User’s Manual

1. Required Software
   * Google Chrome (or equivalent browser)
   * NodeJS command line tool (“node” from the command line)
   * Node Package Manager command line tool (“npm” from the command line)
   * Git source control (“git” from the command line)
2. Quickstart
   * Clone or otherwise obtain the repository data.
   * Create a directory called “data” in the top level of the project (adjacent to app.js, heatmap.js, etc.), this step is unnecessary if you create the directory when extracting the data.
   * Download the data zip archive and extract them into the data directory you just created.
   * Install NodeJS with npm on the server machine.
   * From the command line, change into the top-level project directory and run the command “npm install” – this will install relevant software packages.
   * From the command line, change into the top-level project directory and run the command “node index.js” – this will start the server. Wait for a message which reports the number of records loaded into memory. By default, the server will use the smallset.csv file for the application model. This can be changed by editing line 47 of index.js.
   * With the server running on localhost, simply navigate a browser to <http://localhost:3000/> to run the application.
3. Getting the Source Code

Those wishing to acquire the source code for this project can find the relevant files on GitHub at: <https://github.com/Bret-Finley/crime>. The source code is available on a public repository. The repository can be cloned by running the command:

“git clone <https://github.com/Bret-Finley/crime.git>”, or by simply downloading the repository as a zip file. While this repository contains all relevant source files, there are some additional files which are required to run the project.

1. Getting the Data

Prior to executing the application, one must acquire the data upon which the application operates. This project utilizes data pulled from a public dataset provided by the city of Chicago, Illinois. The full, growing dataset can be found at: <https://data.cityofchicago.org/Public-Safety/Crimes-2001-to-present/ijzp-q8t2>. However, the actual data which I used in the development of this application is somewhat different. I have created my own sample datasets which the application can be run against. I have not included these in the GitHub repository due to their large file size. Instead, I have created an archive which contains several different sample files, this archive can be downloaded at: <https://www.dropbox.com/s/21np13s77fy02u3/data.zip?dl=0>. This archive contains a number of different samplings of the main data set: test.csv (3MB), smallset.csv (100MB), halfset.csv (500MB), fullset.csv (1.4GB). All of these files can be injected into the server in order to have the application run with these datasets. It is important to note that the server expects a directory called “data” within the same directory. Please create a so-called directory at this time and place the extracted data files within.

1. Running the server

To run the server, the user must download and install the NodeJS software package. This software can be found at: <https://nodejs.org/en/>. If you are running the server on linux, it may be easier to download and install Node with your package manager of choice. Mac and Windows users are provided nice binaries which usually contain both Node and npm, but on Linux they are usually separated (ex. Fedora users: “sudo dnf install nodejs npm”). Once this software is installed, the user should find several new command line utilities at their disposal. Simply change to the top-level project directory and run the command “npm install”. Once Node has installed the required software dependencies, the server can be run with the command “node index.js” This will start a local server listening on port 3000.

1. Running the Application

To run the application, simply navigate a browser (preferably Google Chrome) to <http://localhost:3000/>.

1. Using the Application

The application consists of a number of discrete features.

* **Filter Bar** – This appears at the top of the application. This tool can be used to trim down the total dataset into something more manageable or interesting. Users may select a starting year, ending year, as well as any number of community areas and or crime types. If the user does not select any crime types or community areas, then the application will select all of them by default. This is not recommended and can result in a large amount of data being sent to the client, potentially causing the application to crash. Pressing the refresh button will send the request to the server and refresh all of the other views in the application. The x button resets the application to its initial state.
* **Map View** – This appears in the upper left hand corner of the application. This view consists of a Google Map view with a number of controls and view elements. The main view elements on the map take the form of colored and numbered circles. These circles represent a number of crimes and their locations in the world. Clicking on the circle will zoom the map down a level, where the user can continue to click these circles until individual markers appear. When clicked, these markers display an info window containing details about that specific crime. Cross and dash buttons reside on the lower right hand side of the window which allow the user to manually zoom in and out of the view, this functionality is also mirrored by the scroll wheel on the mouse. Additionally, there is a stick figure button above the cross button. This stick figure can be dragged anywhere on the map to immediately zoom the user to a street-view of that location, where the user can pan and explore the location. Finally, a tabbed button appears in the upper left hand corner of the window. The map option of the button allows the user to see a stylized representation of the map view, while the satellite generates a map view based on satellite imaging, giving the map a more realistic look.
* **Table View** – The table view appears next to the map view and is responsible for giving the user a snapshot of what’s going on in the map view. The table view updates as the map view is panned or zoomed. The table view shows visible crime reports on the map and gives some high level information for each. The table displays the particular type of crime, the block on which it occurred, as well as the community are of Chicago in which it occurred. A button on the end of each table row can be clicked to immediately move the map to that crime. At the top of the table is a query bar. From here, users can filter the visible table rows by either community, crime type, wildcard, or block. Typing in the input bar updates the table in real-time. Clicking the x button will reset the form and table.
* **Heat Map View** – This view is located below the map view and displays the density of crimes in relation to the type and location of the crime. The rows and columns in this view directly reflect the types and communities which the user selected from the filter bar. Community areas are shown abbreviated across the top of the view, while crime types are shown abbreviated across the side of the view. If the user picks too many crimes or communities, then the view will allow for both horizontal and vertical scrolling. Squares within the view itself show the density of that particular crime at that particular community. Hovering over any of these squares shows the community, type, and frequency that particular square represents.
* **Bar Graph View** – This view is found below the table view. The bar graph is meant to show a different perspective of the same data represented by the heat map. Abbreviated communities are shown across the bottom of the graph, while the frequency of the crime is show on the y-axis. Below the graph is a select box which allows the user to select the type of crime being shown with the graph. Communities on the x-axis as well as types in the select box represent the selections made from the filter bar at the top of the application. Hovering over a bar brings up a contextual hover box which shows the location and the frequency of the crime.

1. Relevant Files
   * data/ - Directory which contains datasets for the server. Must be present for the server to run.
   * images/ - Directory required by the MarkerClusterer library for placing icons on the Google map.
   * lib/ - Directory which contains third party libraries necessary to the application.
   * node\_modules/ - Directory created by Node which holds various software packages for the server.
   * .gitignore – Ignore file for GitHub.
   * app.js – Main angular module declaration file.
   * barchart.html – Template HTML file for the barchart directive.
   * bargraph.js – JavaScript file containing the bargraph rendering logic.
   * controller.js – Single controller which acts as a sort of glue between view and model.
   * heatmap.html – Template HTML file for the heatmap directive.
   * heatmap.js – JavaScript file containing the heatmap rendering logic.
   * index.html – Main view file which defines the markup for the application.
   * index.js – Main server file which contains data loading and API routing logic.
   * package.json – Node project file containing a list of project dependencies.
   * README – Documentation file for the project.
   * services.js – This file contains all of the Angular service modules.
2. Miscellaneous

I was not able to install a comprehensive test suite for this application, so there very well may be various interface and data combinations which will cause the application to enter an error state, simply refreshing the application should render it usable again. Various datasets can be used in conjunction with this application, simply change the line within the server script (index.js) which loads the file from disk to. Depending on which dataset the user is applying, one can find themselves loading very large amounts of data directly into the client. More often than not this will cause the browser to be unresponsive. In light of this, the user is encouraged to apply as many filters to the data as possible in order to trim down the dataset. Deselecting either or both the community and type selection boxes is not recommended as this can result in a large number of records being pulled and rendered in all four of the views, causing the application to be sluggish.