



DUBLIN INSTITUTE
of **TECHNOLOGY**
Institiúid Teicneolaíochta Bhaile Átha Cliath

CrimeViz: Irish Crime Data Visualization

Final Year Project Report

DT228

BSc in Computer Science

Max Curtis

James Carswell

School of Computing

Dublin Institute of Technology

13/04/2018



Abstract

The intention of this project is to design and develop a system to allow for the visualisation of Ireland's crime statistics. This data is an untapped resource in its current state. The goal was to create an application that would let users make sense of this mountain of data that is available. Say if someone were writing a report on fluctuations in the crime rates in the country. Or if someone were curious as to whether burglaries in their area, or the area they are looking to move to, is on a decline or incline. Or for the Garda themselves to see areas that have been experiencing increases in certain types of crime in recent years. The design of this project is to convey this information quickly and easily, through a straightforward user interface and let people get some use out of all of this data on the crime rates in Ireland.

Declaration

I hereby declare that the work described in this dissertation is, except where otherwise stated, entirely my own work and has not been submitted as an exercise for a degree at this or any other university.

Signed:



Max Curtis

Max Curtis

13/04/2018

Acknowledgements

I would first like to thank my supervisor James Carswell for the assistance and direction he provided me throughout the duration of this project, and for keeping me in check with regards regular progress and not leaving everything until the last minute.

I would like to thank my parents for their support both emotional and monetary during my time at DIT, and for their help in establishing connections with people at the Garda to run the project by in the early stages and later to test the application.

Lastly, thank you to all my friends for being there for the last few years for the ups, the downs, and all the in-betweens, and for testing out the application when it was finished. In particular the friends I made at college as the days would have been that much longer without them.

Table of Contents

1.	Introduction.....	10
1.1	Project Overview and Background	10
1.2	Project Objectives.....	10
1.3	Project Challenges	10
1.4	Structure of the Document	11
2.	Research	13
2.1	Background Research	13
2.1.1	Data Visualization.....	13
2.2	Technologies Researched	17
2.2.1	Maps	17
2.2.2	Javascript Libraries	17
2.2.3	Databases.....	18
2.2.4	Cloud Computing Services	18
2.2.5	HTTP Servers	19
2.2.6	Other Technologies	19
3.	Design.....	21
3.1	Design Methodology.....	21
3.1.1	AGILE.....	21
3.1.2	Prototyping	22
3.2	Component Design	22
3.2.1	Database	22
3.2.2	Django API.....	23
3.2.3	Web Application.....	24
3.2.4	Source Code Layout.....	26
3.3	Features and Use Cases	26
3.3.1	Feature List.....	26
3.3.2	Use Cases	27
4.	Architecture & Development	29
4.1	System Architecture	29
4.1.1	Architecture Diagram	29
4.2	Project Components	30
4.2.1	Database Instance	30
4.2.2	Data	30
4.2.3	Django Droplet	32
4.2.4	Website Droplet	34
4.2.5	Web Application.....	35
4.3	External APIs and Libraries.....	41

4.3.1	OpenStreetMap.....	41
4.3.2	JavaScript Libraries	42
4.3.3	Django.....	44
4.3.4	NGINX	44
4.3.5	Own Code.....	44
5.	System Validation	45
5.1	Testing.....	45
5.1.1	User Feedback.....	45
5.1.2	Changes due to Feedback.....	49
5.2	Demonstration	51
6.	Project Plan.....	55
6.1	Initial Plan.....	55
6.2	Future Work	56
7.	Conclusion	59
7.1	Database	59
7.2	Django API.....	59
7.3	Web Application	59
8.	Bibliography.....	60

Table of Figures

Figure 1 Simple graph example, created using beam.venngage.com (2).....	13
Figure 2 Screenshot of the trulia.com displaying crime rates in New York	14
Figure 3 Screenshot of geographicprofiler.com crime mapping tool	15
Figure 4 AIRO Crime Mapping Tool showing Burglary rates for 2016 (5)	15
Figure 5 Close up of map section (5).....	16
Figure 6 Crime data example in database	23
Figure 7 Example of Django models	24
Figure 8 Example of Django serializers	24
Figure 9 Initial design for front-end main page	25
Figure 10 Initial design for the graphing page. Note: Crime rates are fabricated for rough draft.....	26
Figure 11 Use Case Diagram	28
Figure 12 Architecture diagram	29
Figure 13 Sequence diagram.....	30
Figure 14 Screenshot of vertical prototype	31
Figure 15 Crime data before rearranging	31
Figure 16 Crime data after rearranging	32
Figure 17 Example Dockerfile for Django image	33
Figure 18 Django API	33
Figure 19 Kidnapping data displayed via Django	34
Figure 20 DigitalOcean networking page	35
Figure 21 Code for handling crime values in loadData()	35
Figure 22 UI for selecting crime and year for heatmap.....	36
Figure 23 Map after selecting a Garda station	36
Figure 24 Code to find the nearest station	36
Figure 25 The menu for selecting additional stations	37
Figure 26 The graph generated from selected stations.....	37
Figure 27 Menu showing ability to compare multiple crimes.....	38
Figure 28 Data table	38
Figure 29 Function for populating table and making it selectable	39
Figure 30 makeChart function.....	40
Figure 31 Example of chart code	40
Figure 32 Code for initialising map with map tiles.....	41
Figure 33 Code for creating markers for stations	42
Figure 34 Example of AJAX code	42
Figure 35 Example of using jQuery to change something in the HTML	42

<i>Figure 36 Introduction page</i>	44
<i>Figure 37 Question 1 of the survey</i>	45
<i>Figure 38 Question 2 of survey</i>	46
<i>Figure 39 Question 3 of survey</i>	47
<i>Figure 40 Question 4 of survey</i>	48
<i>Figure 41 Question 5 of survey</i>	48
<i>Figure 42 Question 6 of survey</i>	48
<i>Figure 43 Question 7 of survey</i>	49
<i>Figure 44 Colour coded markers for stations</i>	49
<i>Figure 45 Some of the feedback regarding font</i>	50
<i>Figure 46 Old chart selection</i>	50
<i>Figure 47 New chart selection</i>	50
<i>Figure 48 Application at the start of testing</i>	51
<i>Figure 49 Application at the end of testing</i>	51
<i>Figure 50 Crime and year selected</i>	52
<i>Figure 51 Crime and year dropdowns expanded</i>	52
<i>Figure 52 Max value toggle</i>	52
<i>Figure 53 Illustration of station selection</i>	53
<i>Figure 54 Expanded chart dropdown and chart types</i>	53
<i>Figure 55 Crime comparison</i>	54
<i>Figure 56 Gantt chart from start of development</i>	55
<i>Figure 57 Variables used for crime rates</i>	57

1. Introduction

1.1 Project Overview and Background

Data visualisation is the method of presenting data in a manner that makes it easier and more clear to understand (1). The ideas behind data visualisation and why it is important will be discussed in greater detail later, but in essence it is a way of making use of something that has become one of the largest untapped resources in the world. Big data, a commonly heard word in the world today, has been compared to oil in regards its usefulness and the amount of it laying untapped just below the surface (2). There is so much data that has been gathered, with the amount of new data created and gathered daily constantly increasing. Companies the world over are upping their investments in the analysis of this data so that they can fine tune their business models. According to International Data Corporation global spending on big data analytics is set to grow beyond \$200 billion by the year 2020 (3).

Despite all of this there are still so many areas where data is collected and left unused. One such area is the crime statistics of Ireland. This project is intended to take this data, or what is available of it, and make it more accessible to the public. The data will be used to generate heatmaps illustrating the hot zones for various crimes across the country and allow users to create graphs of this data to facilitate analysis of the data.

1.2 Project Objectives

The ultimate objective of this project is to have an application that will allow users view the data on Irish crime rates in a more easily consumable manner. There are a lot of sub-objectives that will be focussed on that will lead to this. The datasets will need to be gotten and transformed into a structure and format that will allow them to be used in this project. The data will need to be housed somewhere that can be accessed by the application remotely. The application itself will need to be user friendly and not overly cluttered. It will need to have functionality for generating the heatmaps from the data and for creating graphs based on selected data. Ideally the graphs will be able to be exported for external use. Additionally, the ability to include census data of the areas would be beneficial to avoid situations where the area around Dublin is red with no colour anywhere else due to the high population density of the area. Though the last two objectives are secondary to the main functionality the goal is to have all of these features complete.

1.3 Project Challenges

A major problem in this project is the transformation of these datasets that were created completely independently of one another to facilitate using them together. They are distributed in different data formats and different structures, so a lot of time will need to be spent working on the datasets to allow them to work together.

Due to the multiple different functions of the application several libraries will need to be made use of. Not all libraries will work well together so there is expected to be some research needed into which libraries will best serve their individual need while also being able to be used with others. Primarily the more specific libraries like the heatmap and graphing ones. There are a multitude of libraries out there that will serve these functions, but it is not unheard of that they may cause bugs in one another. There

is also the issue with keeping the user interface simple enough that end users will be able to make sense of it and interact with it without too much hassle.

The biggest potential issue is due to the scale of the system. Because it will require the database, the method of accessing the database, and the front-end application it will not always be apparent where errors are occurring. For instance if data is not displaying properly in the heatmap it could be due to the heatmap not being configured correctly. It could also be due to an oversight somewhere in the method of accessing the database. It could be due to the database itself not being set up properly. It could even be due to an error made in the earliest stage of transforming the data. There is a lot of room for error in a system like this and it will not always be apparent where the error is, so it is expected that a lot of time will be spent troubleshooting.

1.4 Structure of the Document

Section 2 – Research

This section begins with a discussion of the topic of data visualisation and its importance both in industry and in research. Then it goes on to discuss some existing examples of the intended result of this project from other countries, along with one in Ireland that could stand some improvements. Followed by an explanation of the technologies researched for potential use in this project, the ones that were chosen and the ones that were not. These include map systems, databases, javascript libraries, cloud computing services for hosting, HTTP servers, and miscellaneous other technologies.

Section 3 – Design

In the design section some of the early thoughts behind the design of the overall system and its components are discussed, starting with the design methodologies that were examined. It continues on to look at the structure of the database and the considerations that went into that after examining the datasets. Then the creation of the API to match the database and its tables. There are some early prototype images of the web application shown, along with the user scenario that was written up to establish a list of features, and the use case diagram of these features.

Section 4 – Architecture and Development

Section 4 looks at the architecture of the overall system and how it interacts along with an in depth examination of the development process of the individual components. To illustrate the architecture there is a system architecture diagram and sequence diagram showing the functions that call to the back-end. For each component there is an account of the process of implementing it in the system, as opposed to section 3 where it was just a look at the design of these components.

Section 5 – System Validation

This is the section where testing is discussed. There were two separate types of testing seen throughout the development of the project. The first being integration / regression tests performed by the developer regularly to maintain the complete functionality of the application. Second is the user tests, performed at the end by the intended end user groups. Included are the tabulated results of surveys taken by tester, graphed results of the surveys, and some of the noteworthy comments and recommended modifications along with changes that were made because of this feedback.

Section 6 – Project Plan

In this section there is a look at the initial planning that went into the project, along with an examination of the Gantt chart that was drawn up after the interim demonstration of the prototype. This discussion includes how well the schedule was adhered to, intended features that made it in, features that had to be left out and why, changes that were made from the original plan.

Section 7 – Conclusion

The conclusion takes a look at the project as a whole and the individual components in their finished states. There is a mention of how well the finished product performs and comparison to the original intention. Some discussion is had on where components do not meet expectations.

Section 8 – Bibliography

This is where the references used throughout this report are found.

2. Research

2.1 Background Research

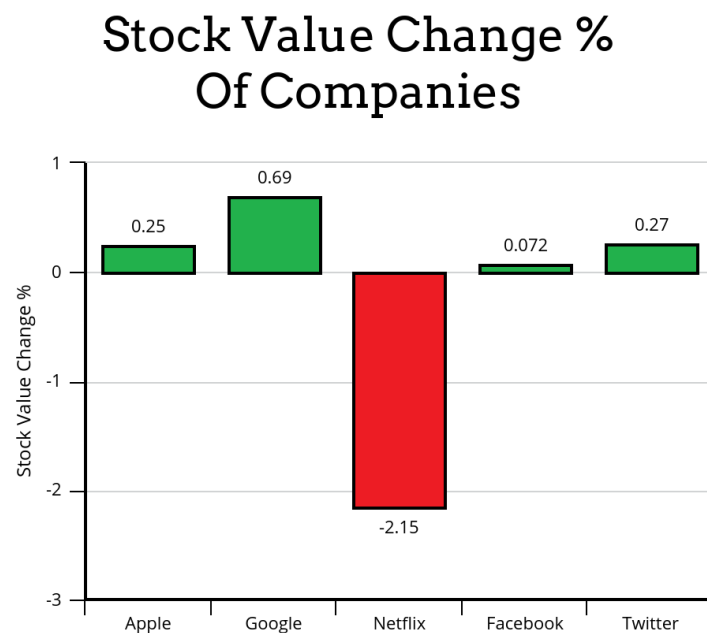
2.1.1 Data Visualization

After decades of researchers gathering data and over the years the process being optimized by technology and yielding greater results, now more than ever data visualisation has become of great importance to businesses and administrations alike. All the data that has been gathered is of no use if it cannot be properly analysed and have meaning extracted from it. Through data visualisation users can observe and manipulate these large datasets in a multitude of ways looking for trends, correlations, anything that might be of use in improving business and increasing profits (4).

Data shown in its basic form can be difficult to find meaning or interest in. Take for instance data on stock value changes from the Forbes website which is referenced above:

Apple +0.25%, Google +0.69%, Netflix -2.16%,
Facebook+0.072%, Twitter +0.27%.

The information is all there, but it does nothing to grab the attention of the reader and needs to be really looked at to compare it. This is not a difficult task in this case with 5 companies represented but think if it were an indication of 50 companies, or 500 companies. These are realistic possibilities in the world of stock exchange and it is important for people to be able to see the changes quickly. Now look at the same data represented in a hastily created bar chart.



Source: www.forbes.com

BEAM venngage.com/beam

FIGURE 1 SIMPLE GRAPH EXAMPLE, CREATED USING BEAM.VENNGAGE.COM (5)

For someone who knows what they are looking at like a stock broker it is much the same thing in this small example. For someone who would be less used to seeing stock information it gets across the information more quickly and is more attention grabbing. Useful if presenting the data to the public or to a higher up at a company whose main concern is to see that their company is seeing an increase in value and not another red bar on the chart. It allows for easier comparison as well. For instance one can quickly tell that Apple and Twitter saw a roughly equal increase, while Google saw over twice the increase of them.

Likewise, this can be of use in areas like research on climate change with tracking trends in temperature or weather with a line graph, in medicine tracking spreads of diseases with a heatmap, or to show crime rates across the country and look for trends therein.

Early research into this area revealed a lack of user-friendly methods of viewing Ireland's crime data. The data itself is available to all on the Central Statistics Office (CSO) website in its raw, spreadsheet form alongside some pre-made graphs with small subsections of the data illustrated.

Broadening the search to include tools available to other countries revealed an example from the United States the website trulia.com (6), a property focussed website that allows users to view crime rates in multiple areas of the country.

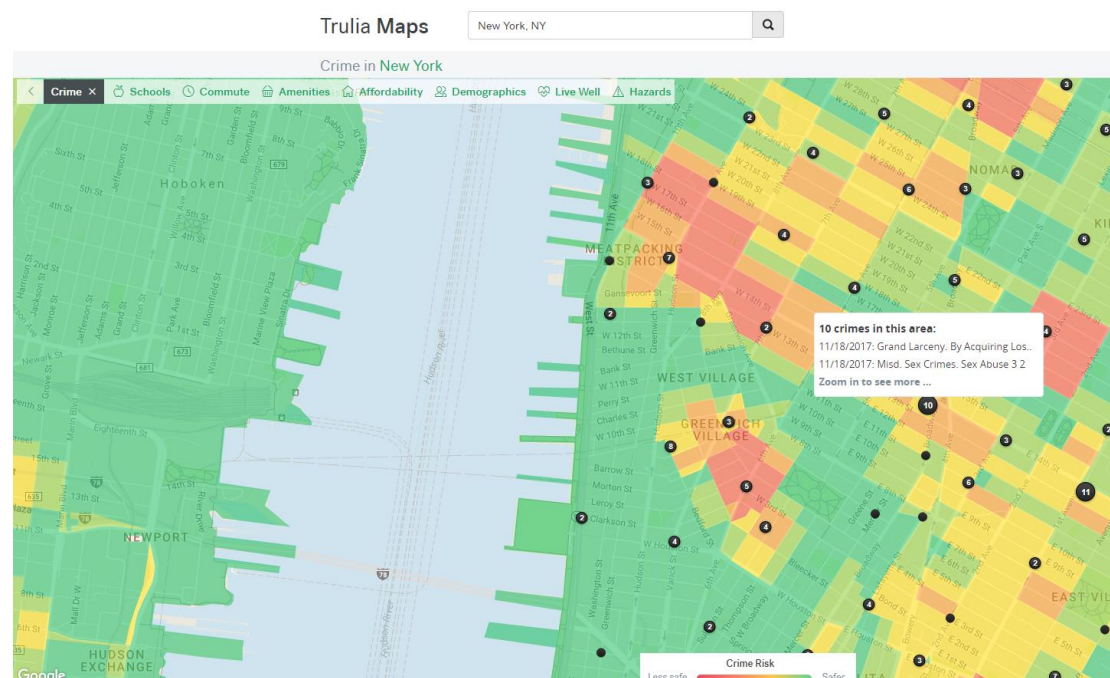


FIGURE 2 SCREENSHOT OF THE TRULIA.COM DISPLAYING CRIME RATES IN NEW YORK

This tool implements a heat map system as is the goal with this project. It also shows the actual location of the crimes occurring, as indicated by the black circles above. This is a functionality that is beyond the scope of this project due to the data available at the moment. Another key feature in this application is the ability to change the dataset being visualised. There are options for schools, housing costs, and demographics. Although there is no option to use these datasets at the same time as the crime dataset it still allows for quick comparisons between the visualised data. There is also no option for selecting specific crimes, or for viewing different timeframes.

