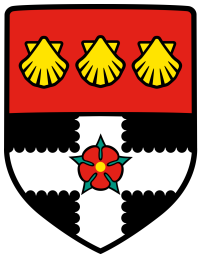


University of Reading

Department of Computer Science

Crime Mapper

Daniel Pitfield

Supervisor: James Ferryman

A report submitted in partial fulfillment of the requirements of
the University of Reading for the degree of
Bachelor of Science in Computer Science

May 10, 2020

Declaration

I, *Daniel Pitfield*, of the Department of Computer Science, University of Reading, confirm that all the sentences, figures, tables, equations, code snippets, artworks, and illustrations in this report are original and have not been taken from any other person's work, except where the works of others have been explicitly acknowledged, quoted, and referenced. I understand that if failing to do so will be considered a case of plagiarism. Plagiarism is a form of academic misconduct and will be penalised accordingly.

Daniel Pitfield
May 10, 2020

Abstract

Abstract Text

Table of Contents

Chapter 1 - Introduction

1.1 Background

This project is to be a computer-based crime mapping tool, primarily focused at facilitating the process of crime mapping, that is identifying where a crime has taken place so it can then be mapped and visualised. The tool will also strive to work with any such data provided using further techniques such as crime analysis, a process which aims to identify the existence of both short and long term patterns in crime, as well as crime prediction or prevention which is the process of identifying and then preemptively putting a stop to potential crime in the future, to provide information about past, present and future crime to all.

A challenge faced with crime mapping is having to balance the sensitivity of crime data with how readily it is made publicly available, which often means access to crime mapping tools is limited. This project aims to break this trend of the select few being able to benefit from effective crime mapping tools with a purpose of ensuring greater transparency, so that the benefit of crime mapping can be offered to more potential stakeholders each with their own reasons to use such tools, ranging from property buyers looking to understand the safety of neighbourhoods around homes for sale or insurance companies and security firms trying to better inform themselves over how to price their policies.

This motivation to improve accessibility has been driven by difficulties that have been experienced firsthand, it stems from past and personal educational study into the effectiveness of an urban regeneration project being implemented for a nearby, less affluent geographic area (a local town). At the time of the study, it was thought that access to crime information for the area over time would have been invaluable as evidence for any improvements as a result of the project, to use in this study. But unfortunately, the information or a relevant crime mapping solution hadn’t been developed or made available for this purpose at the time (and whilst some progress has been made in this area), there is still opportunity for this project to further address this problem.

1.2 Aims and Objectives

1.2.1 Aims

1) To catalogue individual occurrences of crime along with attributes detailing the crime
2) To select and highlight smaller subsets of crime that only exhibit certain characteristics
3) To visualise geographic areas that experience proportionally lower or higher levels of crime
4) To catalogue multiple occurrence of crimes from other formats of data

1.2.2 Objectives

1) Implement a process of plotting crime locations, by the end of the calendar year (31 December 2019), achieving data input times (for a single crime) of no more than a minute
2) Identify a minimum of three relevant crime attributes used when recording crime and assign these attributes with crime locations through further data input by 7 January 2020, whilst also preventing the entire data input process being extended by any longer than an additional minute
3) Develop a method to present all information recorded for a crime, that requires only a single action from when the location of a crime has been identified, all by 10 January 2020
4) Develop a method to amend the information recorded for a crime, that takes less time to complete than plotting a new crime location, within the same timeframe as the last objective

5) Identify applicable crime attributes to isolate specific occurrences of crime and classify them as descriptive or numeric, discrete or continuous
6) Provide a process to specify crime attribute values for crimes to isolate crime with using only lists for descriptive attributes or value ranges for numeric attributes
7) Hide (temporarily) the plots with assigned attributes not matching those specified through the deliverable aimed at meeting the previous objective, achieving filtering times of less than a second per 100 crime occurrences, all before a timeframe deadline of 31 January 2020

8) Investigate analysis methods relating to spatial distribution of crime and compute the result of the analysis when applied to the locations of any plotted crime
9) Apply at a minimum one suitable analysis method with the results being clearly visualised (fully and correctly interpreted by greater than 9 out of every 10 users) with direct consideration to the possible target audiences it may be presented to, by 31 February 2020

10) Investigate the different formats crime data can take and the extent of their use in the field, identifying the most commonly used format
11) Implement the selected data format to map crime on a large scale, achieving to import crime from at least one official crime data source before one month of the project deadline or 20 March 2020.

1.3 Problem Statement

The range of individuals or parties with access to crime mapping tools and thus those able to undertake the direct process of mapping crime on a large scale has stagnated as of modern times, as have the analytical techniques that process or take advantage of the vast amounts of raw data made available through crime mapping. There needs to be a way of shifting the administrative power of crime mapping that law enforcement agencies currently hold by widening those who are able to view mapped crime and even, quite innovatively, in controlled circumstances, those are able to record and add mapped crime. Furthermore, the methods used to analyse and predict crime continually evolve but are either not made publicly available or are not provided for use in any way and so there needs to be ways of more transparently or openly presenting and conveying the important results these methods produce to those who may significantly benefit from it, meaning any valuable insight in to past, present or future crime is no longer confined to law enforcement agencies or alike.

1.4 Solution Approach

The methodology proposed to meet the aims and objectives is a web-based approach, involving the development of a website which can be used simply with access to a web browser. This eliminates the barrier of entry faced by alternative approaches such as developing a standalone desktop or mobile application, for which a download and installation process would be required. It also additionally means that mobile devices are indirectly supported (as after all these devices would still be able to access a website) but it must be said that, despite being at the early stages in the software development cycle, the smaller screen sizes of these devices don't bode well for the large amount of information and visualization the crime mapper is envisioned to provide and it may be that a proportionally small amount of development time is spent catering to these devices.

The approach is chosen with the core vision that users will be able to visit this same website to collaboratively map crime, where they will be able to see the additions or changes made by other users in real time as opposed to each opening a program and seperately using their own individual versions of a crime map. The idea is that an interactive map will be provided to users, acting as a blank canvas to be added to over time, similar to a physical crime mapping board and pins, with the locations of and details about crime being displayed on this map. In essence, the idea is that crimes being added to the mapper are made to persist and are able to be viewed by any and every user at any time and importantly are not lost or discarded when a single user decides to close the website.

1.5 Organisation of the report

This report is organised into seven chapters. This first chapter (Chapter 1) has introduced the background of the project, its aims and objectives, an articulation of the problem investigated as well as a brief mention of the approach to meet these aims and objectives.

Chapter 2 will present a literature review looking into existing literature and solutions relevant to the field of crime mapping. Chapter 3, is split into two subsections and adopts a software engineering methodology approach of a design and an implementation stage. The results of the implementation are shown (as evidence for the objectives being met) in Chapter 4 and are tested in Chapter 5 before reflection on the project outcomes and process being detailed in both the chapters of Chapter 6 and Chapter 7.

Chapter 2 - Literature Review

2.1 Introduction

Whilst Chapter 1 introduced only the theory of crime mapping, the scope of this review is aimed at broadening the field of research by introducing two related processes, those being crime analysis and crime prediction (sometimes also called crime prevention). Crime analysis is the process of evaluating crime data in an attempt to identify the existence of both short and long term patterns in crime and crime prediction/prevention is the process of identifying and then preemptively putting a stop to potential crime in the future.

As an early on indication, as the review was conducted, it was apparent that there was a high availability of literature (and existing solutions) for the initially introduced concept of crime mapping but less availability for these newly introduced concepts of crime analysis and crime prediction/prevention. This begins to suggest how well established and researched crime mapping is and how the other process may perhaps either be in their early stages of adoption or have yet to have been fully explored (providing an initial understanding as to the gaps of knowledge currently present in the broad area of research).

2.2 Literature

2.2.1 Crime Mapping

Police services and other law enforcement agencies used pins on a board as an early form of crime mapping to help visualise crime distribution. Although they were useful in showing where a small number of crime events had occurred, the physical nature of using boards had many limitations. They typically become hard to read or understand as more crime events were added to the board (ineffective crime analysis), offered little to no benefit in the way of storing historical information (data loss) and they also caused complications due to the large amount of physical space the boards and mapping occupied.

Nowadays, GIS applications have widespread use in crime mapping, that is software tools that allows users to modify, visualise, query and analyse geographic data. Such systems help to overcome prior limitations, an example being that the digital data they use and produce is far more easily archived as compared to before. The benefits that GIS applications provide help not only with crime mapping but also with other related processes (namely the concepts alleviated to above) by allowing the use of more efficient techniques and methods. One such technique is geocoding, the process of taking information either in the form of street addresses or location descriptions from a database and then translating it to a location usually involving longitude and latitude coordinates on a digital map. Although useful in improving the speed in which information can be provided to and be used within a GIS as well as being widely sought after and adopted by law enforcement agencies **[1]**, its accuracy and reliability when compared to manual translation of points, has frequently been disputed. Inaccuracy is firstly said to be caused by the geocoding system misinterpreting or failing to understand abbreviations or different spellings of street names as well as incorrectly placing a point tens or hundreds of metres from the true location due to a lack of appropriate training data or due to general geocoding imprecision **[2]**, but the relevancy of these claims should be strongly questioned, when compared to the time of writing, they originate from sources written when geocoding was in its infancy. More recently, the limitations of geocoding seem to be few and far between with being just confined to the discrepancy in accuracy for urban areas, for which there are higher levels of accuracy due to reference data being more

complete and less likely to change over time, as compared to peri-urban or more rural areas, which typically suffer from lower accuracy with fewer reference points such as road networks or plots of land to go by **[3]**.

A consideration alluded to when using GIS applications for crime mapping is the degree of abstraction present in a map, the extent to which some information of the map is removed in order to better present its important information, with there being a tradeoff in the way that more abstraction provides better legibility and simplicity but inherently means some complexity is lost. Consequently, mapping applications have started to provide multiple levels of abstraction by offering the ability to choose between lower levels such as street outlines and higher levels such as satellite imagery, interchangeably, to cater to different needs and users, an example being Google Maps **[4]**.

The way in which information is visually communicated through mapping can also similarly vary. The two types of maps most frequently used in mapping and analysing crime are:

- ❏ Point/Dot - using either a point or dot as a symbol on a map **[5]**

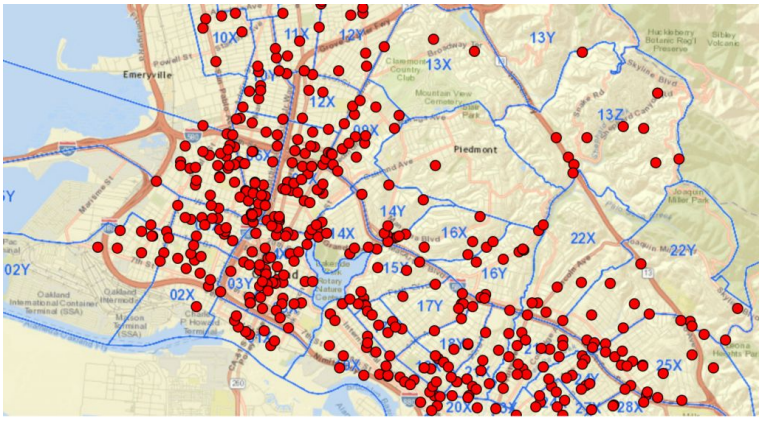


Figure 1: Point/Dot crime map example

- ❏ Choropleth - the shading of areas or regions to express information **[6]**

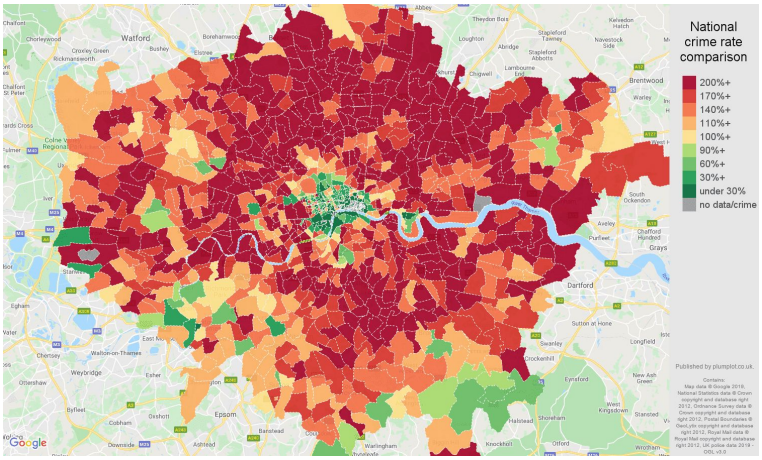


Figure 2: Chloropletht crime map example

Point/Dot maps are prone to overcrowding or overlapping of data points and area maps can be easily misinterpreted, highlighting the importance of the choice of map type.

2.2.2 Crime Analysis

2.2.2.1 Strategic Crime Analysis

This type of crime analysis entails looking at the long-term patterns of crime activity; spatial data clustering is the most useful and most widely used method for this type of crime analysis as of today with it being a way of identifying crime hotspots, areas which experience more crime than other places or where people are at greater risk of being a victim of crime, a process which as a whole is known as hotspot detection. The techniques which are frequently used in conjunction with one another involve grouping crimes that occur within the same geographical areas, with clusters which are 'densely populated' going on to be treated as crime hotspots. The process of looking at why crime events happen at specific locations, form what are known as 'place-based' theories, but without using other spatial data or information as part of the technique, the output presented can often be misleading, exhibiting poor precision and accuracy with an example of such limitation being how highly populated geographical areas are sometimes at risk of being incorrectly identified as crime hotspots despite how they will inevitably see a larger number of crimes than sparsely populated geographical areas. This oversight or limitation is present in an implementation in which the states or regions of India were clustered into crime zones using a technique known as K-means clustering **[7]**. The techniques used in this example involved treating the different states as the center of clusters and then allocating points of crime (data points) to the nearest cluster. Low, medium and high risk crime zones (based solely on the total number of crimes) were then identified, whilst not accounting for or accommodating for the large difference in population between these regions.

2.2.2.2 Tactical Crime Analysis

The limitation in solely using basic spatial clustering for crime analysis seems to be a general consensus and such is why more advanced and refined methods which build upon the technique have been devised. Tactical crime analysis looks past just the crime's location and makes use of other crime characteristics such as how and when the crime occured. An example of this approach **[8]** named 'Attribute Oriented Induce Method' involves the pre-processing of additional information about crimes in the form of attributes (similar to the characteristics listed above). The similarity of crimes is then determined through an overall classification of these attributes (a taxonomy) to better determine crime trends and/or hotspots. Although it is highlighted that further work into the design and optimisation of the taxonomy is needed in order to improve the accuracy of crime analysis using this method, it has not gone unnoticed that the concept of the preprocessing of data (regardless of method) is common in the research field, with another implementation despite using a different algorithm deciding to similarly use attributes and attempt to remove less relevant data before actually performing any analysis on data **[9]**.

2.2.2.3 Administrative Crime Analysis

This area of crime analysis refers to the presentation of findings about crime events with statistics, printouts, charts or maps (to some extent there is overlap with both crime mapping and other types of crime analysis). In terms of the intended solution, it is likely that forms of administrative crime analysis will be the front facing components attempting to inform users with tactical and strategic crime analysis being used in the background, determining and providing the information that needs to be shown.

2.2.3 Crime Prediction/Crime Prevention

As with between the different types of crime analysis, there can be overlap between crime analysis and crime prediction, an example being the identification of hotspots as crime analysis (or crime prediction) but reacting to these presence of hotspots being crime prevention. Inducing factors such as the reassignment of police officers, quicker response times, identifying future victims at risk or general reactive arrest policies are typically what pushes a method be seen as being within the domain of crime prevention **[10]**. Crime prevention is pushing the agenda from using retroactive methods (that is using existing/previous crime data to tackle crime) to proactive methods (that is stopping future crime before it happens). Despite this currently most implementations that make use of crime prediction or prevention are closed and opaque in their nature (no or very few implementations involve information from crime prediction and prevention techniques being shared to other parties other than within or between law enforcement agencies). There is a significant gap in the field in which the determined information from crime prediction and prevention is shared with an effort of being highly transparent, giving further thought to where the intended solution is to fit within but also how it will attempt to supersede what has already been provided in the field.

2.2.3.1 Risk Terrain Modelling (RTM)

Whilst hotspot detection in crime analysis uses crimes that have already happened and can show where crime is happening (it shows indications of crime), it doesn't tell us *why* it's happening (directly addressing crime) **[11]**. Risk Terrain Modelling attempts to explore why crime happens by exploring the relationship between crime and geographical features, whether that be obvious types of places such as parks, houses, or public transport hubs **[12]** or other factors such as the accessibility of roads. A combination and weighting of these features and factors are then used in determining the probability of criminal behaviour occurring. From observation there is strong indication that the nature of technique lends well to being used with GIS, probably due to the large amount of geographic information readily at hand that they provide, as well as being a technique that is self-contained and which can be implemented independently of other techniques probably due to not requiring inputs such as past crime data to perform its operations. Having no reliance on other data is beneficial but it also questions how the technique can accurately pinpoint future crime with such little information to go by, a claim supported by the statement: "The disadvantage of RTM is that it does not create obsolete scenarios where crime will occur" **[13]**. A similar (but independent of risk terrain modelling) technique to quickly mention is geographic profiling, the process which attempts to identify where crime is originating from in the form of the crime base of offenders (essentially where they live). By being able to deduce where criminals live, it becomes quickly apparent why some information determined through this and similar techniques is rarely or not at all shared or released into the public domain but there is no reason why some smaller components or concepts such as 'buffer zones', areas that have been identified for it to be unlikely for criminals to offend again there, could not be **[14]**.

2.3 Existing Solutions

2.3.1 Police Crime Map (1)

As part of the recent coalition government's plans to reform policing, crime data for every street in England and Wales was made available publicly in the early months of 2011 **[15]**. The information was released on the official crime and policing website (police.uk) for England and Wales in the form of a crime map but it was also strongly emphasised and encouraged that third parties could develop their own ways of presenting the data with the aim of improving the transparency of information with the public and the accountability of police forces.

The proprietary implementation developed **[16]** first prompts the user to find and select a neighbourhood using either a postcode or location name with a search element which is always present along the banner of the website. Once a neighbourhood is selected, summary information of the policing team and crime for this neighbourhood is shown along with the ability to navigate on to a crime map. The map highlights the area in question and then shows marker icons representing the number of crimes in a particular location with small information panels about the crime(s) being shown when these icons are clicked. The information shown on the map can be filtered by both the time period and the type of crime by choosing options located just above the map and by confirming to update the map. The implementation as seen in both the source code and the bottom of the map element uses the mapping platform, Mapbox, as a way of providing its functionality.

