**Criminal Activity in Chicago, Illinois**

**Project Architecture**

This project employed several different technologies over the course of development. I created several different modules which were in charge of different facilities for the project in order to separate the various concerns of my project as much as possible. Firstly, I decided that an MVC framework of some kind would be useful to separate the business logic, the data, and the view. This decision came about because of the overall complexity of the project. I was going to have multiple views, all of which needed their own data and would present it in a different way from the others. I opted to use AngularJS, a JavaScript MVC framework provided by Google. I have had a fair amount of JavaScript and Angular experience in the past. This, coupled with the fact that I knew I was going to be using Google Maps JavaScript API, as well as D3 made Angular an obvious choice. My Angular application consists of a single html file: index.html. This file provides all of the markup for the application. I used Bootstrap, a CSS library provided by Twitter. This allowed me to better organize the different views and move the screen elements exactly where I wanted them. Behind the view sits the Angular controller. My application consists of a singular controller which handles all of the view events.

The controller provides an interface for view elements to interact and fire off triggers. The various buttons and forms all rely on the controller for their functionality. When a button is pressed on the view, the controller executes the corresponding function, once this function is finished executing, the view is updated with any new information that was generated during that time. When the view requests more data, the controller passes through requests to the API and initializes the view-model which the view has direct access to. Behind the controller sits the Angular services.

Services act as reusable pieces of code which different Angular components can include and interact with. My application makes use of several services which I constructed in order to modularize the application. The UtilSrvc provides miscellaneous utilities for other pieces of the application. This includes data structures which hold the different types of crimes that are available, as well as the different communities that are also available. The Crime service provides a constructor function for creating new Crime objects. Crime objects act as the client-side model for my application. This provides a universal interface for interacting with and accessing different data members. The DataSrvc provides a uniform interface for interacting with the API. I wanted to separate my implementation from my interface, and an Angular Service seemed like a good way of doing this. That way, if the API ever changed, then I could simply change the code inside of the function without the controller knowing. The DataSrvc simply takes a few parameters which allows the user to filter data from the server. The last service is the MapSrvc. The reason I constructed this service was just in case I wanted to use a different map provider in my application. If, for example, I changed from Google Maps to Leaflet, then I would have to go through my controller and remove any and all Google maps code. I would also have to figure out how to replace this code with equivalent Leaflet code in order to get the map working again. By creating this service, I simply exposed a few, key functions, which would work the same way regardless of implementation. The final pieces of the application were the directives. Rather than try to create the heat map and bar graph within the markup itself (as I knew that they were going to be D3 components), inside of script tags, I decided to push both of these entities into Angular directives.

Angular directives help to modularize frontend DOM elements by basically packaging up JavaScript and HTML into a single element which can be injected at any time into the DOM. I created one directive for each D3 element. The first element I worked on was the heat map. The heat map displays a matrix of rectangles which indicate the density of a certain crime for a certain community of Chicago. The heat map consists of three different rows and four different SVG canvases. The heat map gets notified of when the controller has made a request to the API, it is at this time that the heat map redraws itself. The heat map iterates over the data and aggregates it into a format that it can use. At this time, the set of crime types and communities are decided, and the colored rectangles which make up the map itself are constructed. Finally, the legend for the map is created. Rather than try to update the map each time, I found that it was easier to essentially clear the map on each update. From what I’ve seen, drawing the map from scratch isn’t very intensive, and clearing and totally redrawing the map simplifies the D3 code considerably. The bar graph directive only makes use of a single SVG element. The bar graph gives the user a different representation of the same data as the heat map. The bar graph shows the number of crimes for a given, single type of crime across the domain of communities which the user selected. Essentially, the heat map is a slice of a single row from the heat map, but allows for a different type of comparison than the heat map. The bar graph comes with a select box which allows the user to select from the available crimes. Clicking submit will propagate the change to the bar graph.

Figure 1 below provides an overview summary of the entirety of the project architecture. Note the direction of the arrows. Program flow moves in the clockwise direction until it reaches the server, at which point the flow moves back in the counter-clockwise direction.