Many other examples were found including one as close as the United Kingdom, an application on geographicprofiler.com (7) that provides a heat map of different areas based on reports from the police.



FIGURE 3 SCREENSHOT OF GEOGRAPHICPROFILER.COM CRIME MAPPING TOOL

In this tool there is a hard divide between the interactable area on the left where users can specify the crimes, the year, and even the month they wish to view the data of. Unlike the American application there is no specific data on the locations where the crimes occurred, so the output of the application more closely resembles the end goal for this project. The added layer of depth to the dataset that is the months allows for a better plotting of crime rates, as opposed to just seeing the rate for a year the user can see the fluctuation through the months, as the weather changes, more closely map changes to occurrences in that year etc.

The Gardaí themselves were contacted to see if any such application exists but is not open to the public. Upon description of the project idea they said that they have no such tool and were interested in the idea. Given the way the dataset is laid out the mapping system is not able to be as precise as some of the examples from other countries, giving exact locations of crimes plotted on the map. But the data will still be sufficient for the purposes this tool looks to serve, these being the construction of a heatmap and allowing users to generate their own charts showing the crime rates of areas.

There is a tool that exists known as the AIRO (All-Island Research Observatory) Crime Mapping Toolkit (8). It was created in 2011 and while useable there are improvements that could be made upon it. For instance, the data is presented by placing a node point (in this case a circle on the map) at each Garda station and then colouring the node depending on the range it falls into.

Recorded Crime Monitoring Tool

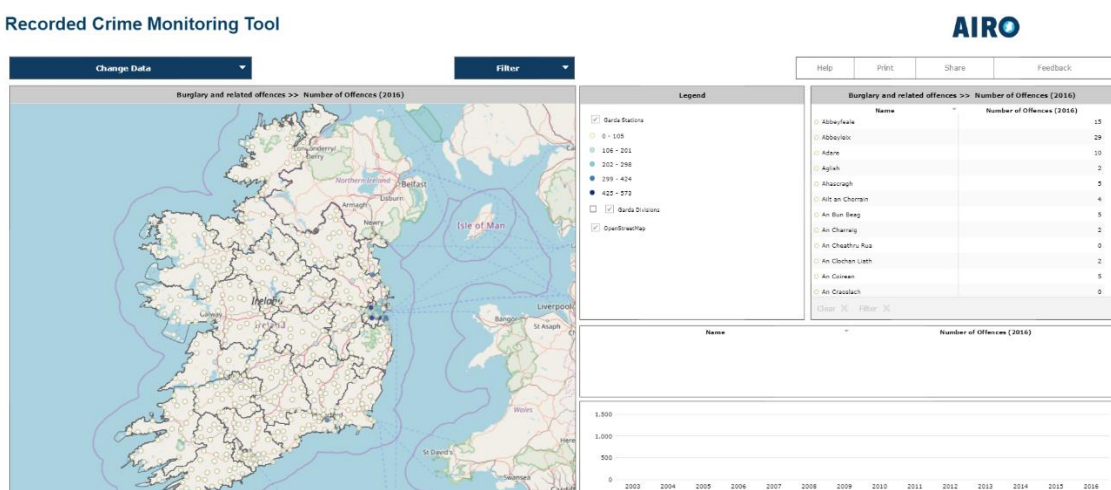


FIGURE 4 AIRO CRIME MAPPING TOOL SHOWING BURGLARY RATES FOR 2016 (8)

The map screen is shown on the left half with a node for each station, these nodes are coloured based on the range of occurrences they fall into. The colours are explained in the legend to the right of the

map. The numbers can then be viewed to the right of that again and sorted in order, with a graph of the years available below. This graph system allows the selection of multiple areas.



FIGURE 5 CLOSE UP OF MAP SECTION (8)

The nodes might not be the most effective way to communicate the information. Due to the population density difference between Dublin and the rest of the country most selections of data will be like the example shown above with a few darkened nodes in Dublin and largely blank nodes everywhere else. For this reason, a heatmap could serve better. Showing a larger range while still being clear where the changes occur and where the hotbeds of these crimes are. The nodes can still be present in some form as a way to view the actual numbers for an area when selected. Also, as a way to show the user the areas that they have selected for more in-depth analysis. This also provides an argument for the inclusion of census data. Census data will help mitigate skewed results like this by taking the population of an area into account.

Below is a table showing the main functionality found across the different crime mapping tools examined along with the intended functionality of this project.

	Trulia	GeographicProfiler	AIRO	This Project
Heat map	Yes	Yes	No	Yes
Crime selection	No	Yes	Yes	Yes
Year selection	No	Yes	Yes	Yes
Month selection	No	Yes	No	No
Locations of crime occurrences	Yes	No	No	No
Graphing functionality	No	No	Yes	Yes

The big things that will be missing from the application being designed here is the ability to view rates for specific months and the ability to see more specific information on where the crimes occurred. Both

of these shortcomings are due to the nature of the data available and could in future be fixed if the datasets were to be improved.

2.2 Technologies Researched

2.2.1 Maps

Google Maps is Google's web mapping service, the most used web mapping service in the world today boasting over 1 billion users per month since 2015 (9). There is an API that is available for developers to use when developing web mapping applications (10). There is a free tier that allows for up to 25,000 map loads per day for free in a web application. While this project will not be approaching those numbers in the interest of creating an application that could be future-proofed (to a degree) a different mapping system was chosen.

Mapbox is another leading web mapping service. Reaching 300 million a month, smaller than Google Maps' numbers but still a large number of people. Mapbox is the mapping service of choice for several large companies such as Airbnb, Snapchat, and National Geographic (11). While having a free tier as well it has a significantly lower cap than that of Google Maps at 50,000 map views per month. The decision was made to go with something completely open-source and free provided that the company are credited.

OpenStreetMap is another web mapping service. It is of a high quality and kept up to date largely by the user base and the OpenStreetMap Foundation, an international, non-profit organisation that exists for the support and maintenance of OpenStreetMap without trying to completely control the system (12). OpenStreetMap allows any use free of charge provided the service is credited within the application and for this reason it was chosen as the mapping system of this application (13).

2.2.2 Javascript Libraries

React is a well-known JavaScript library. A JavaScript library is a bundle of pre-existing functions that have been written by other people. They are simple to include in a project and help to expand the functionality of said project while limiting the need for new code (14). React is a library that is largely used for designing user interfaces that allow for an increased level of user interaction over that of one designed in basic HTML and Javascript (15). React, while powerful, did not fully suit the needs of the project.

Leaflet is another JavaScript library. It is open-source and intended for use in creating interactive, map-based applications. This made it the ideal choice for this project. It facilitates using different layers to overlay things on top of a base map image. It also has functions for displaying markers which will be suitable for marking Garda station locations. It has functionality for reading and working with JSON and GeoJSON, two data types which are used to store the Garda station locations and the associated crime data.

JSON is a data type that allows the storing of many different types of data inside of a single data object. This is done in the form of key / value pairs, i.e. A key that is used to identify the type of data and then the corresponding value of that data item. An example of a key value pair in a JSON object would be

“colour : blue” or “age : 24”. Both different types of data could be stored in and used from the one JSON object (16). GeoJSON is much like JSON but with the added bonus of being able to store geometric data as well, such as points (like a point on a map) and polygons (a shape with many sides) (17). This geometrical functionality makes it ideal for this application.

Leaflet is also mobile-friendly so any future expansion to develop a mobile application as well will be made easier by using Leaflet (18).

jQuery is a third JavaScript library that was looked at to potentially be used in this project. It has several benefits such as its fast nature and small file size for including in projects. It is an expansive library with a multitude of available plugins that would make it better suited for dealing with maps. jQuery is used by several large corporations, for instance IBM and WordPress. While it was not the library that was chosen for the main part of this application there is some use of jQuery for its AJAX methods and for handling some interactions between the JavaScript and HTML of the web application. AJAX allows the application to make calls to a URL, in the case of this application it is used to make calls to the API that retrieves data from the database. (19).

2.2.3 Databases

MySQL is an open-source database. It has an extensive functionality for an open-source tool and is a competent alternative to paid database systems. It is used by many large businesses like Facebook, YouTube, and Google (20). MySQL does have the capability to work with geo data such as coordinates (21), but alternatives were found that are better suited for this task.

PostgreSQL is another open-source database. While not as well known or as popular as MySQL it is a very powerful database tool in its own right. Completely free to use, with no limit on database size or items stored in a table of a database. It also can make use of an extension for its databases called PostGIS which allows it to work with geo data (22). PostGIS turns a regular PostgreSQL database into one that has functionality for dealing with geo data. Most notably for this project it can take in and put out GeoJSON data, as well as converting from other data types to GeoJSON (23).

2.2.4 Cloud Computing Services

Amazon Web Services (AWS) is a cloud computing service. What this means is that Amazon offer the ability to set up a computer system with a specific functionality like say a server machine, a system for backup storage, or a system for some extra computing power. This machine will be set up according to the specification of the user, using predefined options, as a virtual machine on Amazon servers.

Essentially a virtual machine is another computer running inside a physical computer, using the technological resources of that physical computer, i.e. the RAM, storage, processor. AWS servers are very powerful machines that allow the construction of multiple smaller, less powerful but still respectable virtual machines on them (24). When you create an AWS system a server machine at one of Amazon’s server sites creates a virtual machine meeting your specifications and with the necessary software installed, like database management software.

The user will be provided with information to use to access this system from their own location (25). In this case AWS is used for hosting the PostgreSQL database. The information for accessing is used by the backend of the application, the part the user does not directly interact with that manages the database,

to set up and make modifications to the database. The front end will use the information to connect to the database and query information from it, for instance finding the burglary rates for 2013, when the user needs it.

DigitalOcean is a similar cloud computing service. It can also be used for hosting and interacting with databases despite not offering explicit database setups like AWS. Boasting a fast deployment, a simple and intuitive API, and expansive storage up to 16TB DigitalOcean is a viable alternative for what is needed in this project. DigitalOcean is used for hosting the Django API that accesses the database. This API forms the bridge between the front end of the application and the back end and allows communication between them which is key to the functionality of the application.

2.2.5 HTTP Servers

The purpose of HTTP Servers is to allow remote access to the web application across the internet through the IP address or preferably through a domain name. In the case of this project a domain name was procured through the GitHub Student Developer Pack, courtesy of Namecheap.

Apache HTTP Server was the first one looked at due to previous experience with it in other projects. Apache was created and is managed by volunteers to provide a HTTP web server to people free of charge. These contributors number in the hundreds, having helped with ideas, implementation, or documentation. Administration over the maintenance and development of the project is handled by the Apache HTTP Project Management Committee (26). It is quite straightforward to use given the thorough documentation of the project and very capable, but in the end was not chosen for this project.

NGINX is a similar free-to-use HTTP Server that is used by several large, noteworthy companies e.g. Netflix, Wordpress, and Dropbox (27). The process of setting it up is quite straight forward and also well documented in many locations, primarily just starting the service and dragging and dropping the website files (html, JavaScript, CSS, images etc.) into the folder NGINX uses. Research into the two showed that use of NGINX is increasing, and the service is well regarded so the choice was made to use it over Apache. The NGINX server is deployed on to another DigitalOcean droplet.

2.2.6 Other Technologies

Django is a web application framework. In much the same way that the JavaScript libraries discussed above take some of the workload off of the developer by allowing the use of pre-written functions this allows the use of pre-made components. These components can be put together to create the basis of a web application with more advanced functionality like user authentication (a login system), an admin screen where administrative users can make changes to the web application, or in the case of this project a REST API.

An API, Application Programming Interface, exposes data and functions that allow systems to transmit data between themselves (28). REST stands for REpresentational State Transfer. It is an architectural style used in web services. The purpose of REST was to define a standard across the web that systems could adhere to, making communication between these systems more straightforward. One key aspect of REST is how the client and server are kept as separate from each other as possible. This is so that if there is a change to the code in one it does not have a direct effect on the other. It may effect the communication being output, such as a modification to the exact message being transferred, but it will not have a direct impact on how the other application performs. In a REST architecture a client can

make requests of a server. This request can be one of four things. A GET request, where the client wishes to receive something, a POST request, where the client wishes to create something on the server, a PUT request, where the client wishes to change something on the server, and a DELETE request, where the client wishes to remove something from the server (29).

3. Design

3.1 Design Methodology

A design methodology is essentially outlining the structure to be followed during the development lifecycle of a project as a basic loop. The goal of a methodology is to end up in a position where the initial idea of the project has been completed, so the foremost step is to define what a completed state is. From this definition this overall problem can be broken down into smaller problems which can be completed one by one, each being another step towards this completed state (30).

Each methodology outlines a smaller cycle that is repeated throughout the overall development generally encapsulating some combination of a research stage, a design stage, an implementation stage, a testing stage, and a feedback stage. Not every methodology will use all of these but a lot of common ones in the field of software development use some mixture of these or similar steps.

Some key requirements of a good design are that it should be as simple as possible without making compromises, make use of only available resources and not interfere with time restraints, meet the requirements of the end goal, be responsive to requirement changes, and should be possible to implement given the information available (31)

3.1.1 AGILE

In an Agile methodology functionality is implemented in what are known as “sprints”. A sprint is a shorter development cycle inside of the overall development cycle. At the beginning of a sprint a developer will decide upon a piece of functionality to be implemented in the sprint (32).

A typical sprint is thirty days long, approximately one month. Due to the varying degree of difficulty in the parts of this project to be implemented the sprints would need to vary in length. The rough designs have been done as part of the prototyping. For some aspects, like the heatmap, this would be enough to go off. Others may require some refinement before commencing coding. At the beginning of any sprint that requires finalising of designs, like the User Interface, a short period of a few days would need to be spent designing what the finished product will look like. Then the coding and actual implementation begins.

The coding should be completed by the allotted deadline date, set out in the early stages of the project. Past this date that functionality should be considered completed and no more development should be taking place on it.

The schedule defined has time allotted for user group testing near the end of the project’s development cycle. This is when outside users like those representing the general public and the Garda will be testing the finished functionality. There will also be smaller scale testing while moving through development. When each new piece of functionality is implemented a series of tests on previous functionalities will be conducted to ensure that the new code has not caused anything previously working to break, known as regression tests.

Agile is better suited to a larger scale development and is largely based off of regular meetings to discuss progress, hold ups, next steps etc. It is intended for use in teams of people as opposed to individuals. While this does not automatically discount it, and it was intended to be used early on in development, an alternative in a similar vein was inevitably chosen and adhered to.

3.1.2 Prototyping

In the prototyping methodology the overall result of the project is built up gradually through the development of smaller prototypes. This facilitates better, more accurate feedback from end users as instead of just having a discussion of the requirements or the functionality needed there can be quick, practical demonstrations of these developed to see and use. After showing the prototype there is another meeting and feedback is given, adjustments are decided upon and new functionality is outlined. Another prototype is then developed with the decided upon revisions and rough drafts of new functionality and another meeting takes place. Each prototype is generally developed in a quicker time, ranging from a few days to a week (33). This is more in line with the schedule of this project with weekly meetings where the project's current status is shown, feedback is taken in, and new functionality for the next meeting is discussed.

Each cycle incorporates the usual phases of design, research, implementation, testing and feedback on a much more condensed scale. Essentially there is only one phase, each week being a phase. The phases continue until there is no new functionality to implement or improve at which point the project is completed.

3.2 Component Design

3.2.1 Database

The construction of the database seemed initially simple but proved problematic quite quickly. The data needed was the crime rates for the country. Available from the CSO website, provided by the Garda in the according to the Garda station they were reported to. This means that to map the data there is also a need for the location data of each Garda station in the country. These two were not available together so a second dataset was used for the location data.

The location data for placing the Garda station markers was procured from the Garda website (34). On a page allowing the user to view all Garda stations is a link to a .kmz file with the location data inside it.

A .kmz file is an archive type, like a .zip file, used to transmit multiple files bound together. This particular archive type is used to transmit .kml files, or Keyhole Markup Language files. This is a file type used by Google for their applications Google Maps and Google Earth (35). Due to the fact that this project is not using Google Maps this file on its own was of no use. However it was able to be converted into a usable format. Through the use of an online tool, mygeodata.cloud (36). Originally the file was transformed into a GeoJSON file as discussed earlier. There were difficulties encountered with this format when bringing the data into the database.

The idea was that the GeoJSON would be easily read in by the Django application. In reality, there were some complications. An error occurred relating to the datatype of the data stored in the GeoJSON. The process that would import the data into the database requires defining fields with the appropriate datatype for the data being imported, e.g. a string (a series of letters, numbers and / or other characters) being imported would go into what is called a "Character Varying" field. This field takes in a group of characters of varying length. When the fields were set up and the data was meant to import the data type seemed to all be of a different, undefined type.