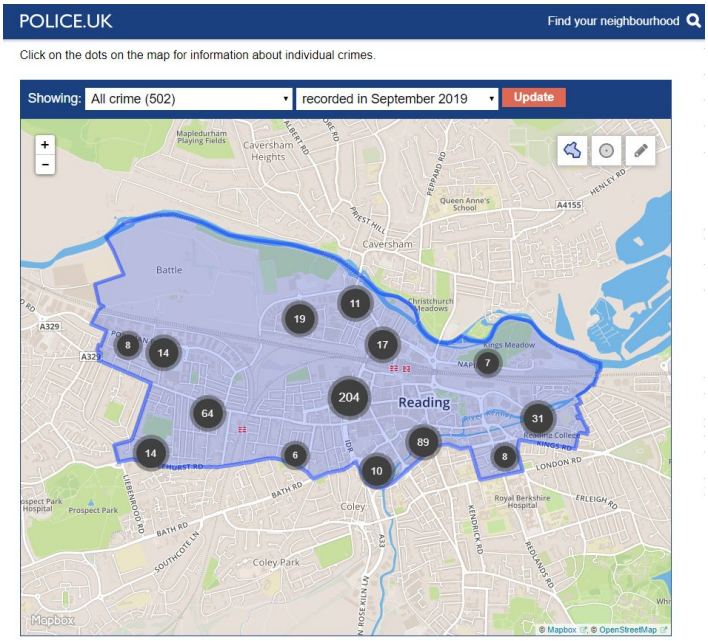


Figure 3: Filter crimes by month

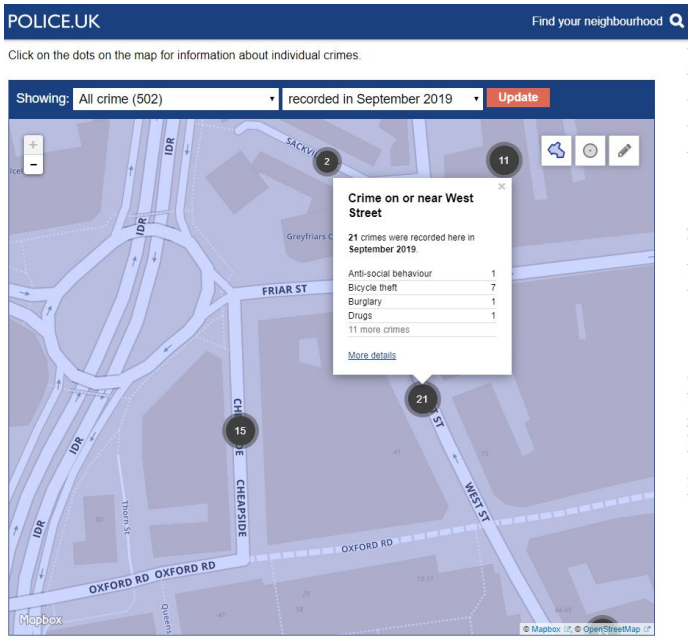


Figure 4: View crimes with information panel

2.3.1.1 Observations

+ Search bar for locational search
+ Low level representations of crime using information panels
+ Robust mapping platform as opposed to a custom made platform
- Only crime information for the neighbourhood selected is shown at any one time (information is segmented)
- Limited filtering options (only one type of crime or a specific month as a time period can be chosen)

2.3.1.2 Effect on solution

The existing solution being limited to viewing information for only a single neighbourhood is restrictive and hinders the user experience, highlighting the importance of enabling users to view (and map) crime across different geographical locations more intuitively. In terms of filtering, providing such functionality is useful but a better approach for the intended solution would perhaps be a checklist or allowing multiple different values to be chosen at once (not just one). The search bar and information panels are useful features too, the search bar especially as it was observed that the navigation of the map was significantly streamline with this being able to be used and so similar features may see use in the proposed solution.

2.3.2 CrimeMapping.com (2)

This online crime mapper **[17]** is provided with crime data from participating police agencies and plots newly committed crimes each day (when it was maintained) and is intended to be used as a platform for members of the public to understand the extent of crime in their neighbourhoods:

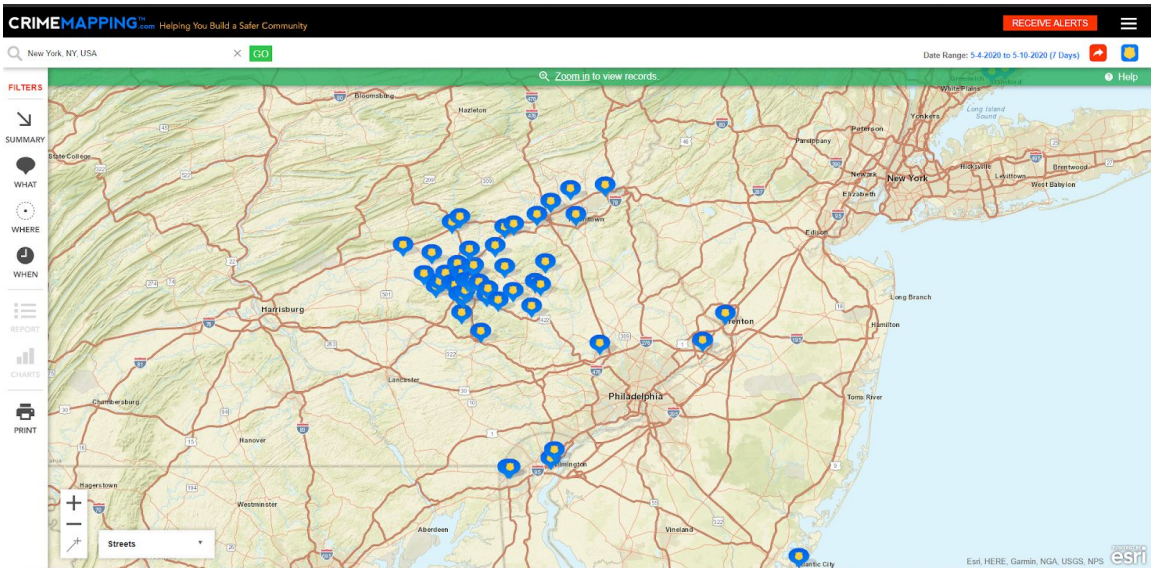


Figure 5: CrimeMapping.com (existing solution) crime map

Similarities with the other existing solution include the use of a search bar and the providing of filtering options but the options provided this time around are far improved with the ability to filter by type of crime, location and date/time of crime with the symbols used (it is a point/dot map) also reflecting the type of crime, as opposed to being generic:

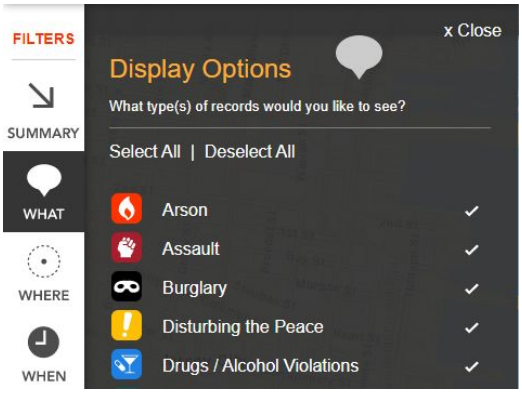


Figure 6: CrimeMapping.com type filter

Figure 7: CrimeMapping.com time filter

2.3.2.1 Observations

+ Unique point/dot symbols for different crime types
+ Extensive filtering options
- No analysis or processing of crime data information
- Interface elements obstruct large areas of the map

2.3.2.2 Effect on solution

Having researched this existing solution, the different types of values that are used for filtering could be and unique icons seem useful features which should be strongly considered for implementation of the intended solution but to improve on what is already out there, this filtering functionality if implemented should somehow be devised as not occupying so much of the screen's space

2.4 Conclusion

Simply mapping crime where it happens as locations or points is the barebones of crime mapping and is just the beginning in a series of processes that tackle crime. Whilst the mapping of crime is an important basis to build on and bears great significance in how effective the later processes are, it's important to stress the importance of innovative crime analysis or prediction/prevention techniques for the intended solution if it wants to stand out in the field. The intended solution could also be madeWhether this be through better transparency of information or other means, consideration to the sensitivity or confidentiality of information must be given (although overapplied there is still reason as to why most information from crime prevention as of now is kept mostly under wraps).

Chapter 3 - Methodology

3.1 Design

3.1.1 User Interface (UI)

From background research, the most prominent hindrance to the usability of crime mappers was the size of the mapping component in proportion to the screen layout and it was apparent that in most cases, the map's size was being unnecessarily diminished to make space for a large number of static user interface elements (usually both at the top of the screen and down each side of the screen). Although such an approach provides clear mapping as to what each UI element actually performs, it results in the discoverability of the user interface being severely affected with a large amount of wasted time being spent trawling through differently labelled buttons before finding the desired functionality. Interface designs like this also limit critical operations of crime mapping such as navigating the map and the visibility of information through data visualisation.

3.1.1.1 Context Menu (Initial Design)

An early design for the proposed solution aimed to overcome this problem by not relying upon static elements and instead making use of dynamic elements which come and go as and when they are needed thus allowing for the mapping component to occupy the entire screen space. For instance, a context menu could be made to appear when requested, with its options opening relevant popup windows, if further user action is required, or simply performing the functionality, if not required. This way means the map is only ever temporarily overlaid as the popup windows can be made to close once finished with. The options anticipated to be used most frequently are positioned towards the top of this context menu acting as a signifier to the perceived, relative importance of the option. Taking this concept forwards, a low fidelity prototype and a high fidelity prototype (a computer based, interactive prototype) of this proposed context menu were produced:

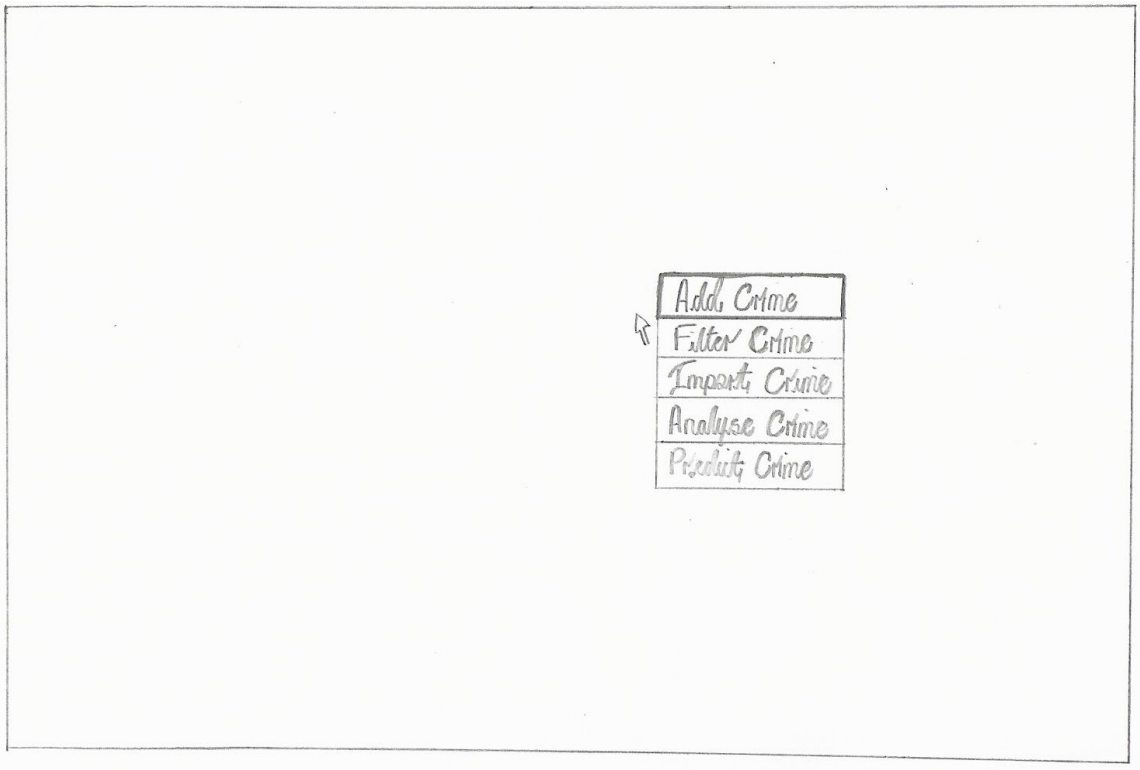


Figure 8: Low fidelity prototype for a context menu design

Figure 9: High fidelity prototype for a context menu design

Whilst the context menu seemed for all intents and purposes suitable initially, when the high-fidelity prototype underwent usability testing, the problem of users accidently choosing the wrong option was frequently reported and along with it uncertainty as to what the precise nature of this problem was. After some clarification from the testers, contrary to what was first thought, this was a problem of frequently misclicking the desired option (a slip error).

In hindsight, this should not have come as a surprise with the options not being separated (no gaps between them) and having such a small target area. Making the options larger wasn an option but this comes with the setbacks or tradeoff with travel time as larger sized options would mean the last options in the context menu are further away from the top of the context menu. In other words, the benefit of a larger target to click on would likely be outweighed by the increased distance of navigating to options (this is the concept of Fitts's Law) **[18]**. But ultimately, thinking ahead to how some of these options' functionality may be implemented, it was considered how the 'Add crime' option may well be the only option which could actually benefit from being requested from the map (for instance the user could click on the map and select to add a crime, where they actually wish to add the crime) and so having the other options as part of this context menu is not necessary and they can be migrated elsewhere.

3.1.1.2 Toolbar and Context Menu (Revised Design)

With this in mind, the next revised design made a small compromise in terms of the screen space for the map by adding a single navigation bar or toolbar across the top of the screen and migrating all but the ‘Add Crime’ option to it (leaving just the ‘Add Crime’ option as part of the map context menu):

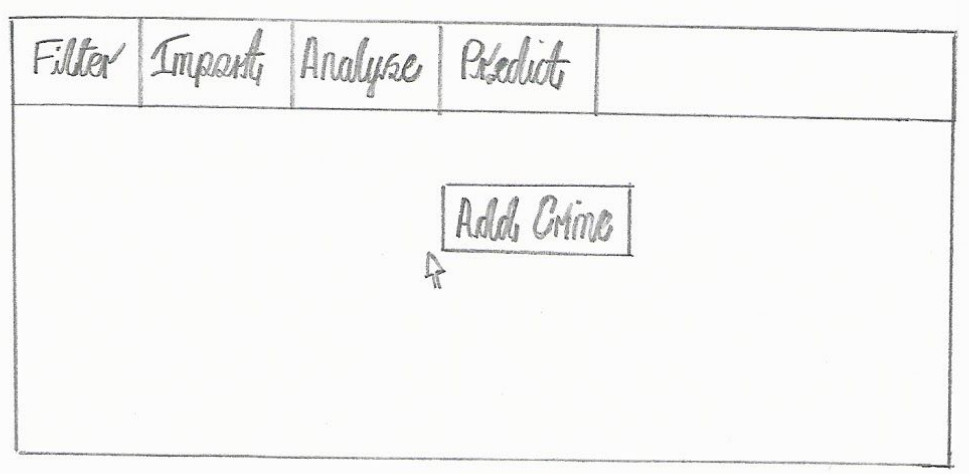


Figure 10: Low fidelity prototype for a toolbar and smaller context menu design

But the user feedback at this stage resembled how the user interface looked incomplete/unfinished, probably due to the large amount of unused space and so another (final) design iteration was needed.

3.1.1.3 Toolbar with Search and Context Menu (Final Design)

This final iteration of design involved increasing the size of options as well as using the toolbar’s remaining empty space by including a search bar:

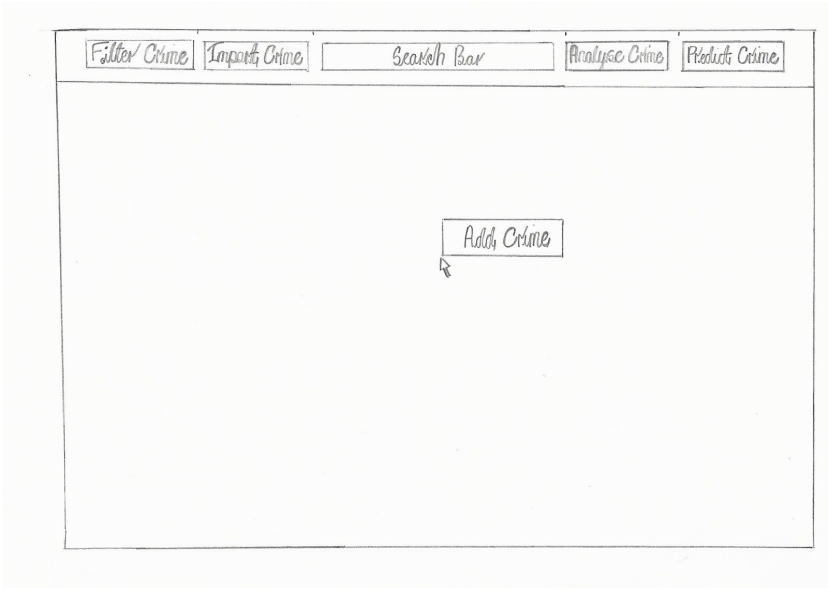


Figure 11: Low fidelity prototype for a toolbar with search bar design

3.1.1.4 Popup Window - Base Design

Before designing the interface for each individual popup window which are intended to be opened either by the options in the toolbar (in the case of ‘Filter Crime’ and ‘Import Crime’) or the sole context menu option (‘Add Crime’), a base design was devised (in the form of a low fidelity prototype) to ensure a standard of consistency, an important principle of interaction design **[19]**. To further ensure consistency of the interface the designed colour for the headers of these popup windows was chosen to be the same colour as the toolbar. The confirmation buttons were intended to be coloured green which semantically indicates progression of a successful action:

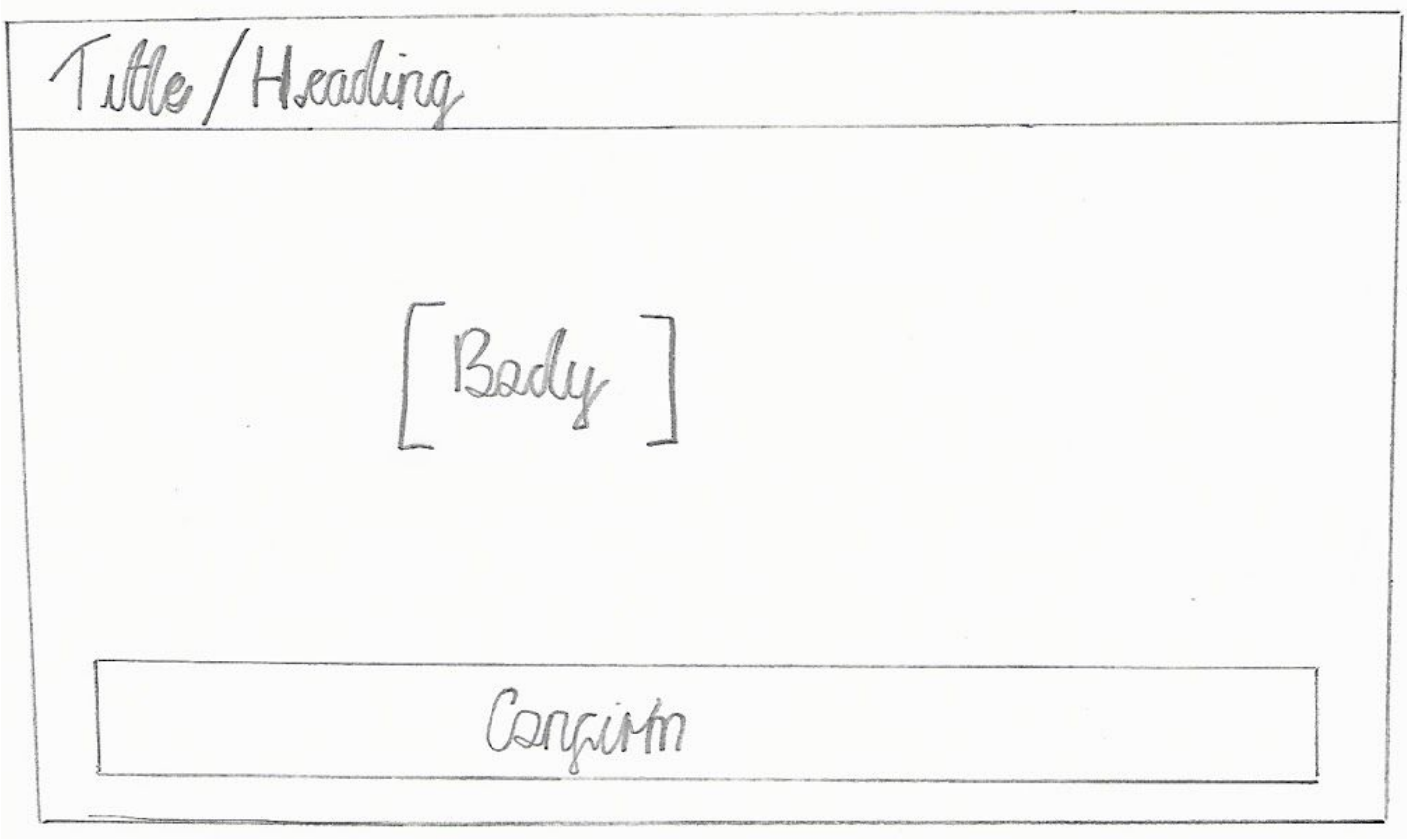


Figure 12: Low fidelity prototype for the base popup window design

Being just a base design, the title would be changed based on the functionality the individual popup windows provide as would the content of the body and perhaps even the text of the confirmation button. The size of the title and confirmation button were designed to be kept constant for all the windows but the overall size of the window can change and be as large as the body requires it to be (but no larger than necessary).