The basic interaction loop starts with the user. When the user interacts with a certain control or user-interface element, this fires off a chain of events within the application, depending on the nature of the interaction. Most of the diagram represents what happens when the user changes the filter settings on the top bar of the application. This changes the overall dataset which the visualization represents. In this case, the controller uses he DataSrvc service to send an HTTP GET request to the node server. This request is intercepted by the Express API which matches the route to a predefined routine. At this point, the server defines a single API route with several different available options. The API uses the query string sent in the request to filter down the dataset which resides in an in-memory JavaScript array on the server. The newly constructed, filter array is then sent in the request body back to the DataSrvc. At this point, the DataSrvc pipes the received data through the Crime constructor function to build a new array consisting of model objects. These objects have the structure that the rest of the application expects, as such, they are sent back to the controller, which disperses the data throughout the application. Listeners on the heat map and bar graph modules watch for changes in the controller and update the D3 visualization if such a change is detected. This all results in an updated user interface for which the user can continue to interact.

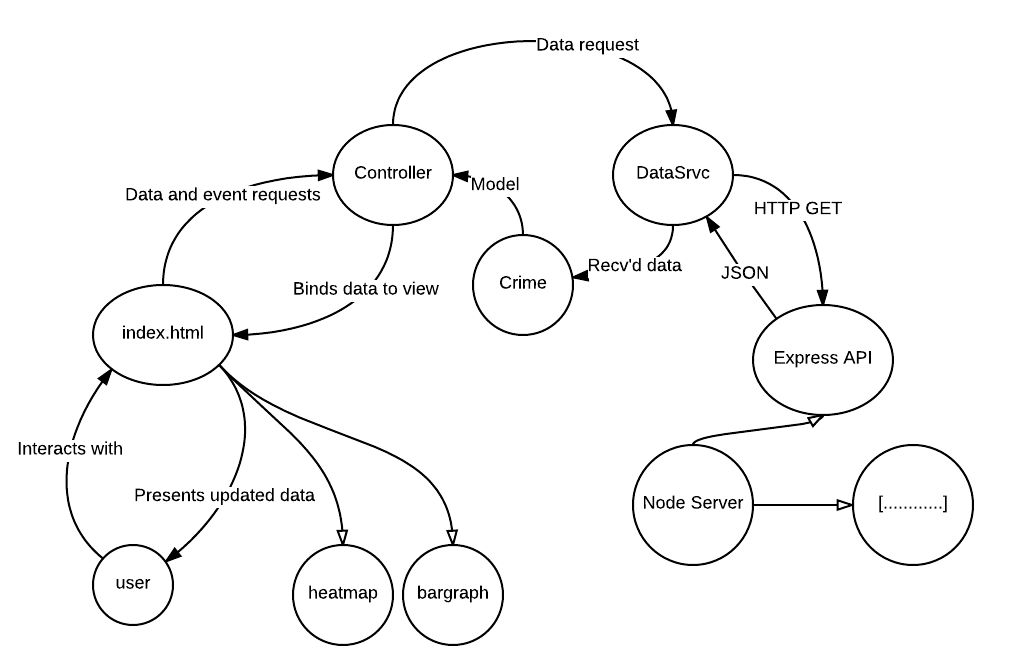


Figure 1: Project Architecture

**Project Details**

The project itself consists of four main data visualization views, along with several controls which the user can interact with to change the data and visualization. At the top of the main application window, the user will find a control with several different inputs. This is the control that the user can interact with in order to filter the total dataset found on the server. From here, the user can select year constraints for the data, as well as multiple selects which allow the user to select various crime types and communities. These selects allow the user to select multiple values from the list, rather than a single type/community. This way, the user can select a range of criminal activity to be check against a range of community areas. If the user does not specify either a type or a community, then the program will automatically use the entire set of communities and or types. This can prove to be quite computationally intensive, so the user is encouraged to filter the results as much as possible. From here, the user can press the refresh button to update the view, or the cancel button to reset the filter form to its initial state. The refresh button executes a routine on the server which pulls down all of the pertinent data. At this time, all four views are refreshed with the new data which the user can explore.

The first view consists of a Google Map window. This window displays the geographical locations of each of the crimes in the selected, filtered, dataset. This can show density, as well as crimes in relation to one another. Crime locations are displayed as clusters in order to make the map more readable. Clicking on a cluster will zoom the map a level lower and spread the clusters out. Eventually, the user will be left with single markers which they can click on in order to read the individual details regarding that particular crime.