These difficulties can certainly be circumvented but for this part of the project it was more straightforward to use what is known as a Shapefile instead.

A Shapefile is a format for storing geographic information much like GeoJSON. It will store locations, shapes, and attributes associated with the geographic features (37). The Shapefile was read into the database by the Django application and the data was stored.

The difficulty with the database came with how to relate the different datasets. There were a few main options looked at. First was to join the two datasets together and be left with one big database table. This table would consist of a row for each Garda station with a column for name, latitude, longitude, and then a column for 2003 – 2016 for each crime listed of which there are 12. This means there would be 171 columns for each of the 561 rows. This also means that the way the Django API is set up a call to the database would retrieve every bit of this information and it would have to be dealt with on the front-end. There would be benefits to this, namely that there would only have to be a single call to the back-end on initialising the application. However, the drawback would be that the initial load of the application would be slower, potentially a lot slower depending on the connection available.

Second, and the way that was chosen in the end, was to create a separate data for the location data and for each crime. The result is 13 separate tables each with 561 rows. This was the more ordered format and allowed for requests of smaller amounts of data. With this structure the application can take in the crime type the user wants to see and make a request to the API for just that crime type, meaning it only needs to transmit a fraction of the data of the other method, although it will have to transmit it each time the user changes the crime data they wish to see.

	id [PK] serial	v2003 character varying(4)	v2004 character varying(4)	v2005 character varying(4)	v2006 character varying(4)	v2007 character varying(4)	v2008 character varying(4)	v2009 character varying(4)	v2010 character varying(4)
1	1	29	26	34	43	34	27	32	13
2	2	0	0	1	2	5	0	4	1
3	3	9	4	9	14	11	20	30	20
4	4	6	7	4	3	3	2	7	2
5	5	1	4	4	6	2	3	2	6
6	6	167	166	216	158	152	156	166	129
7	7	0	2	9	3	3	0	3	4
8	8	3	5	2	5	1	5	3	2
9	9	32	48	60	52	65	61	50	65
10	10	9	6	15	12	10	12	7	20
11	11	9	8	10	8	7	2	8	3
12	12	4	12	24	11	5	3	13	4
13	13	24	16	35	37	35	36	40	18
14	14	1	2	2	4	10	5	8	5
15	15	118	111	95	120	122	69	74	75
16	16	25	14	24	11	15	17	17	8
17	17	4	3	5	5	3	6	8	31
18	18	90	87	74	81	91	128	133	102
19	19	48	38	41	41	27	33	19	32
20	20	7	14	23	17	4	7	6	25
21	21	15	27	18	19	40	23	20	9
22	22	8	4	6	2	4	1	7	3
23	23	7	17	19	25	28	27	34	35
24	24	3	6	6	4	14	7	5	10
25	25	210	201	220	288	229	188	170	144
26	26	141	102	88	81	74	134	151	128
27	27	14	16	18	30	17	17	14	11
28	28	11	10	20	15	8	19	11	11
29	29	27	13	16	15	25	22	14	26
30	30	60	95	86	95	162	136	123	146
31	31	7	7	10	3	12	8	6	4
32	32	19	14	10	12	22	12	28	21
33	33	61	40	42	34	62	59	57	94
34	34	18	7	9	6	17	7	17	24
35	35	5	2	5	3	1	6	13	8
36	36	36	31	51	49	46	40	63	67

FIGURE 6 CRIME DATA EXAMPLE IN DATABASE

Figure 6 shows the structure of the table for each different crime type, in this instance showing the “Burglary” table, each row being a different station in line with the order of stations in the table with the location data.

3.2.2 Django API

The Django API was constructed after the database as it relied upon the structure of the database. First a model was designed to correspond to each table.

```

s.py x load.py x 0001_initial.py x models.py x
from django.contrib.gis.db import models

# Create your models here.

class GardaStations(models.Model):
    fid = models.CharField(max_length=80)
    name = models.CharField(max_length=80)
    blank = models.CharField(max_length=80)
    Sun = models.CharField(max_length=80)
    Address = models.CharField(max_length=80)
    Longitude = models.CharField(max_length=80)
    Latitude = models.CharField(max_length=80)
    County = models.CharField(max_length=80)
    Phone = models.CharField(max_length=80)
    StationID = models.CharField(max_length=80)
    Station = models.CharField(max_length=80)
    Mon_Fri = models.CharField(max_length=80)
    Sat = models.CharField(max_length=80)

    point = models.PointField()

    def __str__(self):
        return self.name

class Burglary(models.Model):
    v2003 = models.CharField(max_length=4)
    v2004 = models.CharField(max_length=4)
    v2005 = models.CharField(max_length=4)
    v2006 = models.CharField(max_length=4)
    v2007 = models.CharField(max_length=4)
    v2008 = models.CharField(max_length=4)
    v2009 = models.CharField(max_length=4)
    v2010 = models.CharField(max_length=4)
    v2011 = models.CharField(max_length=4)
    v2012 = models.CharField(max_length=4)
    v2013 = models.CharField(max_length=4)
    v2014 = models.CharField(max_length=4)
    v2015 = models.CharField(max_length=4)
    v2016 = models.CharField(max_length=4)
    name = models.CharField(max_length=80)

```

FIGURE 7 EXAMPLE OF DJANGO MODELS

Each model needed an attribute for the fields in the database that would be represented, which as seen above was exclusively CharFields to correspond to varchar fields in the database.

```

s.py x load.py x 0001_initial.py x models.py x admin.py x Dockerfile x serializers.py x
class Meta:
    model = GardaStations
    fields = ('url', 'id', 'fid', 'name', 'blank', 'Sun', 'Address',
              'Longitude', 'Latitude', 'County', 'Phone', 'StationID',
              'Station', 'Mon_Fri', 'Sat')

class BurglarySerializer(serializers.HyperlinkedModelSerializer):
    class Meta:
        model = Burglary
        fields = ('v2003', 'v2004', 'v2005', 'v2006', 'v2007', 'v2008', 'v2009', 'v2010',
                  'v2011', 'v2012', 'v2013', 'v2014', 'v2015', 'v2016', 'name')

class DangActsSerializer(serializers.HyperlinkedModelSerializer):
    class Meta:
        model = DangActs
        fields = ('v2003', 'v2004', 'v2005', 'v2006', 'v2007', 'v2008', 'v2009', 'v2010',
                  'v2011', 'v2012', 'v2013', 'v2014', 'v2015', 'v2016', 'name')

class DrugsSerializer(serializers.HyperlinkedModelSerializer):
    class Meta:
        model = Drugs
        fields = ('v2003', 'v2004', 'v2005', 'v2006', 'v2007', 'v2008', 'v2009', 'v2010',
                  'v2011', 'v2012', 'v2013', 'v2014', 'v2015', 'v2016', 'name')

class FraudSerializer(serializers.HyperlinkedModelSerializer):
    class Meta:
        model = Fraud
        fields = ('v2003', 'v2004', 'v2005', 'v2006', 'v2007', 'v2008', 'v2009', 'v2010',
                  'v2011', 'v2012', 'v2013', 'v2014', 'v2015', 'v2016', 'name')

```

FIGURE 8 EXAMPLE OF DJANGO SERIALIZERS

The models also needed a serializer for each that allows the data to be converted into Python datatypes and then changed into common data formats (38), in this instance JSON. The JSON is what is then sent to the front-end on retrieval from the database.

3.2.3 Web Application

The front-end web application was the last part of the project designed, but also the part that took the longest as it is the part that the end users will be seeing and interacting with. Initial designs were

adhered to up to a point but midway through development some suggestions were made on how to re-tool it.

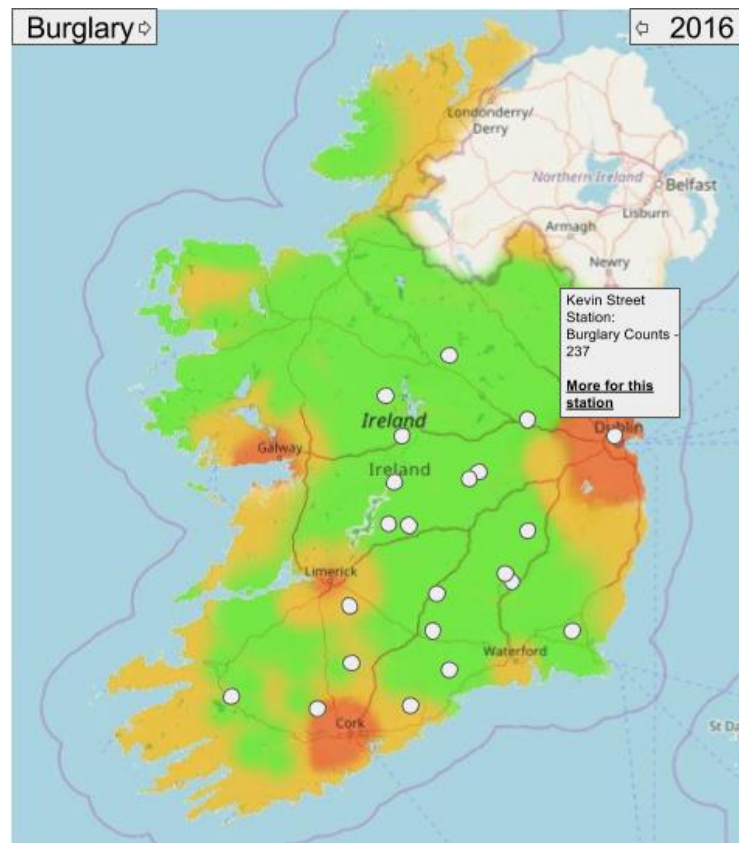


FIGURE 9 INITIAL DESIGN FOR FRONT-END MAIN PAGE

At first it was planned to have the application load straight in to the map with a dropdown in each top corner allowing users to select a crime and a year, to then make a request to the API and generate a heatmap with the data received from the API.

The plan in this design was for graphing functionality to be contained on another page, which would be accessed via the “More for this station” link seen in Figure 9. This would bring the user to a screen like the image below.

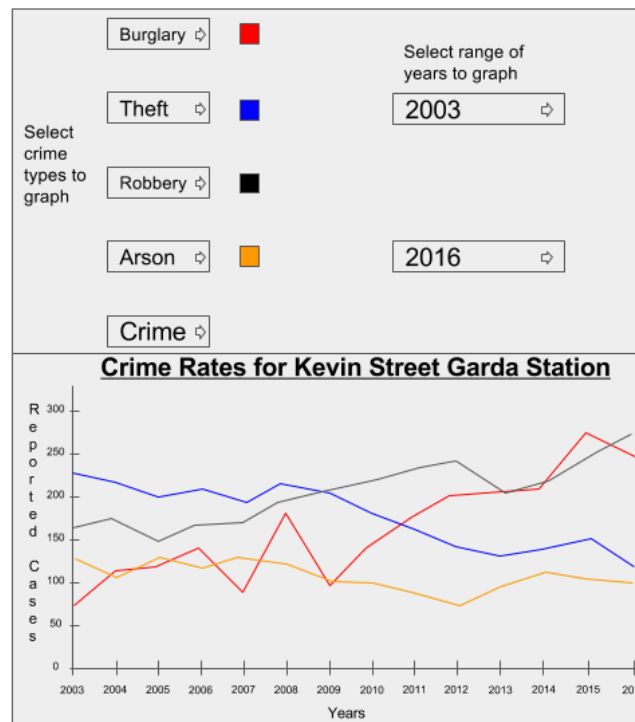


FIGURE 10 INITIAL DESIGN FOR THE GRAPHING PAGE. NOTE: CRIME RATES ARE FABRICATED FOR ROUGH DRAFT

In this design the user would only be able to graph multiple crimes for the selected station. To change the station the user would need to go back to the map and select a new station, then go to this page again for the new station. This had the benefit of allowing full focus on the desired task but was somewhat restricting and slow.

Eventually a discussion was had in a meeting and the decision was made to put the main functionality all on to one page, to avoid the user having to change back and forth between pages when they want to use the different aspects of the project. The revised design had the initial load of the website bring the users to an intro page with a brief description of why the project was undertaken and then instructions on how to use it. On the next page is the actual application divided up into a section for dropdowns and buttons, the map, a table with the data, and the graphs of selected Garda stations.

3.2.4 Source Code Layout

Due to the project being a web application most of the code takes the form of JS, CSS, and HTML files. These are available on the project GitHub. The backend is also hosted on a separate GitHub to make it clear what code is for what part of the project, front-end or back-end.

Web Application code: <https://github.com/MGCurtis/FinalYearProject>

Django code: <https://github.com/MGCurtis/FYPDjango>

3.3 Features and Use Cases

3.3.1 Feature List

A preliminary step in prototyping and deciding the features before development was the writing of a user scenario. Alongside this user scenario, images were designed to give an idea of the rough look of the project when it is in a useable state. The main examples of these images are viewable above in Figure 9 for the main map screen and Figure 10 for the graphing screen.

A user scenario is a story detailing a made-up user interacting with the system. From this story the developers can extract the necessary functionality and a design can start to take shape through the imagining of this scenario. User scenarios can get much more detailed than this example, and there can be multiple scenarios created for one system so as to further build out the functionality. User scenarios can also help developers understand the key areas to focus on when reaching the testing stage, and what areas will have the most variables at play and thus require more varied testing (39).

Note: In the user scenario parts that detail application functionality are highlighted in red.

John loads up the web application and sees a **map of Ireland** with **markers indicating different Garda stations**. John is thinking about moving to a new area, so he wants to check the recent burglary rates in that area to see if it is something he would need to be concerned about. To do this John clicks the **dropdown labelled crime**. He selects the burglary option, then clicks the **dropdown for year** to select the year of data he wants to see. John wants to see the most recent burglary rates so he selects 2016. The map then is **overlaid with a heatmap** showing the rates of burglary across Ireland in 2016. John **selects a Garda station** of the area he is thinking of moving to (for this example Kevin Street) and is shown the actual value and a **“More” option**. John selects the **“More” option** and the **screen changes** to one with multiple options and a blank space below. John wants to view the trends in Burglary, Theft, Robbery, and Arson in the area. John selects these **crimes** and the **full range of years**, starting 2003 and ending 2016. The amount of reports of these crimes over the years is **mapped out on a graph** in the blank space, allowing John to look for which are on the rise and which are decreasing in this area.

Secondary Functionality

John can select to **include census data** so that population counts are taken into account for the heat map, to avoid high density population areas having skewed results.

John can **export the graph** that has been generated to refer to later, share with people, or use somewhere else.

This list of features underwent slight revisions throughout development, but the user scenario provided a solid framework for commencing work on the project.

3.3.2 Use Cases

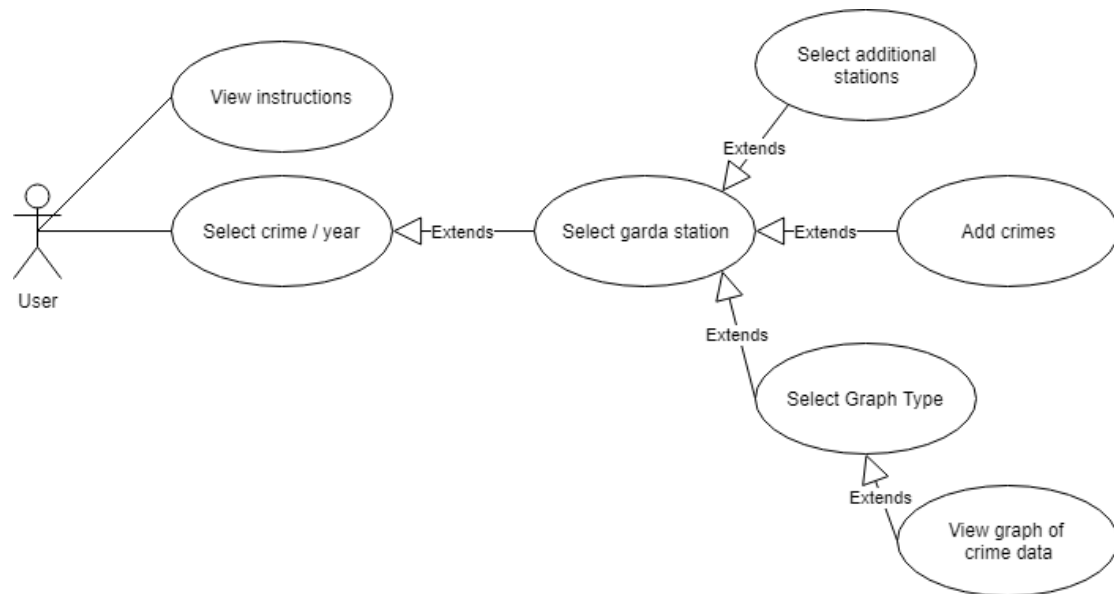


FIGURE 11 USE CASE DIAGRAM

The use case diagram above in Figure 11 illustrates the interactions the user can have with the application along with the workflow of these interactions. This use case was designed far later in the development cycle than the user scenario hence the somewhat differing functionality.

