Motivation:

Crime has become a growing national issue, but an understanding of its actual rates, what causes it, and how to best deal with it, are still opaque to many Americans. We hoped to inform people of the facts of the current rates and types of crime, and analyze it through two important factors: the general demographics and socioeconomic status of the cities, towns, and counties where crime takes place (giving details on what crime happens where), and the police presence in these areas (what kind of measures do the police take, and what is the correspondence between police staffing and crime rate in their area of authority). By doing so, we wished to illuminate what parts of our current crime response structures are effective at dealing with it, and what should be tweaked for a safer America.

User Stories:

Customer suggestions we've taken to improve the website in terms of usability include:

Phase 1

The navigation bar was initially placed in the middle of the webpage. We modified and left-justified the navigation to bring it more in-line with the web standard. The lack of a footer and the fact that it was necessary to scroll back to the site's header to navigate was noted, so to address this, we added a footer with basic information about the website's source, as well as more links. There was a lack of general interconnection in the website beyond the header and footer, so one step we took to improve it was crosslinking the police departments' profiles to the city pages with which they are associated. Finally, we got a request from other developers for something to provide an example for how to improve the functionality of the schema for the API, by implementing a way to query police stations by a certain city. We thus provided a postman API with information from all of our models to allow for scraping of data (although the API was not initially online).

Phase 1 (As Customers)

The first tasks that we gave as customers were simple style additions that were vital to the look and feel of the website. This included the addition of a navbar and a footer that can provide us with clickable links to other pages. We also requested viewable content on the splash page that will provide us with information on what this website is about. As customers, we also noticed that there were no easy links to access other pages within the pages. This was a crucial request since the navbar shouldn't be the only way to access other content. Lastly, as a customer who is interested in using their information for our website, we requested a working API that allows us to easily get information from their website.

Phase 2

The about page was pointed out as being sparse, lacking information and pictures on the developers. This information was promptly provided, as well as a linkage to GitLab to provide stats that way. The home page appeared to be sparse and lacking in information that quickly explained the purpose of the

site, and the text that was there was hard to read. The remedy for this included making the home page more appealing, with a better splash image, as well as making it richer with information. While the added footer did add functionality to the site, the customer further clarified a request to make it have more information as opposed to acting as a mere secondary navbar. Thus, we implemented changes to make it look better and set up the potential to add extra links later (since currently there aren't useful links to add). On the actual data display, a customer requested being able to click on the coordinates listed on a city page to obtain where the city is located on a map, a useful feature that we implemented. There appears to be an error with the "recent crimes" display for the police department page, and not properly displaying the links to all the crimes associated with a department. There also was a request to adjust information display, to show a hierarchy of the information by importance (such as Primary Race).

In trying to help other developers with their website, some of the features we've requested were similar to the ones we received, such as adding a footer with detailed information, including more hyperlinks between instances of different models, adding a properly aligned navbar. Others were more broad features, such as adding an API for data scraping and building out the content on the website.

Phase 2 (As Customers)

Further input we gave as customers included suggestions for improvements to their home page (adding information about the icons, and information in general), and adding functionality to the footer beyond links to sources, and also navigation through the website or contact information. Other suggestions included changing the display of information to not require glancing between a legend to the table to understand acronyms, and this was addressed by formatting the description in the table column name. We requested better formatting on the About page, as currently the photos/descriptions are not aligned with each other, and one is on a row by itself. The About page should also have been moved to the end of the navbar, past the links to the actual content. All these provide quality of life updates so the user can more easily navigate through their pages. We believe that a user should be able to experience a website that is easily understandable and navigable, and that these changes will help their website achieve this.

Phase 3

We ensured that the data displayed on the page were not empty records. This was a serious concern as many crimes were not properly linked and police departments were showing up blank. The footer was also noticed to not be sticky, which was promptly fixed. Now, when reloading a page, the footer stays static. In addition, we formatted the specific record pages to more intuitively display the information. However, our group made some tough decisions to decide not to include some suggestions, such as the county map or the pagination refreshing. Besides this, we feel that the model pages, while not perfect, do have the necessary structure and visual appearance to appeal to users.

Phase 3 (As Customers)

For this phase, requests were more specific and mostly quality of life and design issues. This included adding a favicon to their website so that we the user can easily tell what tab we are currently on. Additionally, we wanted them to have labels on their graphs so that information is more clearly readable. Thirdly, a general search and filter feature to their website was requested so information can be more easily found. Fourth, we wanted them to improve their maps by showing the locations of jobs on the map, however, this request was out of the scope of this project and was declined. Lastly, we requested for an improved pagination feature that allowed the user to easily skip to the front or back of the list. Overall, these changes should elevate their website to be visually pleasing and have improved functionality.

Restful API:

The API is fairly simple since not much advanced SQL code is needed to combine the data into a usable format when the data is already geographically linked. The documentation has been linked below. To see example responses, please click the dropdown in the Example Request row.

https://documenter.getpostman.com/view/12923323/TVYC8zY2

Each model has 2 endpoints, one of which returns every record in the model's table. The second endpoint takes a parameter, which returns the corresponding record based on the primary key. For each model, the linked records are also returned through this call, i.e. a crime record returns the police department it refers to, police department records return the counties it covers, and the crimes occurring in it.