3.1.1.5 Crime Attributes

From the second project objective through to the sixth objective, mention of crime attributes is made and so with these making sense to include in the individual pop-up windows (which were designed to provide the processes described in some of these objectives), this point was a good time to mention what these attributes were and briefly set out what they refer to.

Table 1: Crime attributes and their descriptions

Crime Attribute	Description
Type	The name given to the crime or the type of offence committed
Date	The date of when the crime took place (or failing that when the crime was recorded)
Time	The time of when the crime took place (or failing that when the crime was recorded)
Latitude	Geographic coordinate that specifies the north to south position of a location **[20]**
Longitude	Geographic coordinate that specifies the east to west position of a location **[21]**
Description	Any further relevant details of the crime

3.1.1.6 'Add Crime' Design

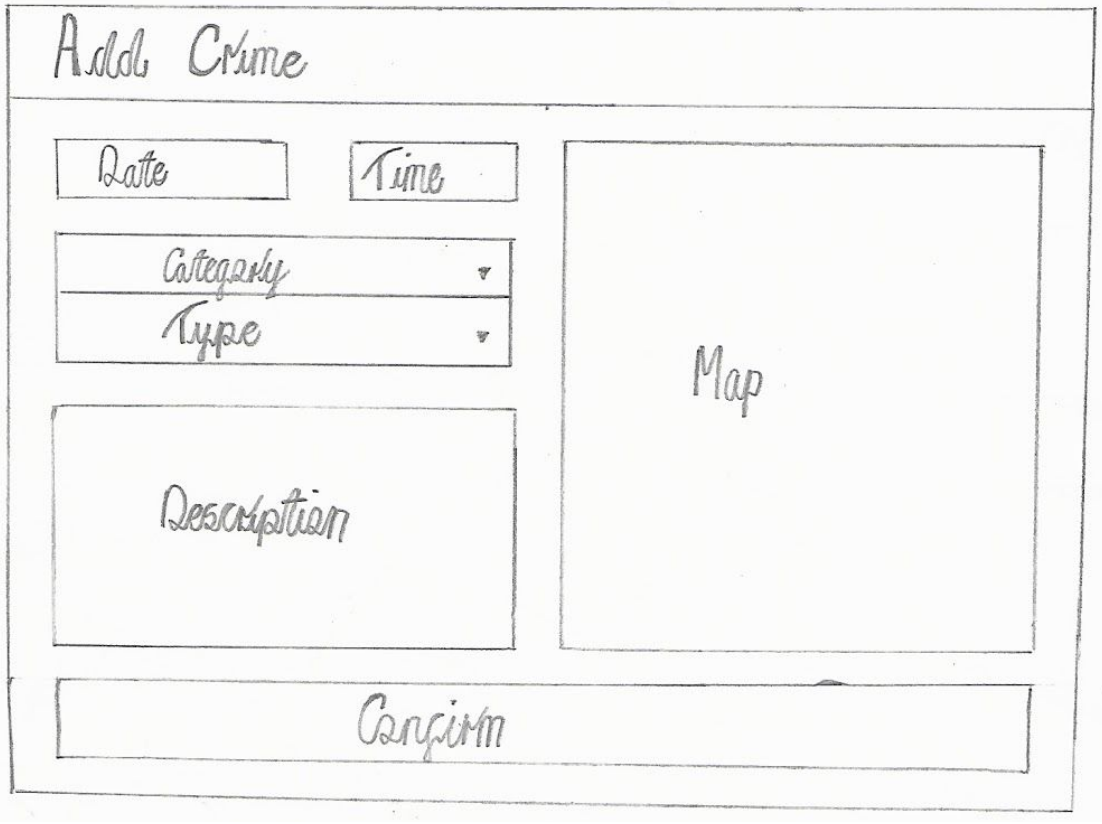


Figure 13: Low fidelity prototype for the 'Add Crime' popup window design

3.1.1.7 Mapping and Viewing Crime

The process of mapping a crime was so far designed as opening a context menu at the location of the crime and opening a pop-up window to specify additional information. But how the direct plotting of a crime and how a crime is viewed on the map, at this stage, were not yet designed and so the following sketch was devised:

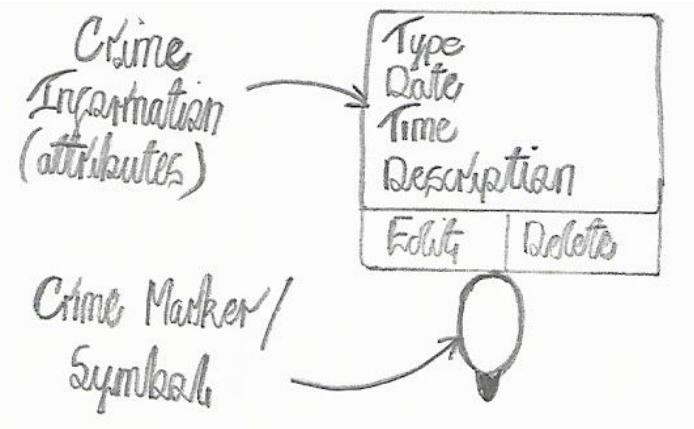


Figure 14: Design sketch for viewing a crime's attributes

This design introduced two additional options, 'Edit Crime' which was designed to open another individual popup window targeted directly at addressing the fourth objective of the project, as well as a 'Delete Crime' option ,designed to remove the symbol or marker shown to represent a crime.

3.1.1.8 'Edit Crime' Design

The popup window for this newly introduced option was designed to match with the 'Add Crime' popup window (Figure 13) as the functionality they were designed to provide was very similar:

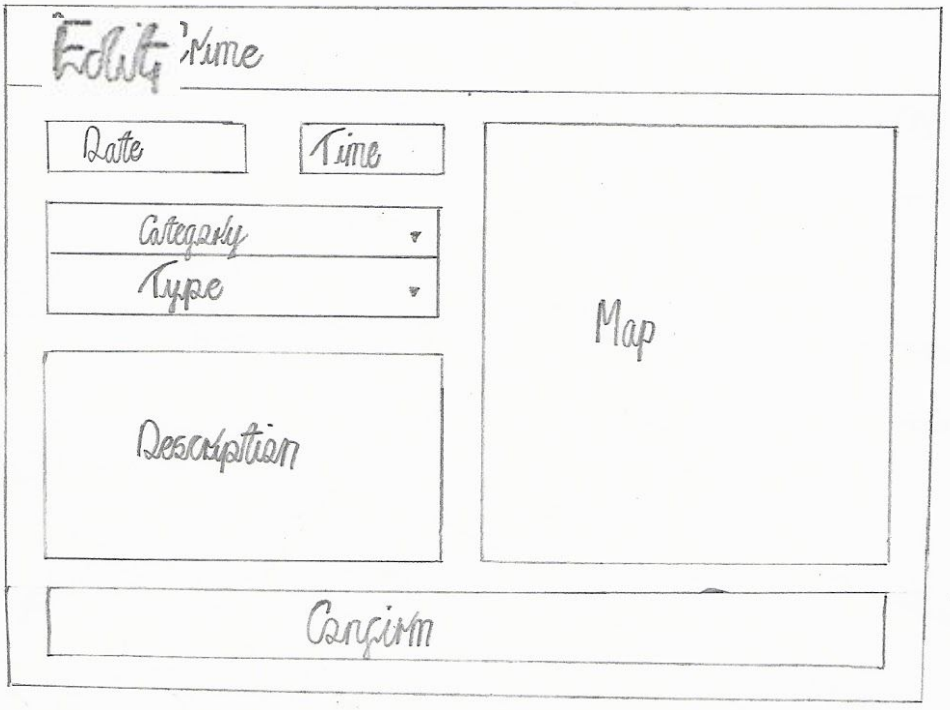


Figure 15: Low fidelity prototype for the 'Edit Crime' popup window design

3.1.1.9 ‘Filter Crime’ Design

This popup window was designed as having a comparatively larger number of inputs because multiple inputs were needed to be provided to allow for a range of values to be specified for some crime attributes. The small map was designed to be used to filter crime by location an input being provided to specify the search radius:

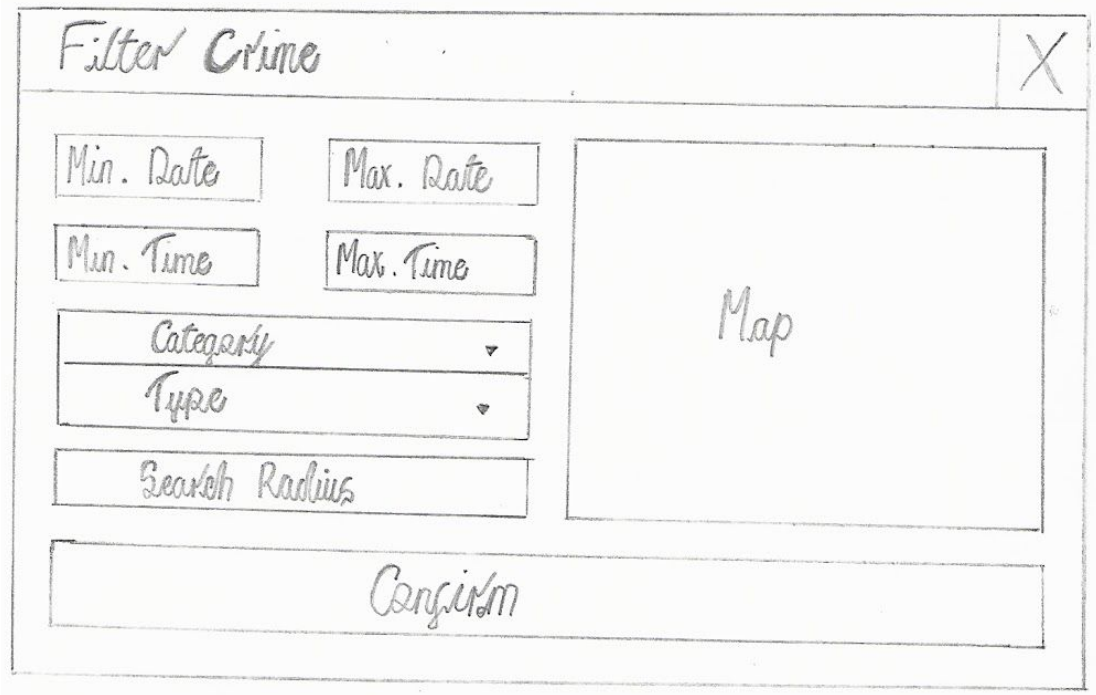


Figure 16: Low fidelity prototype for the ‘Filter Crime’ popup window design

3.1.1.10 ‘Import Crime’ Design

With the functionality of importing crimes only requiring a single input (a file), this popup window was designed as being comparatively smaller in size. A progress bar was also added because it may be beneficial to show the progress of the import functionality/process):

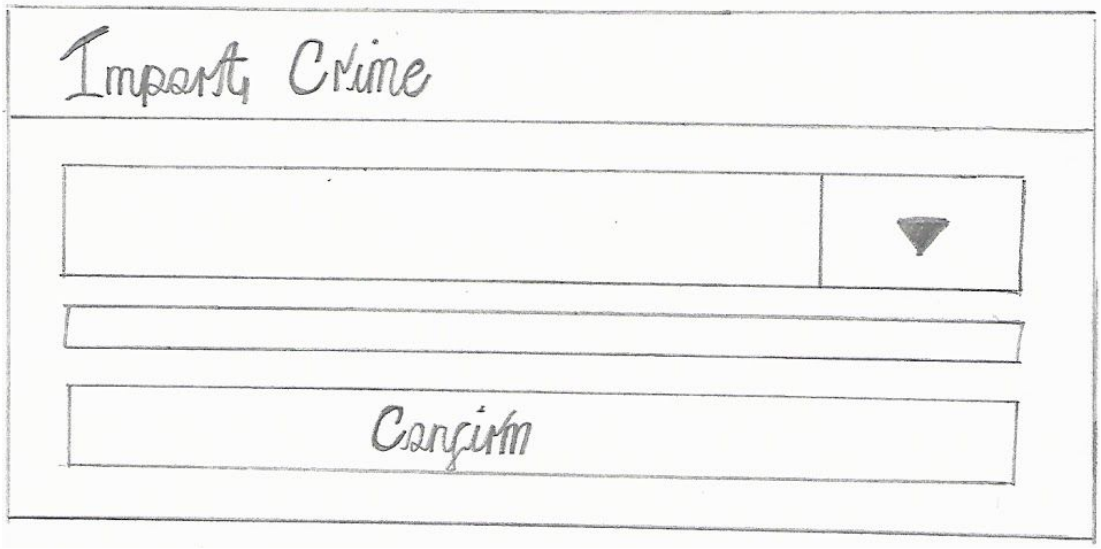


Figure 17: Low fidelity prototype for the ‘Import Crime’ popup window design

3.1.2 Database

In order for the crimes being added to the crime mapper to persist, any and all data relating to the location and information of crimes needs to be saved to an external data store. A database was identified as achieving this by allowing for crimes to be permanently stored and retrieved at will and so the next stage of design was to design a suitable database for the solution.

3.1.2.1 Conceptual Data Model

With only one entity to collect and store data for (crimes), only a single table is required, making for a simple conceptual model of the database. This table was given the name *'markers'*.

3.1.2.2 Logical Data Model

In terms of the columns present in this single table, they should reflect (in their name) the information that is entered by the user when they are adding a crime, those being the crime attributes listed in Table 1, along with any additional columns required to ensure a reliable structure for the database. An example being a primary key column which should uniquely identify each record. Therefore, the designed columns for the table were *'Crime_ID'*, *'Crime_Date'*, *'Crime_Time'*, *'Crime_Type'*, 'Latitude', *'Longitude'* and *'Description'*.

Important or notable design choices included the approach of storing locational information about the crime as two columns (namely '*Latitude*' and '*Longitude*') due to being a precise and well documented standard when working with Geographic Information Systems (such as crime mappers) as well as naming the columns relating to the crime's type, date and time, '*Crime_Type*', '*Crime_Date*' and '*Crime_Time*' respectively. Although redundant to include 'Crime' in the column's names with a Crime ID column present in the table, the reasoning behind the decision was to avoid predicted complications with reserved keywords (either in SQL when creating the table or when using other programming languages for querying/referencing the table).

3.1.2.3 Physical Data Model

The '*Crime_ID*' column was the column identified as having to be enforced as the primary key and being of an integer data type along with important constraints including being a unique value (as no two records should have the same ID) as well as not being a null value.

When deciding upon the possible column data types for the '*Crime_Date*' and '*Crime_Time*' columns, the option of having a column with a datetime or timestamp data type, a data type which combines information relating to date and time together questioned whether having separate columns for date and time was the most suitable design choice but for simplicity purposes the design was left as described with the date column having a *DATE* data type and the time column having a *TIME* data type (treated separately).

With regards to the data type for the '*Latitude*' and '*Longitude*' columns, considering that such values can have up to as many as 15 decimal places **[22]**, the *DECIMAL* data type was evidently required over more primitive numeric data types such as integer. And for the length for this data type, allowing for and storing a greater number of decimal places increases the geographic precision of detailing locations but has the side effect of increasing the storage size of the database (the response time for querying this information would also increase, potentially impacting the user experience if functionality were to depend on or require such information to be returned within a reasonable timeframe without a noticeable delay). With the eighth decimal place providing precision to within 1.1mm, a substantially greater level of precision needed to map a crime, it was decided that this would be the upper limit lengthwise with the intention being to provide the option or capability of such precision if and only when needed and that most values in need of being stored would not reach these level of precision. Although the length after the decimal point is the same for latitude and longitude, the length before the decimal point is importantly not. There is a small difference in terms of the range of values they can take with latitude ranging from -90 to +90 degrees (length of 2 digits) whereas longitude can range from -180 to 180 degrees (length of up to 3 digits) thus this revelation influenced the format for defining the lengths for the decimal data type, which is made up of two components, the total number of digits followed by the number of digits after the decimal point.

In terms of the '*Crime_Type*' and '*Description*' columns, being nominal values and of variable length, the *VARCHAR* data type was used, with it being decided that the max length (domain) should be set to 255 characters for the description and 100 characters for the crime type. Although not critical to the integrity of the database, limiting the length of user input as they are entered, to these lengths, that the database can store, would be good practice (as not enforcing and storing user input larger than the max length enforced for the database would result in any additional characters being cut off, unbeknownst to the user):

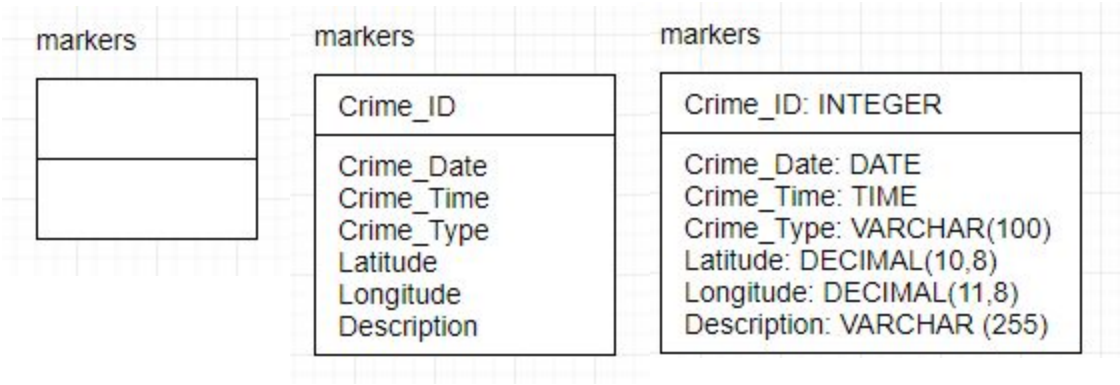


Figure 18: Conceptual, Logical and Physical data models for designed database (left to right)

3.1.2.4 DBMS and Database Administration

With the solution being web-based, MySQL was chosen to be the database management system of choice (due to the scalability and performance it offers) **[23]** and for administration of the database, phpMyAdmin was chosen to be used. The process to create the database and its necessary content was briefly outlined with the instructions that phpMyAdmin should first be used to create a database (with the name '*crime_mapper*') and then secondly used to execute constructed SQL which creates, structures and adds the designed '*marker*' table to this database.

3.1.3 Website Structure/Information Architecture

The website was chosen to be hosted and managed using cPanel with the aforementioned tool of phpMyAdmin provided through its control panel being used to host the database and the provided file manager being used to add the pages of the website onto the hosting server. Any information or data sent to the server such as user input was decided as being sent through using the client-server transmission method of AJAX.