4. Architecture & Development

4.1 System Architecture

4.1.1 Architecture Diagram

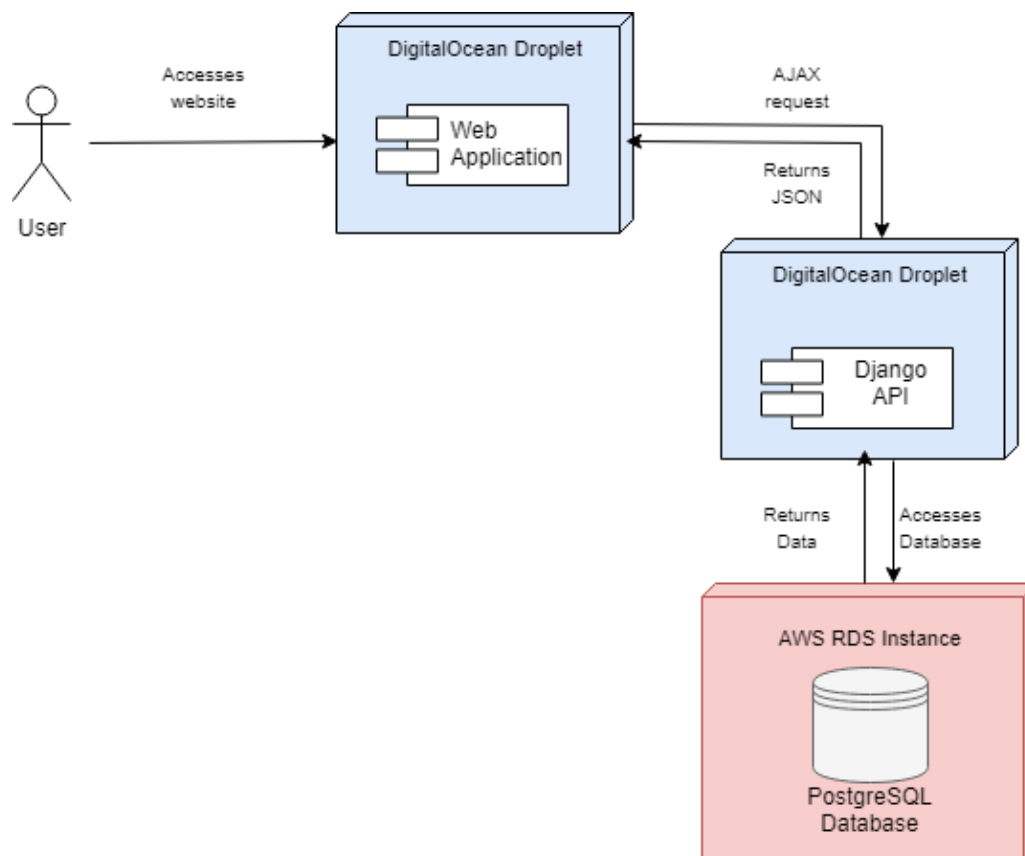


FIGURE 12 ARCHITECTURE DIAGRAM

Figure 12 illustrates the high-level architecture of the system. The system was designed starting with the bottom of this diagram, i.e. database first, then Django API, then the front-end website. The database is hosted on a deployed AWS database. This avoids having to couple a SQLite database or alternative with the application when deploying, keeping the size of the project down. The database is accessed via a deployed Django API, hosted on a DigitalOcean droplet. The front-end application makes use of this API by way of simple GET requests through the AJAX functionality of jQuery. Once a query is made the Django API returns the requested data to the front-end in JSON format which is easily used by the application for its various needs.

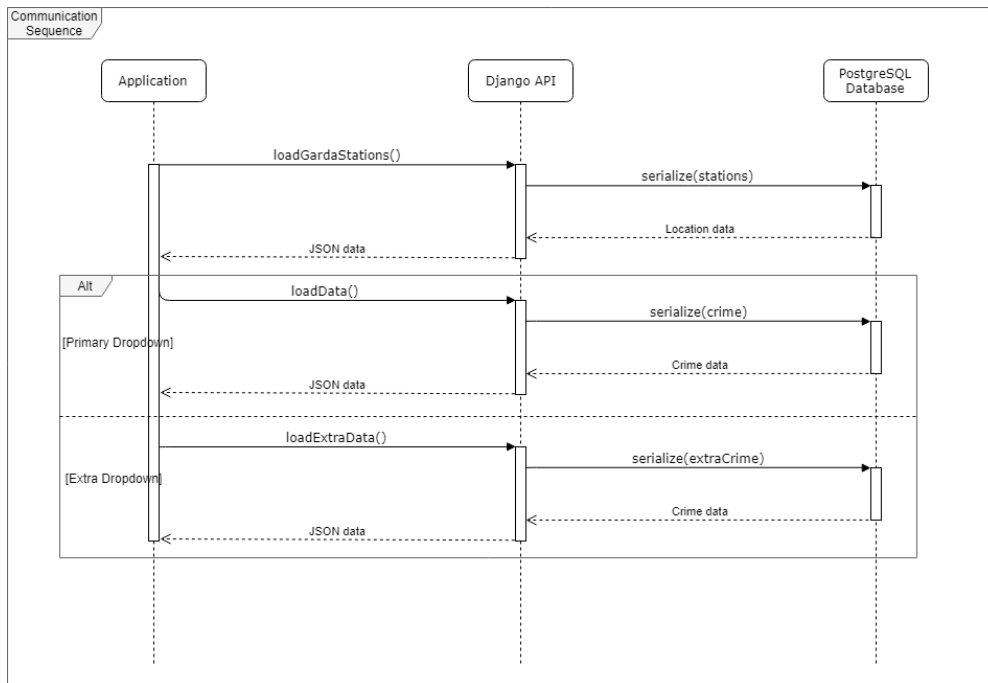


FIGURE 13 SEQUENCE DIAGRAM

The above sequence diagram is to better illustrate the functions that communicate with the backend. On an initial load the `loadGardaStations` function fires and retrieves the data on the Garda stations like the location and names from the database through an AJAX GET request. Then the user will select a crime and year from the primary dropdowns and the `loadData` function will fire sending another AJAX request, this time to the API endpoint for the selected crime. Then if the user selects a crime in the extra dropdowns, which will be elaborated upon in the section on the web application, a similar function fires called `loadExtraData` which requests the data for the selected extra crime. The process of getting the selected year is handled in the front-end.

4.2 Project Components

4.2.1 Database Instance

The AWS Instance required the least modification after launching it, as there is a pre-defined specification for a PostgreSQL RDS instance on AWS. Once it had been created the database connection information was added to the Django project and through the models and serializers the data was imported from the procured files into the Postgres database.

The crime data needed slightly more work than the location data as the Shapefile could be directly mapped and imported, due to the specifically defined fields for the data. The crime data being in JSON format meant that there was no explicit definition of the attributes and the different objects / items could potentially have different attributes. The OGR library included in GDAL, which is used to import the Shapefile, only works for specific spatial data formats (like Shapefiles) so that would not work for JSON anyway. The method used to get the data in was a simple for loop in Python to add the data object by object.

4.2.2 Data

The datasets came with their own set of problems. The first one, the location data, was great on its own. It was added to the Postgres database and used for an early vertical prototype.

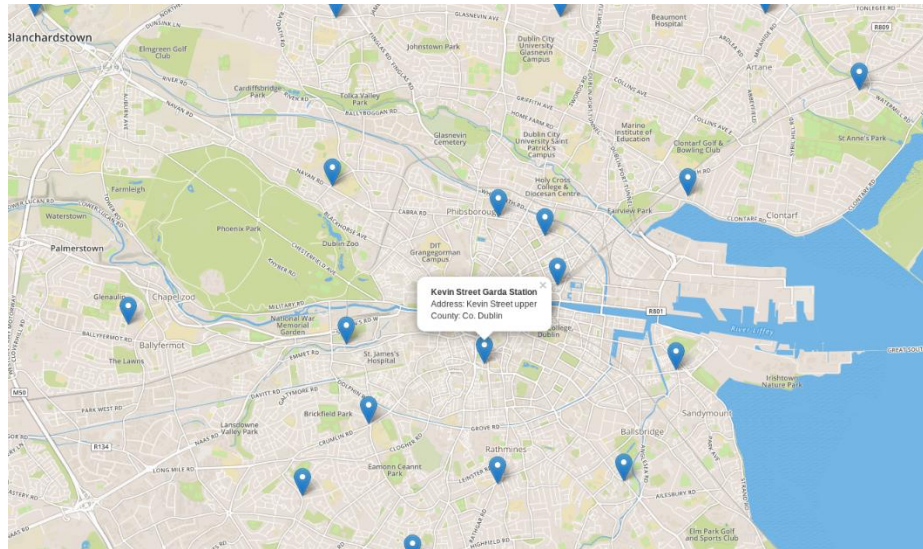


FIGURE 14 SCREENSHOT OF VERTICAL PROTOTYPE

The purpose of this prototype was to load a Leaflet map, get the location data from the Postgres database via the Django API to show the system would all communicate work together, and then use this data to place a marker at each Garda station in the country. A subsection of this can be seen above in Figure 14. Clicking a marker would open a popup with the name, address, and county of the Garda station. In the finished project this popup contains the name of the station and the crime rates for that station.

After this the crime data needed to be added. This is where problems began to arise. The most straightforward way of making use of the two different datasets was to call both in separate arrays so that the location data only needs to be called once and then the crime data can be called as it is needed. This gives two arrays that ideally are in the same order with the same number of entries allowing the use of simple for loops to associate the crime values with the locations. At first glance this seemed like it would be simple as both datasets were in alphabetical order. However, in the crime datasets some of the station names were in Irish, meaning that the alphabetical sorting placed them in a different order than the all-English location dataset.

569	04 ,Dangerous or negligent acts								
570	"Abbeyfeale, Limerick Division"	14,12,19,26,16,16,14,20,6,6,7,9,9,6							
571	"Abbeyleix, Laois/Offaly Division"	23,24,35,38,32,37,22,23,5,6,12,11,9,7							
572	"Adare, Limerick Division"	6,11,14,16,14,12,12,6,5,6,5,4,8,7							
573	"Aglish, Waterford Division"	5,1,0,0,3,3,1,0,2,2,6,4,1,1							
574	"Ahascragh, Galway Division"	0,4,7,6,3,7,4,3,2,5,3,3,1,5							
575	"Ailt an Chorraíni, Donegal Division"	3,3,1,2,12,8,5,6,2,3,1,2,1,3							
576	"An Bun Beag, Donegal Division"	16,24,15,25,25,21,24,13,15,16,5,5,5,7							
577	"An Charraig, Donegal Division"	5,0,3,8,6,8,7,4,5,3,1,1,0,0							
578	"An Cheathru Rua, Galway Division"	4,2,5,5,13,17,12,16,4,10,12,3,8,4							
579	"An Clochain Liath, Donegal Division"	17,9,9,14,21,29,15,6,6,6,5,7,4,5							
580	"An Coirean, Kerry Division"	7,10,9,13,3,4,11,6,5,3,2,2,4,2							

FIGURE 15 CRIME DATA BEFORE REARRANGING

In the above example there is a station “Ailt an Chorraíin” which is the 6th station in the list, in the location data this station is Burtonport so is significantly further down the list in the 118th spot. To solve this the crime data csv file was opened in Microsoft Excel and was rearranged using a Visual Basic function. The function was hastily written and would take in a station name and the number of spaces to move it, then go down through the whole document and move each station by that name. This meant

it was a somewhat manual process, but still took a fraction of the time it would take to do it all by hand as each station appears 13 times, once for each type of crime.

566	04, Dangerous or negligent acts								
567	Abbeyleix, Laois/Offaly Division,	23,24,35,38,32,37,22,23,5,6,12,11,9,7							
568	Gob an Choire, Mayo Division,	6,4,5,8,8,2,2,0,4,1,0,1,1							
569	Adare, Limerick Division,	6,11,14,16,14,12,12,6,5,6,5,4,8,7							
570	Aglish, Waterford Division,	5,1,0,0,3,3,1,0,2,2,6,4,1							
571	Ahascragh, Galway Division,	0,4,7,6,3,7,4,3,2,5,3,3,1,5							
572	Anglesea Street, Cork City Division,	130,126,175,209,176,142,114,107,96,105,50,32,46,53							
573	Annascaul, Kerry Division,	2,1,1,3,1,2,0,2,1,0,1,0,1,0							
574	Ard an Raitha, Donegal Division,	6,3,8,11,12,10,7,5,3,3,1,1,12							
575	Ardee, Louth Division,	22,25,13,15,16,32,29,23,18,13,11,18,13,20							
576	Ardfert, Kerry Division,	14,16,9,12,15,14,8,6,9,4,8,3,10,6							
577	Ardfinnan, Tipperary Division,	4,9,7,2,4,11,16,2,1,2,1,5,3,1							

FIGURE 16 CRIME DATA AFTER REARRANGING

Shown above in the same order as the location data, as if all of the stations were in English. The next step was to divide the csv up into separate JSON files with one file containing the data for one crime type. This was accomplished using array operations in JavaScript. An array was created containing the data for one type of crime, take for instance Burglary. This array was separated out by comma with each object ending at a newline and each item of an object ending at a comma. The result was JSON with an object for each station, and each object with a series of key-value pairs where the key was the year or the word 'name' and the value was the number of the given crime for that year or the name of the station. This process was repeated until there was a JSON file for each crime.

The process of creating models and serializers was then repeated for each type of crime, as it was done for the location data. Then the JSON files were inserted into the Postgres database by a simple Python for loop in which a SQL insert statement is created using the values of entry “I” in the array, where “I” is the current iteration of the loop.

4.2.3 Django Droplet

The Django droplet required some slight set up. This was namely achieved via Docker. Docker is a tool for the containerization of software systems. Essentially it allows the developer to construct an image in which the developer can specify the necessary libraries and other dependencies of a project, allowing more control over its environment cutting down on unforeseen errors and bugs (40). This image is constructed from a Dockerfile.


```

FROM python:3.5

RUN apt-get -y update
RUN apt-get -y upgrade

RUN apt-get -y install libgdal-dev

RUN mkdir -p /usr/src/app

COPY requirements.txt /usr/src/app/
COPY . /usr/src/app

WORKDIR /usr/src/app

RUN pip install --upgrade pip
RUN pip install --no-cache-dir -r requirements.txt

EXPOSE 8000
CMD ["python", "manage.py", "runserver", "0.0.0.0:8000"]

```

FIGURE 17 EXAMPLE DOCKERFILE FOR DJANGO IMAGE

Figure 17 shows the Dockerfile for the Django part of the project. Through the command “docker build” Docker will construct a container with reference to this Dockerfile where the dependencies of the project are specified., for instance here it ensures “libgdal-dev” is installed in the container, this is a library needed for working with spatial data. It also installs what is listed in the requirements.txt file that is provided alongside it. Then it exposes the specified port and runs the command “runserver” on port 8000 which starts the Django application. Docker images are relatively small in size to download, and relatively quick to set up. Since they make certain that applications will execute in the expected manner they are a worthy consideration when it comes to deployment.

This Docker image was deployed via Docker Hub, similar to GitHub but for Docker images. The image was built on the development machine where it was tested, then pushed up to docker hub. Once there the droplet machine was accessed remotely via SSH, docker was installed, and the image was pulled down from Docker Hub and run. The result can be viewed at 139.59.162.120.

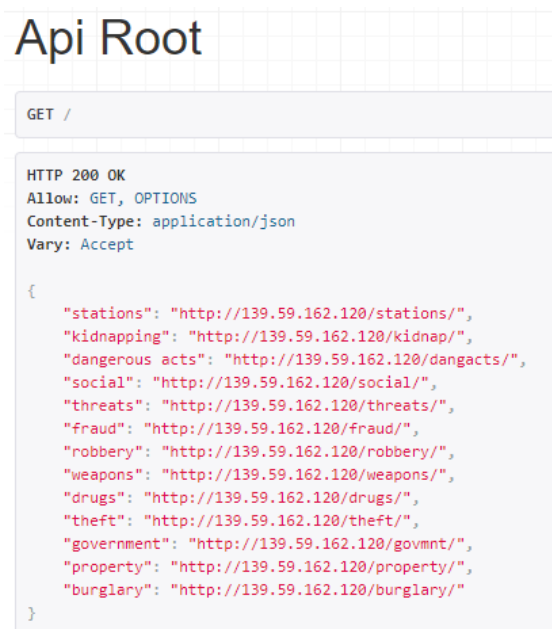


FIGURE 18 DJANGO API

When accessed the user will see the page shown in Figure 18. Going into any of the links listed, e.g. 139.59.162.120/kidnap will display a JSON formatted list of the associated crime data.



```
GET /kidnap/

HTTP 200 OK
Allow: GET, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

[
  {
    "v2003": "0",
    "v2004": "1",
    "v2005": "0",
    "v2006": "0",
    "v2007": "0",
    "v2008": "2",
    "v2009": "0",
    "v2010": "0",
    "v2011": "0",
    "v2012": "0",
    "v2013": "0",
    "v2014": "0",
    "v2015": "0",
    "v2016": "0",
    "name": "Abbeyleix, Laois/Offaly Division"
  },
  {
    "v2003": "0",
    "v2004": "0",
    "v2005": "0",
    "v2006": "0",
    "v2007": "0",
    "v2008": "0",
    "v2009": "0",
    "v2010": "0"
  }
]
```

FIGURE 19 KIDNAPPING DATA DISPLAYED VIA DJANGO

A field for each year and the station name, the v in front of the year is due to Python not allowing variable names starting with a number.