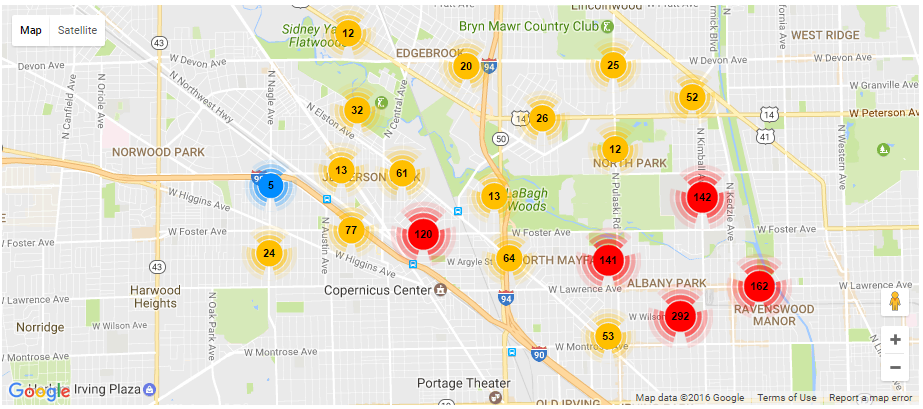


Figure 2: Map View

This takes the form of a Google Maps Info Window, which displays the time, date, location, type, and a brief description of that particular crime. A small icon is also available in the lower right hand corner. Dragging this icon onto the map allows the user to see a street view of the area and navigate through the map as if they were actually there. Zooming out once again aggregates the clusters. The user is able to pan and zoom through the view as they like.

The second view consists of an HTML table. This table represents what the current window of the map is showing at that time. Rows of the table are updated as the user drags the map view. Each row in the table represents a singular crime, of which the table shows twenty initially. If the user would like to view more rows of the table, they need only click the “Load More” button located at the bottom of the table.

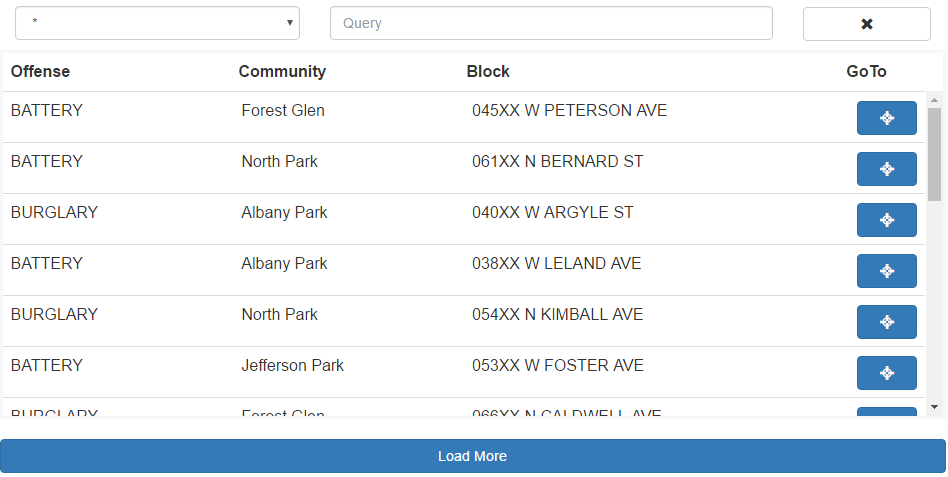


Figure 3: Table View

Each row of the table displays The type of crime, the community area the crime took place in, as well as the individual block. A button sits at the end of each row, when click, the map view is navigated to that particular crime so that the user can get a better idea of where, in relation to other crimes that that crime occurred. Above the table sits a query form which allows the user to search through the table for certain criteria. The user can search by type, community, block, or can search through each criteria using the wildcard option. Typing in the input box automatically refreshes the table with query hits, while pressing the reset button at the end of the form resets the table and query form to their original states.

