CS 411 Team-25 Track 1 Report

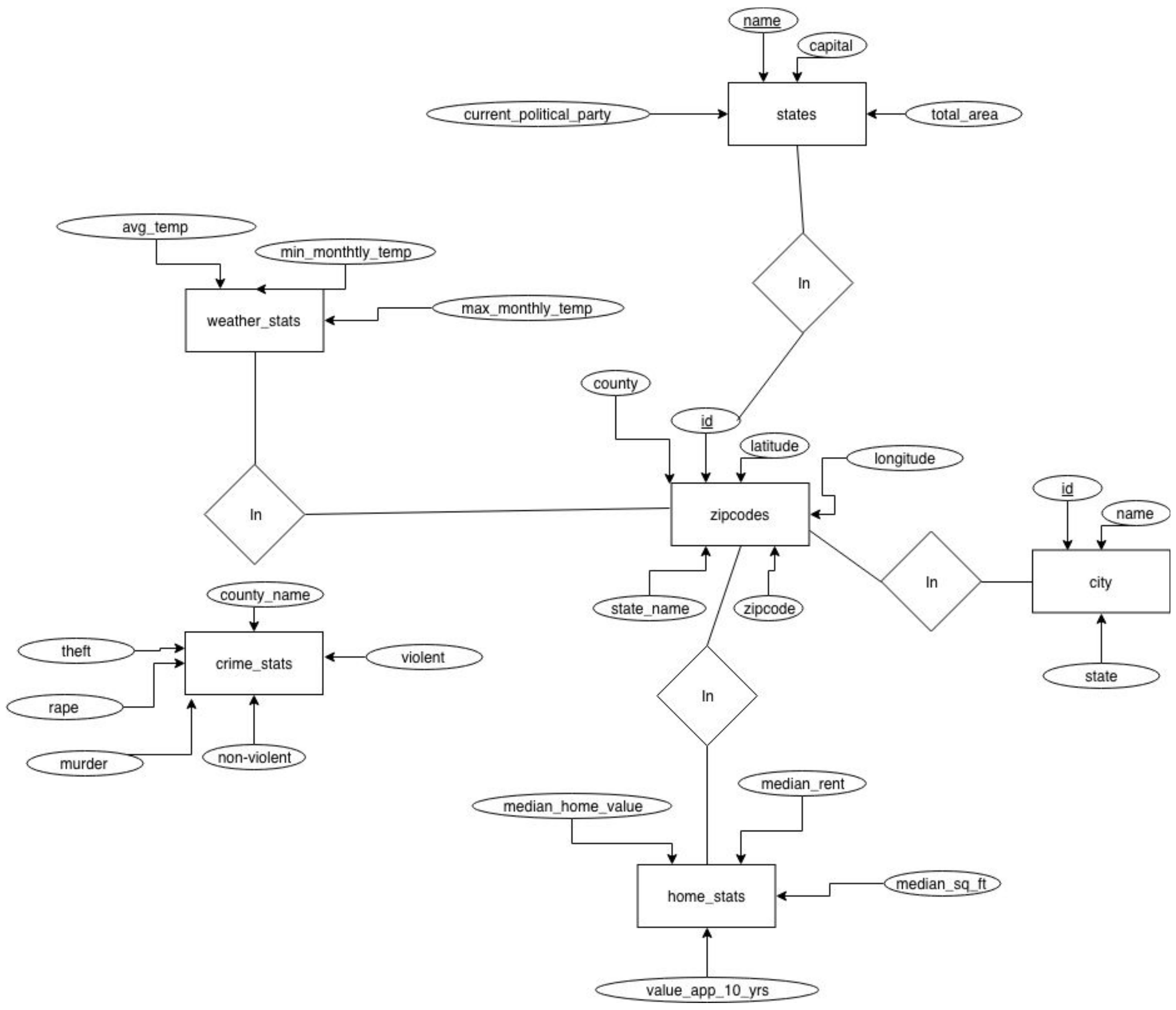
Parker Hess (pihess)
Vivek Bandapalle (vbanda4)
Mohammad Moaaz Ahmad (mmahmad3)

The project is a Flask web application that helps people looking to relocate within the US find locations by filtering for those areas which match their conditions for weather, housing prices, and crime level.