4.2.4 Website Droplet

The website droplet needed some setup as well. First the NGINX software needed to be installed. The firewall needed to be modified to allow NGINX to make connections and expose the ports. Then it was a simple matter of putting the website files (the HTML, CSS, JS, plugins, and images) into the HTML folder where the default NGINX HTML file is located. At this point the web application was hosted and could be accessed entering the IP address into the address bar of a browser.

The next step was connecting the domain name to the IP address. On the website of the domain provider (Namecheap) there is a page to enter in the nameserver of the hosting service. This indicates what service to try and resolve the request with. Then on the DigitalOcean website there is a section to enter the domain name is entered and told which IP address to correspond to.

DNS records

Type	Hostname	Value	TTL (seconds)	
CNAME	www.crimeviz.me	is an alias of crimeviz.me.	43200	More ▾
A	crimeviz.me	directs to 178.62.22.178	3600	More ▾
NS	crimeviz.me	directs to ns1.digitalocean.com.	1800	More ▾
NS	crimeviz.me	directs to ns2.digitalocean.com.	1800	More ▾
NS	crimeviz.me	directs to ns3.digitalocean.com.	1800	More ▾

FIGURE 20 DIGITALOCEAN NETWORKING PAGE

The CNAME type tells the system to treat “www.crimeviz.me” the same as “crimeviz.me”. The A type states to direct requests for crimeviz.me to the IP address indicated. The 3 NS types say that crimeviz.me will direct requests to the 3 different nameservers of DigitalOcean.

At this point the website can be accessed either by web address or IP address.

4.2.5 Web Application

The development of the front-end web application began with the vertical prototype described earlier which would call the location data from the database through the Django API and place a marker at each station on the Leaflet map. This formed the base for the rest of the development. After the reformatting of the crime dataset the next step on the front-end was to call this crime data from the API as well. Once it was being called a for loop was used to add the value for the selected crime in the selected year to the array of station locations and names.

```
for(var i = 0; i < stats.length; i++)
{
  stats[i].value = vals[i];
  if(vals[i] > maxVal){
    maxVal = vals[i];
  }
}

for(var i = 0; i < stats.length; i++)
{
  markers[i].bindPopup("<b>" + stats[i].name + " Garda Station</b>"
    + "<br>County: " + stats[i].county + "<br>Reported Cases: " + stats[i].value);
}
```

FIGURE 21 CODE FOR HANDLING CRIME VALUES IN LOADDATA()

The above code makes a field with the key ‘value’ and the value equal to the number in the ‘vals’ array corresponding to the station. This ‘vals’ array is the result of taking in the value of every year of the selected crime for each station from the AJAX call to the API, then making the ‘vals’ array where each index is the amount of crime in the selected year at the station. This is why the two datasets had to be in the same order, because the station at index 0 gets the value at index 0 and on until the station at index 560. The ‘maxVal’ variable is the highest occurring value in the ‘vals’ array and is recorded so the map knows where to mark as being red, or having the highest amount of crimes, and how to colour the rest of the country based on its proximity to that maximum value. The second bunch of code binds a

popup to the marker at each station with the name, county, and the number of reported cases at that station. This popup is displayed when the station is selected, by clicking the map so that this is the nearest station to the clicked location.

FIGURE 22 UI FOR SELECTING CRIME AND YEAR FOR HEATMAP

These dropdown menus are in the top left of the application and allow the user to select the crime and year to generate the heatmap for. Once there is a value in each dropdown the AJAX call is made to the API for the selected crime and the loop in Figure 21 extracts the values for the selected year. The Clear button removes the heatmap and resets the dropdowns. The checkbox allows alters the heatmap so instead of using the 'maxVal' variable from Figure 21 it will instead use a different variable that is created by looping through the entire crime array to get the most amount of the selected crime in any year at any station. This allows the heatmap to display the state of the country as a whole for the selected crime in relation to the highest the selected crime rates have been.

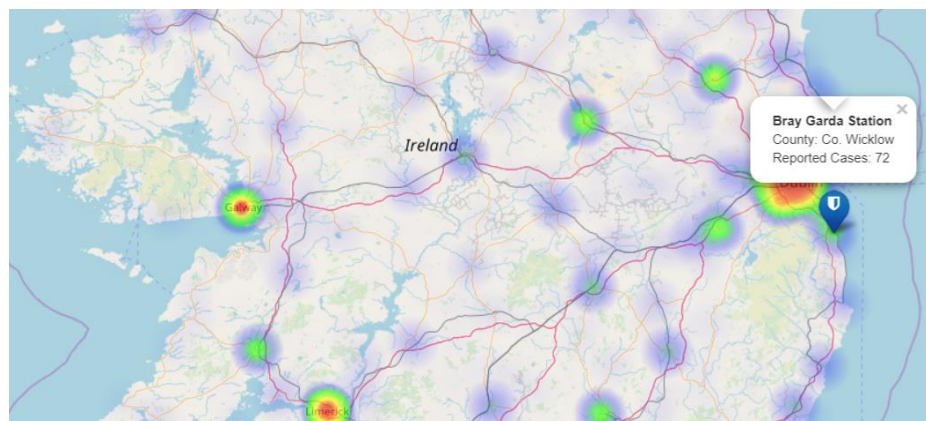


FIGURE 23 MAP AFTER SELECTING A GARDA STATION

Once the heatmap has been generated clicking a location on the map will add the marker and open up the popup of the nearest Garda station.

```
function findMyNearestGardaStation(e) {
    var closestDist = Number.MAX_SAFE_INTEGER;
    var currentDist;
    for(var i = 0; i < markers.length; i++)
    {
        currentDist = map.distance(e.latlng, markers[i].getLatLng());
        if (currentDist < closestDist){
            closestIndex = i;
            closestDist = currentDist;
        }
    }
    var dTable = document.getElementById("dTable").getElementsByTagName("tbody")[0];
    dTable.rows[closestIndex].dispatchEvent(new Event("mousedown"));
}
```

FIGURE 24 CODE TO FIND THE NEAREST STATION

This is the function that fires on the user clicking the map. It runs through the list of markers and finds the closest one to the clicked location. Once finished it adds that marker to the map and opens it. The actual code for adding and opening is contained in the function that fires on clicking an entry in the table (which will be shown further later). This is because clicking a station on the map and clicking a station in the table should have the same results. Clicking in the map should highlight in the table while clicking in the table should add the marker to the map. Because of this the decision was made to have the main functionality in the table function just call that in the function for clicking the map.

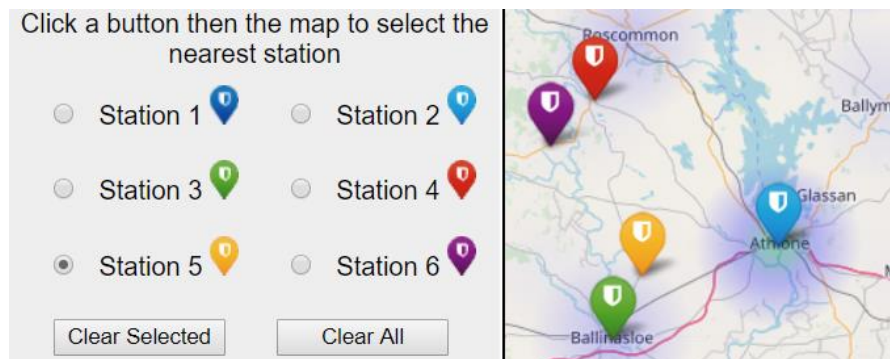


FIGURE 25 THE MENU FOR SELECTING ADDITIONAL STATIONS

The menu shown in Figure 25 allows the user to select up to six different stations. The reason for this is that each station selected is added to a graph down in the bottom right of the application.

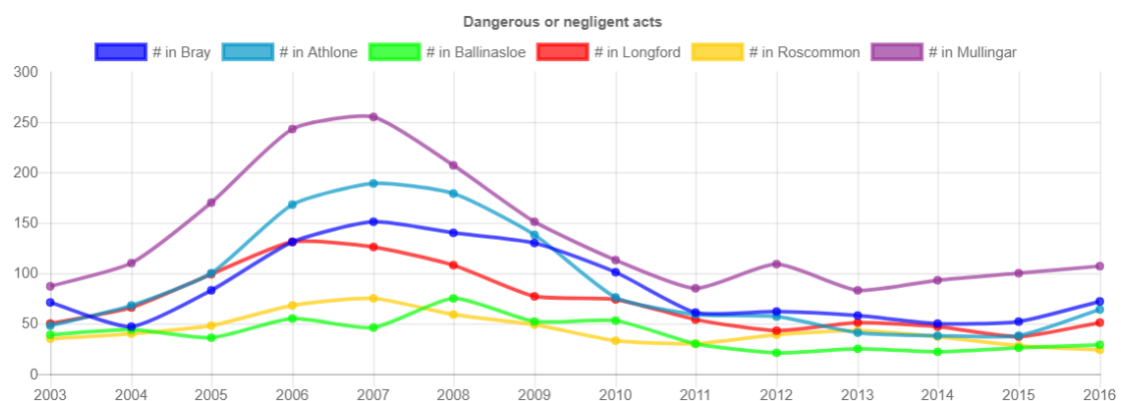


FIGURE 26 THE GRAPH GENERATED FROM SELECTED STATIONS

The example shown is a line graph as it is the default setup of the application, but there is the option to change the graph to a point graph, bar graph, single bar graph with just the first selected station, or pie graph with just the first selected station. The colour of the station in the graph was chosen so each would be distinct from the others. The markers were later reworked to match the colour of the station in the graph. At first all the markers were the blue of the first marker but during testing users stated it was not immediately obvious which marker was which and it would be better to make them identifiable.

Select to compare:

☐ Stations
 ☒ Crimes

Select other crimes to overlay on to map and compare in graphs

Select Crime ▼	Select Year ▼
Select Crime ▼	Select Year ▼
Select Crime ▼	Select Year ▼
Select Crime ▼	Select Year ▼

FIGURE 27 MENU SHOWING ABILITY TO COMPARE MULTIPLE CRIMES

The buttons at the top of the screenshot above change the menu with the six stations into the menu in the image. These additional dropdowns are to implement the functionality for comparing multiple crimes at a single station. When the additional crime and year are selected another heatmap is generated and overlaid on top of the main heatmap. These crimes are added to the graph in the same manner as the additional stations, where each colour represents a different crime and the title of the graph is the selected station. The Clear All button will remove all of the additional crime heatmaps and reset the dropdowns but leave the main one. The Hide Heatmaps button will remove the additional heatmaps from the map but keep them in the graph. This is because the map can get somewhat cluttered looking with so many heatmaps overlaid on top of one another, so it keeps the heatmap readable while letting the user compare the different crimes in the graph.

Station Name	County	Reported Cases
Abbeyleix	Co. Laois	7
Achill Sound	Co. Mayo	1
Adare	Co. Limerick	7
Aglish	Co. Waterford	1
Ahascragh	Co. Galway	5
Anglesea Street	Co. Cork	53
Annascaul	Co. Kerry	0
Ardara	Co. Donegal	12
Ardee	Co. Louth	20
Ardfert	Co. Kerry	6
Ardfinnan	Co. Tipperary	1
Ardmore	Co. Waterford	4

FIGURE 28 DATA TABLE

In Figure 28 the last element of the main application's UI can be seen. This is the table of the selected crime rates for the selected year. This lets the user examine exact rates in stations and the rows of the table are selectable so if a user knew the name of a station but not its location on the map they can find it in the table and select it there. As mentioned earlier when a station is selected in the map the table is selected in the table as well and vice versa. Making the table selectable was accomplished with reference to a useful post on sweetcode.io about selectable HTML tables (41) .

```
function populateTable() {
  var dTable = document.getElementById("dTable").getElementsByTagName("tbody")[0];
  if(dTable.rows.length <= 1){
    for(var i=0; i<stats.length; i++){
      var row = dTable.insertRow(dTable.rows.length);

      row.onmouseover=function(){
        if ( this.className === '' ) {
          this.className='highlight';
        }
        return false
      }
      row.onmouseout=function(){
        if ( this.className === 'highlight' ) {
          this.className='';
        }
        return false
      }
    ]
    row.onmousedown=function(){
      if ( selectedIndex == null ) {
        if ( this.className !== 'clicked' ) {
          if ( selected !== null ) {
            selected.className='';
            markers[selected.rowIndex - 1].remove();
          }
          this.className='clicked';
          selected = this;
          selectNo = selected.rowIndex - 1;
          openStation = markers[selectNo];
          openStation.options.icon.options.markerColor = markerCol;
          console.log(openStation.options.icon.options);
          openStation.addTo(map).openPopup();
          map.panTo(openStation.getLatLng());
        }
      }
    }
  }
}
```

FIGURE 29 FUNCTION FOR POPULATING TABLE AND MAKING IT SELECTABLE

In the code it references the HTML table, changes the class of a row to 'highlight' on mouseover and changes back when the mouse leaves the row, and changes the class to 'clicked' on click. Clicking a row also gets the corresponding marker and adds it to the map, which is the functionality that is also used by clicking the map. It was suggested during user tests that the markers be coloured corresponding to the colours of the stations in the graph, so first is deep blue, second is light blue, third is green etc. Another request from user testing was to centre the map on the newly opened marker, which is what is happening in the last line of the above code.

```

function makeChart(){
    var ctx = document.getElementById("myChart").getContext('2d');
    if(myChart !== null){
        myChart.destroy();
    }
    switch (document.getElementById("chartDd").value) {
        case "pie":
            makePie(ctx);
            break;
        case "line":
            if($("#input[name=compSel]:checked").val() == "crimes") {
                makeLineCrimes(ctx);
            }
            else{
                makeLineStations(ctx);
            }
            break;
        case "bar":
            if($("#input[name=compSel]:checked").val() == "crimes") {
                makeBarCrimes(ctx);
            }
            else{
                makeBarStations(ctx);
            }
            break;
        case "barSing":
            makeBarSing(ctx);
            break;
    }
}

```

FIGURE 30 MAKECHART FUNCTION

Inside of the populateTable function there is a call to the makeChart function which will destroy the current chart if one exists, checks the chart type selected in the menu, and with a switch statement will make the appropriate type of chart any time a new station is selected. The makeChart function then in turn calls the relevant function. The 'ctx' variable is the reference to the HTML canvas for the chart.

```

function makeBarStations(ctx) {
    myChart = new Chart(ctx, {
        type: "bar",
        data: {
            labels: ["2003", "2004", "2005", "2006", "2007", "2008", "2009",
                "2010", "2011", "2012", "2013", "2014", "2015", "2016"],
            datasets: [{
                label: '# in ' + statName[0],
                fill: false,
                borderColor: cols[0],
                backgroundColor: cols[0],
                data: selectData
            },
            {
                label: '# in ' + statName[1],
                fill: false,
                borderColor: cols[1],
                backgroundColor: cols[1],
                data: extraSelectedStat[0]
            },
            {
                label: '# in ' + statName[2],

```

FIGURE 31 EXAMPLE OF CHART CODE

Figure 31 shows an example of the chart construction code. If the user has selected to compare stations instead of crimes and has the bar chart type selected this is the function that will be called from makeChart. There are two for the bar, point, and line charts each having a station and crime variant depending on which of the radio buttons in Figure 27 the user has pressed. The pie and single bar charts only display the data of the primary crime selected. In the station variant each colour on the graph is labelled with the name of the selected station and the colours are defined in an array called 'cols', the graph is titled with the selected crime. In the crime variant each colour is instead labelled with the crime it represents and the graph is titled with the station selected.

One issue that took some getting accustomed to and required a workaround at one point was the nature of how JavaScript does not wait for requests to be resolved before continuing to execute the rest of the code. The main point where this became an issue was in the generation of the heatmap. The code was attempting to set the data of the heatmap before the API had returned the crime data, so was throwing an error. This was only occurring when the heatmap generation was confined to its own function, but what solved it in this case was moving the code for creating the heatmap to the bottom of the loadData function. In this position it would only trigger once the data had been received.

4.3 External APIs and Libraries

4.3.1 OpenStreetMap

The base layer of the map, that being the actual map tiles are from OpenStreetMap (13) .

```
function makeBasicMap() {
  console.log("In makeBasicMap.");
  //Initialize map
  // set up the map latitude and longitude bounds to stop user from scrolling away from Ireland
  bounds = new L.LatLngBounds(new L.LatLng(51, -11.3), new L.LatLng(55.7, -5));

  map = new L.Map('map', {zoomControl:false, maxBounds: bounds});

  // Url for getting map, attribution for map tiles, and tile layer variable
  var mapUrl='http://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png';
  var mapAttrib='Map data © <a href="http://openstreetmap.org">OpenStreetMap</a> contributors';
  var mapLayer = new L.TileLayer(mapUrl, {minZoom: 6, maxZoom: 16, attribution: mapAttrib});

  // Set initial map view to roughly middle of Ireland and add the tile layer
  map.setView(new L.LatLng(53.33743, -7), 7);
  map.addLayer(mapLayer);

  // Add heading to the bottom left of the map
  L.control.heading({ position: 'bottomleft' }).addTo(map);
  // Add zoom control to top right of the map
  L.control.zoom({
    position: 'topright'
  }).addTo(map);
}
```

FIGURE 32 CODE FOR INITIALISING MAP WITH MAP TILES

The above code shows the process of creating the map object and getting the map tiles. The first part is to establish bounds, the latitude and longitude that the map will not let the user scroll past. This is a small range because the application is only focussed on Ireland, so scrolling to anywhere else would serve no purpose. Then the actual map object is created with Leaflet and the bounds are set. The map URL is pointed at OpenStreetMap (OSM) with the URL incorporating the zoom level, {z}, and the x and

y coordinates, {x} and {y}. This tells the map what tiles to pull from OSM and at what level of detail, more detail being needed at higher zoom levels. Then is the attribution, stating that the map is provided by OSM, and the instantiation of TileLayer to be added to the map. The view is set as well, roughly at the centre of Ireland with a zoom level showing most of the island. A heading is set in the bottom left which is used to display the crime and year being shown in the heatmap, and the zoom controls are in the top right.