Below the map view is the heat map view. The heat map shows the number of a certain crime committed in a certain community area. Titles across the top represent different Chicago community areas, while titles across the left side represent different crime categories. The squares in the middle of the heat map are colored in accordance with the density of crime occurrences.

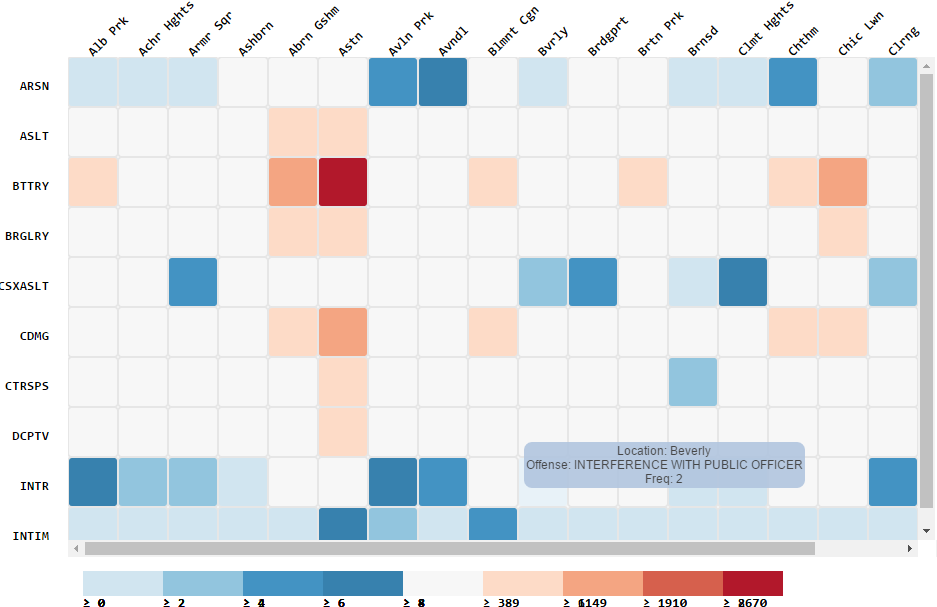


Figure 4: Heat Map View

Denser crime rates are colored red, while less dense crimes are colored in blue. Hovering over a particular square brings up a hover box which displays the location, type, as well as crime frequency for that particular square. This gives the user a brief, textual summary of what the visualization is trying to show at that instance. If the user selects a large number of either crime types or communities, then scroll bars will be provided. Communities and types will be fixed to their respective squares and will scroll with the whole heat map.

The last view of the application is a bar graph. This bar graph shows a lower level representation of the data than the heat map, though the data between them is the same. The x-axis of the bar graph represents the various Chicago communities. This includes every community which the user selected above in their filter options.

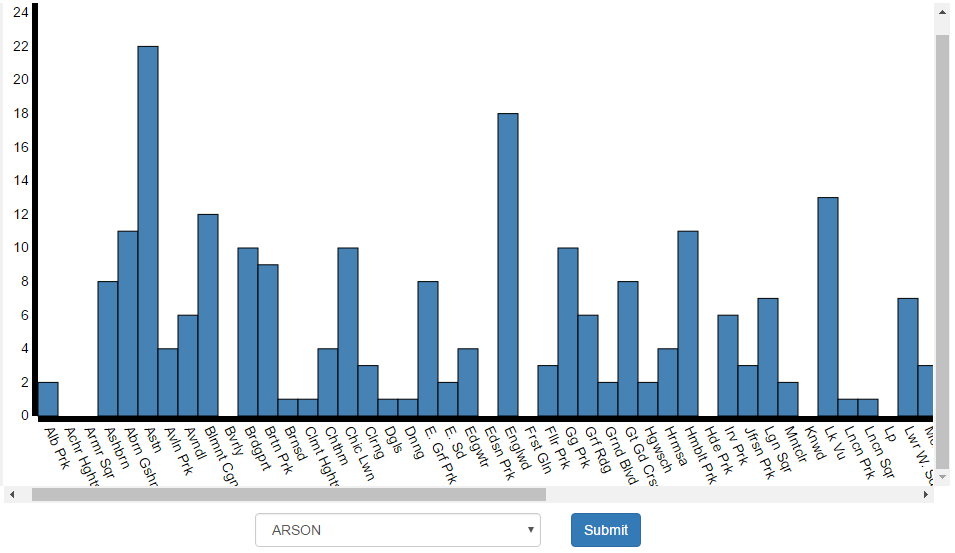


Figure 5: Bar Graph View

The y-axis represents the number of occurrences of a particular crime for that community. Below the graph is a select box and a button. The select box allows the user to choose from the set of selected crime times and change what the graph represents. Pressing the submit button propagates the change to the graph making the graph represent crime frequencies for the new crime type.

