Project Report : All About Austin
Phase 3 - Team Amber

Team Members

Name, E-mail, Github User
Justin DuPont, justindpnt@gmail.com, justindpnt
Canyon Evenson, canyon@utexas.edu, cpe342
John Koelling, john.k.koelling@gmail.com, lucundus
Yixing Wang, yixing.wang@utexas.edu, AlienEdith
Grayson Watkins, graysonwatkins@gmail.com, Graysless
Zach Wempe, zdwempe@gmail.com, zachwempe

Project Leader, Phase 3

Yixing Wang

URL to Website, Git Repository, Google Docs

Deployed Website:
http://www.allaboutaustin.info/
GitHub:
https://github.com/Iucundus/AustinData
Google Docs:
https://docs.google.com/document/d/1R-suefPzFPDNJkwbQ2wc0wpQ5aZYFyTxBlKfVDbHA-4

Tasks Completed:

Tasks for this phase from Project Proposal:

- At least 5 more user stories (15 Total)
- Finalize daily data collection from all 3 datasets
- Refine your dynamic site with many pages hosted on GCP
- Refine and add to your unit tests of the API, refine the GUI tests
- Add to and refine the technical report
- Implement more datasets from Austin Government (time permitting)

Completed Tasks:

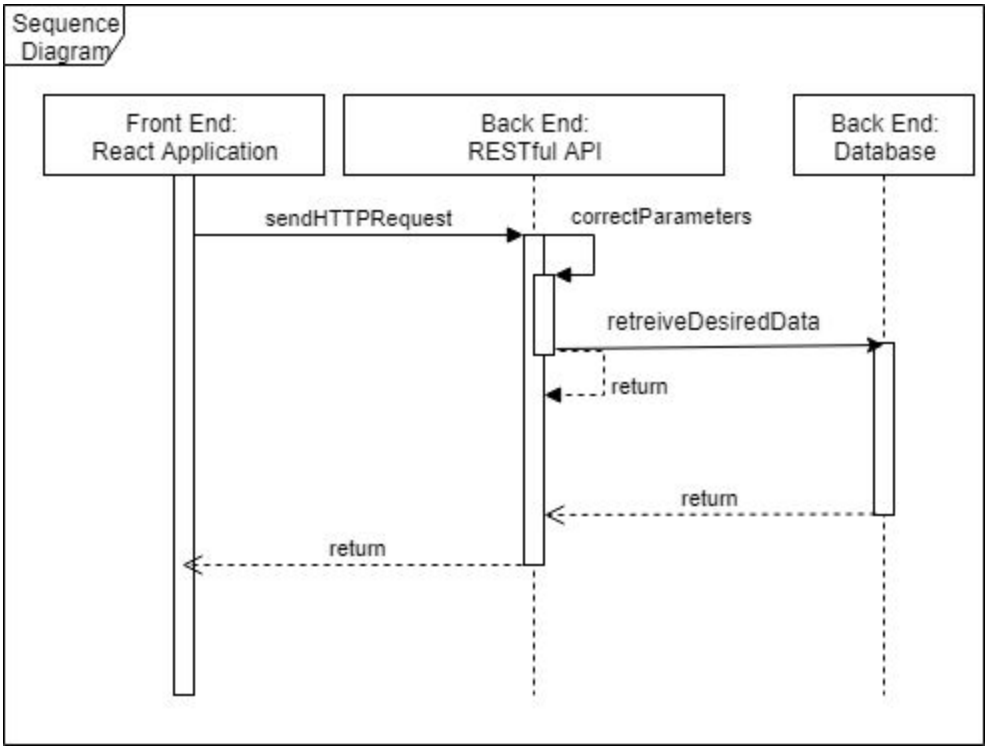
- 5 more user stories (15 Total)
- Finalized automated daily data collection from all 3 datasets
- Refined our dynamic site with many pages hosted on GCP
- Refined and added to your unit tests of the API, make GUI tests
- Added to and refined the technical report

- Implemented more datasets from Austin Government (traffic volume)
- Added Google login support to save user favorite zip codes
- Improved the education score algorithm to look at surrounding zip codes when there is no school data for the current zip code
- Added in cinema and hospital fields to each zip code, along with a list of each
- Added heatmap display for food, traffic, and education data, using Austin City Government data for zip code boundaries
- Created a feature for the user to change their priority rankings after the rankings display has already started
- A complete RESTful API for Austin Zip codes information, hosted on http://api.allaboutaustin.info/. (Documentation:https://documenter.getpostman.com/view/6614125/S1ETQG2B)

Design:

- **Overall Design**

  The core part of our website is our self-implemented backend RESTful API, which connects our front end view part and back end database together. When the front end page is loaded or takes user inputs, it sends an HTTP request to our RESTful API with corresponding requirements/parameters. When the RESTful API receives the request, it will ensure the request is valid, fetch the data from our MongoDB database, then send it to the front end part for display.



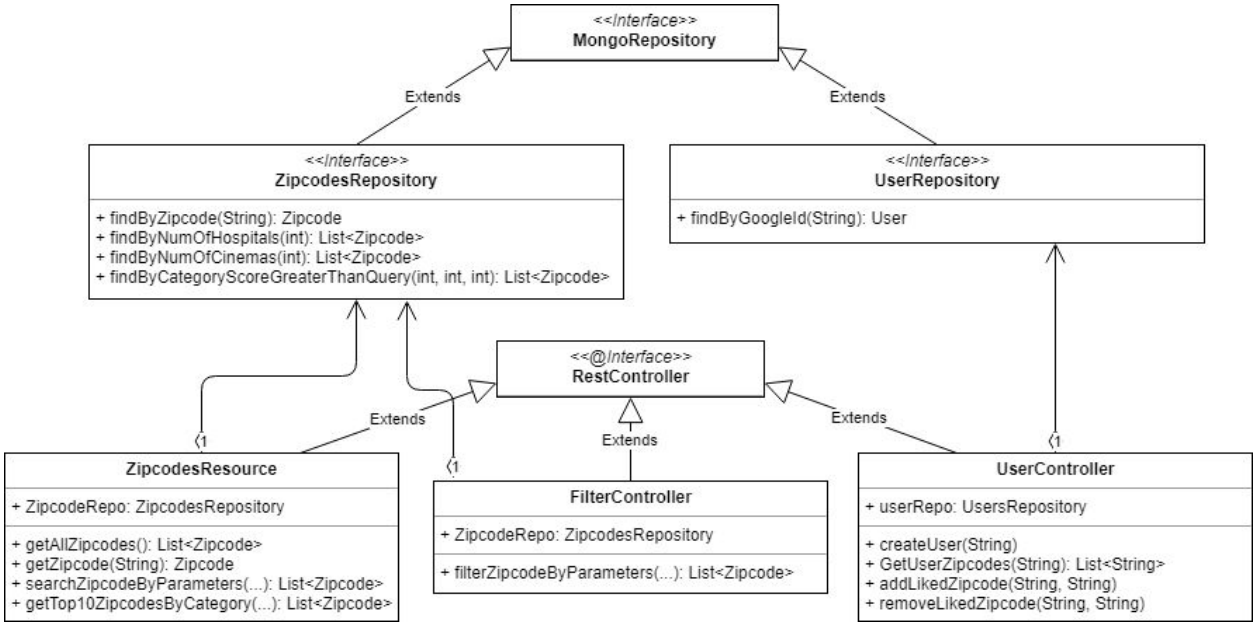
- **Front End Design:**

  Our Front End is implemented with JavaScript React Framework. Based on modularity principle, We implemented components for each user interface with JSX and we used Bootstrap 4 to stylize

our user interface, which also provide screen responsive to our website. The data for each page or component are fetched by sending HTTP get or post requests to our own-developed RESTful API with axios package separately. With React Component Lifecycle and Redux, we could dynamically fetch data and re-render only the a few parts of each page. We used mapbox-gl for building heatmap with our own generated GeoJSON file and google map API for building map display with KML files provided by Austin City Government, thus these two map components are isolated from other data in the page, which follows single responsibility principle.

- **RESTful API Design:**
  Our RESTful API is implemented with Java Spring Boot Framework and we followed the convention of Spring Boot framework. For each data model, we defined our custom Repository and Controller. The Repository contains queries for data, and the Controller maps HTTP requests/urls sent by the website to corresponding methods that fetch desired data from our database and return them in JSON format.
  There are two main models in our Back End API Design:
  - Zipcodes: Contains information of each zip code. Our API supports sorting and filtering zip codes with various kind of parameters and return them in JSON format. The cross-origin of this model's controller is set to all domain name, so other website could also get data from our API
  - Users: Contains only google id and their favourite zip codes list of each user. Since we are using Google OAuth for login system and we only restore the user's google id, there won't be any security issues. Our API supports adding first-login user to our database, adding and removing zip code from users favourite zip codes list. The cross-origin of this model's controller is set to our own domain name only, thus the above functions is only allowed within our website.

  Our API supports not only the basic interfaces for our website, but also supports several further usage. (API documentation: https://documenter.getpostman.com/view/6614125/S1ETQG2B).
  The class diagram below only shows the main part of our backend API design.



- **Database Design:**
  The MongoDB cluster stores data in several forms: Food data, traffic data, education data, and zipcode data. Each of the food, traffic, and education data groups are original data provided by the APIs. We calculate a score for each data point in the range of 0-10, and store it associated with the zip code which that data point is from. Then a second pass on the data is made where we iterate through all zipcodes and fetch the entire set of data for that zipcode, and store it back into MongoDB as zipcode data with all information attached.

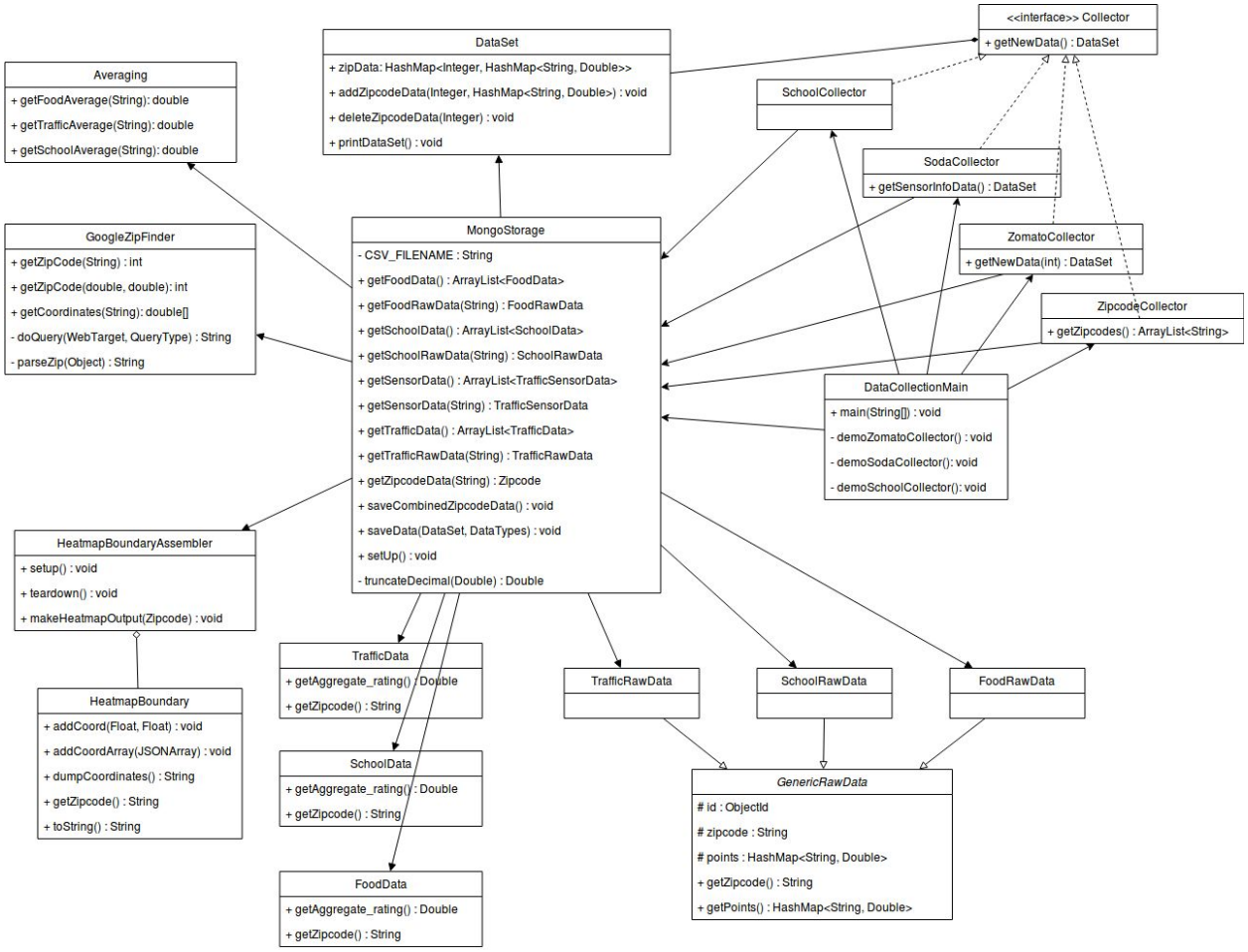
  In that second pass, the MongoStorage class also uploads data from a CSV file. This CSV file holds manually assigned data (such as zipcode descriptions and image URLs) that will be associated with each zipcode, and is uploaded to the combined zipcode data structure.