4.3.2 JavaScript Libraries

This is a section that is discussed in greater detail earlier in section 2.2.2. This will just provide a quick look at the libraries in use and where they are used in the code.

First is Leaflet, the mapping library. Leaflet is used in the instantiation of the map in the section just above this. It is referenced using the notation “L.”, this lets the system know that what follows is Leaflet code.

```
for(var i = 0; i < stats.length; i++)
{
    var marker = L.marker([stats[i].lat, stats[i].long], {icon: gsIcon});
    markers.push(marker);
}
```

FIGURE 33 CODE FOR CREATING MARKERS FOR STATIONS

Here Leaflet is called to create a marker for each station in the received location data. These markers are created with a latitude, longitude, and an icon that is created using Leaflet.ExtraMarkers (42) and Font Awesome (43). These are both relatively self-explanatory and are only used here in the project so there is not much need for their own sections. Suffice to say Leaflet.ExtraMarkers provides additional marker templates for use on Leaflet maps and Font Awesome provides additional icons to be used in said markers. Leaflet is also used when adding the heatmap layer to the map itself through a simple call of “map.addLayer(heatmapLayer);”.

Second is jQuery, used for AJAX calls and interaction with the HTML document. The way to declare that jQuery is about to be used is ‘\$.’.

```
$.ajax({
    dataType: "json",
    url: "http://139.59.162.120/" + crime + "/",
    mimeType: "application/json",
    success: function(values){
```

FIGURE 34 EXAMPLE OF AJAX CODE

In this example jQuery’s AJAX functionality is used to make a call the selected crime endpoint of the Django API, it is told to expect datatype JSON. mimeType is similar to datatype and when not set everything works correctly but the console shows a warning that it is expecting XML and received something else. The ‘success’ line opens a function that will execute on a successful GET request to the url. Other AJAX calls are used for the extra crime data and for the location data.

```
$("#input[name=max]").prop('disabled', false);
```

FIGURE 35 EXAMPLE OF USING JQUERY TO CHANGE SOMETHING IN THE HTML

Here the jQuery gets the HTML element of type input with the name of max. Then it sets the property of disabled to false. What this does is enables the checkbox in the menu below the dropdowns. This is disabled on load because on click it attempts to regenerate the heatmap, and if there is no heatmap

there then it will throw an error. So, until a crime and year are selected and a heatmap is generated for the first time the checkbox is disabled. This ability is used in many other areas in the same manner, such as to reset the dropdowns on clicking the Clear button, or to check which Station button is selected.

The Heatmap-js library (44) is used for the actual heatmap generation . It takes in an array of location data (latitude and longitude) and values to mark at that point on the map. The maximum value needs to be included in this array, which is gotten in the same for loop used to get the selected year's data from the received crime data.

Then is the Chart.js library (45). This library is used for creating the charts of selected stations and crimes. See Figure 31 for a screenshot example of the code for creating a chart with this. Essentially you declare the type of chart (bar, pie, line), label for the y-axis, the different datasets being graphed, the colours for these datasets, labels for the colours (station or crime names), and optionally a title for the graph. There are more options for customisation, but these are the mains ones made use of in this project. Chart.js is not mentioned in the research section as there was no initial research done into the library, it was found mid-development and was very straightforward to implement. There are likely other libraries that could be more powerful and customisable but for the purposes of this project Chart.js is more than suitable.

Lastly there is some light usage of Bootstrap in the introduction page with the application instructions on it.

Crime Data Visualization

When starting my project I looked at what data was available on crime statistics in Ireland. From the Central Statistics Office website I found datasets on the crimes rates from 2003 to 2016, according to the Garda stations they were reported to. However there was little in the way of visualization of this data, i.e. graphs, charts, maps etc. I decided to fix this through the creation of this application.

The goal was to create an app that would let users make sense of this mountain of data available. Say if someone were writing a report on fluctuations in the crime rates in the country. Or if someone were curious as to whether crime in their area is on a decline or incline. Or for the Garda themselves to see areas that have been experiencing increases in certain types of crime in recent years

Note: Heatmap will not function with some adblockers enabled

Instructions

Use arrows to change slides

Select Crime

Select Year

Clear

Use max. number of cases 2003 - 2016


The first step is to use the dropdown menus in the top left of the screen to select a crime and a year. This creates the heatmap of crimes of this type in that year across the country.

Controlled drug offences

2011

Clear

Use max. number of cases 2003 - 2016



Number of Controlled drug offences cases for the year 2011

Go to application

Created By Max Curtis 2018

43

Bootstrap is a popular library for design of the front-end (46). Due to its basic nature this design could likely have been done using plain CSS but the use of Bootstrap allowed the inclusion of some premade components, like the slideshow for the instruction images. This has barely scratched the surface of what Bootstrap can be used for and if development of this application were to continue this page would be reworked to be less barebones, and the Bootstrap could be used to give the main application page a more responsive design.

4.3.3 Django

The Django API was designed with reference to the Django REST Framework website (47). The actual design process of the models and serializers is gone into in greater detail in section 3.2.2. The use of the API provides the method of accessing the PostgreSQL database and avoids the need for constructing SQL queries. This avoids the danger of SQL injection without requiring prepared statements.

4.3.4 NGINX

While there was no specific coding to be done for the NGINX system the setup was done with reference to a DigitalOcean tutorial on setting up NGINX on a Ubuntu system (48) . This setup itself just required the installation of NGINX, setting up the firewall to allow access to the server, and then placing the website files into the 'www' folder NGINX uses for its web pages.

4.3.5 Own Code

The code inside of the HTML, CSS, and JavaScript files is all of my own design except where specified inside of the code that there was reference to websites. The only instances of this are with the populateTable function and the 'cfg' variable that is used for generating the heatmaps (44).

5. System Validation

5.1 Testing

Integration and regression tests were performed as each new prototype was completed and brought together with the overall system. These tests are to check the ability of the new components to work with each other and then to ensure that old components that are already a part of the system are not broken or throwing errors due to the new components.

These tests were suitable for finding out if the functionality was working as intended but they served no purpose in finding out how useable the application would be by the end users. For this purpose, the users themselves had to test the application. This was one of the reasons the application was deployed to a server, so that tests could be performed easier. The testing instructions were for the user to select a crime and year to view, select a station they were interested in viewing in detail like their local station, and then select stations or crimes to compare via the charts. The actual instructions of how to use the application were left to the introduction page of the website, to ensure that a user could understand it and use it without having to ask questions of the developer. Majority of testers were family and friends but there were also some professional developers (contacted through family connections) and a member of the Garda to assess it for use from that perspective.

5.1.1 User Feedback

After testing the users were asked to fill out a survey created on SurveyMonkey.com. The resultant data was then gathered from the Analyse Results screen. In total 20 users responded to the survey.

How satisfied are you with the look of this website?

Answered: 20 Skipped: 0

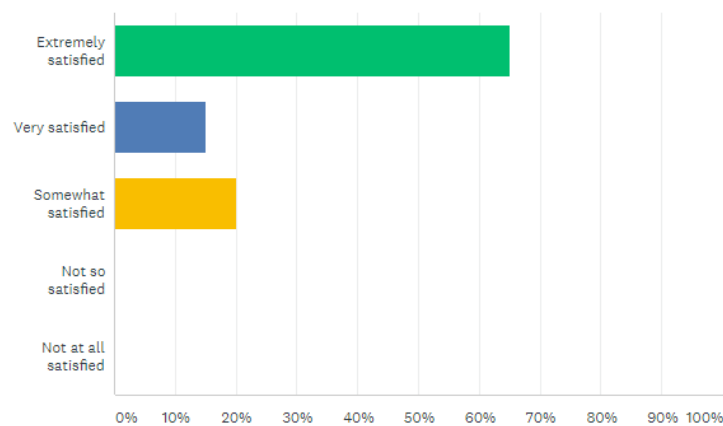


FIGURE 37 QUESTION 1 OF THE SURVEY

Answer Choices	Responses	
Extremely satisfied	60.00%	13
Very satisfied	15.00%	3
Somewhat satisfied	20.00%	4
Not so satisfied	0.00%	0
Not at all satisfied	0.00%	0
Total		20

This was where most of the mixed feedback came from, and it was predominantly early on in the testing, as changes were made based on the feedback quickly as quickly as possible. There is detail on what exactly some of the issues with the look of the website were in the responses to question 7. Once changes had been made addressing this feedback all responses were either very or extremely satisfied going forward so while there are still some improvements that could be made, namely sprucing up the introduction page, the current design and layout are quite suitable.

How satisfied are you with the reliability of this website?

Answered: 19 Skipped: 1

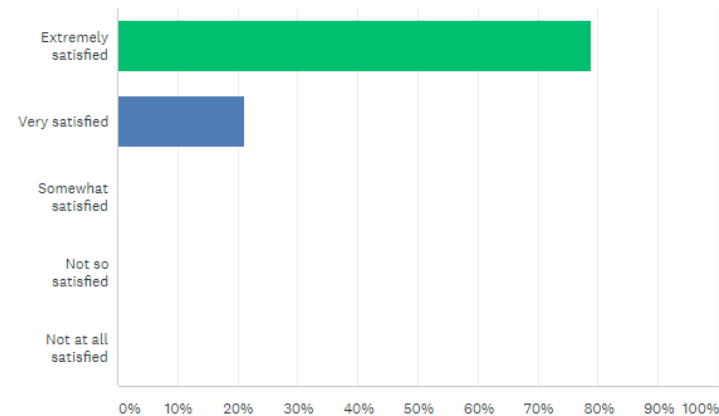


FIGURE 38 QUESTION 2 OF SURVEY

Answer Choices	Responses	
Extremely satisfied	78.95%	15
Very satisfied	21.05%	4
Somewhat satisfied	0.00%	0
Not so satisfied	0.00%	0
Not at all satisfied	0.00%	0
Total		19

This was one of the most important questions as the design, labels, instructions and such could be changed easily enough but it would have been difficult to make changes to the core functionality of the project. Fortunately, all users gave positive feedback on the reliability aside from one user who seemingly forgot to select a response. The regular integration and regression tests seem to have been quite beneficial in the long run despite seeming somewhat monotonous during development cycles.

How difficult did you find the application to use?

Answered: 20 Skipped: 0

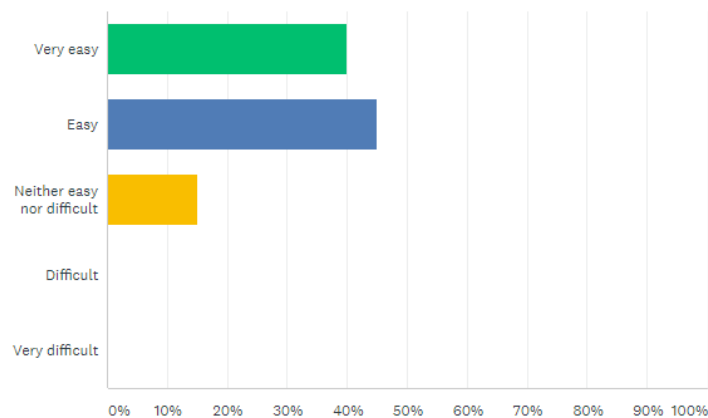


FIGURE 39 QUESTION 3 OF SURVEY

Answer Choices	Responses	
Very easy	40.00%	8
Easy	45.00%	9
Neither easy nor difficult	15.00%	3
Difficult	0.00%	0
Very difficult	0.00%	0
Total		20

Another initially mixed section. This was a very useful source of feedback as it led to the rewording of some of the labels in the menu, rewriting of the instructions, and some slight design changes to make it more apparent what was doing what. The checkbox that tells the application whether to use the selected year's max value or the max value across all years was a particularly difficult point to phrase correctly. It required several attempts at rewording to get the message of its function across while keeping it short and simple. Again, as the changes were made and feedback was addressed the responses shifted to be more positive but clearly there are still improvements that could be made looking at the split of easy and very easy responses.

Were the instructions on the introduction page helpful with understanding the application?

Answered: 20 Skipped: 0

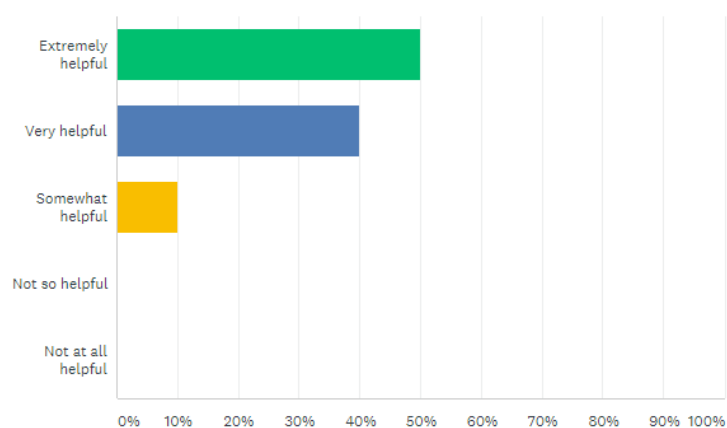


FIGURE 40 QUESTION 4 OF SURVEY

Answer Choices	Responses	
Extremely helpful	50.00%	10
Very helpful	40.00%	8
Somewhat helpful	10.00%	2
Not so helpful	0.00%	0
Not at all helpful	0.00%	0
Total		20

In this section only the very first tests gave mixed responses, their feedback was immediately taken into consideration, and revisions were made to the instruction slides before further testing. From this point all users found the instructions more clear and easy to understand. Even though there are more responses of extremely helpful than very helpful it is a close enough split that the instructions could likely stand some refinement as well. This is expected to come down to them being quite long, but the decision was made to allow for length in the instructions to avoid needing it in the actual UI.

Did the application clearly communicate the information?

Answered: 20 Skipped: 0

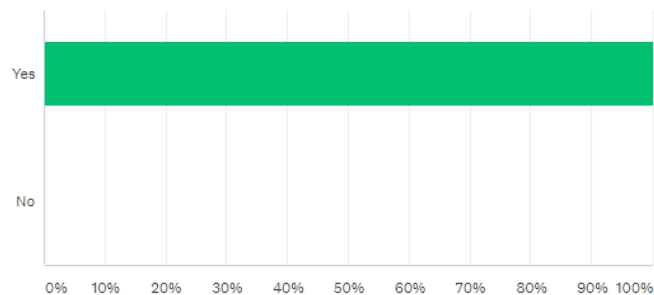


FIGURE 41 QUESTION 5 OF SURVEY

Answer Choices	Responses	
Yes	100.00%	20
No	0.00%	0
Total		20

This was the most encouraging feedback received. It was key to the application that the information was being communicated clearly and was quick and easy to understand. A single negative response in this section would have been worrying, as it would mean that the application was flawed at a fundamental level and was not doing its intended job. A data visualisation would be no use if it did not allow the user to understand the data.

Was there anything that the instructions did not cover that you would like explained, or anything that you thought could have been explained better?

Answered: 15 Skipped: 5

FIGURE 42 QUESTION 6 OF SURVEY

This was the first question that was allowing testers to input their own responses and as such can not be tabulated the way that the previous questions were. Testers were informed to feel free to skip this and the next question if they had no specific feedback on it, some opted to just respond "No",

“instructions were very clear” or a similar response. The main constructive feedback was that there were elements that were not immediately clear, the primary example being that the term “stations” was used in the instructions when stating what could be selected on the map and what a marker is placed at. Users did not understand what stations were until further on, so they were rewritten slightly to ensure to mention Garda stations before just calling them stations. One user was also confused as to why the single bar chart and the pie chart exist in the application as they do not allow for comparison of stations or crimes. The answer to this is that if a user just wanted to view the primary station / crime in a bar chart or see the breakdown of the crime at that station among the years these charts would allow for this more specific look. Not every function in the application will be useful for everybody but it was decided to keep this in for anyone that would want to see this information.

Are there any changes to anything (e.g. the layout or the button labels) you would like to see?

Answered: 15 Skipped: 5

FIGURE 43 QUESTION 7 OF SURVEY

The final question of the survey and another one that allows testers to input their own responses. This was the most useful section in regards specific changes that testers wanted to see and where the actual changes that the users wanted to see could be noted.

5.1.2 Changes due to Feedback

The first response here informed of an issue with the display of the application, that it was too large to view all at once in the browser and had to be scrolled around to see the different parts. This was an oversight as the development was taking place and the integration and regression tests were being done a simple factor in the display had been neglected, that the browser it was being tested in had its default zoom level set to 80%. It was a simple problem to fix but it should not have occurred in the first place, only due to developer oversight. This also called to mind that the layout should be more responsive so that it would always display fully in the browser, which is where the thought of reworking the whole application with bootstrap came in. This redesign could also be used to make the application mobile friendly which at this point in time it is not. At the late stage this realisation occurred this overhaul seemed too great of an undertaking for the time left so it will have to be relegated to work that could take place in the future.