**Data**

The dataset I chose for this assignment consisted of a number of records regarding crime taking place in the municipality of Chicago, Illinois. This dataset is hosted by the city of Chicago proper, and is available publicly. This dataset is aggregated in real-time, I pulled down this dataset some weeks before this writing, so it is not as up to date as the publicly hosted data. The hosted data changes on a day-to-day basis, this project is not concerned with providing a professional-level interface for exploring the millions of rows of data contained in the dataset. Rather, this project seeks to ask questions and act as a proof of concept as to what can be shown and explored through this data. This project is in no way a comprehensive effort to explore the dataset completely and thoroughly. There are multiple ways to view and interact with this dataset. The city of Chicago provides several visualization views on the website itself. These include various map views. Filters can be applied to these views; these filters cover virtually every aspect of the data. The city of Chicago provides exporting of the crime data in many formats: CSV, JSON, RDF, RSS, TSV, and XML formats are all supported. Exporting the dataset essentially means that the data is pulled down from the server in the form of one big file. I decided that this was sufficient for this project and pulled the data down in CSV format. At the time I downloaded the dataset, I opted to download the full dataset which consisted of over six million rows including crimes recorded form January 1, 2001 until the date at which I downloaded the dataset. I figured this was a reasonable way to deal with the data, because I had experience parsing CSV files in the past with D3. At the outset, my plan was to load the data in from a local file using D3. If I ever had to change to a server architecture, then I figured it would be easy enough to incorporate D3 or some other CSV loader into my backend in order to load the data that way.

The format of this data is relatively straightforward yet comprehensive. The data in question is in comma separated value format, making the data easy to read and work with. Thankfully, most JavaScript CSV parsers automatically assign column headers as property keys when converting the data to JSON. A list of the data headers follows:

* ID – Unique identifier for this particular reported crime
* Case Number – Internal identifier used by the city of Chicago when referencing the investigation into this particular crime
* Date – The date and time string which represents when this crime was reported
* Block – Obfuscated city block address representing the nearest city block to which this crime was said to have occurred
* IUCR – Illinois Uniform Crime Reporting, a four-digit code used to classify crimes
* Primary Type – Essentially the category which the crime falls into (assault, burglary, theft, etc.)
* Description – A brief description describing what transpired in the commission of the crime
* Location Description – A brief description of the location of the crime
* Arrest – Whether or not an arrest was made
* Domestic – Whether or not the crime was domestic in nature
* Beat – The Chicago Police Department beat on which the crime took place
* District – The city district in which the crime took place
* Ward – The city ward in which the crime took place
* Community Area – One of seventy-seven community areas in which the crime took place
* FBI Code – FBI shorthand representation of the type of code which took place
* X Coordinate – Cartesian coordinate for the location of the crime
* Y Coordinate – Cartesian coordinate for the location of the crime
* Updated On – Date time string representing the last time this record was updated
* Latitude – Geolocation coordinate representing where the crime took place
* Longitude – Geolocation coordinate representing where the crime took place
* Location – String combination of the latitude and longitude