  While iterating through each zipcode in the second pass, the MongoStorage class will invoke a HeatmapBoundaryAssembler class that creates a GeoJSON file with information about the boundaries of each zipcode. In that GeoJSON file (stored directly as a file on the App Engine server) food, traffic, and education scores are also embedded, allowing the front-end heatmap pages to build an interactive heatmap of Austin for each of the three categories.

- **Data Collection Design:**
  There are three types of data collectors in this part of the program, each for a different data source and its API. The Austin Government collector (SodaCollector) is unique because it collects two databases instead of one: one database for traffic sensor information, and another database for information about those sensors (used to locate them and identify the correct zipcodes.) GoogleZipFinder is a differently structured class that is used to convert latitude / longitude coordinates into zip codes, using the Google Maps Geocoding API.

A DataSet class is used to hold data, and is written generically so that it can hold any kind of data for the zipcodes. FoodRawData, TrafficRawData, and SchoolRawData are the classes used to store this data into MongoDB.

UML Diagram - Database and Data Collection Classes:



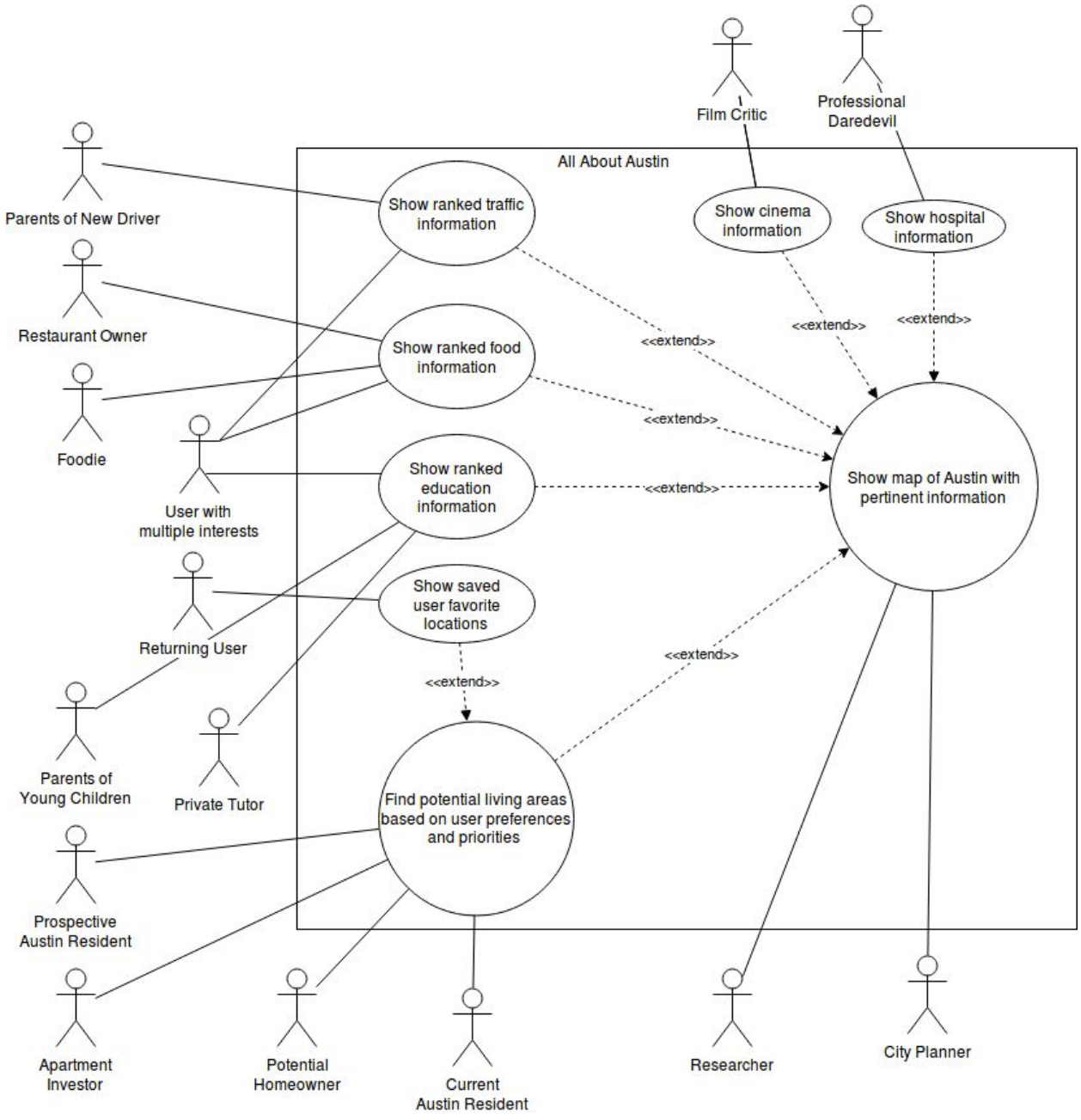
User Requirements:

User Stories for Phase III:

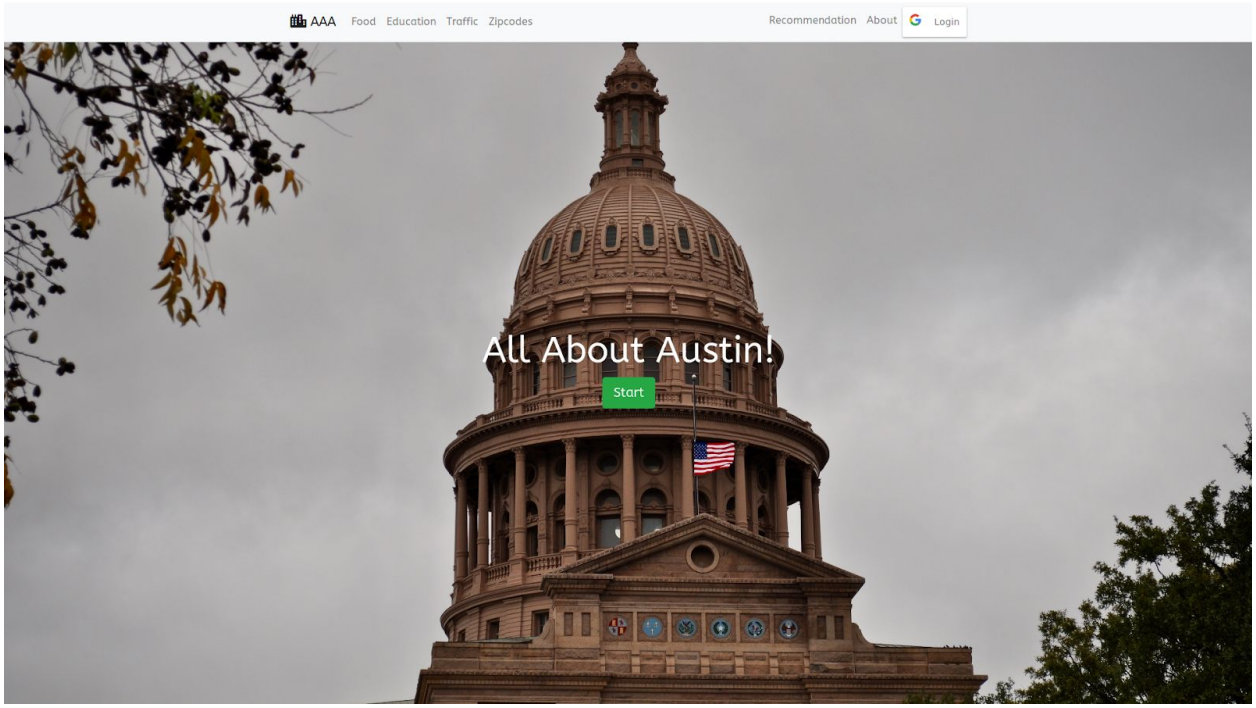
- User Story 11
    - AllAboutAustin user who returns to the site after a few months of downtime.
    - User Story: As an AllAboutAustin user, I want my zip code preferences to be linked to my Google account so that I do not have to retake the ranking test every time I visit the site.
- User Story 12

    - Potential homeowner in the Austin wanting to live in the southern region of Austin, but is not sure what zip code best suits them.
    - User Story: As a potential homeowner seeking to live in South Austin, I want to be able to use a heat map to compare zip codes in this region to find which one best suits my need of a good food scene.
- User Story 13
    - A city planner seeking to diversify the location of their construction projects.
    - User Story: As a city planner, I want to be able to visually compare proximity of the zip codes in the Austin area so I can avoid having my construction projects close together to avoid noise complaints.
- User Story 14
    - Aspiring film critic seeking an area in Austin with not only great food, education, and traffic metrics, but also entertainment.
    - User Story: As an aspiring film critic seeking an area with more than just food, education, and traffic, I want to be able to know if an area has cinemas so that I can have easy access to movie showings.
- User Story 15
    - Professional daredevil visiting Austin to plan where their next venue will take place.
    - User Story: As a professional daredevil planning my next venue, I want to be able to ensure that Hospitals are relatively close to my selected zip code so that I can be transported quickly if anything goes wrong during my act.

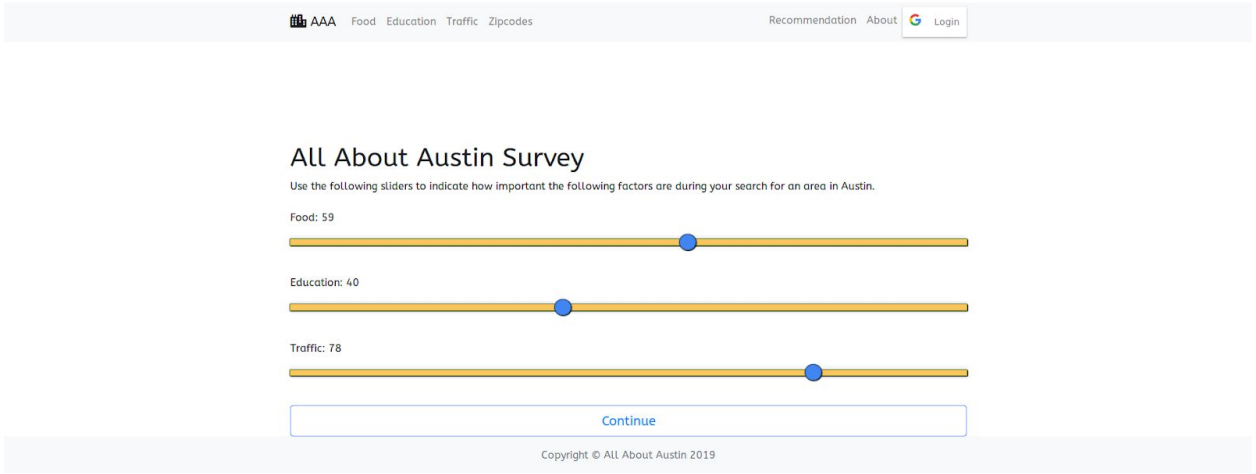
Use Case Diagram:



Screenshots and Walkthrough:

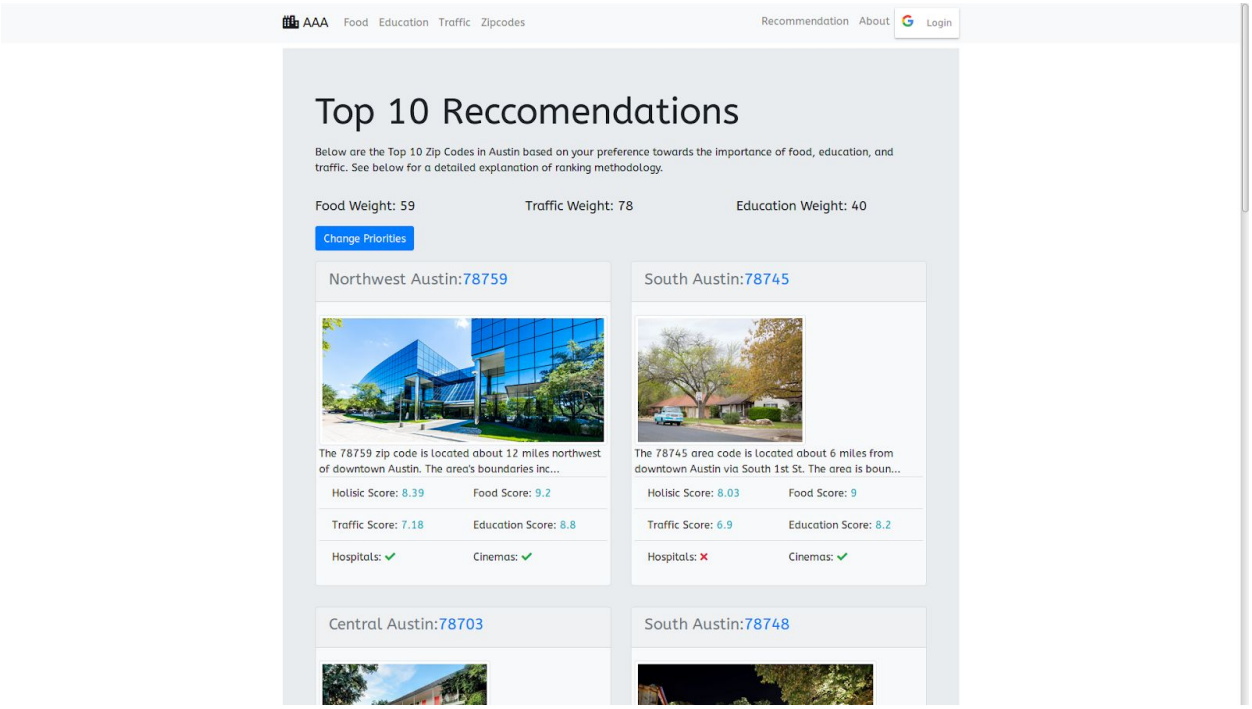


Start page. This lets the user select any mode of the application: Food, Traffic, Education, Zipcodes, or Recommendation (which is also initiated by the Start button.) The Google login option will enable preference saving (detailed later.)