3.1.3.1 Landing page (index.php)

With the design choice of having functionality being presented to the user with popup windows as opposed to alternative approaches such as navigating to and presenting separately designed web pages, there is a need for only a single web page that has front facing/front-end (HTML) elements or components. The interface of this web page will adopt the main user interface design (Figure 11) making use of the Bootstrap framework to aid in providing a responsive design which works across different devices and to reduce the amount of unique or custom CSS needed in development. The toolbar buttons will open the relevant popup window and the Google Maps JavaScript API was chosen to be used to implement an interactive Google Map as the mapping component for this page. This webpage was also designed to include back-end components to facilitate a connection to the designed database (Section 3.2.2). In order to connect to the MySQL server this database will be hosted on, a scripting language will be required. The scripting language decided upon was PHP therefore meaning the filename of this main webpage (the landing page) is to be '*index.php*'. This back-end functionality of connecting to the database will allow for any information in the database to be retrieved and used (such as for placing markers of all stored crime when the website is first opened). Keeping a local copy of any information retrieved would be good practice so that it can be referred to at any time, if needed for any other functionality, as opposed to costly retrieving the database's information several times:

```
1   Include 'dbConfig.php'
2   Include 'layout.css'
3
4   Display Mapping Component (Google Maps)
5
6   Query all database information
7   Locally store database information
8   Place markers on mapping component using information
9
10  Toolbar 'Filter Crime' button clicked
11      Open 'Filter Crime' popup window
12
13  Toolbar 'Import Crime' button clicked
14      Open 'Import Crime' popup window
15
16  Context Menu 'Add Crime' option clicked
17      Open 'Add Crime' popup window
```

Figure 19: Pseudocode snippet for designed index.php webpage

3.1.3.2 Page Styling (layout.css)

In order to control how elements are displayed for this front-facing webpage, CSS needs to be used, which can take the form of either inline styling, internal styling or external styling. A benefit of external styling is that a single file can be used and applied across multiple web pages (preventing the need to define the same CSS in every page) but with only one web page designed as containing HTML elements (index.php), this benefit is negated for this solution. With the popup windows designed sharing some elements that will need the same styling (for instance, the designed style and number of inputs for the 'Add Crime' and 'Edit Crime' are the same), internal styling seemed initially the most applicable form of styling. But saying this, there is still the benefit that with external styling once the user has visited the solution at least once, their browser will cache this external file, improving the loading time the next time they visit the website, as well as how having all the CSS in a separate file improves maintainability and means if any additional web pages were to be developed in the future, the styling could be quickly and easily carried over. And it is for these reasons why an external CSS file was designed to be used, which was given the filename *'layout.css'* along with using some inline styling in the main web page, if absolutely required.

3.1.3.3 Database Configuration (dbConfig.php)

Any remaining back-end functionality related to the database was implemented in separate PHP files. With each of these additional PHP files and the main landing page (index.php) requiring a connection to the database being made, instead of implementing the connection and configuration details of the database in every one of these files, a dedicated file which could be included (or referenced to, a similar concept to the external styling used) was designed instead, with the name of '*dbConfig.php'*. This file would contain information relating to the database host, the database name (which was set in Section x.x as 'crime_mapper') as well as information for a universal user which can not only query information, an action required by index.php, but that can also insert, update and delete information found in the table of this database (actions required by and that each have their own PHP file as designed in the next subsections):

```
1   Host = Name of database host
2   Username = "hq017496_test"
3   Password = "crime_mapper_2019"
4   Name = "crime_mapper"
5
6   Connect to database using configuration options
```

Figure 20: Pseudocode snippet for designed dbConfig.php webpage

3.1.3.4 Add Crime (SaveMarkers.php)

Requesting to add a crime to the map requires for a new record to be made in the database, in order for it to be permanently stored, and so therefore the related information about the crime that is entered by the user in the 'Add Crime' popup window (Figure 13) needs to be sent to the server (in addition to the map location which was specified when first opening the window) and then be added to the database by the way of an SQL INSERT statement. To keep track of which record an added crime on the map correlates to in the database, the unique ID used when adding this new record was designed to be sent back from the server (returned):

```
(index.php)
'Add Crime' popup window confirmation button is clicked
    Entered information is recorded
    Information is sent to server using AJAX

(SaveMarkers.php)
Recieve crime information from AJAX call
Use crime information in SQL INSERT statement
Send ID of inserted record back to index.php
```

Figure 21: Pseudocode snippet for designed SaveMarkers.php webpage

3.1.3.5 Edit Crime (EditMarkers.php)

If a crime were to be requested to be edited that would mean a record for that crime is already present in the database (by the way of '*SaveMarkers.php*'). As opposed to designing this record being deleted and a new record inserted, the updated information about the crime or input entered by the user in the 'Edit Crime' popup window (Figure 15) was designed to be sent to the server along with the relevant ID and be used as part of an SQL UPDATE statement to make the requested changes:

```
(index.php)
'Edit Crime' popup window confirmation button is clicked
    Entered information is recorded
    Information (and ID) is sent to server using AJAX

(EditMarkers.php)
Recieve crime information (and ID) from AJAX call
Use crime information (and ID) in SQL UPDATE statement
```

Figure 22: Pseudocode snippet for designed EditMarkers.php webpage

3.1.3.6 Delete Crime (DeleteMarkers.php)

This same concept of an existing record in the database and an ID being used to identify this record in the database, was also applied for designed how markers are deleted after having viewed a crime (Figure 14):

```
(index.php)
'Delete Crime' button is clicked
    ID is sent to server using AJAX

(DeleteMarkers.php)
Use ID in SQL DELETE statement
```

Figure 23: Pseudocode snippet for designed EditMarkers.php webpage

3.1.3.7 Import Crime (ImportMarkers.php)

With design for adding and recording a single crime with '*SaveMarkers.php*', importing crime (multiple or a large number of crimes at once) was designed as this process being scaled up and performed numerous times using information from an external format as opposed to information entered into a popup window. An objective was set out to identify the most feasible format of external data, which was identified as being text files. With the interface of the relevant 'Import Crime' popup window (Figure 17) including an input to select a file, the file to be imported is designed to be selected and then sent to the server. A PHP file, named 'ImportMarkers.php' could then read the contents of this file and use an SQL INSERT statement to add any identified crime information to the database. Just as before, the ID of every record added is returned to allow for any imported crime which is added, to be later edited or deleted:

```
(index.php)
'Import Crime' popup window confirmation button is clicked
    Send selected file to server using AJAX

(ImportMarkers.php)
Read and process received file's contents
Loop (for every identified crime record)
    Record crime information
    Use crime information in SQL INSERT statement
    Send ID of inserted record back to index.php
End
```

Figure 24: Pseudocode snippet for designed ImportMarkers.php webpage

3.1.3.8 Filter Crime

This functionality was designed as making use of a local copy of crime markers and not requiring any interaction with the database. The information entered into the 'Filter Crime' (Figure 16) popup window can be compared against this local copy with any crimes not matching the criteria being temporarily hidden from view (and not deleted from the database, as it was quickly identified that it would be both strongly unintuitive add/remove information to and from the database as and when it is filtered).

```
'Edit Crime' popup window confirmation button is clicked
    Record filter criteria

Hide markers with related information not matching this criteria
```

Figure 25: Pseudocode snippet for designed filtering of crimes

3.1.3.9 Analyse Crime and Predict Crime

The remaining actions of analysing and predicting crime are reminded as being designed to not open their own popup windows, they also don't have their own separate PHP files associated to them. The chosen and designed analysis method for crimes was clustering, but as opposed to designing and implementing an algorithm, the Google Maps MarkerClustererPlus library **[24]** can be used and applied to the crimes plotted. The analysis provided using the library was then designed to be used in conjunction with other properties of crime (not just their location) in order to predict the locations (and potential properties) of crimes that are yet to happen with these crimes being temporarily plotted (by importantly not being added to the database).

3.1.4 Website Structure Diagram

To summarise the web pages designed and how they are all interconnected to form the structure of the website along with the database, a website structure diagram was produced (for simplicity of the diagram, the index.php is not shown as including dbConfig.php and instead directly queries the database):

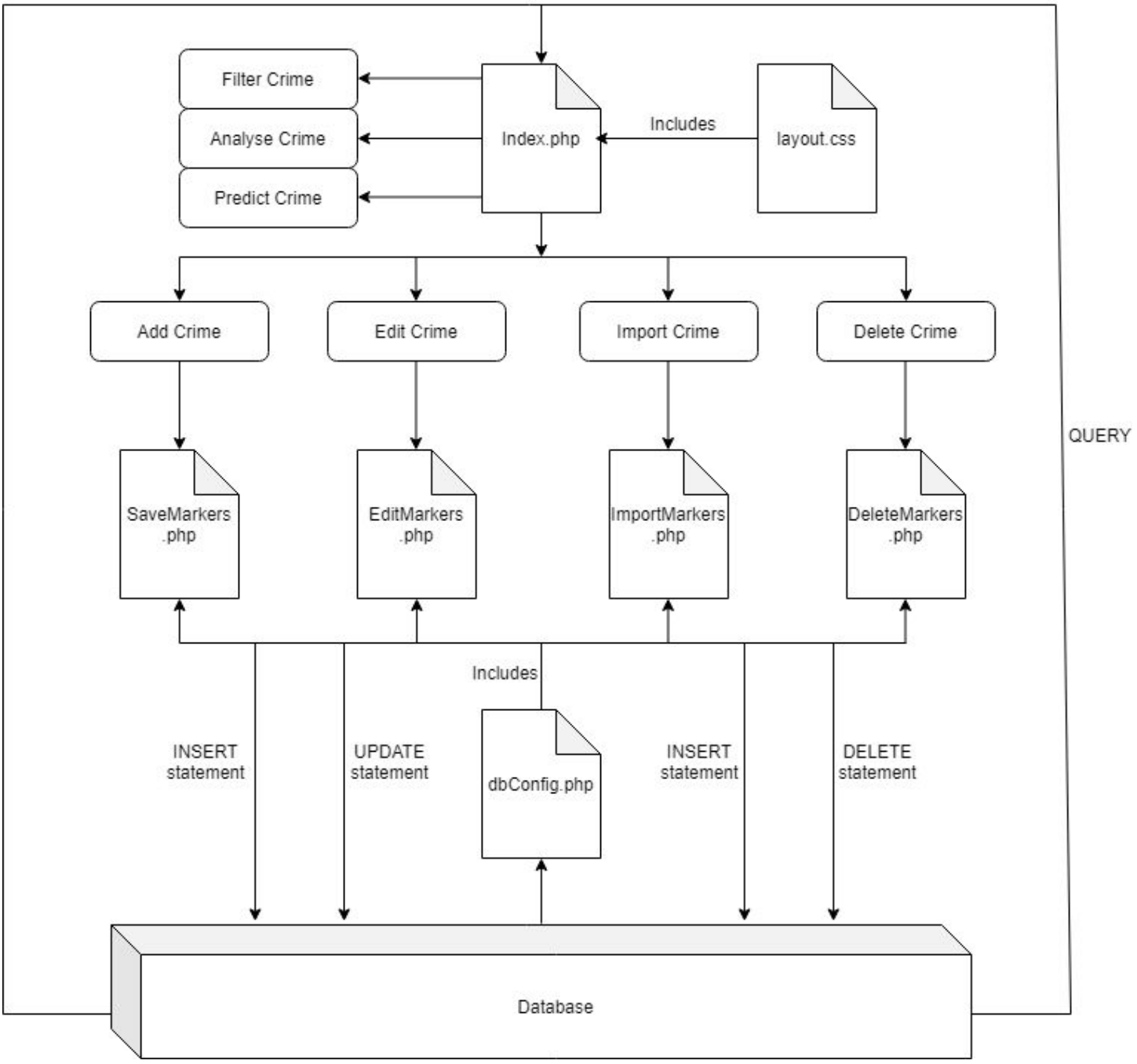


Figure 26: Website structure diagram showing relations between designed web pages and designed database

3.2 Implementation

3.2.1 Preliminaries

To build the structure of the website, the webpages outlined in the design section were created and renamed as needed, with some setup for this page being advisable in order to help ensure maintainable code was produced (which uses a consistent use of style) and the correct displaying and scaling of content.

3.2.1.1 Declaration

As the first line of the page, the version of HTML the page uses or is written with was specified using a <!DOCTYPE> tag. With the most recent version of HTML, HTML5, being the most widely recognised version by web browsers, offering the widest feature set and with it being common knowledge to only use older versions for either compatibility purposes, this was the version specified.

3.2.1.2 Root Element

Whilst the version of HTML the browser should use is stated using the above tag, the browser still needs to be instructed that the document is a HTML document, for which a <html> tag was used. This acts as the root element with any and all elements of the webpage being contained within its opening tag and closing tag.

3.2.1.3 Metadata

Within this root container, another container for any metadata, that being any data concerning the document and that isn't directly displayed as part of the webpage, was created using a <head> element. The style sheets used to control the layout and formatting of elements within the webpage were added to this container w

3.2.1.4 Content

Using <body> tags, a content container for the content of the web page was then lastly defined, acting as the location for where the majority (if not all) the remaining implementation was to be placed, finishing the setup for the main webpage:

```
<!DOCTYPE html> <!-- Declaration -->
<html> <!-- Root Element -->

<head> <!-- Metadata -->

<title>Crime Mapper</title>
<meta name="viewport" content="width=device-width, initial-scale=1">
<link rel="stylesheet" href="layout.css"> <!-- Relative Path -->
<!-- Absolute Path -->
<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/css/bootstrap.min.css">

</head>

<body>


<!-- Content -->


<!-- External Dependencies -->

<!-- Google Maps JavaScript API -->
<script src="https://maps.googleapis.com/maps/api/js?key=?" async defer></script>
<!-- Bootstrap script(s) -->
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.4.1/jquery.min.js"></script>
<script src="https://cdn.jsdelivr.net/npm/popper.js@1.16.0/dist/umd/popper.min.js"></script>
<script src="https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/js/bootstrap.min.js"></script>

</body>

</html>
```

Figure 27: Maintainable structure of index.php webpage

3.2.1.5 Database

A SQL CREATE TABLE statement was used to create the designed ‘markers’ table (Section 3.2.2) in phpMyAdmin:

```
CREATE TABLE markers (
    ID int(5) NOT NULL AUTO_INCREMENT PRIMARY KEY,
    Crime_Type varchar(100) NOT NULL,
    Crime_Date date NOT NULL,
    Crime_Time time NOT NULL,
    Latitude decimal(10,8) NOT NULL,
    Longitude decimal(11,8) NOT NULL,
    Description varchar(255)
);
```

Figure 28: Constructed SQL to create ‘*markers*’ table

3.2.2 Mapping Component

With the required API now able to be accessed, the mapping component decided upon in the design was next implemented for this newly created and setup main web page. To begin with, an area needed to be reserved for the map (within the body container) using a div element. Without any other elements currently present on the webpage, the height and width of this element was set to be the full height and width of the HTML body through using a CSS declaration involving percentage-based values of 100% for the height and width attributes in the linked to style sheet. To know which div element to style, an id selector was assigned and used, appropriately being set as 'map'. These percentage values were subject to change however, as some space needed to be reserved for when the static toolbar at the top of the page was to be implemented.

At this point in time, using HTML and CSS, the created div element was simply an unapparent and empty block. This is because a map needed to be actually added inside the div through using JavaScript and so therefore a new map object (provided with the API) was defined and given the reserved element as its container. Other required initialisation parameters for the map included a center location and a zoom level but with these bearing little significance in terms of the functionality of the map and being easily adjusted at any time in development, an arbitrary (of Reading, England) and a zoom level of 8 were specified.

Figure 29: Visualisation of a div element being used as a container for a map object

With the map being critical in providing the website's functionality, it was in need of being the first component loaded as the page loads, but with the map being dependent on its respective API, only once this API has finished loading is the map able to be displayed. To ensure the map loads and is displayed as soon as it possibly can, a small change was made to the script that loads the API. A callback feature was added, a feature which allows for a function of code to be called once the API is ready. Therefore, a function was developed, which was made to include the JavaScript code to create the map and was called:

```
17  <script>
18  function initMap() {
19      var Reading_loc = {lat: 51.454266, lng: -0.978130};
20      var map = new google.maps.Map(document.getElementById("map"),
21                                    {zoom: 8, center: Reading_loc});
22  }
23  </script>
24
25  <script src="https://maps.googleapis.com/maps/api/js?key=?&callback=initMap" async defer></script>
```

Figure 30: Callback function of map object (adapted from **[25]**)

3.2.3 Toolbar

The static toolbar designed to be positioned along the top of the webpage was implemented next using the provided navigation header of the BootStrap framework (an element named navbar). Within this navigation bar, four navigation buttons were then added (by nesting the required elements within this navbar element) with it being ensured that the order in which they were implemented is the order for which it was desired that they appear along the navbar (or as was designed).

The search bar was also added to the toolbar using an autocomplete widget, namely the SearchBox class (of the Google Maps JavaScript API). In order to construct an instance of the SearchBox class, a text input was required to be made, acting as the location for user input (a search) and for where any results (that match this search) are to be presented or returned. Therefore such a type of element was added to the navbar but importantly was implemented in order to appear centrally in the navbar, that is after the first two buttons but before the last two buttons. The width for this search bar and the widths for the navigation buttons, as of this stage, were left unspecified and so this therefore means that, in the case of the buttons, they occupied the width needed in order to display the entirety of the set text and in the case of the search bar it simply just occupied the remaining width of the screen. Following on from this, without any implemented padding or margins, there was also currently no space in between these elements, which contradicts the intended design:

Figure 31: Initial implementation of toolbar (with no styling applied)

Before implementing the functionality of the search bar, the formatting and enforcing the layout of these elements within the navbar was first implemented. The intended design exhibits the four navigational buttons being of equal width and the search bar being of a considerably larger width along with small (equal) spaces in between these buttons and between the two buttons, either side of the search bar. To meet the design, the buttons were each given (percentage modifier) widths of 16%, the search bar a width of 35% and the spaces were set as widths of 0.25% (using margins away from nearby elements), resulting in the following layout (where the widths and spaces added up to the entire screen's width of 100%):

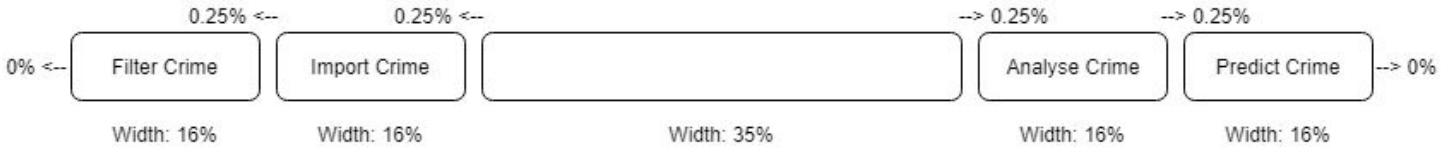


Figure 32: Revised implementation of toolbar (with styling applied)

Before moving on, the height of the mapping component was updated to accommodate for this static toolbar, otherwise the map would exceed the screen's height by being able to be scrolled down. With the static toolbar being 56 pixels in height regardless of screen resolution, the mapping component's height (or technically the container div's height) was decreased by this amount of pixels.

Going back to the search bar, the SearchBox class, as a service, inherently provides the functionality to return relevant place names based on the search made in the referenced search bar (with the predictions being returned as a dropdown list attached to the input element):

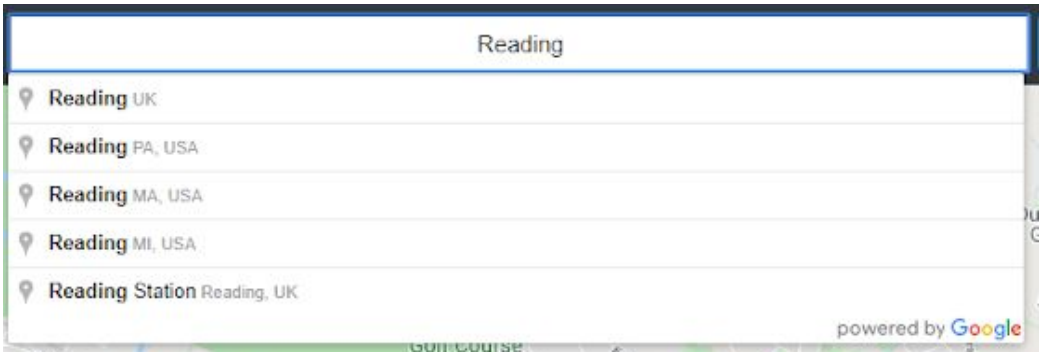


Figure 33: Implemented search bar within the toolbar (and its default functionality)

But the functionality to update the map's location based on the selection of one of these predictions still needed to be implemented. This was achieved through detecting when a predicted location is selected (using an event listener listening for a change in the place selected for the SearchBox) and then updating the bounds of the map to the geographic location the service provides for this selected location:

```
searchBox.addListener('places_changed', function() { // Selecting a prediction from the list
    var places = searchBox.getPlaces(); // Can be more than one place if using text-based geographic search

    if (places.length == 0) {
        return;
    }

    var bounds = new google.maps.LatLngBounds();
    places.forEach(function(place) {
        if (!place.geometry) {
            console.log("Returned place contains no geometry");
            return;
        }

        if (place.geometry.viewport) {
          bounds.union(place.geometry.viewport); // Geocodes
        } else {
          bounds.extend(place.geometry.location); // Location
        }
      });
      map.fitBounds(bounds); // Move map to selected location
    });

}
```

Figure 34: Code snippet for updating map to searches made (code adapted from **[26]**)

3.2.4 Placing Markers

The Google Maps JavaScript API has native Marker objects which can be constructed and added to the map component, which resemble the design in Figure 14. To reach the designed workflow of being able to place a marker at a specified location on the map with a mouse click (and a context menu being opened to bundle information along with the crime location), a divide and conquer approach was adopted for this module of code.