The database is managed by MySQL. It contains tables relevant to the data required for filtering locations:

- County_crime_data: contains information about various crimes in counties within the US
- Cities: contains all the cities' name and state information
- Zipcodes: contains all the zipcodes in the US, along with their geocoordinates, county and city names
- Weather_stats: contains weather information (average temperature, min_monthly_lows, max_monthly_highs) of all zip codes
- Home_stats: contains median house price and population density of each zip code
- Favorites: stores zipcodes marked as "favorite" by users
- Users: stores user credentials
- States: contains state name and state code (e.g. Illinois, IL)

ER Diagram



Relational Schema:

zipcodes(<u>zipcode</u>, latitude, longitude, city_id, state)

- *city_id* is a foreign key from the *city* relation
- *state* is a foreign key from the *states* relation

states(<u>name</u>, capital, total_area, current_political_party)

- *current_political_party* is foreign key from *political_parties* relation city(id, name, state)

- *state* is a foreign key from the *states* relation

crime_stats(zipcode_id, violent, non_violent, theft, murder)

- *zipcode_id* is primary and foreign key from *zipcodes* relation

home_stats(zipcode_id, median_home_value, median_rent, median_sq_ft, value_appreciate_10_yrs)

- *zipcode_id* is primary and foreign key from *zipcodes* relation

weather_stats(zipcode, avg_summer_lows, avg_summer_highs, avg_winter_lows, avg_winter_highs, avg_precipitation_per_yr)

- *zipcode_id* is primary and foreign key from *zipcodes* relation

Data Collection
The data was collected from several sources including:
- Github - For the zipcode / City / State CSV file
- Zillow - For the housing data
- NOAA - For the weather data
- US Census - For the crime data

The source data files had various types of formats and often extra useless data so they were all preprocessed with custom python scripts. The weather data one was particularly complicated because the data was not provided as a zip code by zip code bases but was just a list of data collection facilities that had the data and the Coordinates of the facilities. I then had a script that went through all of the zip code locations and found the closest facility and then associated the data with that zip code. This was done before putting the data into SQL database because if it were not done beforehand it would take a long time to calculate the nearest facility for every zip code every query.

Web scraping was not used because many websites that would provide data that would be scrapped only allowed for a set number of requests per day. That wasn't an option because our data set had nearly 30,000 rows that needed to be populated so downloading the data files was the only reasonable way to get the data.

Feature Specs

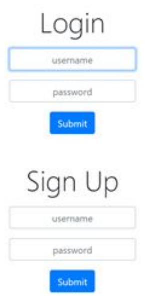
The application firsts prompts users to login or create an account. The application then allows users to filter by average temperature and housing prices or by reported crimes. After filling out the forms, a heatmap of the returned locations is displayed. On this page, the user can also save favorite locations to view at a later time.

One basic function of the application is to pass minimum and maximum values entered by a user from an html form to the application and filter. An example filter is the number of reported crimes in a county. In the example below, %s would be replaced by the values passed in by the user.

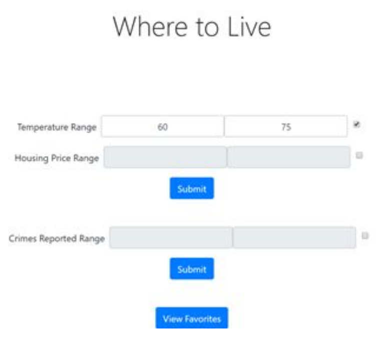
SQL code snippet: 'SELECT cd.county, cd.violent_crimes_total FROM county_crime_data cd WHERE violent_crimes_total BETWEEN %s and %s'

Data flow:

User logs in or creates an account. The form data is sent to a python application connected to our database. The credentials are validated with the database.