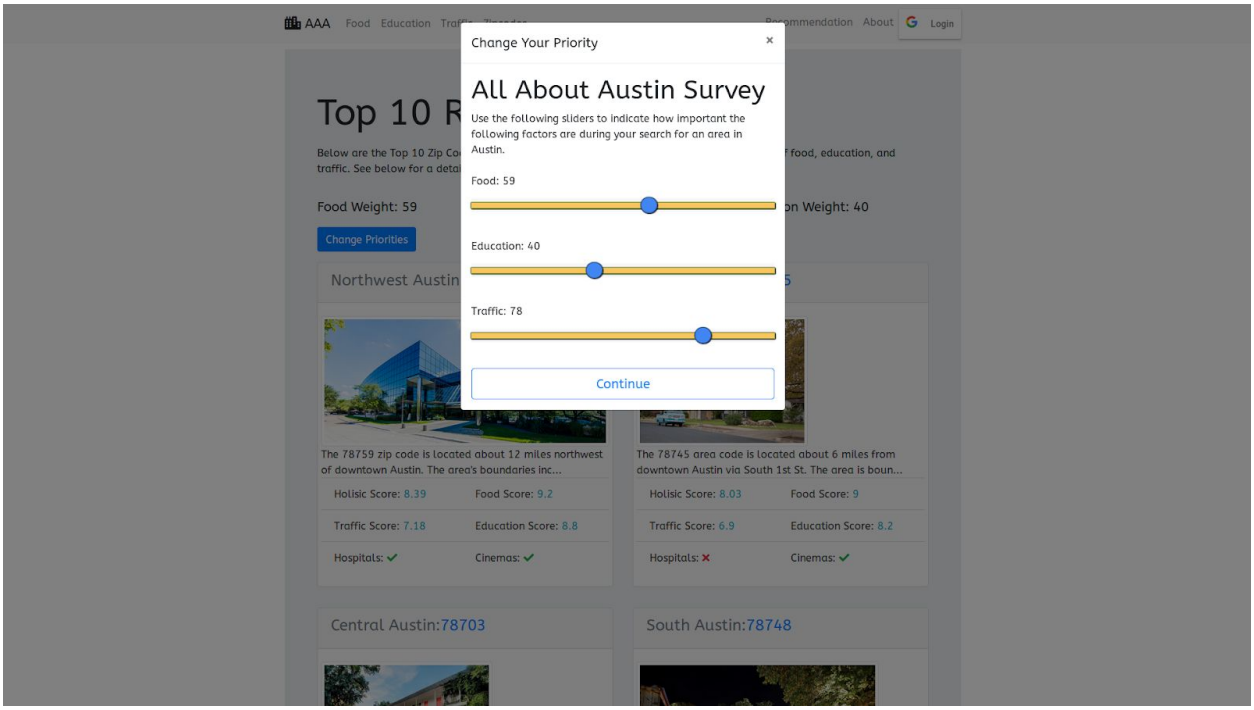


This is the first menu in the Recommendations mode. The user is asked to rank importance of various area factors, and these weights are used to generate the zip code recommendations. The three sliders used as

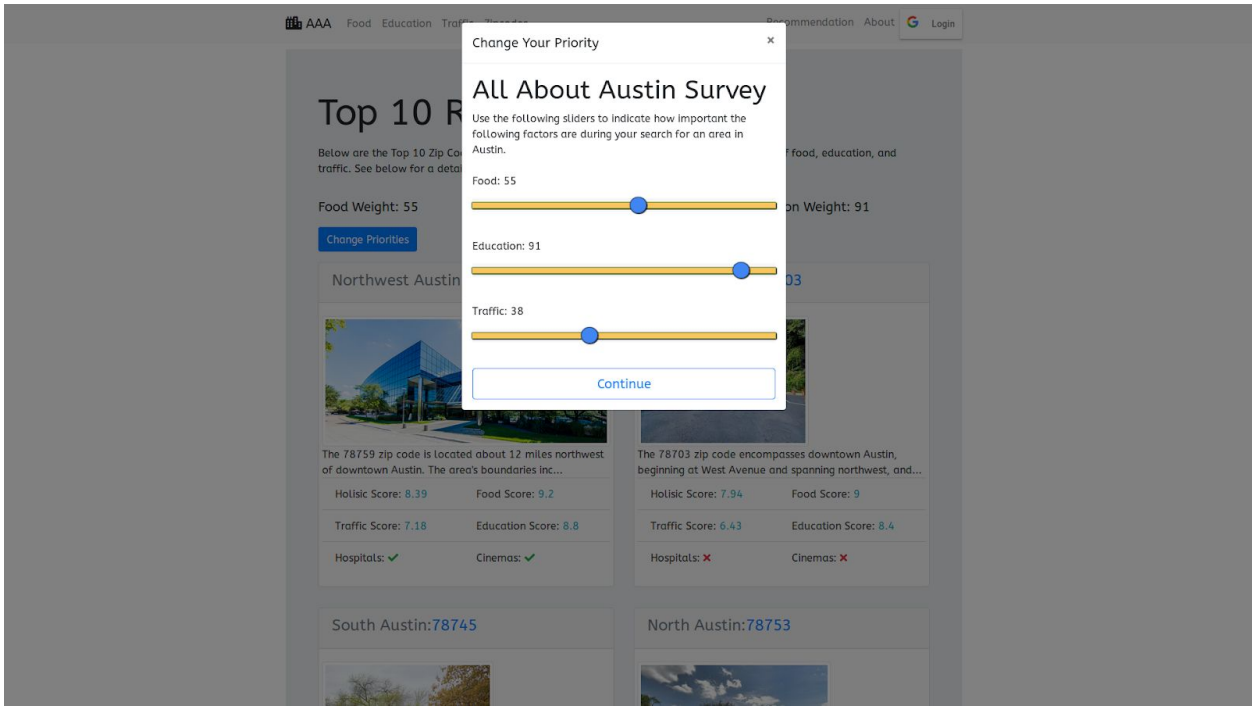
input are the originally intended format for receiving user preferences.



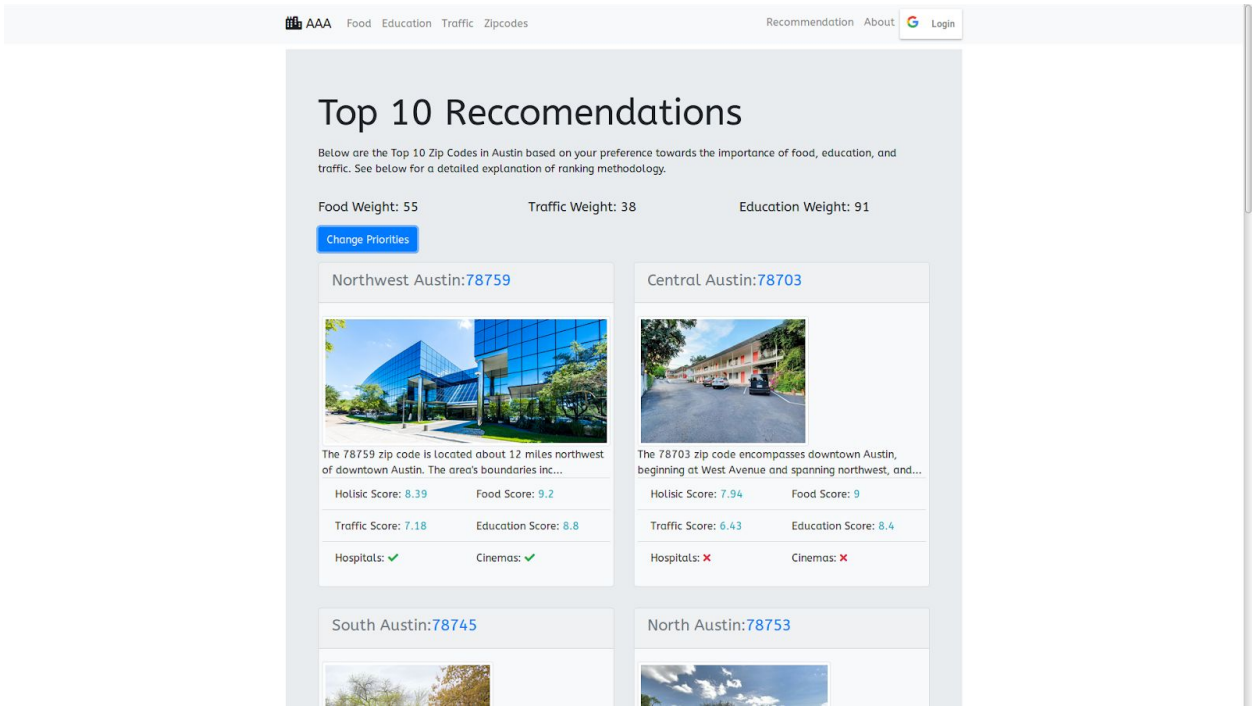
The best zip codes (as determined by the user's relative priorities) are displayed. Each is clickable for more information.



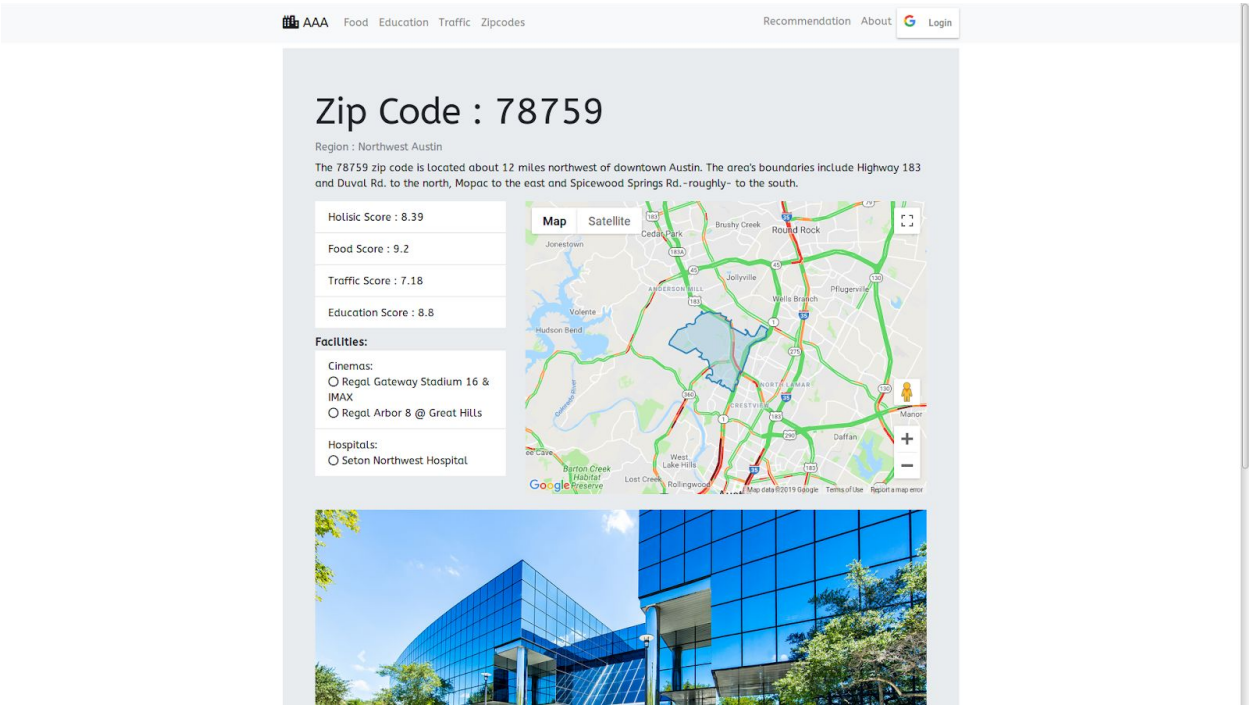
By clicking Change Priorities the user can assign new weights to each of the factors. This is an extra feature not originally planned.



Clicking Continue will provide a re-ranked list.