When working with this data, my first order of business was to decide how I wanted to represent it within the application itself. I have had a fair amount of experience working with JavaScript, so I knew that I wanted to have a way that I could represent the data as JavaScript objects. Thankfully, the parser I was using packaged the CSV data into JavaScript objects out of the box. Unfortunately, the representation of the data left much to be desired. Many of the “keys” being used in the JavaScript objects had spaces in them, requiring access and modification via square bracket notation (ex. Crime[“Primary Type”] === “ASSAULT”). This was rather clunky and not something I wanted to do every time I wanted to access an object property. Additionally, all of the “values” of the objects were in String format. This was not conducive to comparison or arithmetic operations, and I would need to parse and format many of the properties on each object. My solution was to create a sort of “class” in my JavaScript code. While JavaScript does not allow you to create classes in the conventional way, as is the case with languages like Java and C#. JavaScript does allow you to create “constructors”. Constructors are functions which create JavaScript objects from passed in arguments and gives these objects a sort of class-like feel. I came up with a schema of my own for how I wanted “crimes” represented within my application and packaged this into a constructor function which I called “Crime”. Every time I pulled in data from some source, I would run each data value through my “Crime” constructor, this left me with an array of Crime objects which allowed me to interact with the data both uniformly and unambiguously. Latitude and Longitude values were now Numbers, the Date key now represented a JavaScript Date object, and so on. This allowed me to do things like use latitude and longitude values directly when creating markers for a google map. Querying dates for different pieces of the date, and then using this data to format the date. One thing I knew I wanted to do was change how Chicago communities were represented. Chicago community areas are these discrete portions of landmass within the Chicago city limits. Each of these areas has both a name and a number associated with it. In the data, communities are represented as simple numeric values. Seeing these on a map or in a table is not very interesting in and of itself. So I decided to create a mapping between these community areas and the actual names of these areas. I ended up creating a hardcoded JavaScript object which contained both the “Code” for a community, as well as the actual “Name” of the community. References to different keys in this object were held by the crime records themselves and were created within the Crime constructor. This way, each crime object new both the numeric and the string representation of its respected community.

I felt like this data set was a fairly interesting choice because of how relevant and inscrutable much of this data is. Crime occurs on a daily basis, as such, the datasets which record criminal activities grow to be very large. However, actually understanding and learning from this data is a difficult task because much of this data is just numbers and shorthand codes. Latitude and Longitude values are all well and good, but not much can be done with them if you can’t see them on a map of some kind. I decided that I wanted to be able to see where and when crimes were occurring. Chicago is a very large city, which makes it a very interesting case study for crime. Being able to see when and where crimes are committed and at what frequency is, I feel, a very useful tool. And while I don’t think this project executes the idea flawlessly, I think that is nonetheless an intriguing proof of concept to how not just crime data, but data containing time, as well as space values can be visualized in relation to one another.

Working with this data certainly came with its own sets of challenges. I think the most obvious and prevalent challenge was the sheer size of the data. The full dataset that I ended up pulling down was around 1.4GB. The first thing I did was try to trim down this dataset to make it more manageable. Even opening the dataset in a text editor or Excel was a challenge. Most text editor programs are not meant to open such large files, and even Excel could not load every row of the full dataset. When I opened the dataset in Sublime Text, the full RAM usage for the program was around 3GB. I ended up cutting the full file into several different files, eventually creating a file with around 15,000 rows, which I used to accomplish most of the development of this project. At first, I simply loaded the entire dataset into memory on the client through D3. This proved to be relatively sluggish but allowed me to get a few of the features of the project up and running. Eventually, I decided to move to a client-server architecture, which included serving the CSV data from Node. Initially, I had the server open the CSV file and read through it each time a call was made to my backend API. This was quite a bit faster than simply reading from the file on the client. However, I was still a little unsatisfied with this approach. The thinking behind this approach was so that I wouldn’t have to keep the entirety of the file in memory, which would be the case if I simply read the file into memory to begin with. This would lead to basically two copies of the file sitting in memory (at least in the case with development, where the client and the server were on the same machine). However, I was advised not to worry about the server, and pretend as though the server has no shortage of memory. With this in mind I decided to read the file into a JavaScript array on the server-side when the server program was run for the first time. This sped up queries to the API fairly considerably. Setting up the API allowed me to segregate pieces of my dataset so that they weren’t all coming into the client at once.

**Future Work**

Many of the features I originally envisioned for this project are present in the final application. At the outset, I was unsure of what exactly I wanted this project to accomplish. The dataset I selected has a large number of dimensions, with any number of questions that could be asked concerning the data. Narrowing down my expectations for the project was a much larger challenge than I initially expected. However, I feel that there are a number of good features currently present in the application. That being said, I feel that there are several, perhaps obvious improvements that could be made to enhance the usability and richness of the application.