3.2.4.1 Predefined Marker

The first step in this approach was to simply add a marker along with the map when it is created. This was achieved by creating a new marker object and specifying the previously made map object as the map for it to be displayed on. The only other property required for a marker was the location for where on this map it is placed (similar to how the map object required a center location). With an arbitrary location already defined to center the map, this same location was reused as the location for this marker. For the marker to be shown with the map when it loads, this definition of the marker was implemented within the callback function:

```
17  <script>
18  function initMap() {
19      var Reading_loc = {lat: 51.454266, lng: -0.978130};
20      var map = new google.maps.Map(document.getElementById("map"),
21                                    {zoom: 8, center: Reading_loc});
22
23      var marker = new google.maps.Marker({
24        position: Reading_loc,
25        map: map
26      });
27  }
28  </script>
```

Figure 35: Code snippet showing a predefined marker for the map object (code adapted from **[27]**)

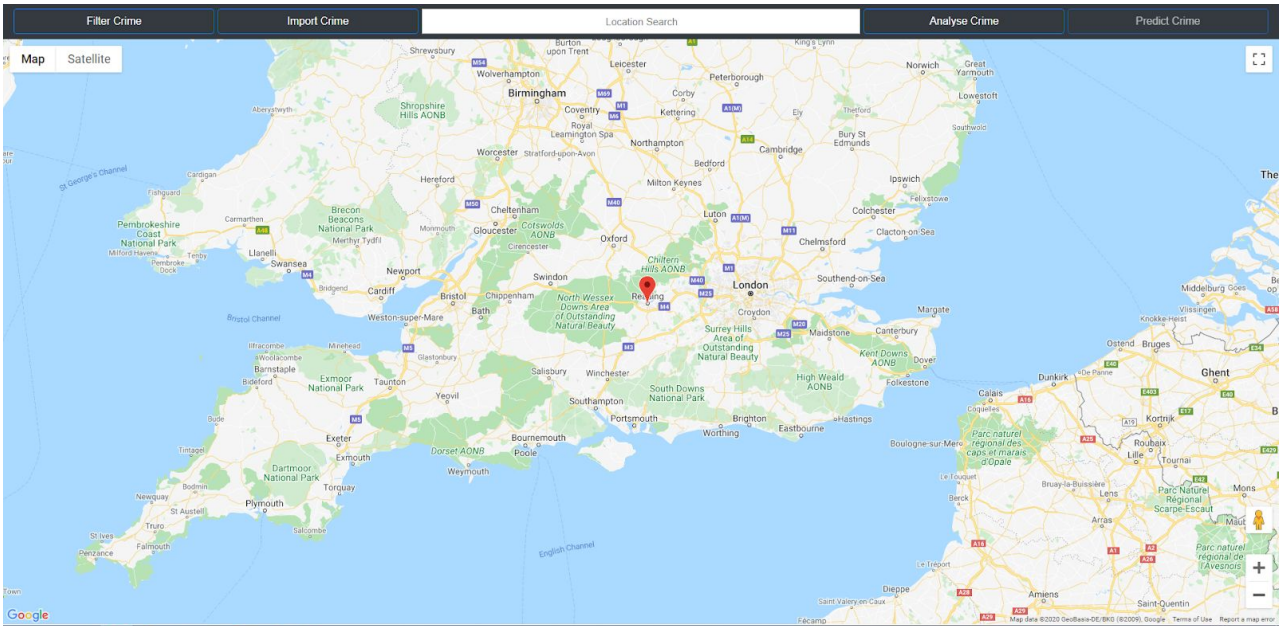


Figure 36: Result of a map with a marker at a predefined location (and implemented toolbar)

3.2.4.2 Marker at map click location

The next incremental step was to develop a marker being placed at the location of any mouse click on the map, not just at a fixed location when the map is created and loaded. The occurrence of a (left) click on the map was detected and handled using the native event listener for the map object. By listening for a left click action, when the action has been heard the latitude and longitude value of where the map has been clicked is recorded. A marker can then be defined with this recorded location:

```
33  map.addListener('click', function(e) {
34      var marker = new google.maps.Marker({
35      position: e.latLng,
36      map: map
37      });
38  });
```

Figure 37: Code snippet for adding marker to the map at a clicked location

But the design clearly states that markers would also be placed on the map when crime is imported from other formats and this import process does not involve clicking on the map. Thinking ahead to when the import functionality is to be implemented, here and now, instead of explicitly and only defining a new marker object when the map is clicked, a placeMarker() function was defined to create and display markers on the map with this function being called when the map is clicked but also being available to be called elsewhere:

```
38  function placeMarker(location, map) {
39      var marker = new google.maps.Marker({
40      position: location,
41      map: map
42      });
43  }
44
45  map.addListener('click', function(e) {
46      placeMarker(e.latLng, map);
47  });
```

Figure 38: Code snippet showing reusable placeMarker() function

3.2.4.3 Context Menu

In order to prevent undesired addition of markers (made by accidental clicks or slips onto the map), instead of immediately placing a marker, the designed context menu (Figure 11) was next implemented as opening with a right click on the map. The HTML (elements) needed to create this context menu were first developed with the entire context menu being derived with a div element which was styled, as required, in the external CSS file):

Figure 39: Result of developed div element (context menu)

With the context menu only designed as appearing dynamically (it is not designed to be shown all of the time) this div element was importantly not made initially visible and and so to begin with it was hidden using CSS (the display property of the element) and positioned off the visible bounds of the page (as an additional precautionary measure). When the context menu is requested with a right click on the map, the context menu is made to be shown by unhiding the element and moving its position to where the screen was clicked:

```
65  map.addListener('rightclick', function(e) { // Right click on map
66      var Location = e.latLng; // Create latLng object of click location
67      for (prop in e) {
68          if (e[prop] instanceof MouseEvent) { // Also record click location in terms of pixels
69              mouseEvt = e[prop];
70              var left = mouseEvt.clientX;
71              var top = mouseEvt.clientY;
72
73              ContextMenu = document.getElementById("menu");
74              ContextMenu.style.left = (left+1) + "px"; // Position context menu at click location
75              ContextMenu.style.top = (top-1) + "px";
76              ContextMenu.style.display = "block"; // Unhide context menu
77          }
78      }
79  });
```

Figure 40: Code snippet for dynamically requesting context menu

Whilst the click location is recorded by screen pixels for the purpose of positioning the context menu, the click location in terms of latitude and longitude (just as with previous subsection) must also be kept track of as this is where a marker will need to be placed:

```
81  const add_btn = document.getElementById("btn_add"); // 'Add crime' button (in context menu)
82      add_btn.addEventListener('click', event => {
83          placeMarker(Location, map); // Place a marker with latLng object
84      });
```

Figure 41: Code snippet for a marker being placed at the recorded location

3.2.4.4 Pop-up Window

The penultimate step for this module of code, involved the designed ‘Add Crime’ pop-up window being made to appear when the context menu option is selected and before the marker is placed, as a way to bundle any information entered along with the marker and to act as an additional form of confirmation.

Using BootStrap’s JavaScript modal plugin, this pop-up window was created with the inputs that make up this window being placed in the body of the modal. Looking back to the design of the popup window (Figure 13) these are inputs for a date value and a time value which are standard types of HTML input elements, two dropdown lists, that resemble HTML select elements, for the category of crime and type of crime, an input to enter a description which a textarea element can provide, as well as a smaller map (another div element and map object) to make small adjustments to the marker’s location and a confirmation button (button element).

In terms of the two select elements, with the selection of the category of crime envisioned as narrowing down the possible types of crime that can be selected (categorising them), the first select element used for selecting the category of crime was implemented as dynamically changing the selectable options of the second select element (after assigning the desired categories of crime to the first list as options):

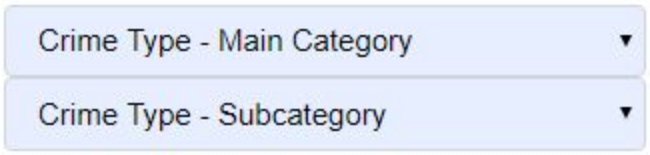


Figure 42: Select elements developed as inputs for the category and type of crime

This involved detecting when the value of this first list is changed and then determining which value it had been changed to. Based on this value, the options of the second list were then assigned, making sure this assignment replaces the options provided and doesn’t append them (otherwise the list would begin to show crime types or options from multiple categories of crime):

```
90  $("#Add_Crime_Type").change(function() { // When main category is changed
91      $('#Add_Crime_Type_sub option:not(:first)').remove(); // Remove all but the default hidden value
92      var el = $(this);
93
94      if(el.val() === "Violence against the person") { // Check which main category was chosen
95          AddOptions(add_sub_select,violence_sub_options); // Update sub categroy accordingly
96      }
97      else if (el.val() === "Public Order") { // ...
```

Figure 43 (above): Code snippet showing addition of select options

And in terms of the smaller map (shown right), that is implemented as having a draggable marker placed at the same location as where the map was originally clicked on to open the context menu (and then the popup window), which if and when moved will update the position used when placing the marker.

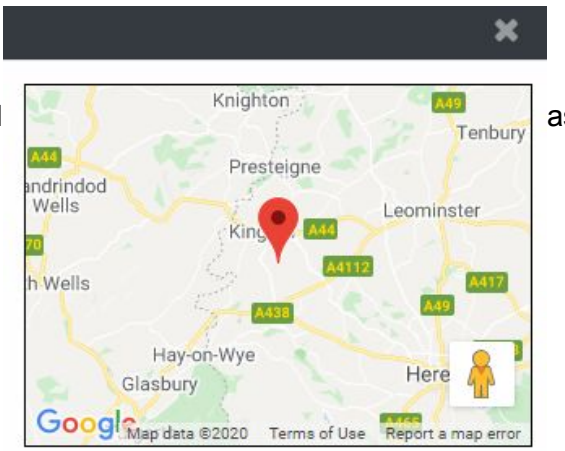


Figure 44 (right): Smaller map implemented for adjustments

This smaller map and draggable marker are defined in almost exactly the same way as the main larger map and the markers used in earlier stages of this code module were:

```
101  var map2 = new google.maps.Map(document.getElementById("map2"), SmallMapOptions); // Show smaller map
102
103 var Draggable_marker = new google.maps.Marker({ // Add a single draggable marker to smaller map
104      position: Location, // At location of click location
105      draggable: true,
106      map: map2 // On this smaller map, not the main map
107  });
```

Figure 45: Code snippet for a draggable marker being placed on the provided smaller map

All the popup window's inputs were nested inside a form element to accommodate for their values to be later sent to the server when needed, as well as the allowed values that can be entered into these elements being specified and enforced where possible (validation). But at this stage, after repurposing the click event of the context menu to now just open the popup window (and no longer place a marker), a marker was implemented as being placed on the map when the popup window's confirmation button is clicked, at either the original location (if no adjustment is made) or at the location of the draggable marker (if an adjustment is made):

```
109 google.maps.event.addListener(Draggable_marker, 'dragend', function (evt) {
110      Location = evt.latLng; // Adjust (update) position
111  });
112
113 $("#add_submit_form").submit(function(e) { // Confirmation button is clicked
114      placeMarker(Location,map); // Place a marker on the main map
115  }
```

Figure 46: Code snippet for a marker being added at a specified (and adjustable) location

However, this development didn't yet bundle the other information (the crime attributes) with the marker and so to associate information with the marker objects, some properties were assigned to them. With the only way to create marker objects being to use the placeMarker() function, this function needs to be added to, in order to support markers having properties:

```
38  function placeMarker(Crime_Type,Crime_Date,Crime_Time,Description,Location,map) {
39      var marker = new google.maps.Marker({
40          Crime_Type: Crime_Type,
41          Crime_Date: Crime_Date,
42          Crime_Time: Crime_Time,
43          Description: Description,
44          position: Location,
45          map: map
46          });
47  }
```

Figure 47: Code snippet for the placeMarker() function being updated to accommodate marker properties

The values of the inputs in the popup window, when the confirmation button is pressed, were implemented as being to these properties, using this newly updated function.

3.2.4.5 Storing Markers

The last stage for this module of code, of adding crime to the mapper, was to store the information of added crimes or the markers and their properties, which are now one and the same, to the database. In other words, the information used to set the properties of the markers needed now to be also stored to the database. The data is sent by being bundled together by taking the result of serializing the form (which surrounds the inputs of the popup window) and combining it with the latitude and longitude values (due to the map not being included in this form). This complete bundle of data is then sent to the server using an AJAX call, targeted at being handled by the '*SaveMarkers.php*' file:

```
114 $("#add_submit_form").submit(function(e) { // Confirmation button is clicked
115     /* Send to database */
116     var formData = $("#add_submit_form").serialize();
117 
118     var Vars = {Latitude: Latitude, Longitude: Longitude};
119     var varsData = $.param(Vars);
120 
121     var data = formData + '&' + varsData;
122 
123     $.ajax({
124             url: 'SaveMarkers.php',
125             type: 'POST',
126             data: data,
127     });
128  }
```

Figure 48: Code snippet of AJAX call sending marker information to server

In this PHP file, all the values sent in the data bundle are checked to see if they have been received successfully, server side. In the case of the serialised form (and all values within it) and the latitude and longitude values all being successfully received, a query is made to the connected database with this query including an INSERT SQL statement that uses these received (or sent) values. If this query successfully executes, the ID used when inserting this row into the database is returned:

```
31  // ... Check arrival of other information
32 
33  if(isset($_POST['Longitude'])) // If received
34 {
35      $longitude = $_POST['Longitude'];
36  }
37 
38  // Insert information into database
39 $sql = "INSERT INTO markers (Crime_Type, Crime_Date, Crime_Time, Description, Latitude, Longitude)
40         VALUES ('$crime_type', '$date', '$time', '$description', '$latitude', '$longitude')";
41  $db->query($sql);
42 
43  $id= mysqli_insert_id($db);
44  echo $id;
45  ?>
```

Figure 49: Code snippet of SQL INSERT statement used to store the marker information in the database

A final implementation change for this module included ensuring that a marker is only ever placed on the map when the associated query for it is known to have been successful (i.e that marker is now stored in the database). This was implemented by working within the success function of the implemented AJAX call (Figure 48), but what is important to note is that this function is called on the condition that the transmission of data to the server has been successful and importantly would still be called even if the query to add the information to the database was unsuccessful. And so within this function, an additional check that an ID has been returned from the PHP file was made (as an ID will only ever be returned if a record has been inserted). If this further check is met, the marker is then placed, but with one additional property as compared to before, this returned unique ID, acting as the keeping of a reference to the record the marker correlates to in the database:

```
123    $.ajax({
124            url: 'SaveMarkers.php',
125            type: 'POST',
126            data: data,
127            success: function(result) // Successful data transmission
128            {
129                if (result) { // Successful insert
130                    placeMarker(result,Crime_Type,Crime_Date,Crime_Time,Description,Location,map);
131                    // Add crime marker with additional ID property
132                }
133            }
134    });
```

Figure 50 Code snippet of marker being placed only through success of the AJAX call

3.2.5 Loading Markers

With implementation to store added markers to the database now added, retrieving and displaying the currently stored markers, each time the website is opened, was next implemented. This was implemented using PHP and a SQL query from within the main webpage (a separate PHP file is not required). The query involved the entire contents of the database being retrieved using the SQL * operator and stored to a data structure (a JS array), value by value and row by row:

```
149    var markers = [ // Array holding parsed result
150        <?php
151        $result = $db->query("SELECT * FROM markers"); // Returns output of statement
152        if($result->num_rows > 0){
153            while($row = $result->fetch_assoc()){
154                echo '['.$row['ID'].',"'.$row['Crime_Type'].'","'.$row['Crime_Date'].'","'
155                    .$row['Crime_Time'].'","'.$row['Description'].'",'.$row['Latitude'].','
156                    .$row['Longitude'].'],';
157            }
158        }
159        ?>
160    ];
```

Figure 51: Code snippet of database contents being retrieved and parsed

The array which is now populated with the contents of the database (the array is used as a convenient way of parsing and handling the data) is then looped over with its values being used to place markers:

```
162     for( i = 0; i < markers.length; i++ ) {
163     // For each row index (or information relating to each record in the database)
164         var ID = markers[i][0];
165         var Crime_Type = markers[i][1];
166         var Crime_Date = markers[i][2];
167         var Crime_Time = markers[i][3];
168         var Description = markers[i][4];
169         var Point = new google.maps.LatLng(markers[i][5], markers[i][6]);
170         placeMarker(ID,Crime_Type,Crime_Date,Crime_Time,Description,Point,map);
171     }
```

Figure 52: Code snippet of markers being placed with information in populated array

3.2.6 Viewing Markers (InfoWindows)

All markers, whether that be markers loaded from the database or newly added markers, have properties associated with them. Whilst these are available to be referenced and used in code, they were currently not shown or displayed in any way at this point. Therefore, development moved to implementing a way of presenting the properties for a marker when they are requested to be viewed.

This was achieved by creating an InfoWindow object (a native object of the Google Maps library) for every marker as it is being placed and the created object for a marker being shown when that respective marker is clicked on. These objects are a type of overlay and resemble small windows or information panels which can be shown above (overlaid on top of) the map.

The content of these InfoWindows determine what is shown within them and so it was implemented that their content are set as the property values of the markers along with other elements such as text elements (sub-headings stating the names of these properties), breaks (used to format and space the content) and buttons (for future designed functionality):

```
300     marker.info = new google.maps.InfoWindow({
301         content: '<div id="iw-container">' + '<div class="iw-content">' +
302                 '<b>ID: </b>' + marker.ID +
303                 '<br> <b style="word-wrap: break-word;">Crime Type: </b>' + marker.Crime_Type +
304                 '<br> <b>Date: </b>' + MarkerDate +
305                 '<br><b>Time: </b>' + MarkerTime +
306                 '<br></br>' + '<i style="word-wrap: break-word;">' + marker.Description + '</i>' +
307                 '<br></br> <button>Edit</button>' + '<button>Delete</button>' + '</div>' + '</div>'
308     });
```

Figure 53: Code snippet of creating InfoWindow objects for markers (code adapted from **[28]**)

Whilst most of these properties could be mapped directly and displayed as they were, the date and time properties needed to be manipulated before being able to be consistently and correctly displayed. In the database, dates are stored in the format 'MM-DD-YYYY' but when a marker is added using the 'Add Crime' pop-up window, the date is in the format 'DD-MM-YYYY' (this is the default format of a HTML input date element). In order to reflect the design choices that had already been made in other parts of the solution and to ensure consistent standards, this later format was the format decided to be shown universally, with the conversion being achieved using an external library (moment.js) **[29]**, which provides inbuilt functions to convert dates between formats. To highlight the importance of this change, without this conversion being made, a marker loaded from the database and a marker that has been newly added using the pop-up window would have different or conflicting date formats. A similar mismatch occurs with the time property, time values stored and retrieved from the database include seconds with the format 'HH:MM:SS', whereas times entered in the pop-up window do not, they are of the format 'HH:MM'. Therefore any marker which has a time property of length 8 (the length of the 'HH:MM:SS' format including the colons) is trimmed by 3 characters, that is, the seconds and the accompanying colon are discarded before being displayed in the InfoWindow:

```
277     var MarkerDate = moment(marker.Crime_Date).format("DD-MM-YYYY"); // Convert to UK format
278
279     var MarkerTime = marker.Crime_Time;
280     if (MarkerTime.length == 8) { // If time is retirved form database which includes seconds
281         MarkerTime = MarkerTime.substring(0, MarkerTime.length - 3); // Remove the seconds for display purposes
282     }
```

Figure 54: Code snippet of converting the date and time properties of markers for display purposes

Whilst, the InfoWindows are created for markers when they are placed, in order to be shown these InfoWindows need to be actually opened. This was achieved by using the native event listener for a click on a marker and implementing it so that when the action has been heard, the InfoWindow for the marker is opened:

```
310     google.maps.event.addListener(marker, 'click', function() {
311         marker.info.open(map,marker);
312     }
```

Figure 55: Code snippet of opening the created InfoWindow objects for markers

As the last addition to this module of code, although the 'Crime Type' property is explicitly shown in the InfoWindow, it was also intuitively implemented as showing when the marker is hovered over:

Figure 56: The crime type property being shown on hover of a marker

This is achieved by assigning this 'Crime Type' property as the value for the marker's 'title' property, a reserved marker property which provides the described and shown hover functionality,

```
264     marker.title = marker.Crime_Type; // Shown on hover
```

Figure 57: Code snippet of assigning title property for markers

3.2.7 Deleting Markers

The functionality provided by these recently implemented InfoWindow objects was next extended by making use of the buttons placed inside them. One of these buttons is labelled by the text ‘Delete’ and so when this button is pressed, it should be implemented so that the associated marker (the marker clicked on to open the InfoWindow) is deleted from the crime mapper.