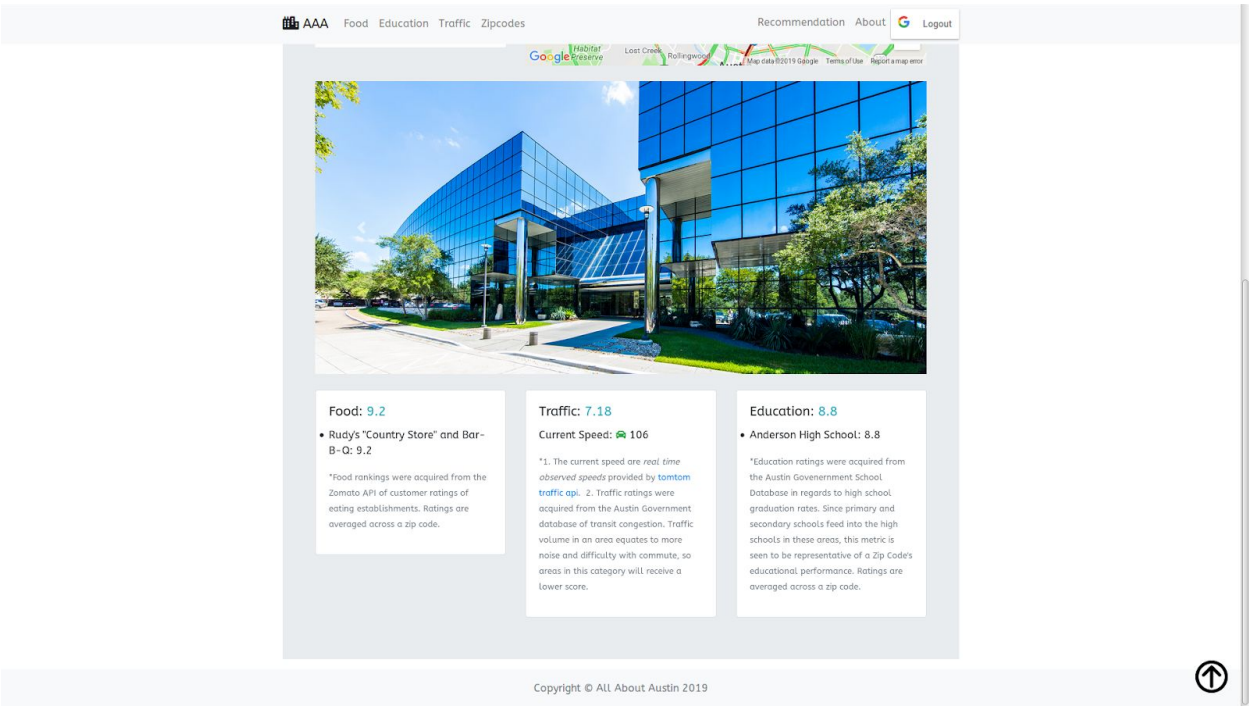


The selection of zip codes is changed. Let's click on one, 78759.

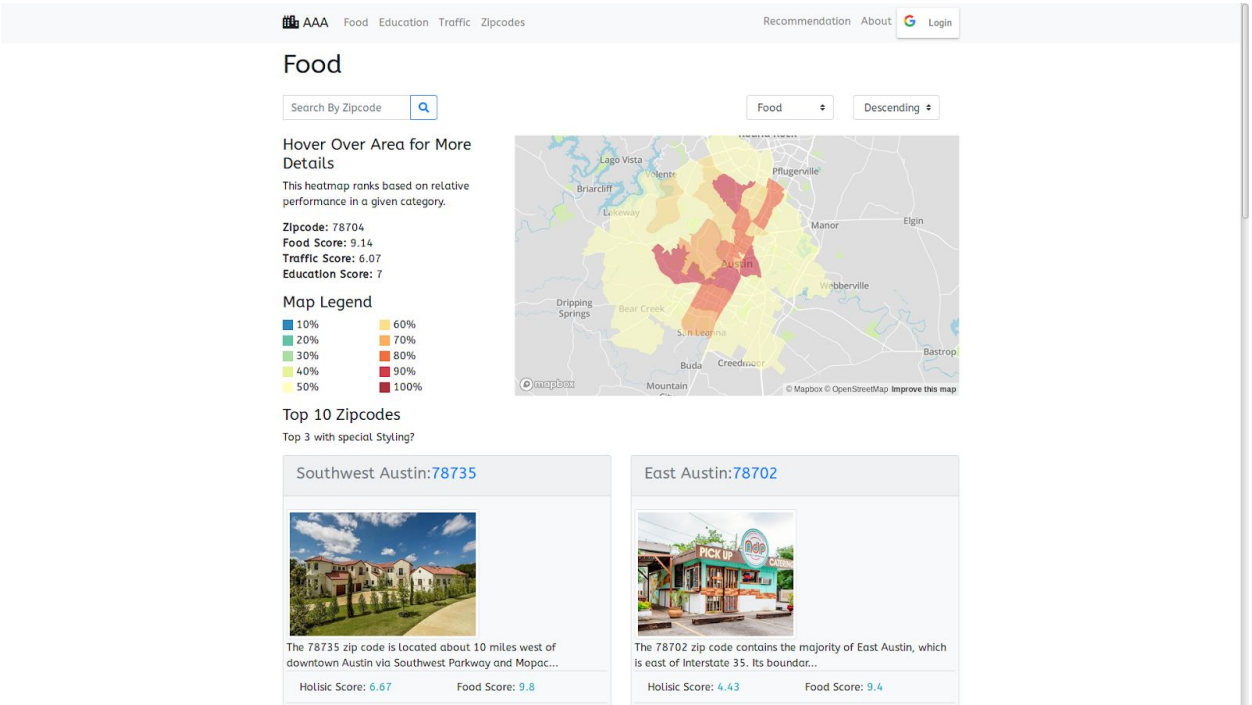


The individual zip code page provides a deeper dive into the details of each zip code. Whether the zip code is reached through recommendation by the AllAboutAustin survey, or by search functionality in the Zipcodes mode, this page provides holistic, food, traffic, and education scores in depth. The facilities tab displays the extraneous information of nearby Cinemas and Hospitals to the user outside of the scope of our Food, Traffic, and Education focus. Below the Facilities is a live look at the traffic in a given zip code. The blue region is the boundary of the zip code and the map is full functionality to allow the user to view traffic in the surrounding areas as well. Below the map and individual picture for each zip code is the detailed look at the Food, Education, and Traffic metrics. For the food and education categories, the user will be presented with the names and rankings of nearby locations, and the traffic is based on congestion as observed by the TomTom API.

The Holistic score is an average of the other three scores. The Food score is an averaged restaurant rating for restaurants in the area. The Traffic score is inversely correlated with traffic volume. The Education score reflects the graduation rates of schools in the area. This uses our data as specified in the Project Proposal, and additionally integrates data for Cinemas, Hospitals, and TomTom traffic data.

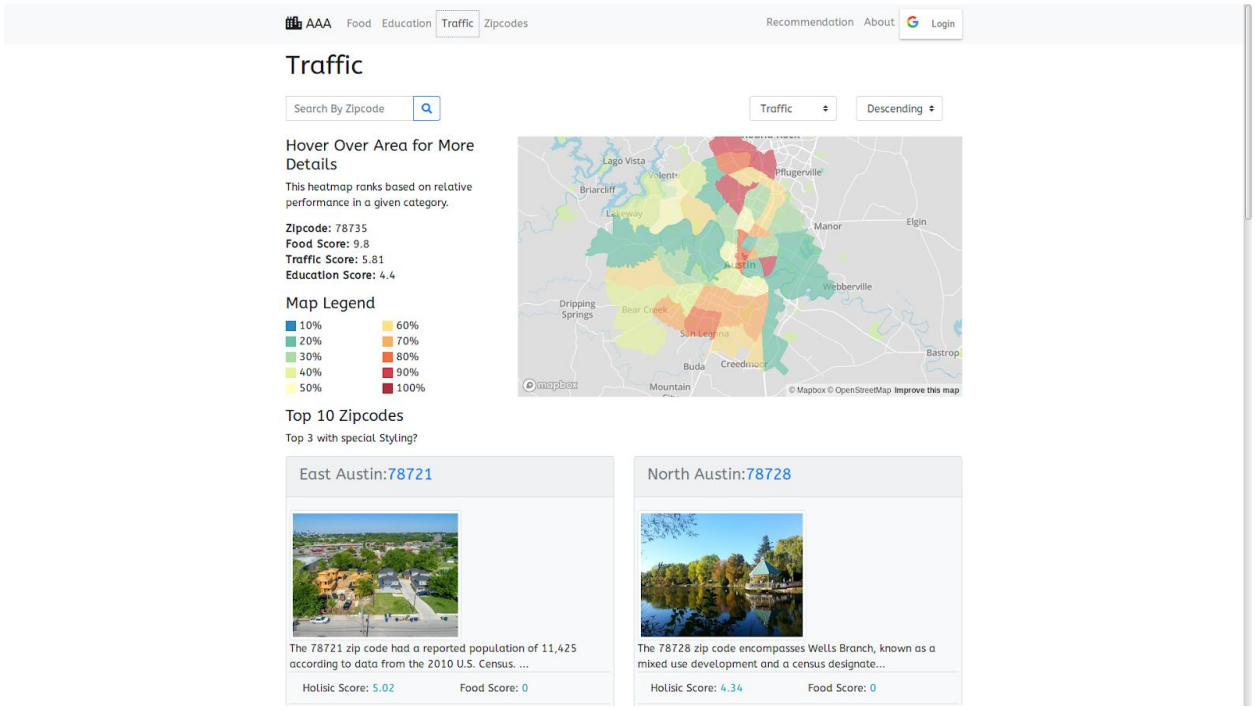


Detailed data is given below. The current speed of traffic is also shown, and a color-coded car icon will indicate the quality of this score.

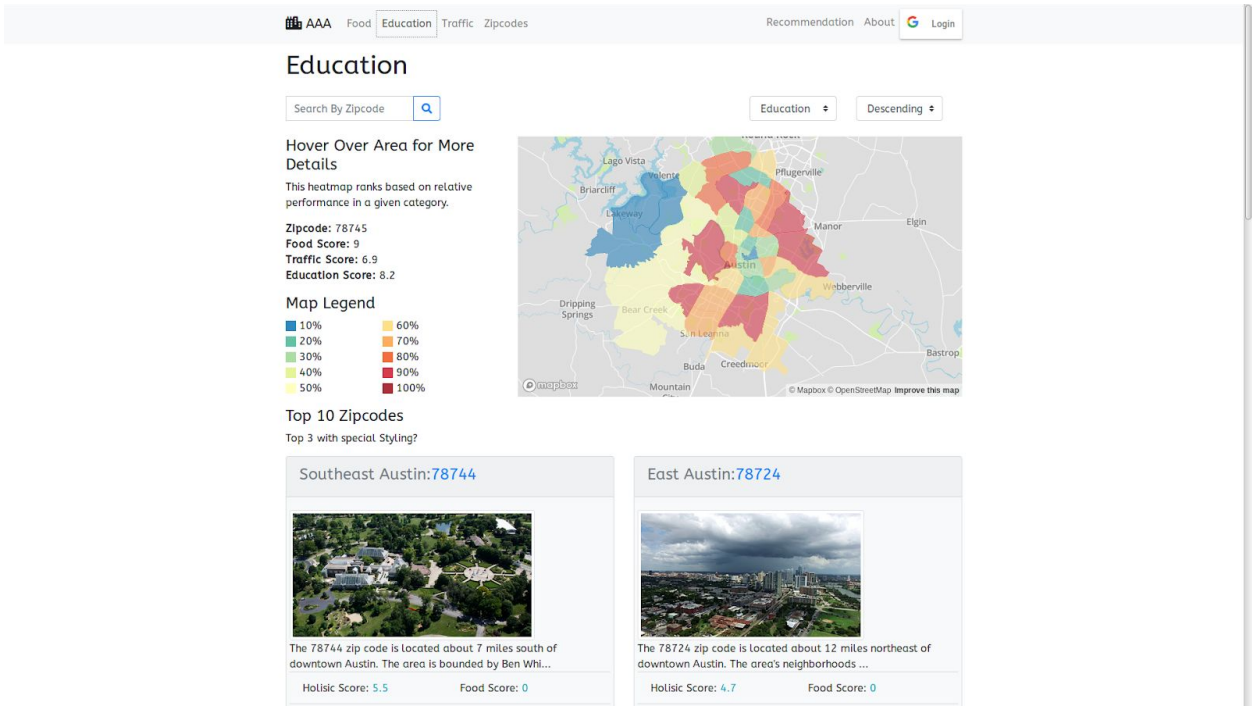


This is the Food mode of the application. The zip codes with the best food score are shown. A heatmap plots all zip codes with a color coding of how good their food scores are. The heatmap was another

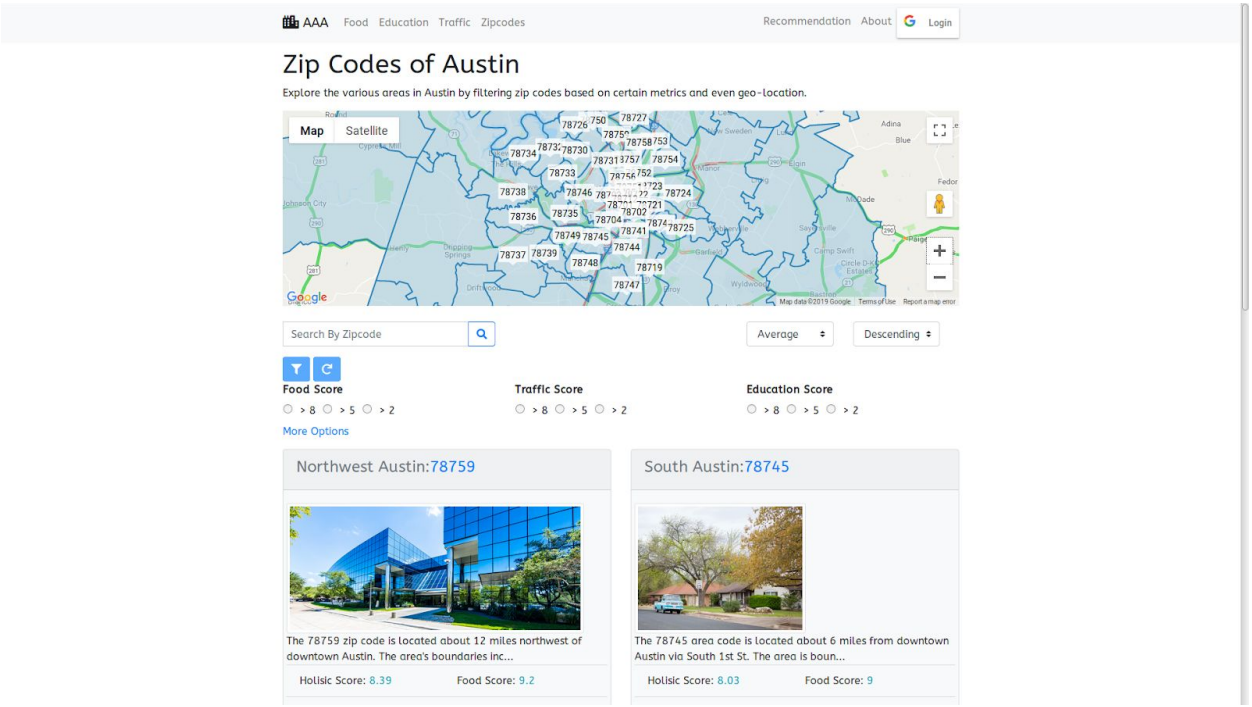
planned feature, and is also scrollable.