Another aspect added based on early feedback was the colour coding of the markers and showing the markers next to their respective station buttons in the menu.

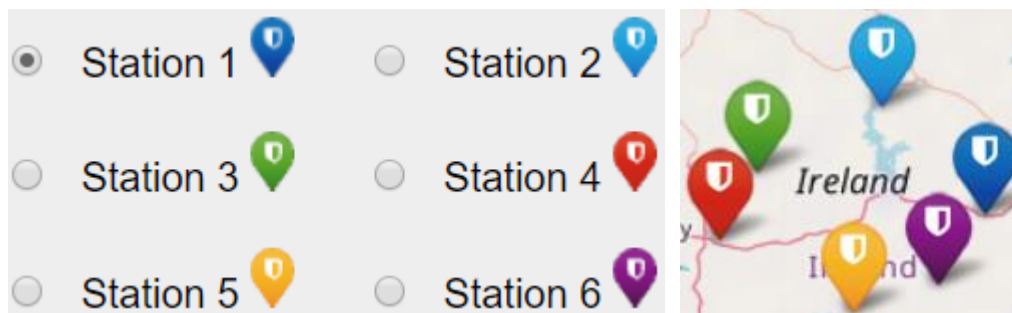


FIGURE 44 COLOUR CODED MARKERS FOR STATIONS

Briefly touched on earlier, initially the markers were all the deep blue of Station 1's marker. This made it unclear which marker was which on the map and the user would have to either to check it against the graph to see or just remember. This was remarked to be an inconvenience and another thing that had not been considered during development. The fix was to create an array of colours matching the

colours used in the graph and when a station button is selected it will colour the marker accordingly before placing it as opposed to using the default blue, e.g. if placing Station 4 the marker will be coloured red before adding it to the map.

The font used was also a point of note for most of the early users. This had not been considered during development as it was far from a pressing issue but it came up in several responses.

The writing on the application page could be formatted to make it more attractive

3/27/2018 9:34 PM

I think a more attractive font could work.

3/29/2018 9:25 PM

Maybe slightly modern fonts possibly, but what's currently used is clear and professional

4/3/2018 7:23 PM

FIGURE 45 SOME OF THE FEEDBACK REGARDING FONT

It was not a high priority thing to change compared to some of the criticisms, so it took longer to get to but once it was changed (Times New Roman to Helvetica) some of the testers were asked to look again and they were pleased with the change.

There was some requests to add search feature for the stations as well. This is a good idea and something that could be added in with some slight modification to the menu but due to time constraints it was left out of the current project.

One of the professional developers requested a meeting in person to discuss their feedback as it was quite extensive. No large scale changes were proposed just multiple smaller ones and largely based on clarifying what the user is to do, even down to the addition of a note at the instructions on the intro page to use the arrows to scroll through instruction slides. The largest change proposed by this tester was to replace the radio buttons for selecting the chart with a dropdown menu.

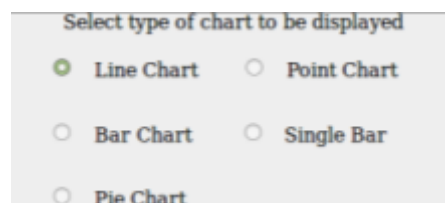


FIGURE 46 OLD CHART SELECTION

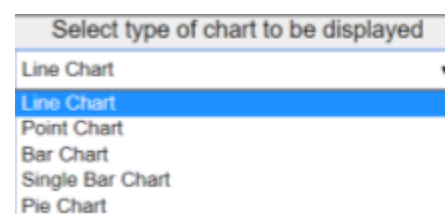


FIGURE 47 NEW CHART SELECTION

This meant that this section of the menu took up less space as the chart dropdown expands over part of the data table when clicked and most of the time occupies a fraction of the space of the radio buttons. After this the menu and map were reduced in height slightly to allow more focus on the graph. Some of this extra space on the graph was used to add a title stating the station / crime selected as a quality of life improvement for users.

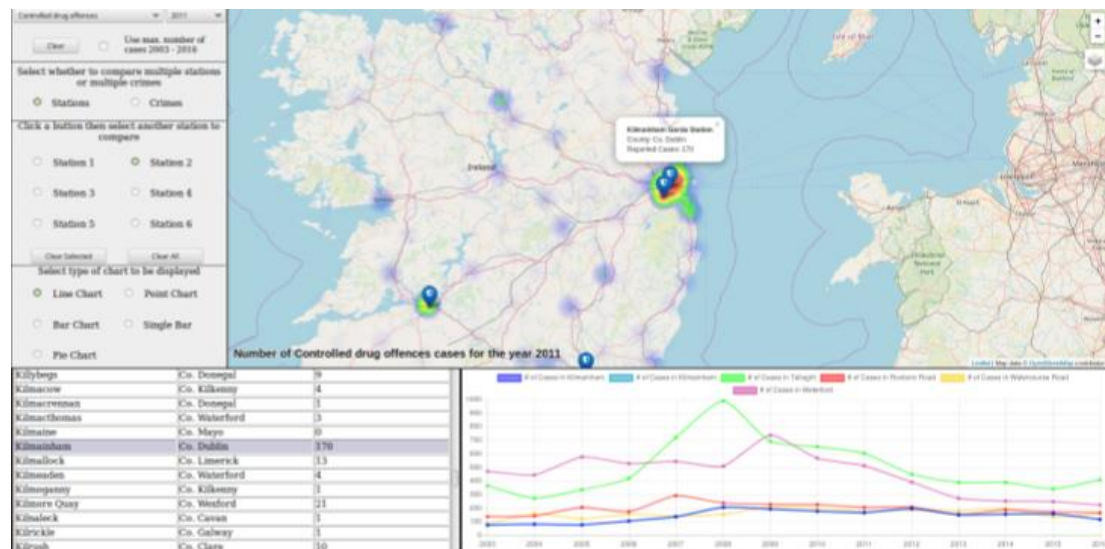


FIGURE 48 APPLICATION AT THE START OF TESTING

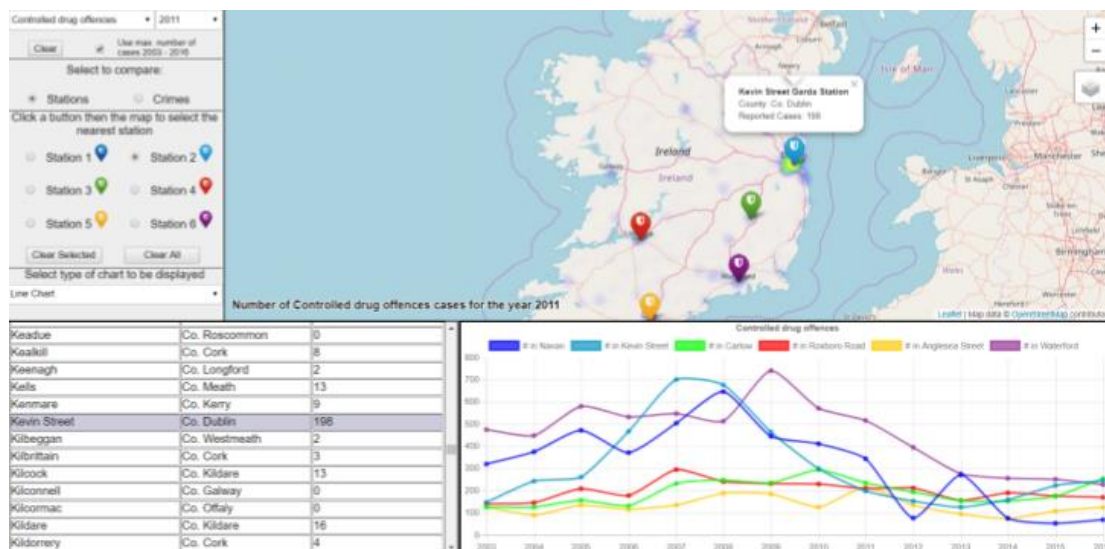


FIGURE 49 APPLICATION AT THE END OF TESTING

The application has been made significantly better due to these user tests, though the changes seemed minor when addressed one by one they add up to make the application far more user friendly and interesting to use.

5.2 Demonstration

A video demonstration of the project has been recorded and can be found [here](#). All of the features of the web application are demonstrated in this video. To ensure clarity there are screenshots below demonstrating the features as well.

Starting at the introduction page it briefly displays the instruction images before proceeding on to the main application. First demonstration is the latitude and longitude bounds on the map to prevent users from scrolling away from Ireland the zoom constraints because there is little to be seen if the map is zoomed out to far.

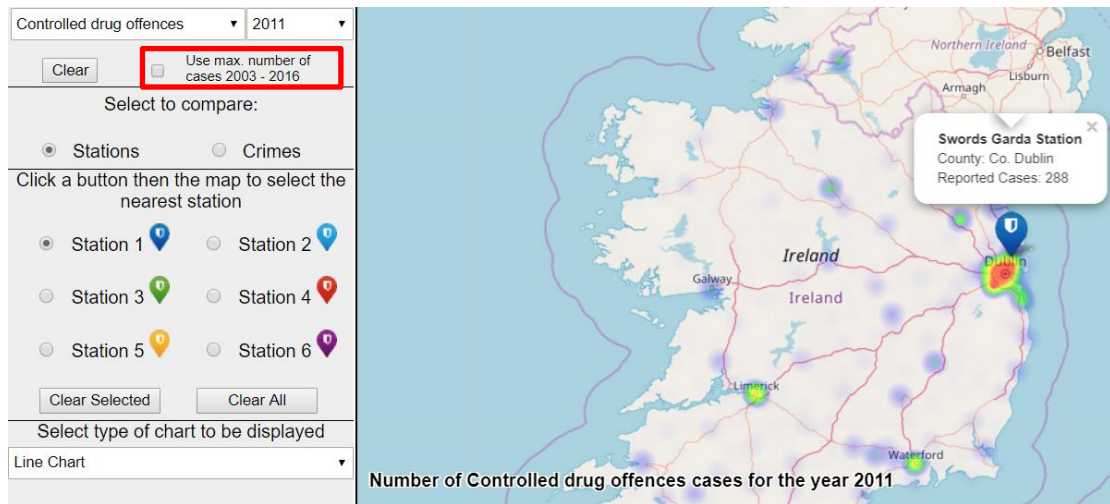


FIGURE 50 CRIME AND YEAR SELECTED

A crime and a year have been selected from the dropdowns at the top, so a call was made to the Django API for the robbery crime rates, then the values for 2016 were entered into an array that is then used to generate the heatmap.

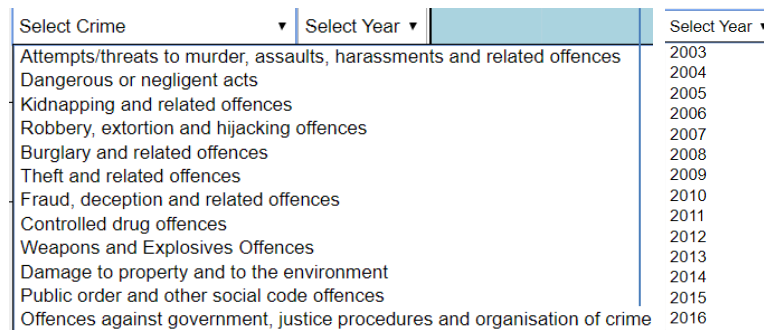


FIGURE 51 CRIME AND YEAR DROPDOWNS EXPANDED

The screenshot above is included due to the software used to record the video demonstration not capturing the expanded dropdown menus.



FIGURE 52 MAX VALUE TOGGLE

The checkbox is toggled on to illustrate the effect on the heatmap. Though the same values are used for generating it there is a noticeable difference in the colours, mostly around Dublin with the chosen values. This is because with the checkbox unticked the max value in the array is set to be the highest amount of robberies that year, with the checkbox ticked the max value in the array is set to be the highest amount of robberies across all years. So, the red area shrank due to these Garda stations no longer being in the upper range of robbery cases, relative to the all-time high for robberies.

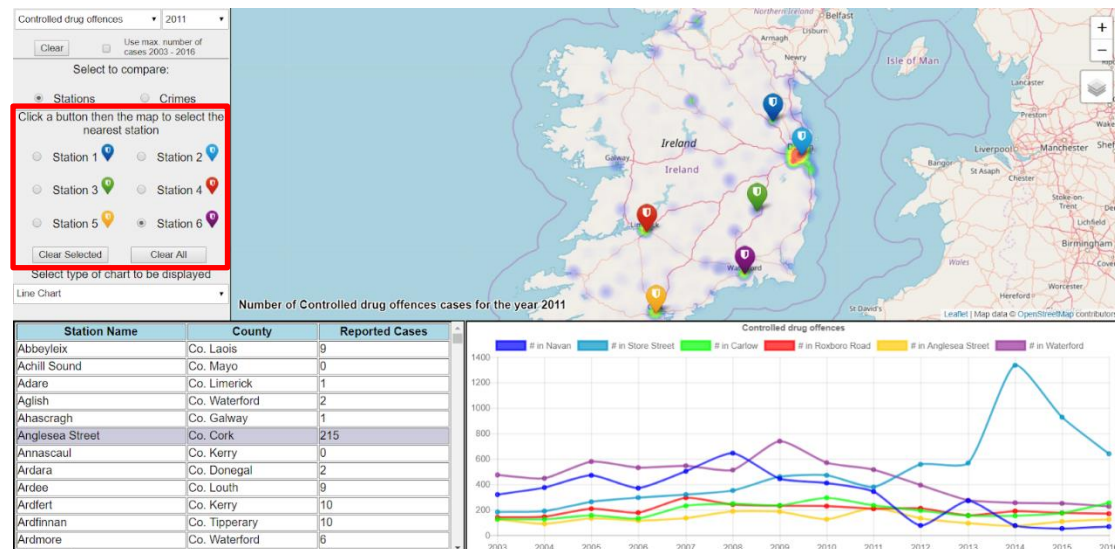


FIGURE 53 ILLUSTRATION OF STATION SELECTION

Using the highlighted radio buttons the user can select up to six stations on the map by either clicking in the area of the station they wish to select or clicking it in the alphabetised table in the bottom left. When the map is clicked the marker snaps to the nearest Garda station to the cursor location. Once a station is selected it is added to the graph in the bottom right of the screen.



FIGURE 54 EXPANDED CHART DROPDOWN AND CHART TYPES

The screenshot displays the interactive map interface for the year 2016. The map shows the island of Ireland with heatmaps indicating the density of robbery, extortion, and hijacking offenses. A red box highlights the 'Select to compare:' dropdown menu, which is set to 'Crimes'. Below it, the 'Select other crimes to overlay on to map and compare in graphs' section is also visible. The 'Drogheda Garda Station' is highlighted with a red circle, showing 30 reported cases. The bottom left shows a table of reported cases by station and county for the year 2016.

Station Name	County	Reported Cases
Abbeyleix	Co. Laois	1
Achill Sound	Co. Mayo	0
Adare	Co. Limerick	1
Aglish	Co. Waterford	0
Ahascragh	Co. Galway	0
Anglesea Street	Co. Cork	16
Annascaul	Co. Kerry	0
Ardara	Co. Donegal	0
Ardee	Co. Louth	1
Ardfert	Co. Kerry	1
Ardinnan	Co. Tipperary	1
Ardmore	Co. Waterford	0

Clicking the Crimes button in the “Select to compare:” menu will change the menu below it from letting the user select stations to compare to letting the user select extra crimes to compare. This generates extra heatmaps that overlay on top of the primary one. If the heatmaps are getting cluttered clicking the Hide Heatmaps button will remove each heatmap layer apart from the primary one but keep the extra crime data in the graph. Alternatively you can select the heatmap layers to toggle off one by one in the menu beneath the zoom control, it expands once the user hovers the mouse on it. The Hide Heatmaps button is functionally identical to manually switching all but the primary heatmap layer off in this menu. Changing between the Stations and Crimes menu clears all data of the menu being switched from. I.e. if there are several stations selected and the user then clicks into Crimes all but the Station 1 are removed, and if there are extra heatmaps overlaid on top of the main one then switching to Stations will clear the extra ones off the map.

6. Project Plan

6.1 Initial Plan

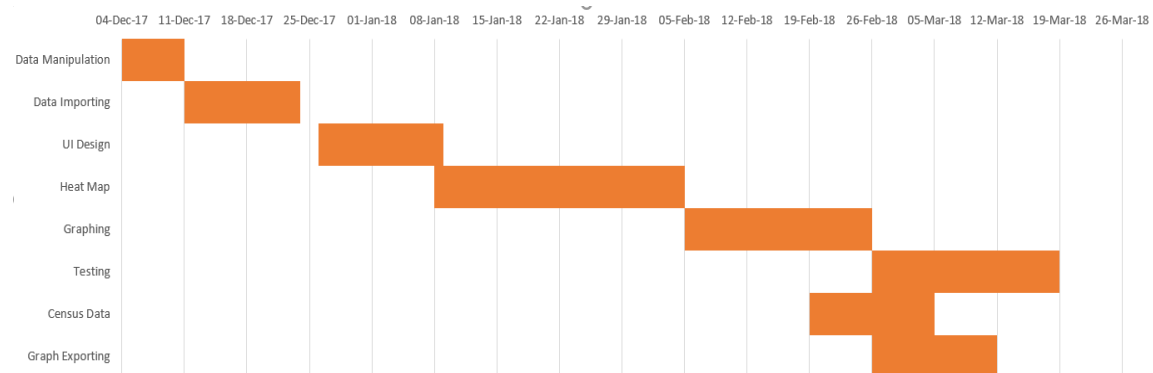


FIGURE 56 GANTT CHART FROM START OF DEVELOPMENT

This Gantt chart was designed prior to the interim demonstration of the project. Because of this there is no accounting for research time as all the significant research was done. The first steps were to manipulate and import the data which wound up running over the budgeted time due to the amount of manipulation that was required. Then the UI design was finalised, during which the idea of the single screen application occurred, later to be finalised in a meeting.

