: Github/Google OAuth process (the "Sign in with…" button) leads to an error page from Github/Google...

- On the live site - Github Solution:
    1. For Github, you'll need access to the CAM2 WebUI Git account. Navigate to the profile menu (with the picture) in the top right corner -> Settings -> Developer Settings and select the app that matches the CAM2 site.
    2. Make sure the field Homepage URL matches https://cam2project.net exactly.
    3. Make sure the field Authorization callback URL matches https://cam2project.net/oauth/complete/github exactly.
- On the live site - Google Solution:
    1. You'll need access to the CAM2 WebUI Google Account. Navigate to console.cloud.google.com and select the project that matches the CAM2 site.
    2. Navigate to APIs & Services -> Credentials using the menu bar on the left.
    3. Select the OAuth2 client ID that matches the live site
    4. Make sure that one of the Authorized callback URLs matches https://cam2project.net/oauth/complete/google-oauth2/ exactly (DON'T FORGET THE TRAILING SLASH).
- On the local deploy - Github Solution:
    1. You'll need to create a project for the local deploy to test the OAuth process on your machine. On Github, navigate to Settings -> Developer Settings and click New OAuth App
    2. Give it a descriptive name, then make sure that Homepage URL matches http://127.0.0.1:8000 (not https, and don't forget the port number) and that Authorized callback URL matches http://127.0.0.1:8000/oauth/complete/github exactly.
    3. Click Register Application to be given a Client ID and Client Secret. You'll need to export these as environment variables in the terminal you use to run the local deployment of the site.
    4. On the command line, type export GITHUB_KEY=YOUR_CLIENT_ID, replacing YOUR_CLIENT_ID with the Client ID listed on the application overview page.

5. On the command line, type export GITHUB_SECRET=YOUR_CLIENT_SECRET, replacing YOUR_CLIENT_SECRET with the Client Secret listed on the application overview page.
6. Test the Sign in with Github button on the local deployment site.

- On the local deploy - Google Solution:
    1. You'll need to create a project for the local deploy to test the OAuth process on your machine. In the Google Developer Console, click on the Select Project button near the top of the window and click New Project. Give the project a descriptive name.
    2. Navigate to APIs & Services -> Credentials and click Create Credentials -> OAuth Client ID
    3. Select Web application, then enter http://127.0.0.1:8000/oauth/complete/google-oauth2/ into the Authorized redirect URLs field. Make sure it matches.
    4. Click create and open up the descriptor page for the credentials to get your Client ID and Client Secret.
    5. In terminal, type export GOOGLE_LOGIN_KEY=<YOUR_CLIENT_ID>
    6. In terminal, type export GOOGLE_LOGIN_SECRET=<YOUR_CLIENT_SECRET>
    7. Test the Sign in with Google Button on the local deployment.

Issue: deleting a recent commit that is not pushed
Solution:

- Run git log and determine SHA key you want to revert to
- Run git reset --hard SHA

Issue: Postgress connection reached maximum limit.
Description: The postgressSQL that you use in development is usually the one used for the staging site. If many people involved in the webUI development, most likely you will reach the maximum connection limit of the DATABASE_URL.
Solution:

(must be done by someone who has access to heroku staging site)

1. Go to Heroku dashboard.
2. Find the associated postgresSQL that you are using in development to get the application name
3. Do the following:

```
$ Heroku git:remote <heroku application name>
$ Heroku pg:killall # to kill all connections

# alternatively, you can restart the application. Some claimed that
it would kill all connections
$ Heroku restart --app <heroku application name>
```

Issue: Migration merge conflict