Firstly, I feel that the application could take advantage of a sort of warning system for the user. The dataset in question is quite large, and unsuited to be loaded into the client all at once. Even around 50,000 results causes the application to slow down considerably. I think that the application would benefit if user were warned when making queries to the dataset which result in a large number of records, causing the application to slow down. I could also devise a way to hide certain data from the users, a sort of pagination of the results that would allow the users to dig deeper into the data if they should so choose. This way I would only display a manageable number of records to the user, even though it appears that the whole dataset has been given to the client. This would be a fairly non-trivial fix compared to the query warnings however.

I also think that it would be useful to be able to explore markers which occupy the same latitude/longitude value. Currently, the MarkerClusterer library keeps markers that occupy the same geolocation together as clusters. Unfortunately, this does not allow for the user to bring up info windows on these particular pieces of data, limiting the experience somewhat. Additionally, I would like to link the other views to markers on the map. Highlighting a certain square on the heat map would display only those markers on the map, likewise with the bar chart. As of this writing, the MarkerClusterer library does not give any way to simply hide markers from view, so a different clustering library may have to be employed to accomplish this.

Ideally, I would like to associate crimes and their frequency more closely with when they took place. Right now, there is not much emphasis on crimes and their relation to different times of the year as I’d like. To remedy this, I would like to have a sort of time line which could display different clusters of crimes organized by the type of crime. This way, users could get a better feel for both when and where crimes are being committed, what parts of the year result in what types of crime, as well as get a feel for how different crime demographics have evolved over the years.

Beyond user interface improvements and feature additions, I believe I could put quite a bit of work into testing and improving the overall stability of the application. During the course of this project I did not endeavor to implement any sort of testing system. Much of this project was developed with a “fix-as-I-go” mentality, with no actual testing plan in place. This, coupled with the large amount of data and potentially limitless number of program states means that there are invariably states which the application cannot handle. Probably the most prevalent error case is if the parsed data ends up in a state which is not accounted for. This can happen if there were errors in entering the data, or if there is a data entry which does not reside within the specified data type domain (entering strings in number columns, misspelling of a crime type, etc.). At this point, I think it’d be best to implement a series of unit test cases in order to combat some of these error states.

Finally, I would like to conduct a user study with the application and receive some actual user feedback concerning the various systems and interface of the project. Right now, I have been the sole user of the application, conducting a user study with a Likert survey would go a long way to making the application act and feel more professional. I wouldn’t necessarily need to conduct a large user study; I feel that a small number of representative users would provide extremely valuable feedback for this project.

**Conclusions**

Overall this project was a very good opportunity for me to strike out on my own, explore and learn something new. There were some difficult challenges over the course of this project. I had little experience working with large datasets and in doing so, had to be careful how I used and presented this data to the user. Furthermore, several different software packages went into the creation of this application. I can’t recall a project which forced me to employ such a varied and comprehensive set of software tools. At times it was very difficult to have the different pieces communicate with one another. Passing data and commands back and forth between the different modules of the project became hectic at times, but Angular went a long way to keeping the various concerns of the application separate.

However, there are still many ways in which this project can improve. For instance, it is still possible for the user to pull in the entirety of the dataset from the server. This will invariably cause the application to be sluggish, or even crash. In this way, when using the application, the user should be thoughtful in their choices for what sort of data they want to view. Over the course of this project I realized how vast a dataset such as this can be. Not only in the sense that the dataset occupies a large number of bytes on disk, but also that there is almost an unlimited number of questions that can be asked about the data. I was unsure of what exactly I wanted this visualization to show at the outset of the project and had to really consider what views would be best for showing what kind of data. The multiple views portion of this project allowed me to spread the different the data around and allow the user to focus on individual aspects of the data per view. Being able to focus the user’s attention and draw them into individual perspectives helped me to show the user exactly what I wanted to show.

In the end I think this was a very interesting project that allowed me to do and learn things that I haven’t before. While the resulting application is by no means comprehensive or industry level in terms of quality, I feel that it still provides a good proof of concept as to what can be done and explored with this data. I think the dataset itself has a lot of interesting questions that could be asked about it, and given the time and resources, I think this project could grow to be quite interesting and informative.