The largest portion of the Gantt chart is the implementation of the heatmap. The library itself was straight forward to bring in to the project. What consumed the most time was using the data received from the Django API for this heatmap. There was a lot of trial and error in finding efficient ways to do it. This is also where the issues with JavaScript not waiting for processes to finish before starting new ones arose.

By the time the heatmap was finished, development was a week behind schedule. However, the Gantt chart was designed to be finished three weeks prior to the actual deadline to account for setbacks so this was no major concern. The implementation of the graphing functionality went as long as expected with the first week using sample data, second week spent getting the data for the selected station across all years and graphing that, and third being the addition of multiple graph types while trying to avoid too much code repetition.

Deployment of the application to a server where it can be freely accessed was never planned on in the initial stages of development. Even coming towards the end of the time scheduled in the Gantt chart it still was not the plan. It was decided upon, researched, and implemented across just a few days during what was intended to be the testing phase. This addition made the testing stage that much simpler. Instead of having to physically go to people and ask them to test the application in front of me the introduction page was added with the instructions, a survey was designed, and the links were sent out along with a brief description of what they should test.

The testing itself wound up being pushed back due to some of the extra features added to the project, only commencing on the 19th of March. Due to the previously mentioned accounting for setbacks this meant there was still 3 weeks to perform tests and refinements on the system before the final draft needed to be ready. There were refinements made to the application daily at first as more results were coming in with more comments and recommendations. After the first week there were fewer recommendations so there were only redesigns after a few of these critiques had stockpiled. By the third week there was no new features or refinement requested so the application was considered finished.

The graph exporting was initially planned but frankly other extra functionality that was not thought of at the start proved more important to the project, like the ability to compare stations as well as crimes. This resulted in this feature being placed on the backburner and ultimately deemed unnecessary, but if development were to continue after the submission then it could be implemented without too much trouble using some additional JS libraries.

6.2 Future Work

It is unfortunate that the census data was unable to be included. The idea was to counteract the high population density of Dublin skewing the heatmap by having a feature to let the user include census data in the process and get the crimes per 1000 people or something to that effect, the exact number would be tested and adjusted. However, the census data is structured very differently to the other datasets so there would need to be a lot of preparation and manipulation of the dataset before it would be in a usable state for this purpose. For example, there is not a direct correspondence between the number of Garda stations and number of towns in the census data, so some would require grouping together for it to accurately reflect the population under the jurisdiction of each Garda station. The project schedule did not allow for this, but it would be a great boon to the application and its usefulness if it were to be implemented in the future.


```

37 var stats = [];
38 //Array for the selected crime rates for the selected year
39 var vals = [];
40 //Array for the selected crime rates for all years, used in charts
41 var allVals = [];
42 //Array for creating the heatmap, populated with max value
43 //Location data, and crime values for the selected year
44 var hmap = [];
45 //Array of arrays for the extra selected crime and year
46 var extraVals = [];
47 //Array of arrays for the selected crime across all years
48 var extraAllVals = [];
49 //Array of arrays for extra heatmaps
50 var extraHmap = [];
51 //Vars for storing selected crime and year
52 var year = "";
53 var crime = "";
54 //Array of extra selected crimes and years
55 var extraYear = [];
56 var extraCrime = [];
57 //Var for storing selected station index
58 var selected = null;
59 //Var for storing the selected station's position in the table
60 var selectNo = null;
61 //Var refers to the opened marker for the selected station
62 //Used to remove marker when another position is clicked
63 var openStation = null;
64 //Var for the selected crime rate for the selected station across all years
65 //Used for plotting selected station on the chart
66 var selectData = [];
67 //Array for extra selected crime rates for the selected station across all years
68 //Used for plotting extra crimes for one station on the chart
69 var extraSelectData = [];
70 //Array for storing extra selected stations indexes
71 var extraSelected = [null, null, null, null, null];
72 //Var refers to the opened marker for the selected station
73 //Used to remove marker when another position is clicked
74 var extraOpenStation = [];
75 //Array for selected crime rates for extra selected stations across all years
76 //Used for plotting extra stations on the chart
77 var extraSelectedStat = [];
78 //Var for storing which station radio button in the UI is selected
79 //Lets the application know which selected station to update
80 //When a new location / table row is clicked i.e. if Station 3
81 //is selected then the map is clicked update the stored Station 3
82 var selectedIndex = null;
83 //Array for storing the extra selected stations' positions in the table
84 var extraSelectNo = [];
85 //Array for storing names of selected stations, used for labelling chart
86 var statName = ["", "", "", "", "", ""];

```

FIGURE 57 VARIABLES USED FOR CRIME RATES

Figure 57 contains the variables used for getting dropdown values, storing the received data from the Django API, formatting data into the heatmap format, referencing the selected station and its place in the data table, the currently clicked Station button (Station 1, Station 2 etc) and the names of all the selected stations. Before adding the functionality for comparing stations most of this list did not exist and the functions for handling these processes were significantly shorter than they are now with it included. Everything beginning with “extra” was added to account for the additional data being stored. This could be cleaned up and the number of variables roughly halved if the project were to be tackled again with this feature in mind and the knowledge gained from creating the system once. Even without redesigning from the ground up it could be done, but it would require reworking of a lot of the core functionality and there was not enough time left once this realisation had come to light for it to be acted upon. Most of the variables that do not begin with “extra” could be gotten rid of and instead just have the primary station, crime, year, and associated data occupy the first space in each of the “extra” arrays.

Another lesson that was learned late into the development cycle that was briefly touched upon earlier is the layout and design of the application. It is not as responsive as it ideally could be. It displays at a fixed height and width meaning that if a monitor has a lower resolution or different aspect ratio than the one used in development the application will either appear slightly zoomed in or slightly zoomed out. This causes no issues with the functionality, everything still works as intended. It just means the user needs to scroll a small bit to see the edge of the graph or the bottom of the data table. It makes the project look somewhat unprofessional and unfinished. This responsive design could be facilitated by expanding the use of Bootstrap in the project beyond just the introduction page as Bootstrap is intended to be used to create websites that display across a whole range of mobile devices and computer browsers alike.

7. Conclusion

7.1 Database

The PostgreSQL database is in the exact state it was expected to be at the outset of this project's development. The location data is all in, the crime data is divided into its separate tables allowing for smaller, more efficient data retrieval, and the whole thing is deployed up on to an AWS machine. The data itself is cleaned up to work both with the application and with the other dataset. The only slight modification to be considered would be the removal of the station names from the crime data. It was included due to its usefulness in troubleshooting whether the data was being received in the intended order and matching up with that of the location data. Once this was verified there is no purpose to these tables having a column for the station names anymore as the position of each row in the crime tables is the same as the corresponding data in the station location table and this table has the station names already.

7.2 Django API

The main thing that needed to be checked in the Django API was that it was retrieving the data correctly from the database. This was first verified through checking the IP address (<http://139.59.162.120>) to examine the API root. From here each table's endpoint was looked at to ensure they were all displaying in the same order. A few values were chosen at random for each endpoint to be checked against the original, local file of the dataset to verify the contents. The API is performing its task quickly, efficiently, and correctly. There is little that could be improved on this aspect of the project.

7.3 Web Application

The web application is where there are the most considerations of things that could have been done differently. The changes that could be made if the development were to start over features that could be added on still either new ones or secondary features that wound up being left out. While all of the primary functionality is in along with some extra there is still significant room for improvement here. As mentioned in section 6.2, not being able to get the census data in due to running short on time was the greatest disappointment for this. This one extra function would have given the project a whole new application and made it a greater interest to more people.

The structure of the code could be somewhat improved and made neater, along with cutting out a lot of unnecessary variables after the inclusion of extra crimes and extra stations for comparison. This is something that could have been avoided with some more detailed planning out of the application. There was some planning in the initial stages but a lot of the actual planning of code was done as the need for it arose which is not great practice and could lead to major problems on a bigger project than this.

Another aspect touched on in section 6.2 is the reworking of the actual design to allow for a full display across different screen resolutions and aspect ratios. Added to this the project could be made significantly better by facilitating use from mobile devices. If users were able to pull up the application and view this data from their phones interest in the website would greatly increase simply due to ease of access.

8. Bibliography

1. Kirk A. Data Visualisation: A Handbook for Data Driven Design. SAGE; 2016. 369 p.
2. Singh AASA. Is Big Data the New Black Gold? [Internet]. WIRED. [cited 2018 Apr 13]. Available from: <https://www.wired.com/insights/2013/02/is-big-data-the-new-black-gold/>
3. Double-Digit Growth Forecast for the Worldwide Big Data and Business Analytics Market Through 2020 Led by Banking and Manufacturing Investments, According to IDC [Internet]. IDC: The premier global market intelligence company. [cited 2018 Apr 13]. Available from: <https://www.idc.com/getdoc.jsp?containerId=prUS41826116>
4. Clark D. Data Visualization Is The Future - Here's Why [Internet]. Forbes. [cited 2017 Nov 8]. Available from: <https://www.forbes.com/sites/dorieclark/2014/03/10/data-visualization-is-the-future-heres-why/>
5. Chart Maker - BEAM [Internet]. [cited 2017 Nov 8]. Available from: <https://beam.venngage.com/>
6. Trulia Local [Internet]. [cited 2017 Nov 8]. Available from: <https://www.trulia.com/local>
7. UK Police Data Geographic Profiles [Internet]. [cited 2017 Nov 8]. Available from: <http://geographicprofiler.com/crimes/uk-police>
8. Recorded Crime Monitoring Tool | All-Island Research Observatory [Internet]. [cited 2017 Nov 8]. Available from: <http://airo.maynoothuniversity.ie/external-content/recorded-crime-monitoring-tool>
9. Insider JD Business. Here are all the Google services with more than a billion users [Internet]. Business Insider. [cited 2017 Nov 13]. Available from: <http://uk.businessinsider.com/google-services-with-1-billion-users-2015-10>
10. Google Maps APIs [Internet]. Google Developers. [cited 2017 Nov 12]. Available from: <https://developers.google.com/maps/>
11. About [Internet]. Mapbox. [cited 2017 Nov 13]. Available from: <https://www.mapbox.com/about/>
12. About | OpenStreetMap Blog [Internet]. [cited 2017 Nov 13]. Available from: <https://blog.openstreetmap.org/about/>
13. OpenStreetMap [Internet]. OpenStreetMap. [cited 2017 Nov 13]. Available from: <http://www.openstreetmap.org/>
14. What's a JS library? [Internet]. Khan Academy. [cited 2018 Apr 4]. Available from: <https://www.khanacademy.org/computing/computer-programming/html-css-js/using-js-libraries-in-your-webpage/a/whats-a-js-library>
15. React - A JavaScript library for building user interfaces [Internet]. [cited 2017 Nov 14]. Available from: <https://reactjs.org/index.html>
16. JSON [Internet]. [cited 2017 Nov 29]. Available from: <https://www.json.org/>
17. GeoJSON [Internet]. [cited 2017 Nov 29]. Available from: <http://geojson.org/>
18. Leaflet — an open-source JavaScript library for interactive maps [Internet]. [cited 2017 Nov 14]. Available from: <http://leafletjs.com/>
19. jquery.org jQuery F-. jQuery [Internet]. [cited 2017 Nov 14]. Available from: <https://jquery.com/>
20. MySQL [Internet]. [cited 2017 Nov 14]. Available from: <https://www.mysql.com/>

21. MySQL :: MySQL 5.7 Reference Manual :: 11.5 Extensions for Spatial Data [Internet]. [cited 2017 Nov 14]. Available from: <https://dev.mysql.com/doc/refman/5.7/en/spatial-extensions.html>
22. PostgreSQL: About [Internet]. [cited 2017 Nov 14]. Available from: <https://www.postgresql.org/about/>
23. Creating GeoJSON Feature Collections with JSON and PostGIS functions - Postgres OnLine Journal [Internet]. [cited 2017 Nov 14]. Available from: <http://www.postgresonline.com/journal/archives/267-Creating-GeoJSON-Feature-Collections-with-JSON-and-PostGIS-functions.html>
24. VMware vSphere 5.1 [Internet]. [cited 2017 Nov 14]. Available from: https://pubs.vmware.com/vsphere-51/index.jsp?topic=%2Fcom.vmware.vsphere.vm_admin.doc%2FGUID-CEFF6D89-8C19-4143-8C26-4B6D6734D2CB.html
25. What is AWS? - Amazon Web Services [Internet]. Amazon Web Services, Inc. [cited 2017 Nov 14]. Available from: <http://aws.amazon.com/what-is-aws/>
26. About the Apache HTTP Server Project - The Apache HTTP Server Project [Internet]. [cited 2018 Apr 7]. Available from: https://httpd.apache.org/ABOUT_APACHE.html
27. nginx [Internet]. [cited 2018 Apr 7]. Available from: <https://nginx.org/en/>
28. Massé M. REST API design rulebook: designing consistent RESTful Web Service Interfaces. Beijing: O'Reilly; 2012. 94 p.
29. What is REST? [Internet]. Codecademy. [cited 2017 Nov 29]. Available from: <https://www.codecademy.com/articles/what-is-rest>
30. Zhu H. Software design methodology. Amsterdam ; Boston: Elsevier Butterworth-Heinemann; 2005. 340 p.
31. Parnas D., Weiss DM. Active design reviews: Principles and practices. J Syst Softw. 1987 Dec;7(4):259–65.
32. Agile Methodologies for Software Development [Internet]. VersionOne. [cited 2017 Nov 29]. Available from: <https://www.versionone.com/agile-101/agile-methodologies/>
33. Burns RN, Dennis AR. Selecting the appropriate application development methodology. ACM SIGMIS Database. 1985 Sep 1;17(1):19–23.
34. Station Map [Internet]. [cited 2017 Nov 29]. Available from: http://www.garda.ie/Stations/Map.aspx?gardaKMZFilename=Garda_Stations.kmz&Mode=gardaViewAll&Args=undefined
35. KML Tutorial | Keyhole Markup Language [Internet]. Google Developers. [cited 2017 Nov 29]. Available from: https://developers.google.com/kml/documentation/kml_tut
36. KML to SHP Converter Online - MyGeodata Cloud [Internet]. [cited 2017 Nov 29]. Available from: <https://mygeodata.cloud/converter/kml-to-shp>
37. Shapefiles—ArcGIS Online Help | ArcGIS [Internet]. [cited 2017 Nov 29]. Available from: <https://doc.arcgis.com/en/arcgis-online/reference/shapefiles.htm>
38. Serializers - Django REST framework [Internet]. [cited 2018 Apr 10]. Available from: <http://www.django-rest-framework.org/api-guide/serializers/>
39. Design Scenarios - Communicating the Small Steps in the User Experience [Internet]. The Interaction Design Foundation. [cited 2017 Nov 22]. Available from: <https://www.interaction-design.org/literature/article/design-scenarios-communicating-the-small-steps-in-the-user-experience>

40. What is Docker? [Internet]. Docker. 2015 [cited 2018 Apr 10]. Available from: <https://www.docker.com/what-docker>
41. Bruno E. A Scrollable, Selectable HTML Table with JavaScript and CSS · Sweetcode.io [Internet]. Sweetcode.io. 2017 [cited 2018 Apr 11]. Available from: <https://sweetcode.io/scrollable-selectable-html-table/>
42. Silva C. Leaflet.ExtraMarkers: Custom Markers for Leaflet JS based on Awesome Markers [Internet]. 2018 [cited 2018 Apr 12]. Available from: <https://github.com/coryasilva/Leaflet.ExtraMarkers>
43. Font Awesome 5 [Internet]. [cited 2018 Apr 12]. Available from: <https://origin.fontawesome.com/>
44. Leaflet Heatmap Layer Plugin [Internet]. [cited 2018 Apr 13]. Available from: <https://www.patrick-wied.at/static/heatmapsjs/plugin-leaflet-layer.html>
45. Chart.js | Open source HTML5 Charts for your website [Internet]. [cited 2018 Apr 13]. Available from: <https://www.chartjs.org/>
46. contributors MO Jacob Thornton, and Bootstrap. Bootstrap [Internet]. [cited 2018 Apr 13]. Available from: <https://getbootstrap.com/>
47. Home - Django REST framework [Internet]. [cited 2018 Apr 13]. Available from: <http://www.django-rest-framework.org/>
48. How To Install Nginx on Ubuntu 16.04 [Internet]. DigitalOcean. [cited 2018 Apr 13]. Available from: <https://www.digitalocean.com/community/tutorials/how-to-install-nginx-on-ubuntu-16-04>