At first, the implementation of this functionality was considered trivial, the marker could just be hidden from view. But of course this marker is still stored in the database and so when the webpage is opened again, this marker would re-appear as it would be loaded back (Section 3.3.6). The marker, even without the webpage being reopened, would also still be present in the local copy of storing markers which is intended to be used to ease the implementation of functionality yet to be implemented (such as the filtering of markers). Therefore, when a marker is to be deleted, it should as initially identified be deleted from view, but it should also be deleted from the local copy (an array) and even more importantly be deleted from the database.

These different levels of deletion are implemented within a DeleteMarker() function which was implemented as being called when the delete button within the InfoWindow is pressed:

```
// ... Other content of InfoWindow
'<button onclick=DeleteMarker('+marker.ID+')>Delete</button>' +
// ... Other content of InfoWindow
```

Figure 58: Code snippet of click function for ‘Delete’ button

This function has a single parameter, providing a reference to the marker that is to be deleted, in the form of the ID (property) of the marker. Using this reference, both the actual marker object and the corresponding record in the database are able to be identified and deleted.

3.2.7.1 View

The marker to delete can be hidden from view using the native setVisible() method for marker objects with the parameter of false. However, whilst this ensures the marker is hidden from view, the associated InfoWindow is not covered by this implementation and would remain behind and so preceding the marker being hidden, the InfoWindow should also be hidden, by being closed.

3.2.7.2 Array

When the marker object is being retrieved from the local array using the ID property, the index in the array that corresponds to the relevant marker is recorded. The element at this index in the array is then removed to remove the marker from the local array.

3.2.7.3 Database

Deleting the record of the marker in the database was implemented using an AJAX call and a separate PHP file. The AJAX call sends the ID of the marker to delete to the server which is handled by the PHP file and goes on to be used in a DELETE SQL statement which deletes the entire row that starts with the received ID.

The deletion process was implemented as treating the deletion of the database with the highest precedence, with the marker only be deleted at the view and array deletion levels if and only when the marker has been successfully deleted from the database:

```
590     function DeleteMarker(ID) {
591         for(i = 0; i < MarkerArray.length; i++){ // Identify marker
592             if (MarkerArray[i].ID == ID) {
593                 var MarkerToDelete = MarkerArray[i]; // Marker Object
594                 var index = i; // Index in array
595             }
596 
597         MarkerToDelete.info.close(); // Close InfoWindow
598 
599         $.ajax({
600             // (Database)
601             url: 'DeleteMarker.php',
602             type: 'POST',
603             data: {ID: ID},
604             success: function()
605             {
606                 // (View)
607                 MarkerToDelete.setVisible(false);
608                 // (Array)
609                 if (index !== -1) {MarkerArray.splice(index, 1)}
610             }
611         });
612     }
```

Figure 59: Code snippet of DeleteMarker() function, the onclick function of the ‘Delete’ button

The code to delete a marker at this database level was implemented within the ‘*DeleteMarkers.php*’ file which only deletes a record when successfully provided with an ID:

```
1   <?php
2   require 'dbConfig.php';
3 
4   if(isset($_POST['ID'])) // Check arrival
5   {
6       $ID = $_POST['ID'];
7   }
8 
9   // Delete marker from database
10  $sql = "DELETE FROM markers WHERE id = $ID";
11  $db->query($sql);
12  ?>
```

Figure 60: Code snippet of deleting a marker (record) from the database

3.2.8 Editing Markers

The InfoWindows were further extended to also include functionality to edit markers using the other button placed inside these InfoWindows by implementing that when this particular button is pressed, the ‘Edit Crime’ popup window appears, with the value of its inputs being initially set to the marker in question’s current properties (or crime attributes), but being able to be changed and these changes then being reflected at an array level and database level (Section 3.3.8), once the popup window’s confirmation button is clicked.

3.2.8.1 Setting current values

For the ‘Edit Crime’ popup window, the current properties of the marker (requested to be edited) are implemented as being set to the relevant inputs as soon the popup window becomes shown (as opposed to these inputs being initially set as having empty values in need of being entered). A simple assignment of these input’s values to their corresponding property suffices for most of the inputs:

```
452     $('#Edit_Crime_Type_sub').val(MarkerToEdit.Crime_Type).change();
453     $('#Edit_Date').val(MarkerToEdit.Crime_Date);
454     $('#Edit_Time').val(MarkerToEdit.Crime_Time.substring(0,5));
455     $('#Edit_Description').val(MarkerToEdit.Description);
```

Figure 61: Code snippet of assigning inputs of ‘Edit Crime’ popup window to marker properties

But for one of the select elements, this approach was not entirely applicable. Whilst the specific type of the crime is a property of a marker and can be assigned directly in this way, the above select element of the category of crime is not and so the set value for this input must be somehow derived from the crime type. As subtly alluded to with the implementation of the ‘Add Crime’ modal, when a value for the main category of crime is chosen, the list of available options is dynamically updated for the subcategory. This is achieved using a number of arrays holding the values of selectable options for these elements, one array holds all the main categories and is assigned as the options for the main category element and numerous other arrays hold the sub-options for each of these main options. And so therefore, whichever of these subsequent arrays which includes the subtype of the crime (the property), indicates and can be used to retrace the main category that should be set:

```
394         if (violence_sub_options.includes(MarkerToEdit.Crime_Type) === true) {
395             $('#Edit_Crime_Type').val('Violence against the person').change();
396         }
397         // ... Checking other arrays
```

Figure 62: Code snippet of assigning category of crime to relevant input in ‘Edit Crime’ popup window

3.2.8.2 Updating values

When the confirmation button of the implemented ‘Edit Crime’ modal is pressed, an almost identical AJAX call, in terms of the data that is sent, to that used for adding/storing markers is implemented, with the small addition of also sending the ID property within the data bundle being sent to the server (as there exists a record of the marker already in the database which needs to be updated as opposed to a new record being made) was implemented. The only other differences being a change in the destination (to ‘EditMarkers.php) which instead includes use of an UPDATE SQL statement that sets the values for the record (which has the ID which was sent) to the newly entered values in the popup window.

3.2.9 Filtering Markers

At prior stages of implementation, the importance of keeping a local copy of markers has been repeatedly highlighted and it is implementing the functionality for filtering markers where the reason(s) for doing this become apparent, it holds information which can be consulted to filter the markers by their properties.

To begin with, the development for this module of code involved the designed ‘Filter Crime’ popup window (Figure 16) and within it any necessary inputs that allow the filter criteria to be specified, being implemented. With this popup window in place, it was implemented that when the confirmation button of this popup window is clicked, the entered filter criteria becomes applied to all current markers with the markers meeting the criteria being kept shown and those that do not, being hidden.

3.2.9.1 Filter Criteria and Validation

As a good starting point in making sense of the specified filter criteria, the inputs in the popup window which have been specified and those which have been left unspecified, were identified. This makes it known which of the properties (that relate to the inputs) should be included in the filtering process and which properties shouldn’t be (that should be left out or not included in the filtering process). The fields with an option to filter by all, namely the category of crime and type of crime fields, if set to this value, can also be treated as not being a part of the filtering process. These values and other values are recorded and then before implementing the direct process of filtering, the combination of values (or lack of values) that have been entered are verified, to ensure they do not meet the any of the following invalid conditions (in which case the filtering criteria is unable be applied):

- ❏ Values being specified for both the minimum date and maximum date fields, but the minimum date is a date chronologically after the maximum date
- ❏ Values are specified for both the minimum time and maximum time fields, but the minimum time is a time after the maximum time (chronologically from 00:00 to 23:59)
- ❏ A value is only specified for one out of the two time fields (one date value is valid because dates are contiguous whereas one time value is invalid because times are recurring)

3.2.9.2 Marker Visibility

If all the search criteria have been determined as valid, the visibility of the markers is then altered based on the search criteria. But first, any previous filters which may be currently applied to the markers must be removed in the way of ensuring that every marker is made visible. The criteria can then be compared with the properties of each and every marker, with a marker being hidden if it does not meet all aspects of the criteria (one or more properties do not match with the criteria):

```
513     if (isMinDate == true) { // If a minimum date was entered
514         if (MarkerDate < minDate) {
515         // And the marker's date is before than that date
516             HideMarker(MarkerArray[i]);
517         }
518     }
```

Figure 63: Code snippet of filtering markers by a specified minimum date

3.2.10 Analysing Markers

The analysis of the markers was implemented by using a MarkerClusterer object (an object within the additional MarkerClustererPlus library) which was given the local array of markers as a parameter for clustering. The object was set to ignore hidden markers (such as the markers implemented as being hidden when they don't meet a specified filter criteria). Other implementation details include the object being toggled between being enabled and disabled whenever the relevant toolbar is clicked and the icons used to display the different levels of clustering being directly specified to those which are desired to be shown:

```
1537    var markerCluster = new MarkerClusterer(null, MarkerArray, mcOptions);
1538    markerCluster.setIgnoreHidden(true);
1539
1540    var Cluster_Active = false; // Clusterer initialised as unactive
1541
1542    const btn_analyse = document.getElementById("btn_analyse"); // 'Analyse' button
1543    btn_analyse.addEventListener('click', event => {
1544        if (Cluster_Active == true) { // If active and button was pressed
1545            markerCluster.setMap(null); // Hide clusterer
1546            Cluster_Active = false; // Alternate variable
1547        }
1548        else {
1549            markerCluster.addMarkers(MarkerArray); // Update markers to cluster
1550            markerCluster.setMap(map);
1551            markerCluster.repaint(); // Redraw and show clusterer
1552            Cluster_Active = true;
1553        }
1554
1555    });
```

Figure 64: Code snippet of implemented MarkerClusterer (code adapted from **[30]**)

3.2.11 Importing Markers

Just as with most other modules of code, creating the related popup window was usually first on the agenda and this was no different for this last main module of code, the designed ‘Import Crime’ popup window (Figure 17) was built and implemented as appearing when the relevant button from the main toolbar is selected. The file input added as part of this window is central to the import functionality that is to be developed.

3.2.11.1 File Selection and Validation

The file input, by default, allows for a file of any format to be selected from local storage which contradicts the design choice of only allowing text files to be imported. The types of text file formats available are numerous and accommodating for importing information from all of them would lengthen development significantly and so to still meet the last objective set out (Section 1.2), the commonly used delimited text file format of CSV files, was chosen to be the only accepted format. The file input was therefore next set to only show .csv files. However, when using it to browse for files, there is nothing to prevent this setting (it is an option of the file input) being changed to show all files again and an alternative file format being chosen. This meant that further checking that the selected file is a CSV file, was required. The extension of the selected file’s filename was evaluated to do this but similarly came back as ineffective with it being possible to spoof the extension (e.g renaming an image file as ending with the .csv extension) and so instead the MIME type of the file was checked, which did prove to be effective.

Once the file type is verified, the file is then opened and the first line is read client side (using JavaScript). The first line of a delimited text file is typically a declaration of the column headers used in the file. The values in this first line are then compared against lists of the expected and acceptable column headers for importing crime. These resemble the crime attributes but some leeway is given with equivocal terms (as well as very small differences such as different capitalisation) also being accepted. When a match is found, the index of the column is recorded along with the attribute the column represents, which therefore accommodates for the column headers to be stated or appearing in any order. But in order for the file’s contents to be considered valid and go on to be imported into the mapper, only two of the crime attributes (‘*Latitude*’ and ‘*Longitude*’) are considered as being required, the others are considered as being optional. This is because a crime is still able to be imported with knowledge of its location and it can just be that there will be no other information related to it. With the bare minimum of these two required columns in the file, the file is then sent to the server using an AJAX call with the progress of this transmission being displayed using a progress bar:

```
633     formdata = new FormData();
634     formdata.append("fileToUpload", file); // File to send
635
636     $.ajax({
637             url: 'ImportMarkers.php',
638             type: 'POST',
639             data: formdata,
640             success: function()
641             {
642                 // Update progress bar
643                 $("#progress_file_upload").css("width", "100%")
644                 .text("File Upload (Complete)");
645             }
646      });
```

Figure 65: Code snippet of AJAX call sending selected file to server

3.2.11.2 Import Process

The PHP file targeted by the AJAX call (Figure 65), '*ImportMarkers.php*', receives the sent file and opens it for reading. The content read is then handled by storing it to an array (in just the same way as information retrieved from the database is, see Section 3.3.6). With the indexes of columns and the type of crime attribute these columns are expected to hold identified, it was then implemented that for each line, the stored values are retrieved from the array and are are extensively validated through checking the inferred type of the value (e.g check whether the values under a header relating to the crime attribute of a date actually represent a date) as well as, in the case of the the required crime attributes, the values themselves being checked as being within their valid ranges (e.g. Latitude between -90 and 90):

```
82      for ($j = 1; $j < $num_rows; $j++)
83      // For every line after column headers
84      {
85          // If a latitude column header was found
86          if ($Latitude_index != 1) {
87              // If value is found
88              if (isset($csvAsArray[$j][$Latitude_index])) {
89                  $latitudeRead = $csvAsArray[$j][$Latitude_index];
90              }
91              if (is_numeric($latitudeRead)) {
92                  // If between valid range
93                  if ($latitudeRead >= -90 && $latitudeRead <= 90) {
94                      $latitudeToSend = $latitudeRead;
95                      $validLatitude = 1; // Flag present latitude
96                  }
97              }
98          }
99          // ... Other atributes
100     }
```

Figure 66: Code snippet of the latitude values for every row in a selected file being processed and validated

If a line has a missing value for one of the two required crime attributes, the line is skipped (not imported. Otherwise, the retrieved values for line are then used together in an SQL INSERT statement to add the crime record to the database:

```
201 if ($validLatitude == 1 && $validLongitude == 1) {
202     $sql = "INSERT INTO markers (Crime_Type, Crime_Date,
203             Crime_Time, Description, Latitude, Longitude)
204             VALUES ('$crimeToSend','$dateToSend',
205             '$timeToSend','$descriptionToSend',
206             '$latitudeToSend','$longitudeToSend')";
207     $db->query($sql);
208 }
```

Figure 67: Code snippet of SQL INSERT statement used to insert valid records in the selected file

The progress bar, at this stage, showed only the progress of the file reaching the server. The showing of the progress for the actual processing of the file was next implemented using another progress bar. To determine the progress made, the number of lines processed in the file in proportion to the total number of lines, is periodically written (as a percentage) to a separate file, named ‘progress.txt’:

```
227   $num_rows = count($csvAsArray); // Number of lines in file
228   $check_interval = $num_rows / 20; // Divided by 20
229   $check_interval = ceil($check_interval); // Rounded up
230
231   // ... After a line in the file is processed
232   $lines_processed++;
233
234 if ($lines_total % $check_interval == 0) {
235     file_put_contents("progress.txt",($lines_processed/($num_rows-1))*100);
236 }
```

Figure 68: Code snippet for writing progress percentage to an accessible (text) file

Then, every second after the file is known to have successfully reached the server, this percentage is read from the file and used to update this new progress bar. This is continued until the percentage read does not change for three successive times, at which point it can be determined that the import has finished (or if because of an error, likely isn’t going to continue):

```
var data_hold = -10;
var NoChangeCounter = 0;
var t=setInterval(CheckProgressFile,1000); // Run below function every second

function CheckProgressFile() {
    $.ajax({
            url: "/progress.txt", // GET from file
            success: function(data) {
                var import_percentage = data;
                if (data == data_hold) { // If current progress check = last progress check
                    NoChangeCounter += 1;
                }
                if (NoChangeCounter == 3) { // Progress not changed in 3 seconds
                    $("#progress_insert_upload").css("width", "100%")
                    .text("Import (Complete)"); // Update progress bar (100%)
                }
                else {
                    $("#progress_insert_upload").css("width", Math.round(import_percentage) + "%")
                    .text("Import (" + Math.round(import_percentage) + "%)");
                    // Update progress bar (progress %)
                }
                prev_data = import_percentage; // Hold progress for next check comparison
            }
     });
}
```

Figure 69: Code snippet of periodically reading the progress percentage from the written to file

3.2.12 SQL Injection

3.2.12.1 Problem

Due to user input being used in several of the SQL statements employed in the solution, thought now moved towards vulnerabilities that could be made possible through SQL Injection (SQLi). With an important concept in Information Security being to treat or assume that all user input is malicious, until proven otherwise, a significant overhaul of how any SQL statement, especially those which make use of user input, was next on the agenda.

The real danger or the main rationale for the overhaul is the possibility of an injection which compromises the back-end database infrastructure (whether that be deleting the table used to store crime records, the database containing this table or other means) which would not only mean that currently stored records would be lost but would also render the solution unable to store records in the future, in turn breaking a significant amount of functionality the solution intends to provide. Whilst not absolutely critical to prevent, it may as well be made for this overhaul to also cover any altering of the database being confined to being performed through the provided channels, as opposed to through unintended or malicious methods.

Before implementing the overhaul, identifying and understanding a possible SQL Injection which must be prevented can help to inform about the possible attack vectors. Although just one of many queries implemented in the solution, consider the SQL query used when adding a new record to the database, which exhibits a common location for where SQL Injection arises (the updated values of an INSERT statement) **[31]**:

INSERT INTO markers (Crime_Type, Crime_Date, Crime_Time, Description, Latitude, Longitude) VALUES (‘$crime_type’, ‘$date’, ‘$time’, ‘$description’, ‘$latitude’, ‘$longitude’);

If the following crafted input were to be entered into the *'Description*' field of the 'Add Crime' popup window (and confirmed using the confirmation button), the input would become used as the $description value in this query, facilitating SQL Injection:

Figure 70: Demonstration of crafted input to perform SQL Injection

This is because the query becomes interpreted as the following:

INSERT INTO markers (Crime_Type, Crime_Date, Crime_Time, Description, Latitude, Longitude) VALUES ('$crime_type', '$date', '$time', 'Text', '50', '50');drop table markers;--', '$latitude', '$longitude');

This is the interpretation made because this crafted input first populates the INSERT statement with the required values to reach the point of vulnerability (in and around the description), completing it with and with a closing bracket and semicolon but then also includes a following statement to drop the 'markers' table. The comment denoted by the symbols -- then make it so that the rest of the query is ignored (by being commented out). Put simply, the crafted input means the query executed includes two statements, one to insert a new record into the database (as would happen normally with unmalicious input) but also an additional statement which deletes the table which stores records of markers (the malicious intent).

3.2.12.2 Solution

To protect against SQL Injection, all currently implemented SQL statements were overhauled to make use of prepared statements. With SQL Injection taking advantage of how user input becomes used as part of a SQL statement (in other words, the mixing of the user input or data with the SQL code or commands), prepared statements nullify this by distinctly separating the query and data. Prepared statements involve a query template, containing parameters in the place where values would usually be directly specified, being sent to the server first (without any data) and a following request then being used to provide the data (the sending of these parameters). The values of this data are then bound to the query through parameterisation and only then is the statement executed.