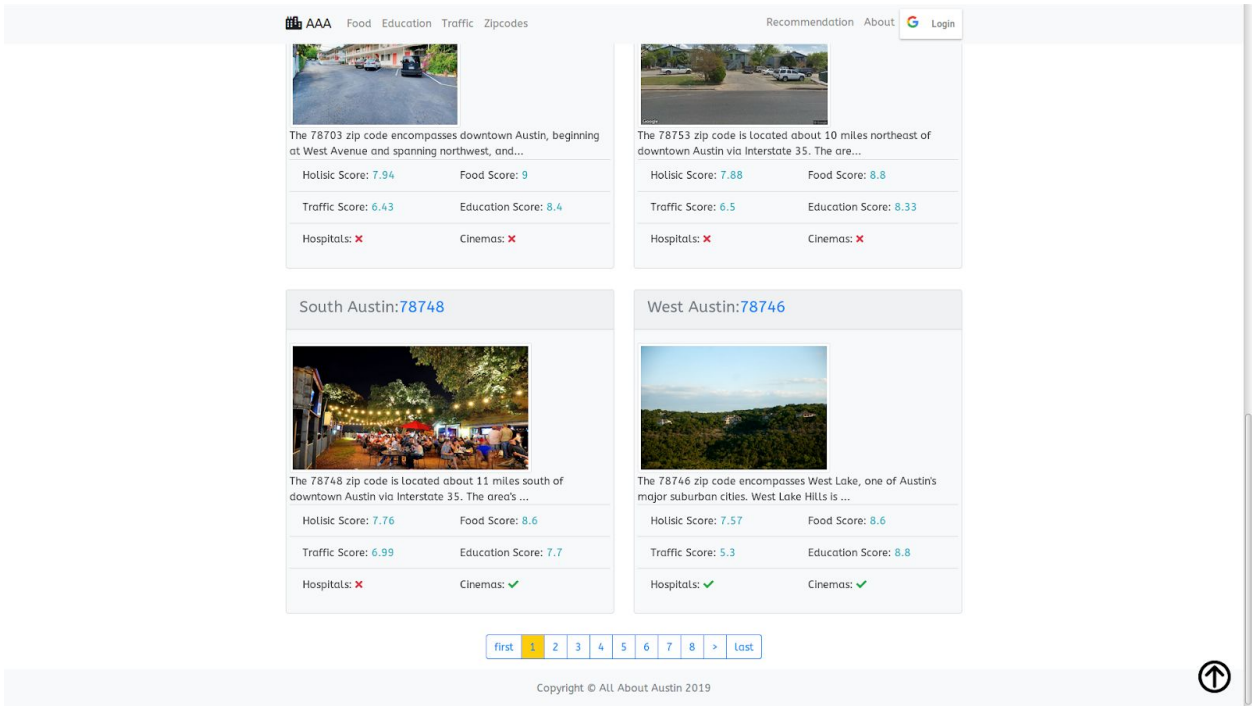
This is the Traffic mode of the application. All zip codes are clickable.



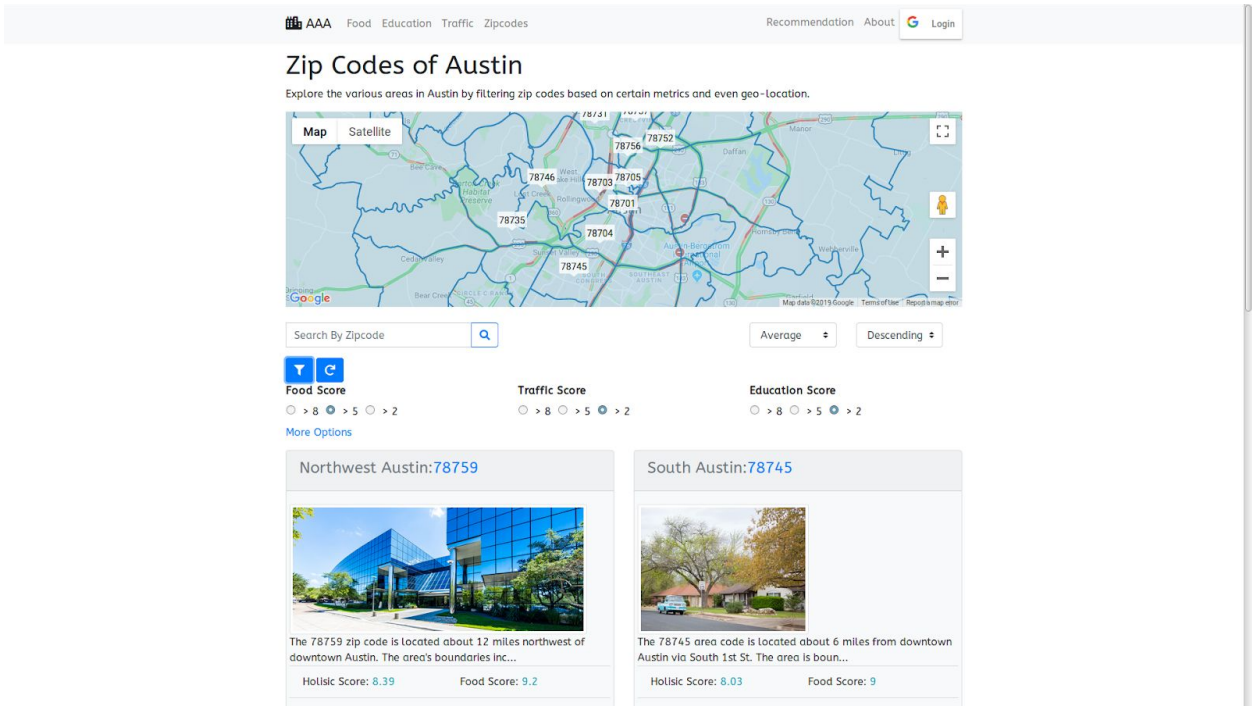
This is the Education mode of the application.



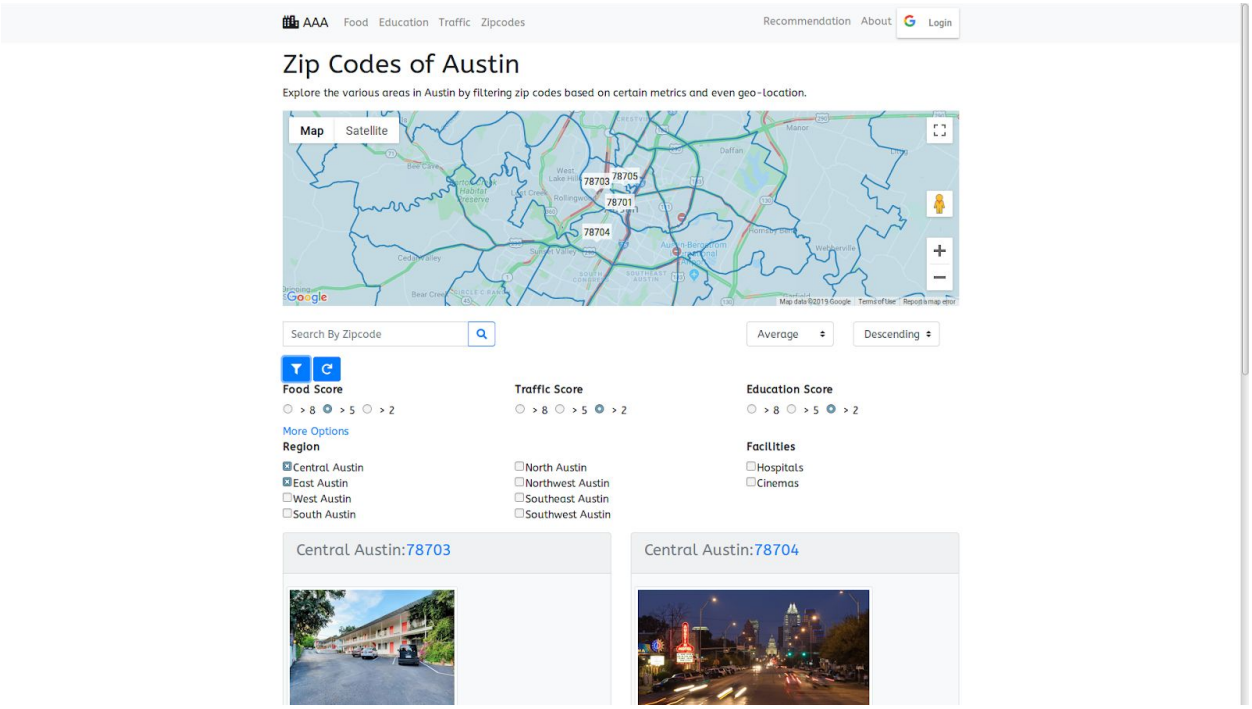
The ZipCodes mode allows for the user to explore all zip codes in the Austin area outside of their recommended 10. This page starts off by presenting an interactive live map of the zip codes in Austin. By clicking on one of these zip code labels, the user will be redirected to the individual zip code page for the selected option. Below the map is the search bar that can access a specific zip code. To travel to the ZipCode 78705, the user would type that number into the textbox and click on the magnifying glass. Below the search bar are the sorting options. By choosing a metric in these categories, users can sort all zip codes in the Austin area in ascending or descending order. In the filter category, we allow the user to limit search results based on certain score thresholds. Under the more options tab, the user is able to specify a certain region of the city or if they wish to see zip codes containing hospitals and cinemas nearby. The filters are applied with the Filter button, and reset with the Reset button. Clicking on any zip code card will redirect the user to the individual zip code page. In addition, the user can favorite certain zip codes by clicking on the heart icon next their preferences which will be tied to their google account.



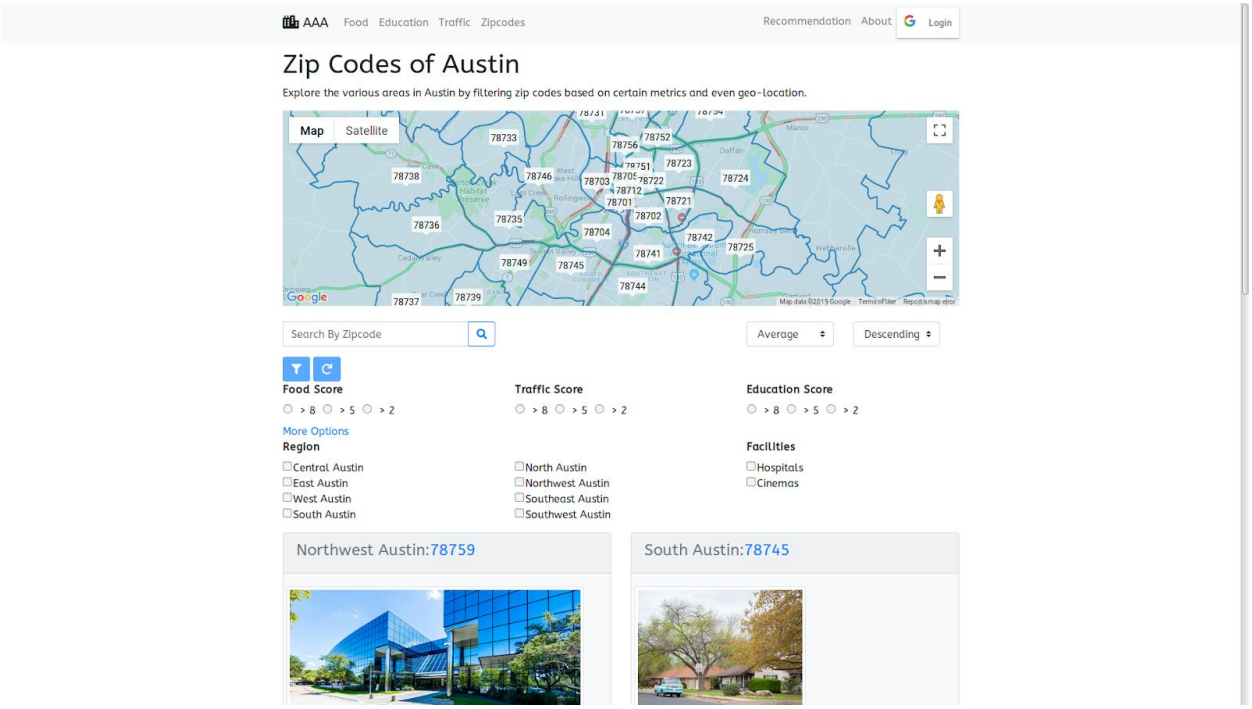
Since the zip codes mode has so many zip codes, the results are paginated with a menu to advance through them.