Hosting:

Our project is hosted on AWS EC2, handling both the React and Flask portions of our website. Amazon Route 53 is also used to direct traffic from our chosen URL to our EC2 instance. Finally, we chose to host the PostgreSQL database on GCP, allowing our backend to connect to the data seamlessly. This was due to some trouble concerning the database setup on AWS.

Models:

As stated above, our primary focus is on crime, and so that is the most important model. Some of the data points that can be sorted in the main table are:

- the type of offense
- the jurisdiction where the data is from
- number of offenders that committed the offense, by race
- number of victims, by race

These crimes, of course, do not exist in a vacuum, they occur in counties and they are reported and addressed by police, which form the other two models. The information on counties is fairly standard:

- Location

- Population
- Size
- Median Income
- Racial make-up

This gives us a framework in which to view the crimes, as it shows us the racial makeup of the county's population. Finally, we analyze police departments with data points such as:

- the number of officers, by gender
- the number of civilian officials
- the region/county they cover
- the density of officers per thousand civilians

This allows us to see the effect of police presence on the volume and nature of the crimes, thus completing the interaction of the three models.

Tools: We've used a variety of tools for this project, mainly Postman, to develop the API that will later be used to scrape the databases to populate our city/crime/police pages. Postman was useful for being able to address other RESTful APIs easily and understand how to write our own. It also helped in setting up a testing pipeline for our API. The databases we used include the FBI's Crime Data Explorer, and its other databases on police employment, and the U.S. Census's broad demographic data, arranged by county. For designing the style of the website, we used Bootstrap, a library/framework of CSS and javascript, to easily implement clean looking style and functionality for the website. React was also used to serve the front end website. We tested using a variety of tools. To test the javascript/typescript functionality, we used Mocha to write the tests and Chai to get a library of assertions to use in the tests. To test the backend, we used the built-in unit test framework in python to test the python code. For testing usage and reliability of the actual website with arbitrary user input, we used Selenium. Finally, we've used Gitlab to manage version control for the site, and it's API to manage the information on the About Us page.

Pagination:

We had a state variable that keeps track of the current page. There are also two buttons, a next button which goes to the next page, and a previous button, that goes to the previous page. Each page contains 10 instances. The pagination also wraps around. If the user selects previous from page 1, it will go to the last page. Further, selecting next from the last page will direct the user to page 1.

Sorting:

We had two state variables to help keep track of sorting. The first state variable was called "sort" and it stored the attribute name that was currently being used to sort. The second state variable was called "asc" and it kept track of whether we were currently in ascending or descending mode. There is one dropdown menu to select the attribute name to sort by, and there are buttons for ascending and descending to select sort order.

Searching:

Searching was implemented with a tool called Algolia. With Algolia, we uploaded our json data to the Algolia website which allows us to make calls during the search. Rather than making a search for each page and requiring a lot of code rewriting, we decided to have a general search page that can be accessed through the search bar in the header. We also believe this is an improvement since the models are connected and many features (Such as name) are shared between the different models. A general search page allows users to easily find attributes across the models. This search also has a fuzzy search feature, so it accounts for user error such as typos. The user's search request is highlighted for an easy to find experience. This feature only displays relevant data to the user, and the table used to display changes in size according to how many matches are found.

Filtering:

We had two state variables to help keep track of filtering. The first state variable was called "filter" and it stored the attribute name that was currently being used to filter. The second state variable was called "val" and it kept track of the value we were currently filtering by. There is one dropdown menu to select the attribute name to filter, and there is another dropdown menu to select the value to filter by. The filter variable defaults to no filter and the value variable defaults to all values. The second dropdown (for values) does not show up when the first dropdown has "no filter" selected, it appears once an actual filter is selected.

Database:

The database is made up of 4 tables. One is the Crimes table, which stores all information about the types of crimes. It has an auto-generated primary key and a foreign key 'ori' which links it to the police table. This table stores information about every police department, using the unique ORI number as a primary key. Since there is a many-to-many relationship between police departments and counties, we used an association table with an auto-generated primary key and two foreign keys, one for police departments and county IDs. The county table uses a primary key of the state number (01-50) + a 3-digit county number to create a 5-digit unique id, and stores all information relating to a county. The relationships are fairly linear, going from crime to police department to the association table to county, and allow us to easily query the database for the needed information.

Testing:

For testing the backend, we used python's built-in unit test framework to test if the Gitlab API returned information in the proper format: a list of dictionaries. Additionally, we tested that each dictionary contained the proper keys (name, commits, issues, and tests). We also tested that the names of all the tables in the database are the names that we were expecting.

For testing the website proper, we used Selenium to run tests on the pages for the 3 models of the website, testing to make sure that site elements and headers existed, and functioned, as well as ensuring

that links in the tables' pages worked (such as from the table to a particular county, police department, or city).

For the RESTful API served by our backend, we used Postman to generate a testing suite collection, which queries all the endpoints one by one, making sure that the status code is correct along with making sure the information requested is present. The collection also ensured that the request returned within an acceptable time period, ranging from 200ms to 400ms depending on the amount of information requested.

For testing the react components of our frontend, we used jest to test components of react. We tested if the App component would render without crashing, which includes all the other components we created within it. We also tested if the contents are accurate.