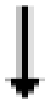
This means that the user input (the data) is always only ever treated as data because the SQL query is already compiled and any data which is parameterized is never directly combined to form part of this query. Critically, this means the data is never interpreted as executable SQL code and thus is never in need of being correctly escaped or directly executed and so even if the user input were to contain malicious SQL commands, they will never be allowed to cause harm or damage by being executed.

The benefit of using prepared statements also extends beyond just improved security. For instance, even if the statement (found within the query) is executed multiple times using different values, the query is only in need of being prepared just once. Additionally, only the changing parameters (the values to be used in the precompiled query) need to be sent each time to the server as opposed to the entire query which would need to be sent every time when executing SQL statements directly (as is currently implemented), saving on bandwidth. In relation to the solution, this benefit would be most substantially observed when prepared statements are implemented for the importing of markers, where the same SQL INSERT statement is executed numerous times for each line in an external file.

3.2.12.3 Solution Implementation

Prepared statements are next implemented to replace the direct statements currently used for every occurrence of a SQL statement (of any kind) in the solution. The binding of the parameters to the query as part of this replacement involves the data type of values being specified in advance using single characters such as ‘s’ (which denotes the string data type), ‘d’ (decimal) and ‘i’ (integer). Using the same query as before (Section 3.3.12) as an example, shown is the query before and after the overhaul:

INSERT INTO markers (Crime_Type, Crime_Date, Crime_Time, Description, Latitude, Longitude) VALUES (‘$crime_type’, ‘$date’, ‘$time’, ‘$description’, ‘$latitude’, ‘$longitude’)



INSERT INTO markers (Crime_Type, Crime_Date, Crime_Time, Description, Latitude, Longitude) VALUES (?, ?, ?, ?, ?, ?)
bind_param(‘ssssdd’, ‘$crime_type’, ‘$date’, ‘$time’, ‘$description’, ‘$latitude’, ‘$longitude’)
execute()

Chapter 4 - Results

4.1 Website Address (URL)

The developed web-based solution (website) is hosted and accessible using the URL:
https://hq017496.webs.act.reading.ac.uk/

4.1.1 Environment

The following results are produced using Google Chrome as the browser and a resolution of 1920x1080.

4.2 Toolbar, Mapping Component and Loading Markers

When this URL is navigated to, the website's elements will begin to load with the toolbar appearing first and the mapping component shortly after. This mapping component will show all the markers or crimes mapped by users (and stored in the solution's database) at the time of the website being opened. For instance, at the time of taking the results for the purposes of this section, only a small number of markers were present on the map but at the time of reading the number of markers may and likely will have changed:

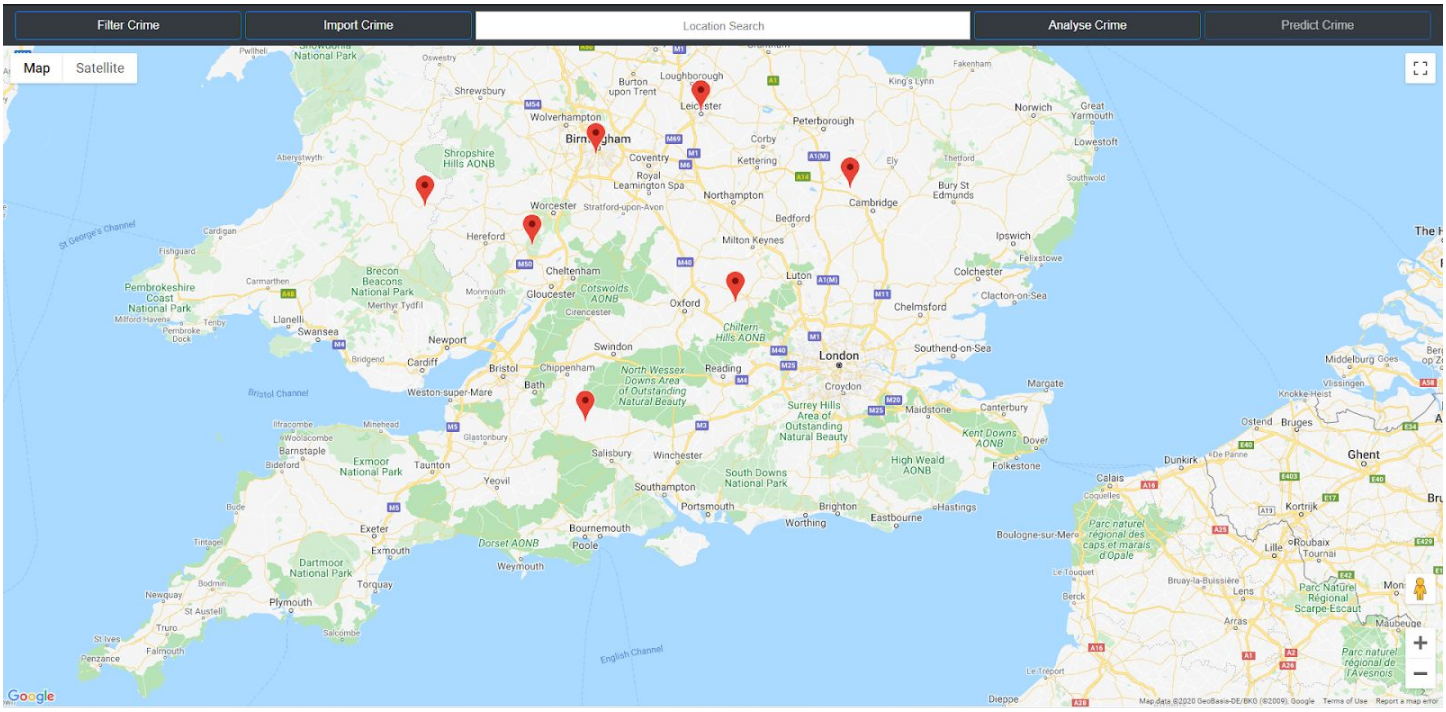


Figure 71: Map and toolbar

4.2.1 Search Bar

The toolbar has a search bar and as a search is made, a list of place predictions that match or resemble the current search is generated on-the-fly and presented as a dropdown list directly below the search bar. When one of these predictions is selected, the map will move and focus to the selected location. Alternatively, a direct search can be made by entering a place name and as long as it can be resolved to a location, when it is submitted by pressing the 'Enter' key, the map will also similarly update, otherwise no change will be made.

4.3 Adding Crime

A mouse right click anywhere on the mapping component will open a one item context menu at the click location. Selecting this option in the context menu will open the ‘Add Crime’ popup window (modal):

Figure 72: Right click context menu

As a quality of life feature, the values of the date and time fields are set to the current date and time respectively but these can be easily changed, if needed, however selecting a date after the current date is prevented, as choosing such a date in all circumstances will almost certainly be an input mistake, as mapping a crime which is yet to happen isn’t exactly feasible or comprehensible. A similar quality of life feature is a smaller map with a draggable marker, used to make small adjustments and confirm the location of the crime/marker to be added:

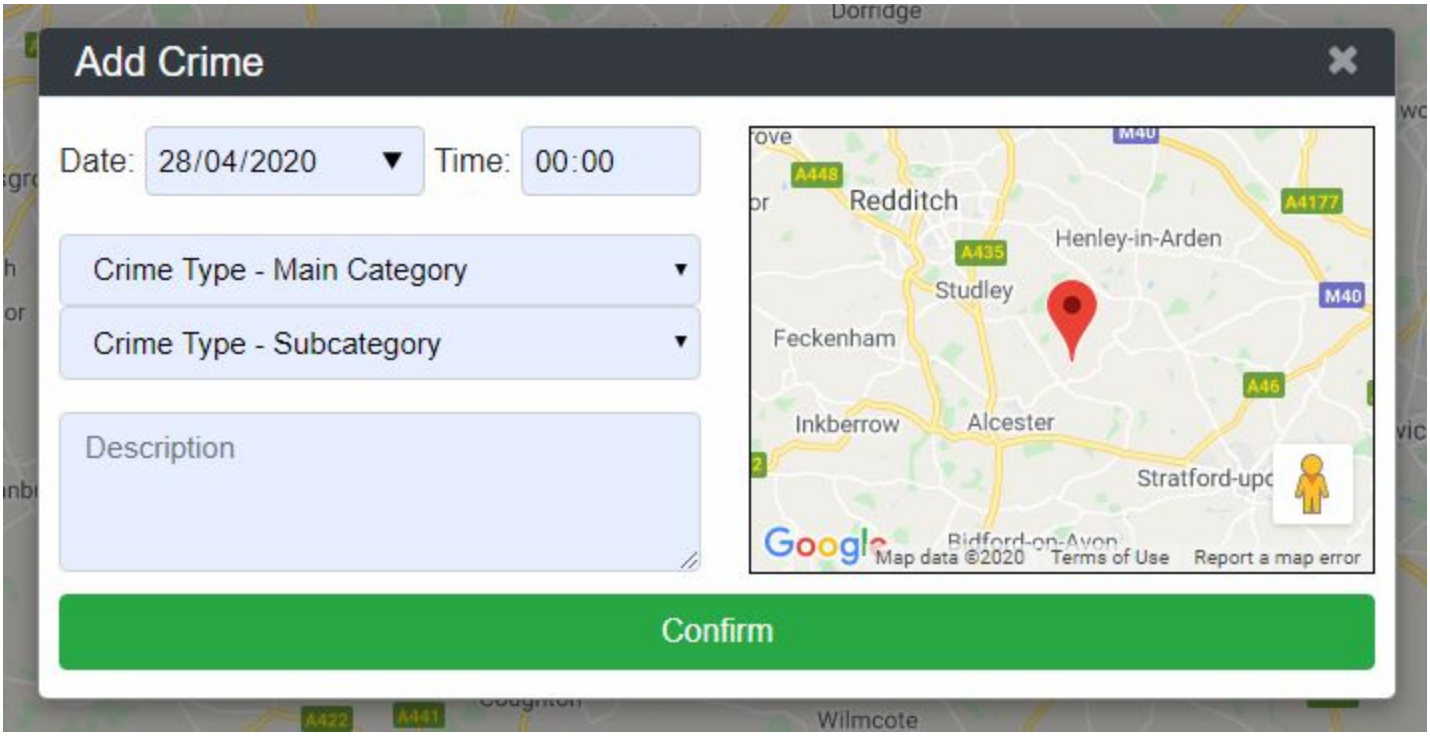


Figure 73: Result of implemented ‘Add Crime’ popup window

4.4 Viewing Crime (InfoWindows)

Left clicking on a marker will open an InfoWindow displaying the crime's current properties. The size of this InfoWindow varies by the length of description in need of being displayed, but is however also importantly limited to a maximum width at which the description will continue on a new line (or new lines). From these InfoWindows, the relevant marker can also be edited or deleted using the respective buttons:

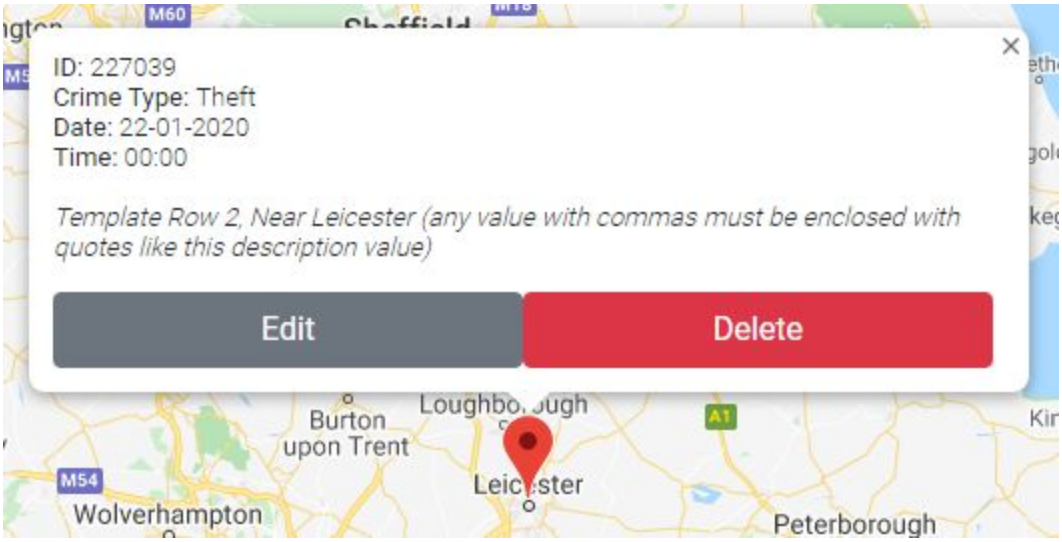


Figure 74: Result of implemented 'View Crime' InfoWindow

4.5 Edit Crime

The 'Edit' button of a marker's InfoWindow will open the 'Edit Crime' popup window. This popup window closely resembles the 'Add Crime' popup window and is provided as an easier alternative to change the properties of a crime as compared to deleting a marker and creating a new one in its place:

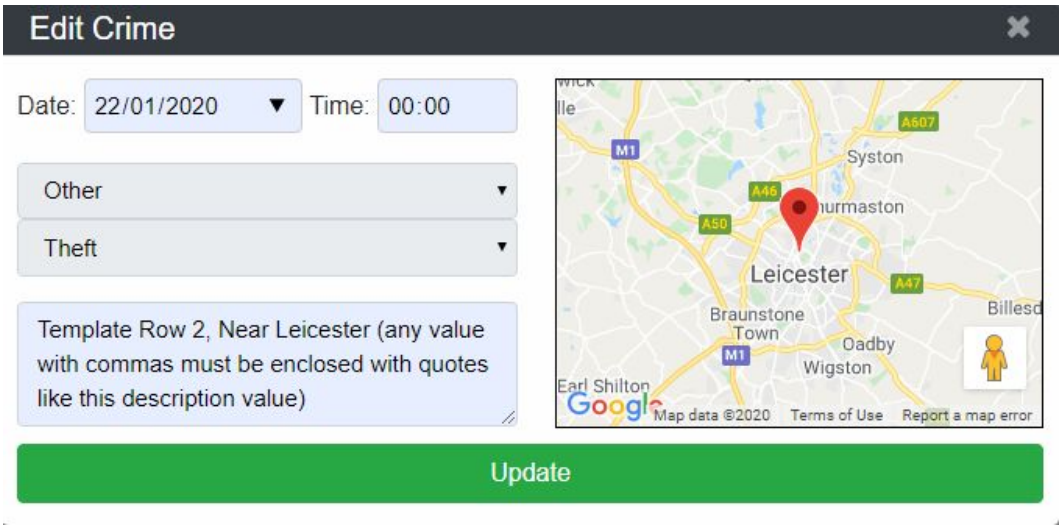


Figure 75: Result of implemented 'Edit Crime' popup window

Unfortunately, the implementation choice of whilst every property can be changed for markers created using the 'Add Crime' popup window or functionality, if the marker was imported from an external file, the crime type properties for this marker can not be edited (the fields are disabled as seen in Figure 75).

4.6 Delete Crime

The ‘Delete’ button of a marker’s InfoWindow will permanently delete that marker from the crime mapper (a noticeable result of the implementation for this functionality is that the mapper does not need to be loaded again for the changes to take effect).

4.7 Filter Crime

The ‘Filter Crime’ button of the toolbar will open the ‘Filter Crime’ popup window (modal):

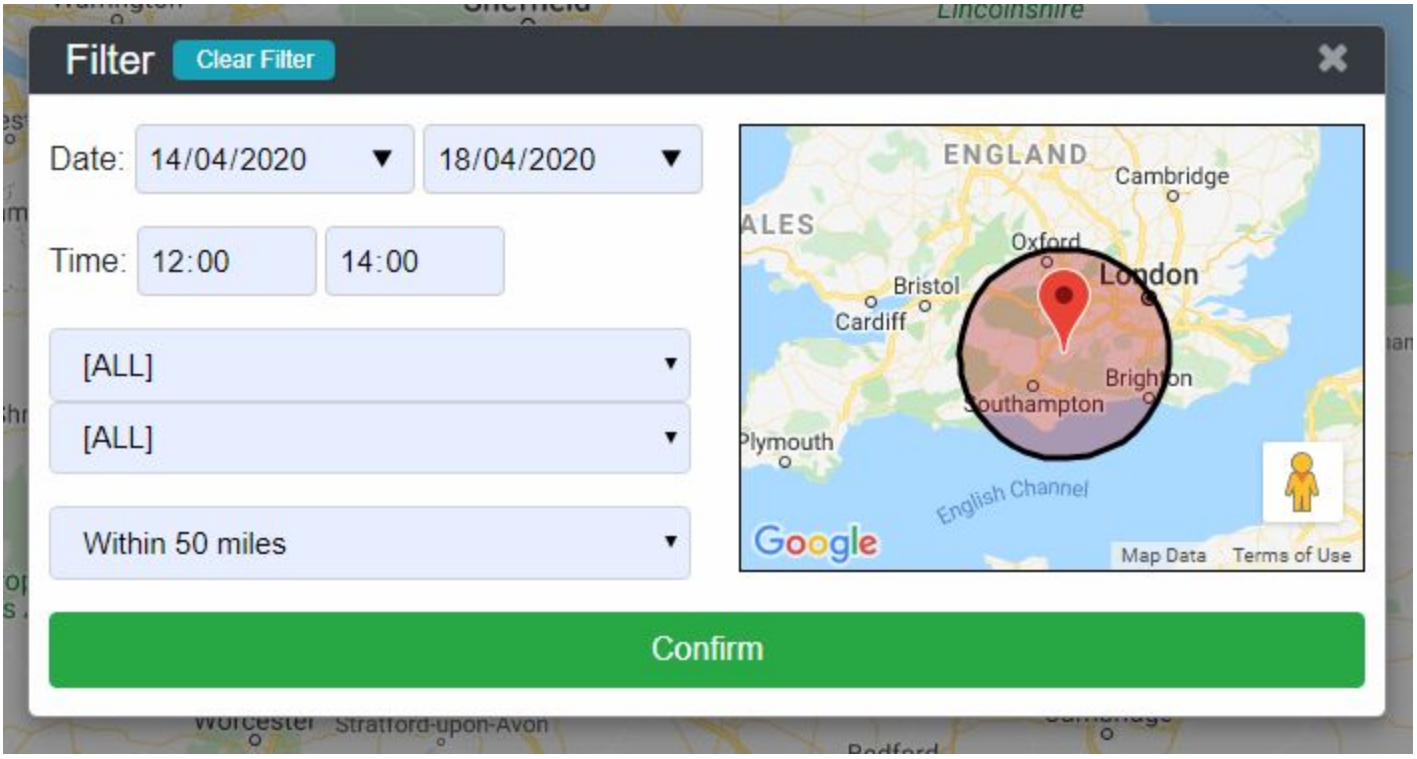


Figure 76: Result of implemented ‘Filter Crime’ popup window (with an option to clear the current filter)

4.8 Import Crime

The ‘Import Crime’ button of the toolbar will open the ‘Import Crime’ popup window. This window allows for a file to be selected and its content to be imported and displayed by the crime mapper, aimed specifically at streamlining the process of adding and mapping a large amount of crimes (such as crime data from official crime data dumps):

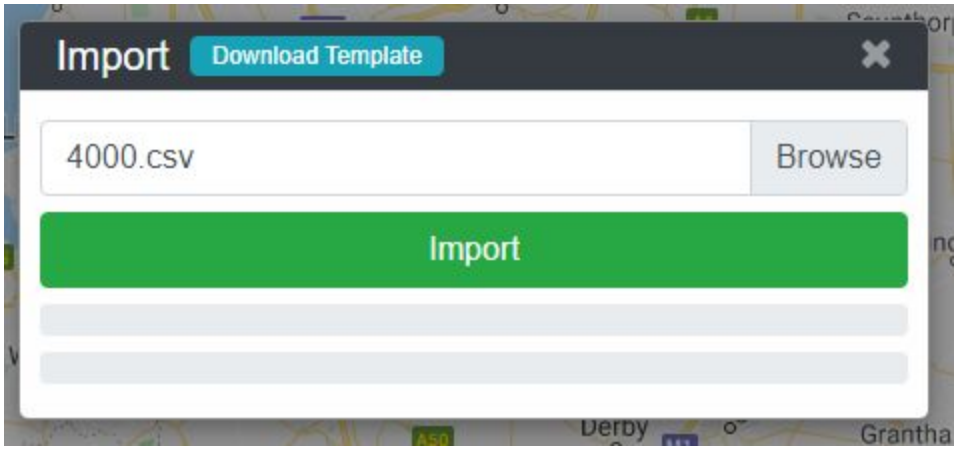


Figure 77: Result of implemented ‘Import Crime’ popup window (with an option to download a import file template)