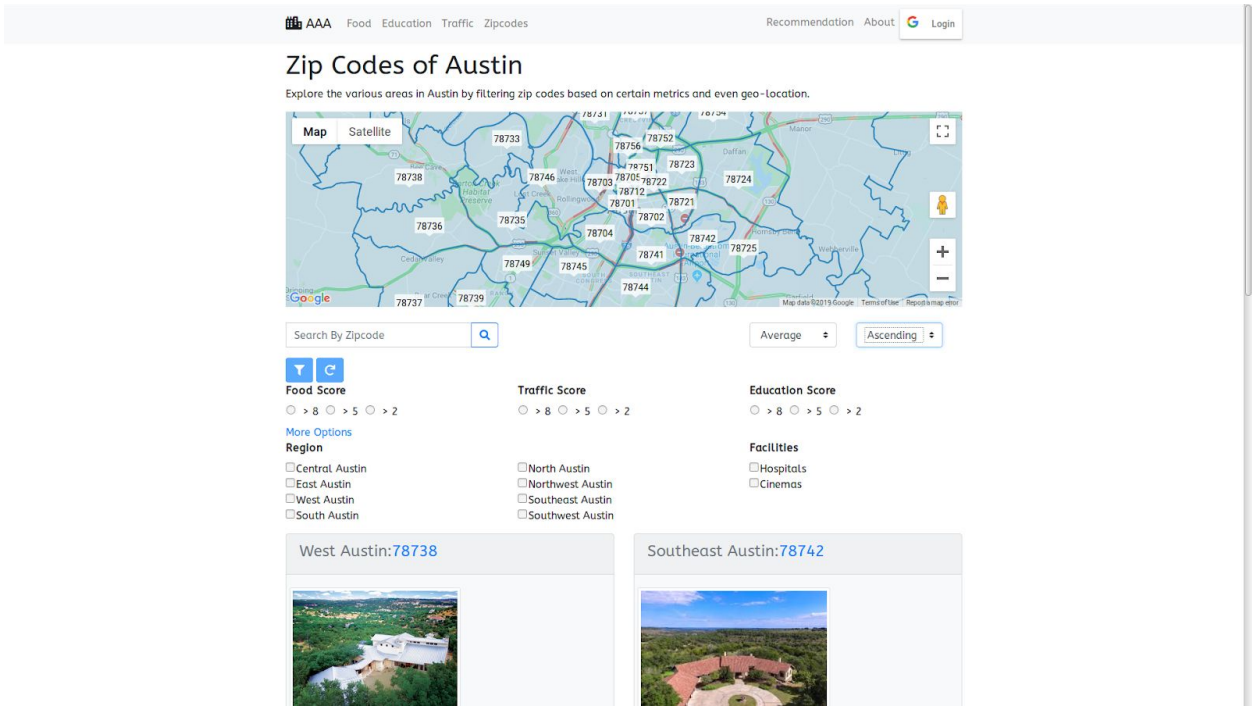
Shows filters applied.



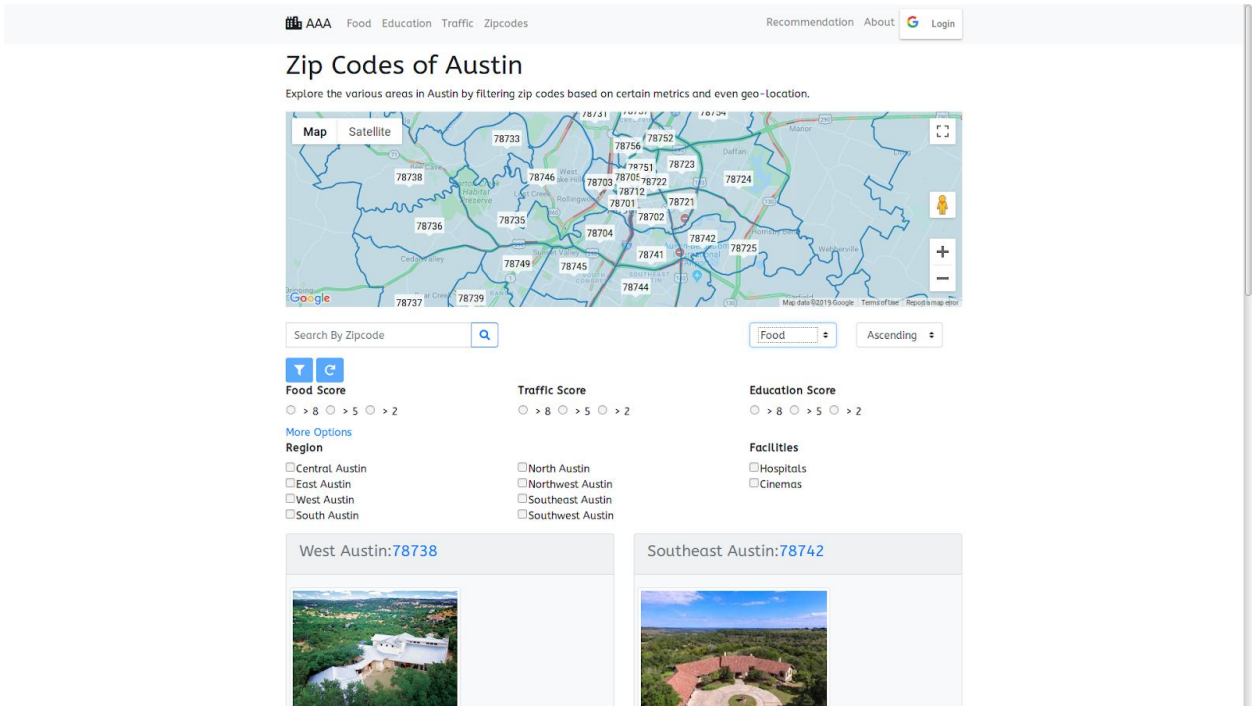
Shows regional filters applied. Limiting zip codes by region was an originally intended functionality per the Proposal. Other filters implemented are additional features.



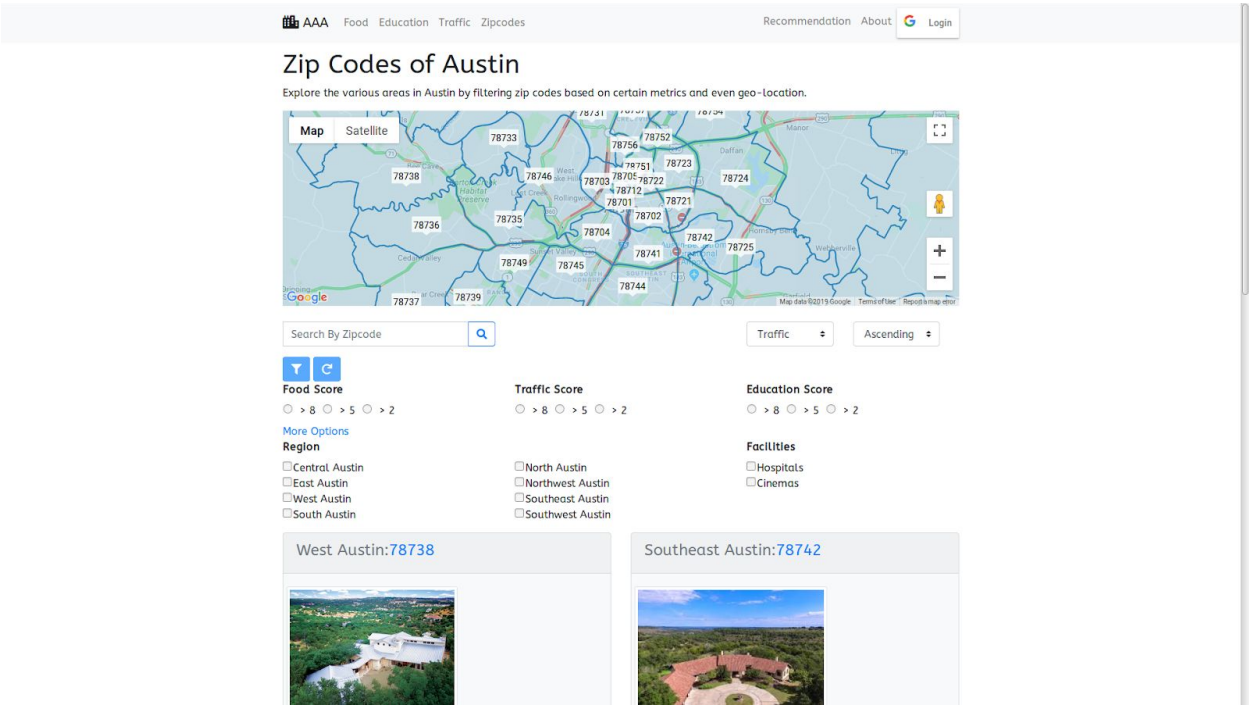
Shows filters reset.



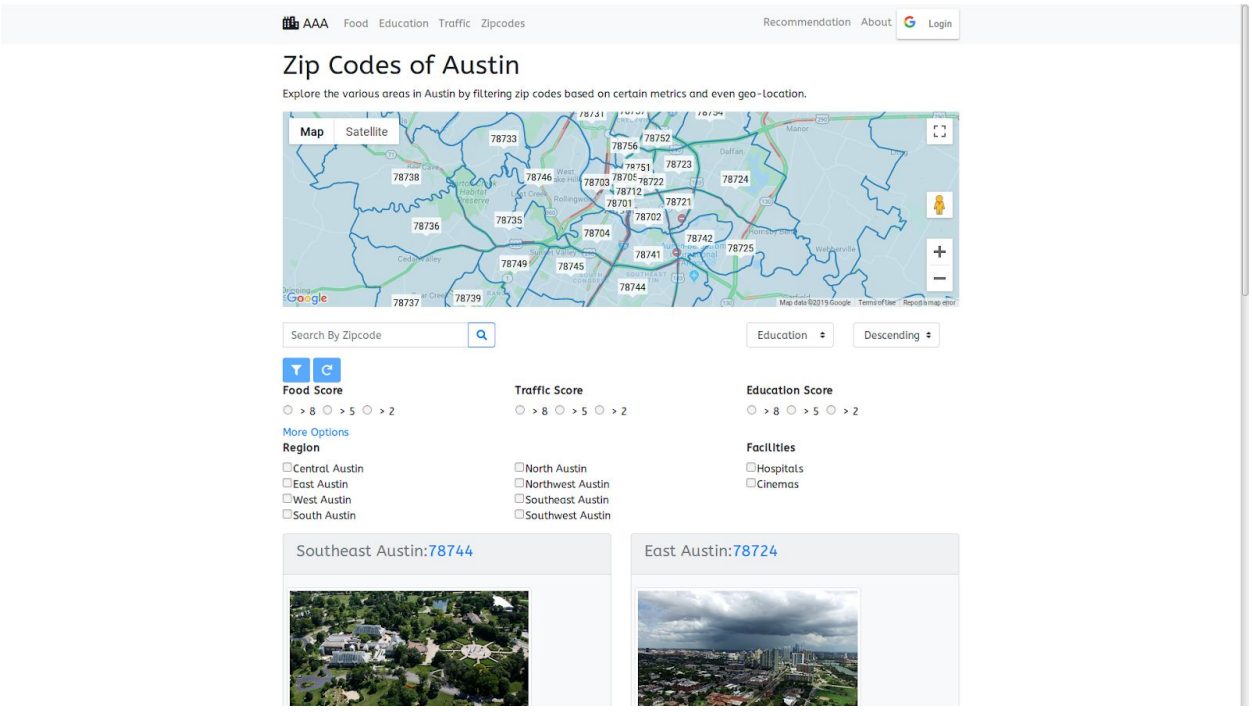
Changed sort order to Ascending.



