Project Overview

This project would be a continuation to build the Tobacco Retailers Permits project to completion- including a user interface for the LMPHW staff who will be conducting enforcement.

LMPHW seeks a way to identify potential retailers and confirm they have paid and received a permit in accordance with the smoke-free law. There are no comprehensive data sets of tobacco retailers or other data sets that may provide a "population" to check the permit data against. Therefore, LMPHW currently has no proactive mechanism to confirm that all retailers are permitted. Instead, enforcement is based on retailers self-identifying and purchasing permits, or reports from Food/Facilities Inspection staff that tobacco is being sold and no permit is present. It is also based on inspection follow-ups from complaints from the general public to LMPHW regarding tobacco retail without a permit.

During the Fall 2023 semester, capstone students were able to build scripts to conduct image processing for retailers, along with scraping websites of retailers to look for indications they were selling tobacco products. Additional work is needed to be able to compare the results of this analysis with the existing database of permitted retailers and identify mismatches—those retailers who seem to be selling tobacco via the image processing/web scraping results, but do not have an active retail permit. We also need an interface to serve results up to our enforcement team so they can visit retailers, reach out to their owners/managers and bring them into compliance.

Initial Implementation

The initial implementation of the user interface of the project was done using ReactJS.

- The code base for this implementation exists here: https://github.com/zsorens/Louisville-Tobacco-UI
- This code currently runs the data through that is saved in a CSV that is stored in the repository.
    - To change the data that is being run, you can either update the CSV (import a new one into the repository), connect the backend services via API, AWS lambda functions, or other similar options.
    - The reason that a CSV is currently used to display and store the data is due to limitations with the Spring '24 groups access to the data. The group did not receive data until well into the project, and then had many updates to do on that code base that was given from the last group.
- Though the UI's currently functionality is limited, it can easily be deployed to AWS, or other web hosting sites. In the next section I will discuss how to do that.

Hosting the Site

- Once a domain name has been purchased and registered, using many different sites, I prefer AWS Route 53 (https://aws.amazon.com/route53/) Here, you can purchase and maintain a domain name that the stakeholders deem fitting for the service.

- Next, ensure that the GitHub repository is hosted in a location where the Louisville Metro Public Health and Wellness department has access and admin ownership. Then following the step below, the site can be hosted effectively and live update based on code pushed to a specific branch in the repository.

- The following YouTube video will give a demonstration on how the code can be pulled from GitHub and hosted on a custom domain that can be managed from AWS.

    - https://youtu.be/gldpw-WNbrU?si=c5Hajfda9ieJlSbU

        - Note: Not all the steps will match if using AWS Route 53 to purchase and deploy the domain name. However, the process will remain rather similar and Zane Sorensen (sorensenzane@gmail.com) is happy to help if needed in the future.

- Once the site is hosted, as mentioned above, the code can live update based on any new changes to the repository which will allow for continuous development to be made by any future employees, or any future students.

    - Note: If there are any *.webp* images in the repository, while it will run locally, the images may not be processed correctly in the browser when hosted. We recommend converting those to *.png* files at a site like this. (https://cloudconvert.com/webp-to-png)

Feature Explanation

- The first and main feature in the application is of course the map and pins on the map that designate the location of a tobacco retailer.

- Each of these pins are generated in the map.js file where the code can be updated if needs of the project change. Each pin when clicked, will populate information in a side bar on the website to get better information about a particular retailer.

- A zip code filter is also implemented on the site, this will allow a user to only display a particular zip code to allow for ease of use.

- There is also an export button that will trigger a CSV file to be created and downloaded based on the filter that is currently in use based on zip code, if no filter is entered, all of the data will be carried to the CSV.

Database / Python Scripts

- The details for running and implementing this application should adhere to the README file contained in the repository here (https://github.com/zsorens/Louisville-Tobacco-UI)

Below is a brief explanation of the project and what has been done.

Throughout our project, we have made substantial updates to enhance the functionality, reliability, error handling, and organization of our codebase across multiple files, ensuring a more robust system. In the main.py file, we introduced modifications such as adding print statements to track the processing of stores in the get_stores function and removing blocks of code that grouped places with and without indicators. Instead, we implemented a new structure to save store information into separate text files organized by zipcode and store type, and improved the main loop to process and save store information using the newly added save_info method. Additionally, we enhanced error handling to skip stores that cause the program to crash and removed unnecessary commented-out code.

In the store.py file, we refined the extract_image_text method to handle exceptions gracefully and accurately store extracted text from images using self.image_review_text. The save_info method was introduced to efficiently manage and write store data, considering invalid characters by replacing them with underscores.

We also addressed rate limiting and exception handling in the places.py file by integrating the ratelimit library to manage API requests in the get_places function and modifying the function to return an empty list instead of None when a rate limit is exceeded. This change ensures smoother subsequent code execution.

In the streetview.py file, we introduced a timeout parameter to handle cases where retrieving streetview images may take too long, ensuring the program skips problematic stores. Error handling and logging were also enhanced to verify the correct saving of images and to create necessary directories.

Lastly, we adjusted the SQL queries in the sql.py file to ensure compatibility with the database system and resolved syntax errors related to database operations. It is important to note that no changes were made to the vision.py and web_scraping.py files during this period.

These collective efforts aim to enhance our programs' ability to process multiple zipcodes efficiently, improve visibility into data processing, gracefully handle exceptions and rate limiting, and bolster the overall reliability and robustness of our system. While these updates are tailored to our recent discussions and the specific needs highlighted, further optimizations may be necessary to cater to additional requirements and constraints of the project.