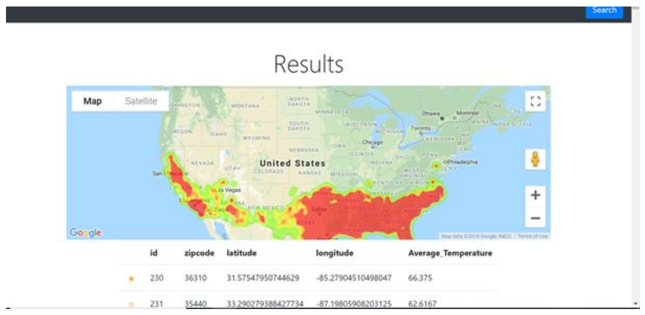


User is prompted for filters for weather, housing or crime data. The user can also check their favorites from this page. The application runs a query using the filter values entered in the form.


Where to Live

Temperature Range	60	75	✓
Housing Price Range			☐
Crimes Reported Range			☐

The results page shows a heatmap with the clustered data values. The user can favorite a location by clicking the star icon in the first column. They return to the search page from the button in the top right. The webpage gets the data, parses it into a json format and creates bootstrap tables based on the data.



	id	zipcode	latitude	longitude	Average_Temperature
★	230	36310	31.57547950744629	-85.27904510498047	66.375
☆	231	35440	33.290279388427734	-87.19805908203125	62.6167

Advanced Function 1 : Maps

We decided displaying a heatmap of the returned data would help users decide what regions match their criteria the best. We passed the coordinates of the zipcodes to the google maps API and also displayed the user saved locations on the map as markers. The advanced portion of function comes from the preprocessing required to get and clean the data the data that should be passed to the API. As described above, collecting the data was done from various sources and required custom python scripts to clean and get the data into the database.

Advanced Function 2: Favorites

We thought allowing users to save locations they have viewed before would assist them in deciding what location to move to in the future. The user can add or remove a saved location by clicking the orange star on the map screen. The difficulty of this function comes from the need to pass the user's login id back and forth between the html pages and the flask application. Each time the user is given the option to view, add or remove a favorite location, the user's login id needs to be passed to the flask application and checked against the database. The format of the data passed to the html webpage also had to be revised to include a column specifying whether the current zipcode was a favorite or not.

Technical challenge

A technical challenge that the team had was getting the weather data into a useable / efficient format. The data was first downloaded as a zipped set of 9839 txt files each one representing a weather station that had recorded temperature normals for the past 10 years. The format of the files was not a well known standard and there was not a direct way to parse it into a data structure so a script had to be made to convert each of the files into a row in a csv file. From there further processing was needed to figure out which station should be used for each of the zip codes because finding the correct station for whenever data was queried would have taken O(N^2) time and because we have almost 30,000 location data values this would lag the server to the point of unusability. The script only had to figure out the data once and then the resulting data would reduce the time cost of a query down to O(N) in regard to the weather data. The script took an hour and a half to run but now the application takes less of a second to get the correct weather data.

Not everything went according to the initial development plan. We initially planned to include data for more filters such as employment, literacy, etc. However, data preprocessing proved to be too time consuming and we were forced to move on to the front-end design. Another unexpected thing was the fact that the crime data was organized per county rather than per zipcode. This forced us to change our initial front-end design and separate the two filters.

Another technical challenge was setting up the environment. After failing to set up CPanel, we decided to use AWS Elastic Beanstalk and RDS services for hosting the Flask application and MySQL database respectively. The AWS setup took long and required additional platform knowledge and and understanding its command-line utility.

Final division of work: Mohammad (mostly back end, some front end), Parker (data collection/cleaning, some front end), Vivek (mostly front end, some back end)

Team Management: We managed teamwork through GitLab and SQL Workbench. We tried to make meaningful progress each week.