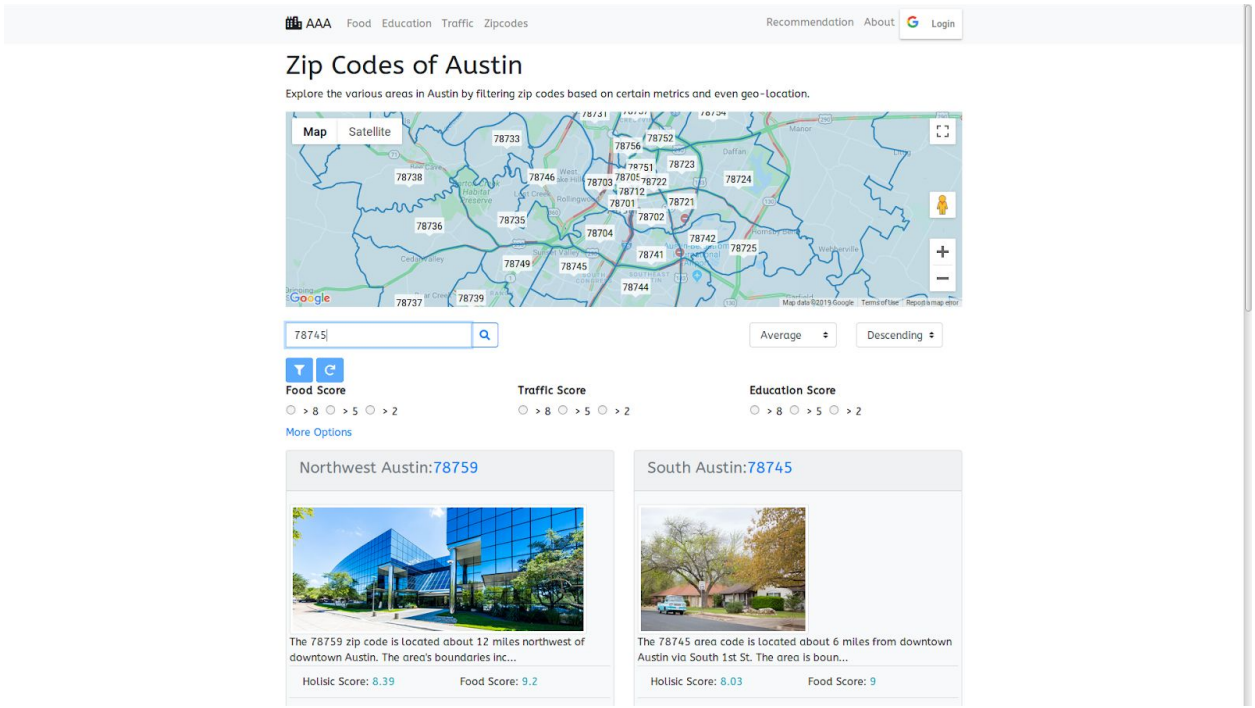
Sorting by Food.



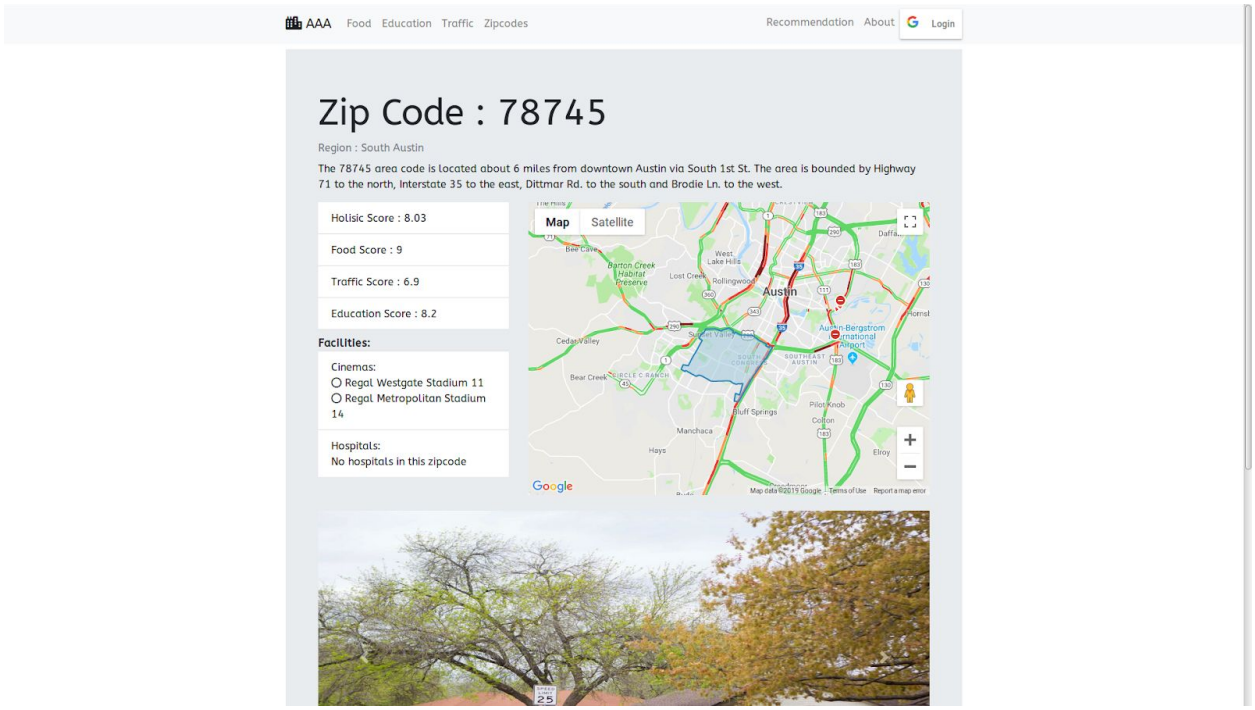
Sorting by Traffic.



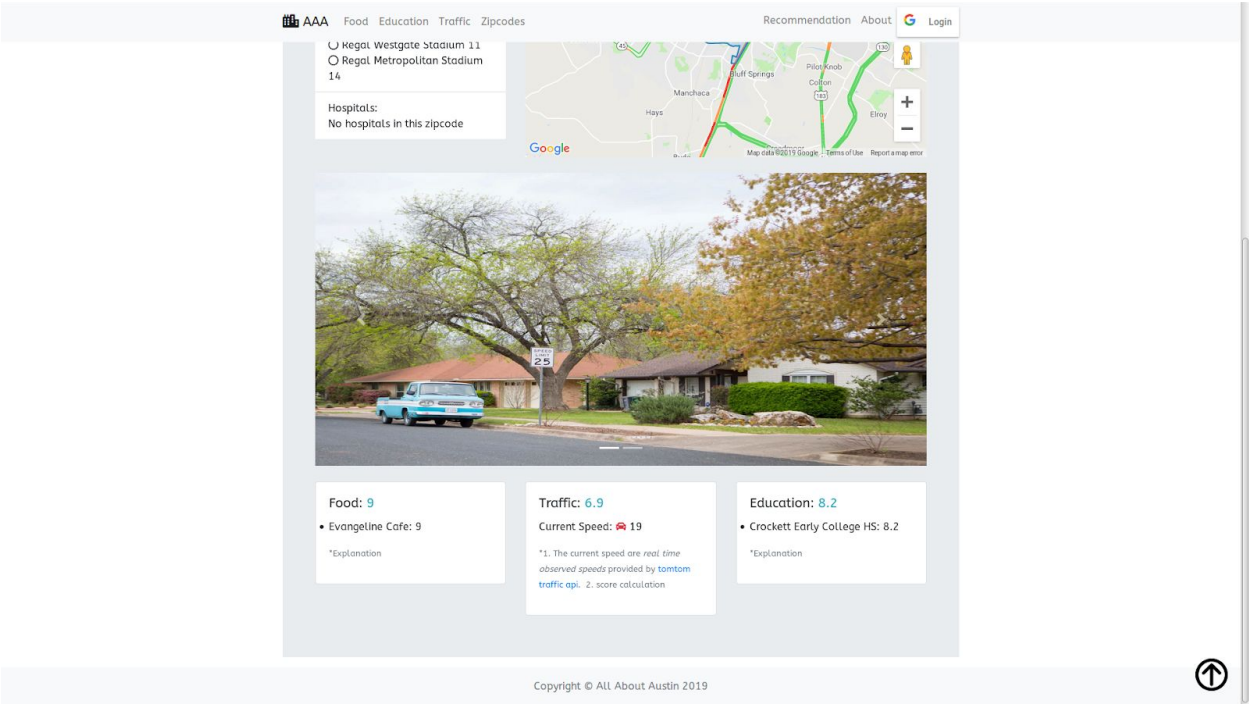
Sorting by Education.



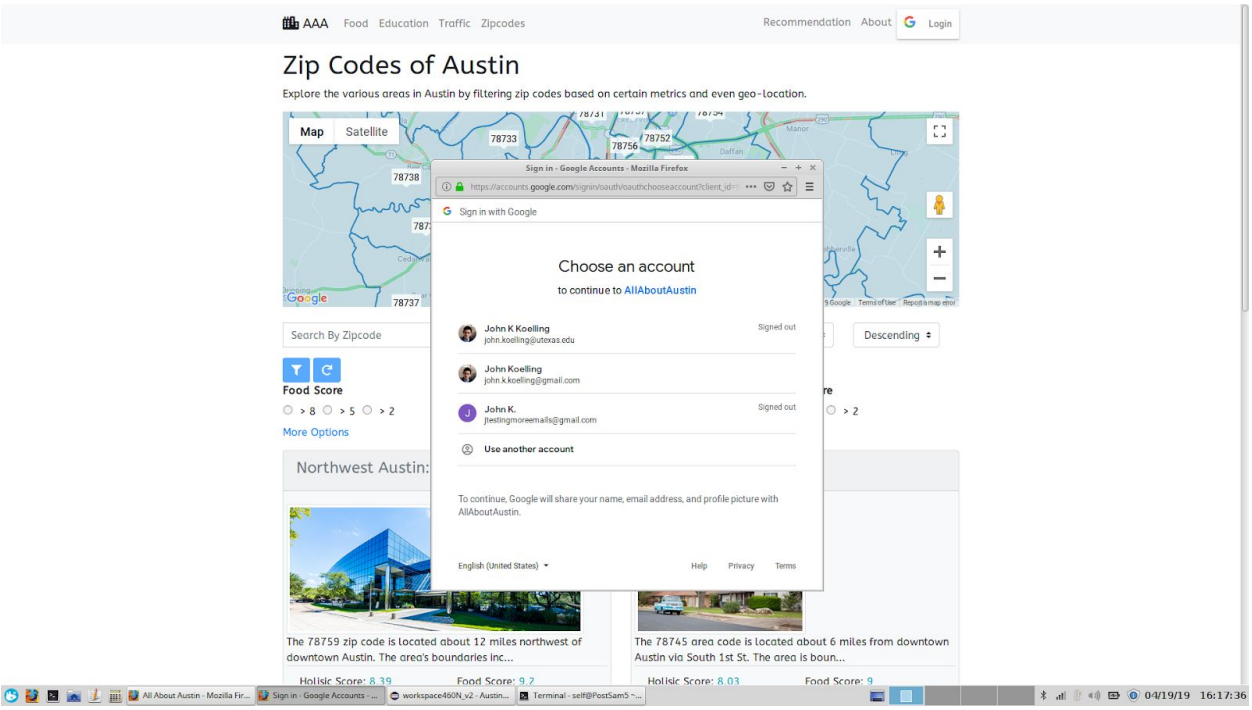
Filters reset. The user has typed in a particular zip code in the search bar, and is about to click Find.



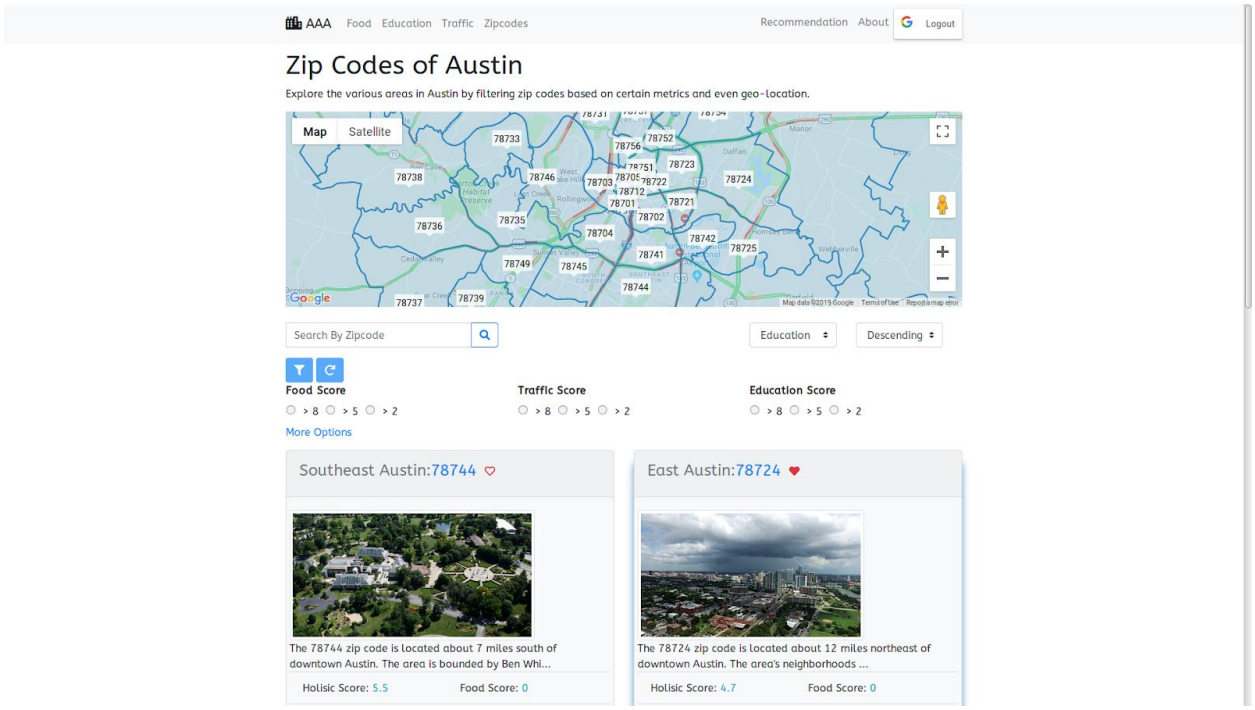
This brings the user to the page for that zip code.



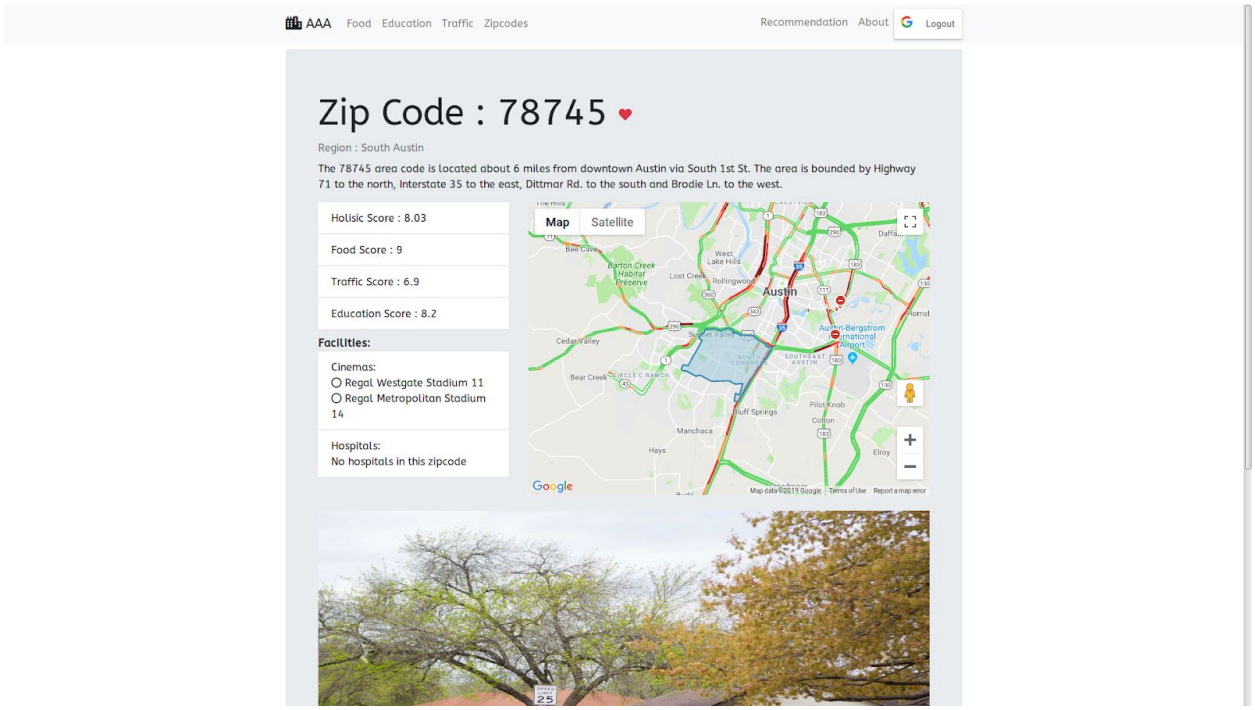
Detailed information is shown at the bottom of the page.



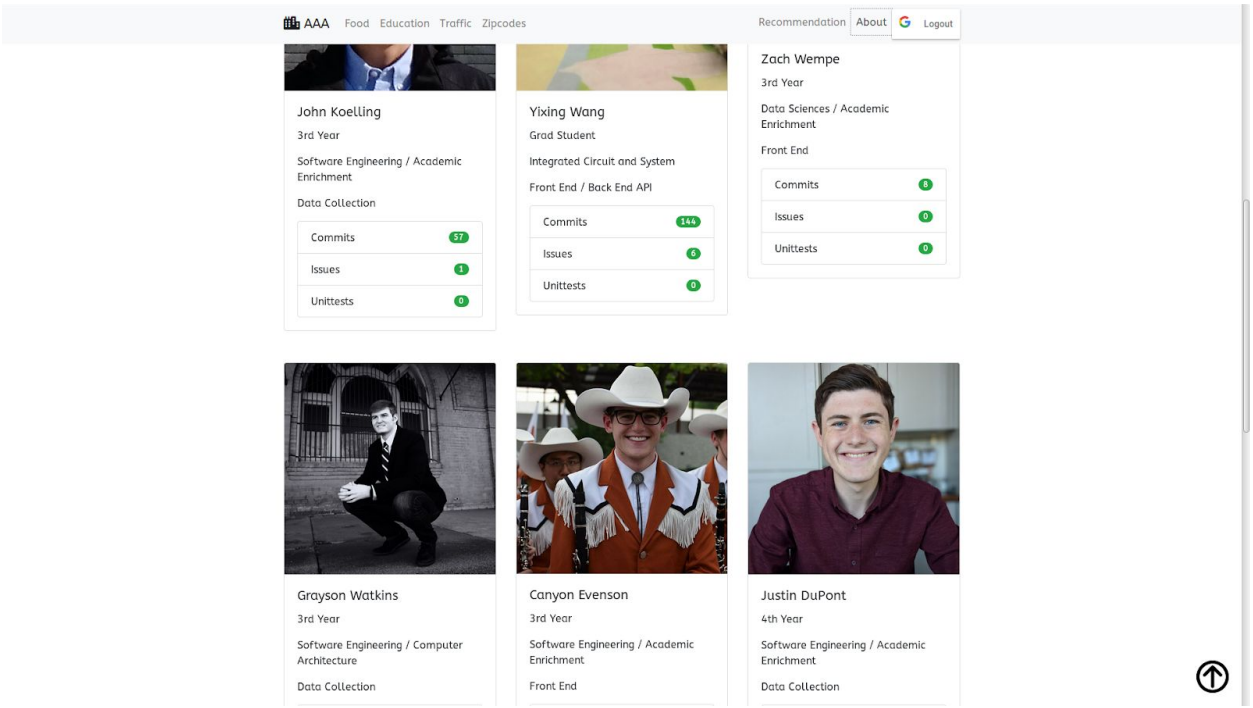
The user can log in with a Google account to save favorite zip codes. When the user logs out and logs back in, these favorites will persist and will be marked with a red heart icon. This was not an originally planned functionality, and both the login and preference saving are additional features.



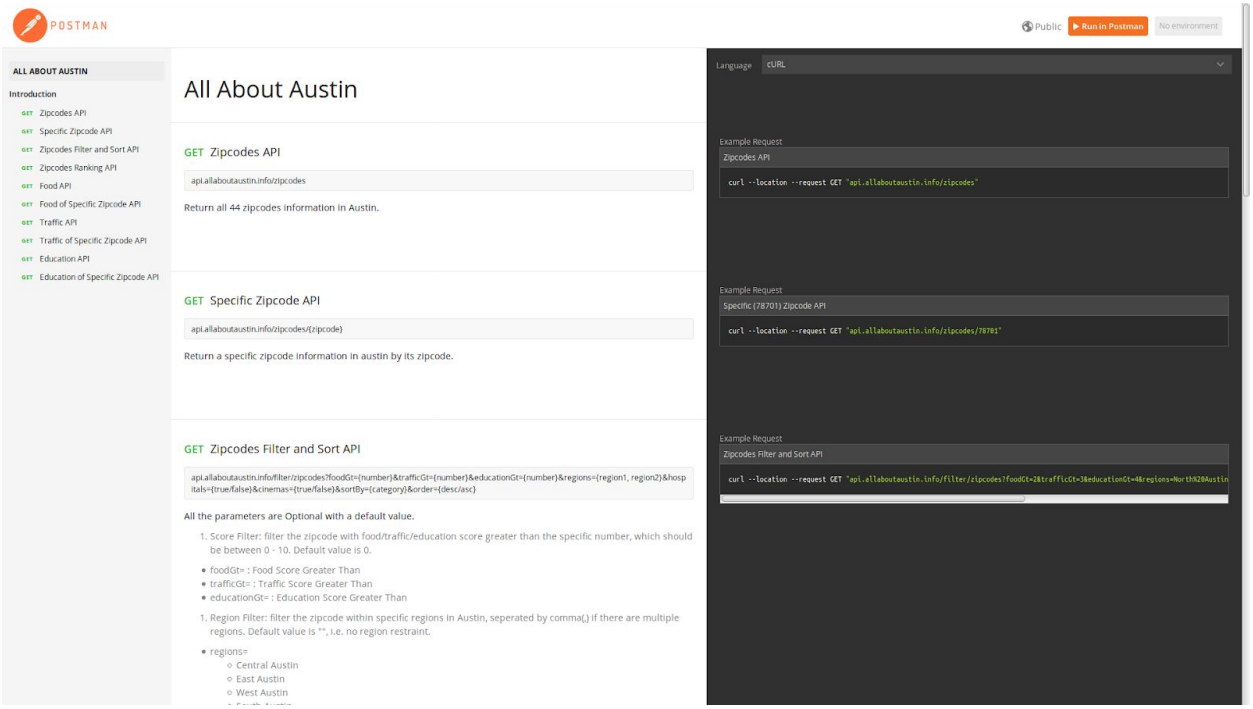
Shows a zip code saved as a Favorite.



The zip code is shown as a favorite on its own page as well.



This about page gives a clear description of our project vision in addition to introducing the team. In our report, we promised live updates as to each member's contribution to the project, as visible under each of the names. In addition to offering insight to commits, issues, and unittests, this about page gives insight to the tools and API's we utilized throughout this project.



Logging onto www.api.allaboutaustin.info presents you with our postman API homepage. The screenshot above shows the documentation interface of our postman API. There are a variety of GET actions that

can be used by an end user to access information for individual zip codes, regions, rankings, or even access to our distinct Education, Traffic, and Food API's.

Tools, software, and frameworks used:

- Front End
    - npm: Node Package Manager that works with Javascript (React).
    - React: A frontend javascript library for building user interfaces.
        - react-router-dom: A package that makes routing available for React.
        - axios: A package for HTTP request in React
        - react-redux: A package for state management for React Application
        - react-map-gl: A package for building interactive map in react with mapbox-gl
        - react-google-maps: A package that wraps google map api for react application
        - ...
    - Bootstrap 4: A front-end Web framework that provides HTML and CSS-based design templates to beautify the user interfaces.
    - Mocha / Jest: JavaScript testing tools.
    - Selenium: A GUI testing tool.
- Back End
    - Spring Boot framework: a Java backend framework. Used for developing RESTful API connecting to our MongoDB database.
    - Postman: A tool for testing our RESTful API.
    - Lombok: A java library that to facilitate writing standard functions in our Java classes.
- Database
    - MongoDB / MongoDB Atlas: A NoSQL database, Atlas allows us to easily set up and run a cloud database.
    - Morphia: A JVM Object Document Mapper for MongoDB, which allows us to easily interact with the MongoDB cluster in Java.
    - SODA, Socrata Open Data API: Data gathering API for Austin Government data.
    - JSON.simple: Java library to streamline processing of JSON objects from online sources.
    - Maven: Project build and dependency management.
- Other
    - Google App Engine / Google Cloud Platform: A cloud platform we used for hosting our website and API.

Testing Plan:

- Front End:
    - Unit testing for local state of each component with Mocha: Some of the components contain its own local state object for storing data, Setting state properly is essential for component rendering, thus we wrote unit testing with Mocha for state changes of each component.

    - Unit testing for redux actions and reducers with Jest: Jest is a testing engine recommended by Redux development team, thus we used Jest for unit testing of redux part. For Redux Actions, we wrote unit tests to test whether desired actions are created and whether the type is valid for both plain action and async actions. For Redux reducers, we wrote unit tests to test whether it return the new state after applying the action to the previous state.
    - GUI testing using Selenium WebDriver: Selenium WebDriver simulates user actions on a webpage and, this, combined with assertions, make up the GUI testing. Tests include map integration, sorting functionality, site navigation, data display, and proper display of GUI components when using the site. Tests were written in Python, and used Python's native assertion library. Currently tests are run in both Firefox and Chrome, with the potential for other browsers to be added as we refine our deployment strategy.

- RESTful API:
  Since we used Spring Boot for API development, there are mainly two part of testing.
    - Unit Testing for custom-defined query methods of Repositories with JUnit: For each repository, we implemented several custom-defined query methods for better data filtering. We wrote JUnit tests for every query methods to fetch data according to given parameters to verify that the returned data is valid.
    - Integration Testing for APIs with Postman: We varied the parameters and zip code of the path variables or URL parameters to ensure that the API gave correctly formatted data under all supported forms of request.

- Data Collection Application: We created JUnit tests for individual parts of the data collection application, testing individual modules at a time. Zip codes used for testing were: with data, without data, and invalid names. All of the data collectors were tested to ensure they provided non-empty data in a valid format. Test classes are in the Git repository at https://github.com/Iucundus/AustinData/tree/master/DataCollection/src/test/java/database/datacollection

Final Reflection:

We found that our some of our databases were sparse. Our database for traffic only had data for a few of the zipcodes, so we had to find a new database and convert our scoring system to the new traffic volume data. We also had to modify the education scoring system to look at surrounding zip codes education scores to make up for lack of education data.

We should start look into deployment on Google App Engine earlier, not at the last day of Phase II, since the deployment configuration requires much more debugging.

We found we had to deal with new forms of data: cinema information, hospital information, images, manually written descriptions, and other forms. To store these effectively, we needed to create a new system that would upload a CSV file into properties associated with the zipcodes in MongoDB, and merge this with our other dynamically derived data, which turned out to be somewhat complicated.

If we were to start over, we would approach this with a more test-driven development mindset, and create more tests before the main classes.

The system meets and exceeds all of our goals, as can be seen from the Project Proposal.