The file can then be confirmed to be imported by pressing the ‘Import’ button, if the file and its contents are not considered valid, an alert message is displayed stating a reason why the file is unable to be imported and/or what must be changed with the file in order to be valid:

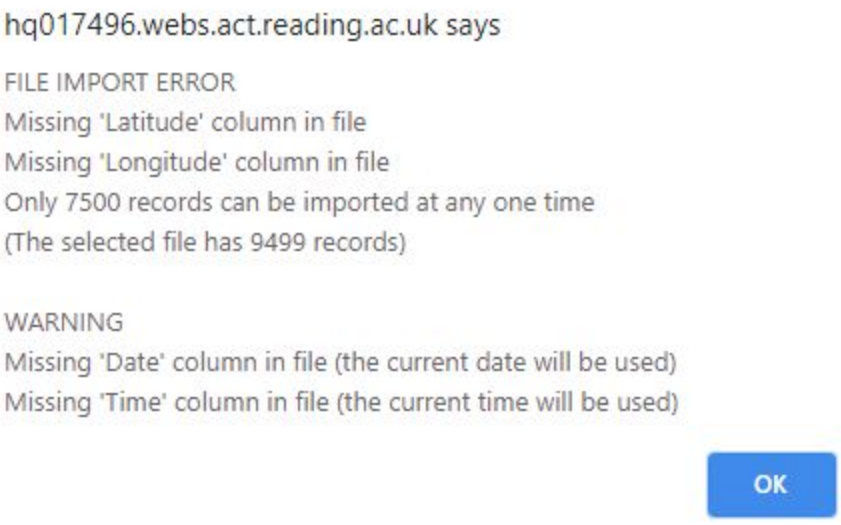


Figure 78: Informative displaying of warnings and errors when importing crime

The two progress bars below the ‘Import Button’ will show progress of the import, if allowed to go ahead, the first showing the progress of the file being uploaded (to the server) and the second showing the progress of processing and mapping each record in the file. These progress bars themselves can show errors with their respective processes in which case the progres bar will change from its default blue colour to an error red colour. Once (and if) both progress bars reach full width, after a short delay the page will refresh and the new imported markers will be able to seen on the map:

Figure 79: Result of importing 4000 crimes from an official crime data dump (police.uk) for the Kent constabulary

4.9 Analyse Crime

The ‘Analyse Crime’ button of the toolbar unlike the other buttons in the toolbar does not open a popup window and instead can be used to toggle the clustering of markers on or off. When toggled on, nearby markers are clustered together into a grouped icon, the colour of which is determined by the number of markers in the group. These groups are dynamically created and change with different zoom levels, showing the spatial distribution or density of crimes:

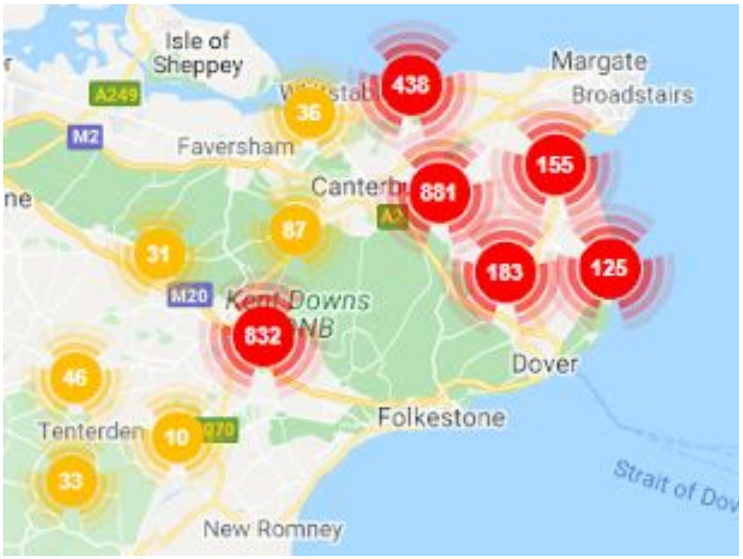


Figure 80: Analysis of Figure 79 result (low zoom)

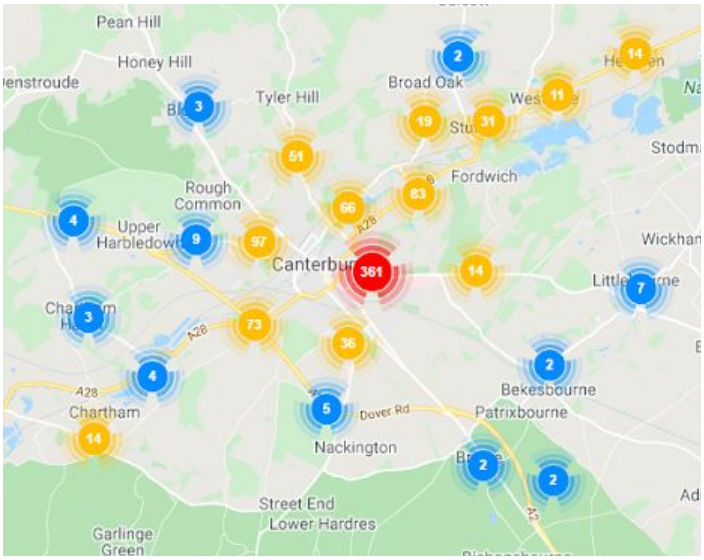


Figure 81: Analysis of Figure 79 result (high zoom)

Chapter 5 - Testing and Validation

5.1 Cross Browser Testing

The solution was first tested using various different web browsers with the latest versions (at the time of testing) of four common browsers along with a constant screen resolution of 1920 x 1080 being used. Most testing consideration for this testing is directed towards the layout of user interface elements (UI) and the website responsiveness of the solution.

5.1.1 Google Chrome (v81.0.4044.129) - **PASS**

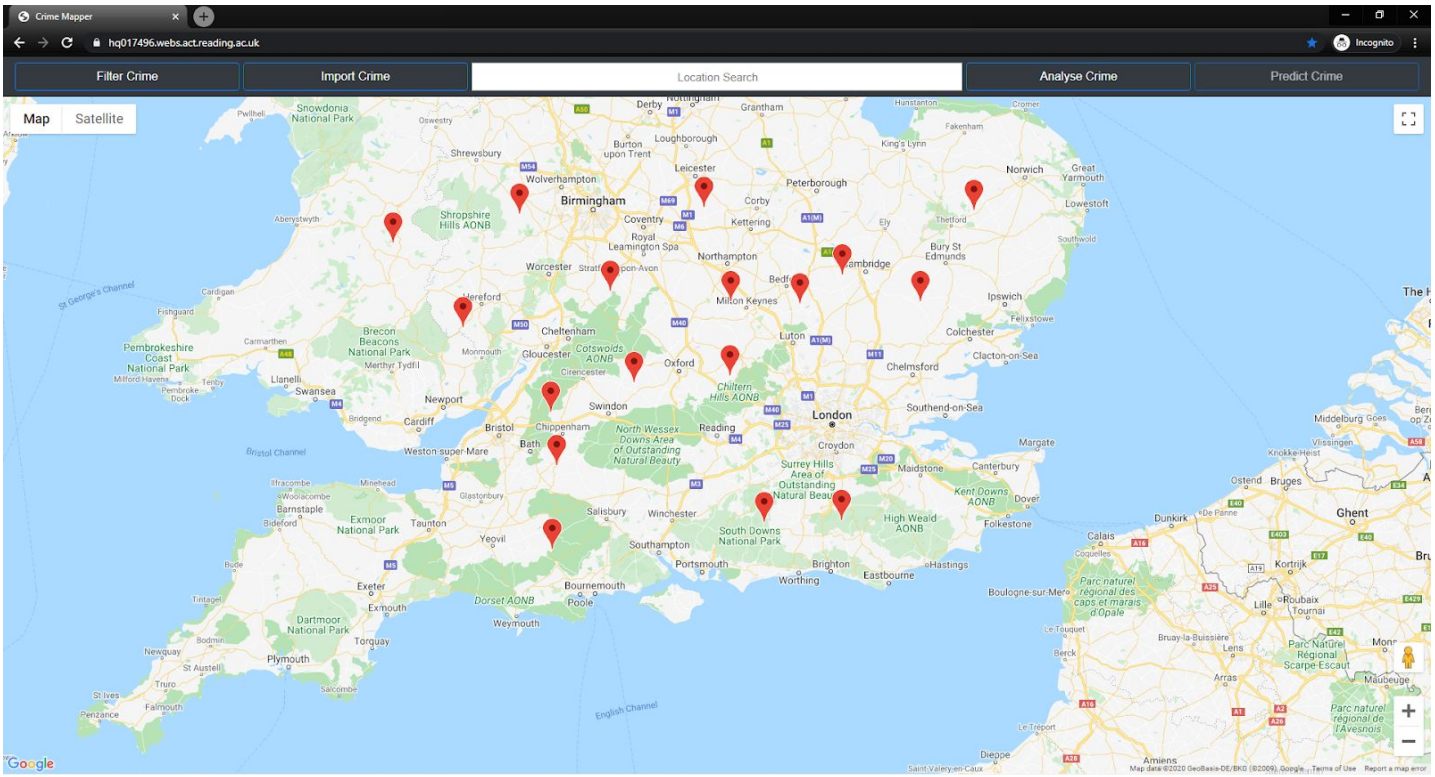


Figure 82: Solution website opened with Google Chrome browser

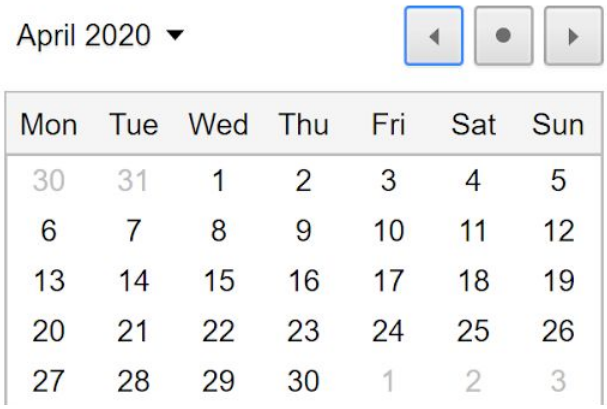


Figure 83: Google Chrome date control

5.1.2 Mozilla Firefox (v75.0) - **PASS**

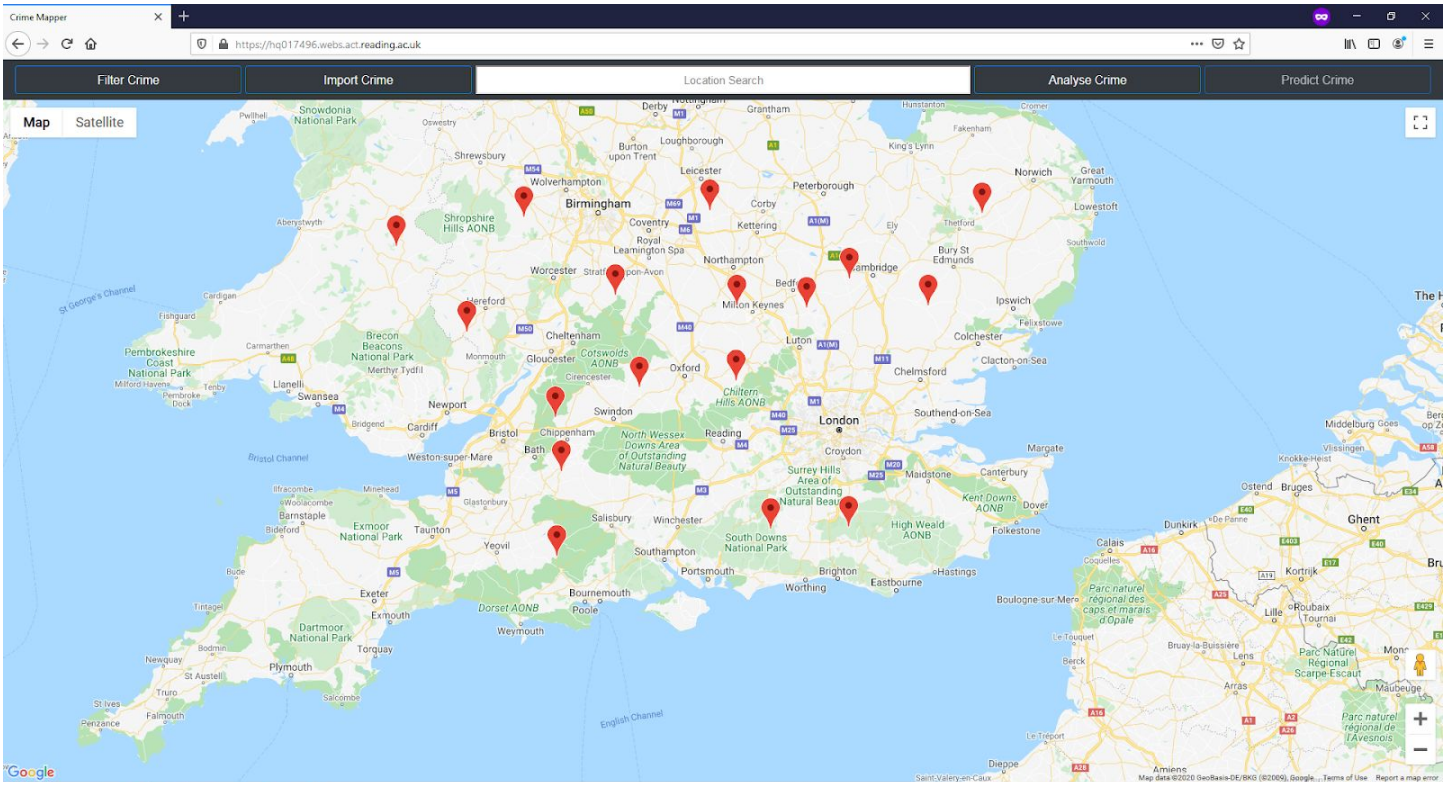


Figure 84: Solution website opened with Mozilla Firefox browser

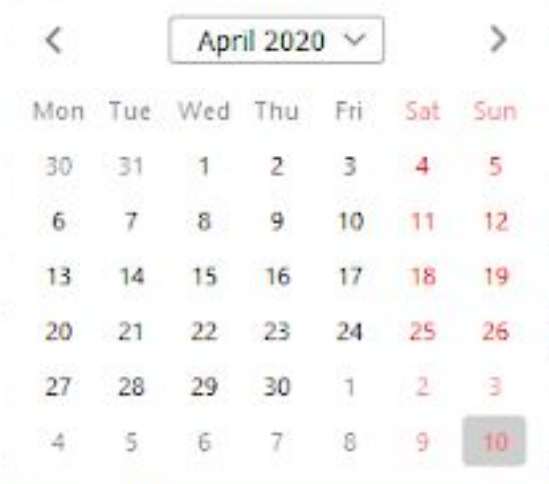


Figure 85: Mozilla Firefox date control

5.1.3 Microsoft Edge (v44.18362.449.0) - **PASS**

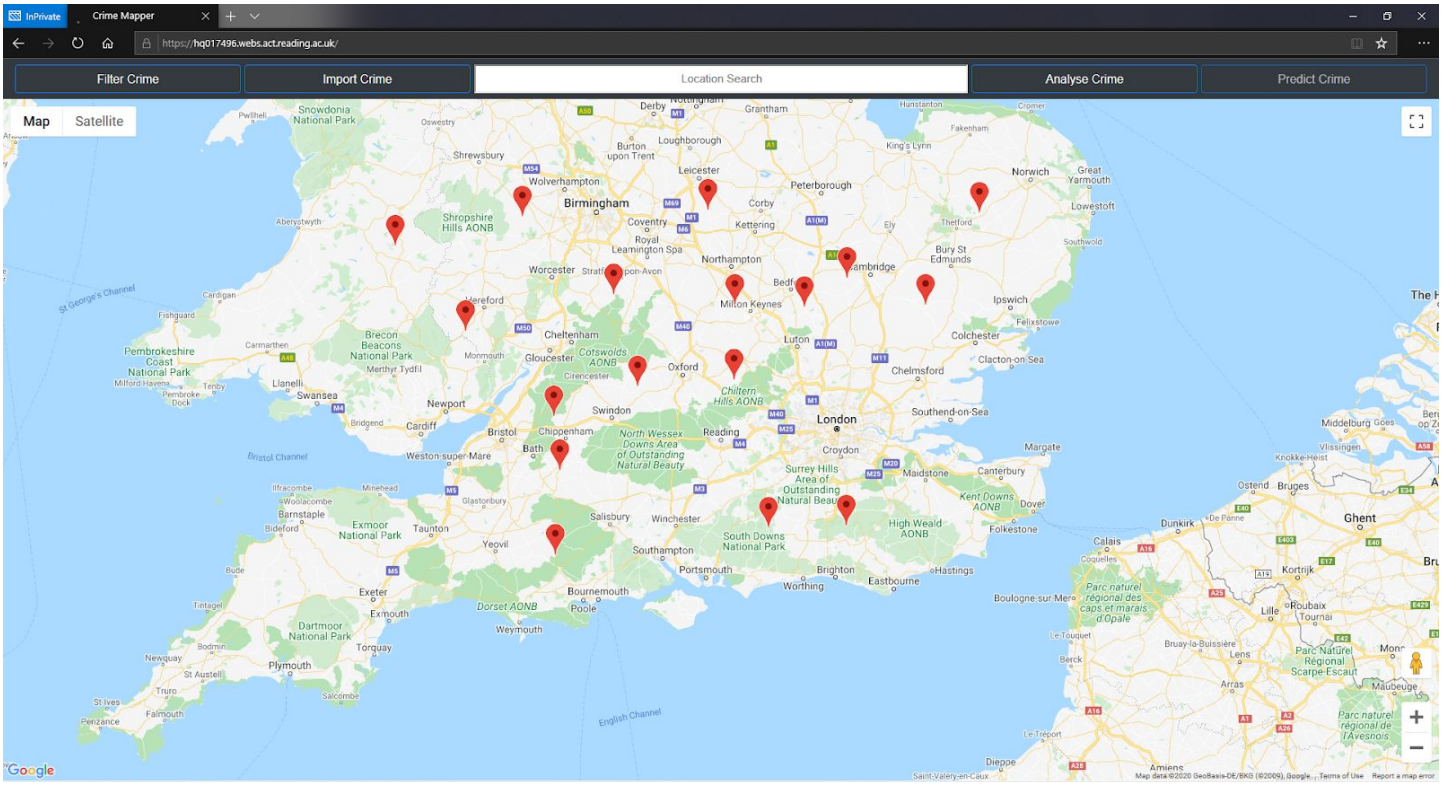


Figure 86: Solution website opened with Microsoft Edge browser

Figure 87: Microsoft Edge date control

5.1.4 Internet Explorer (v11.778.18362.0) - FAIL

The mapping component does not load with this browser and only the toolbar is shown:

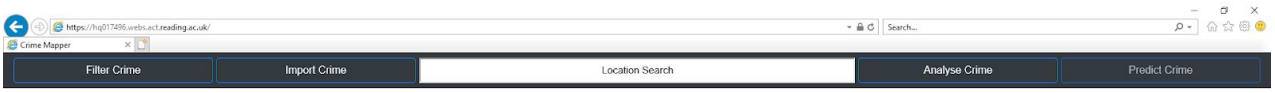


Figure 88: Solution website opened with (Microsoft) Internet Explorer browser

The inputs of the popup windows do not function correctly, some allow for any type of values to be entered where unintended and for which is not the case with the other modern browsers, whilst others such as the dropdown lists do not provide any options to select:

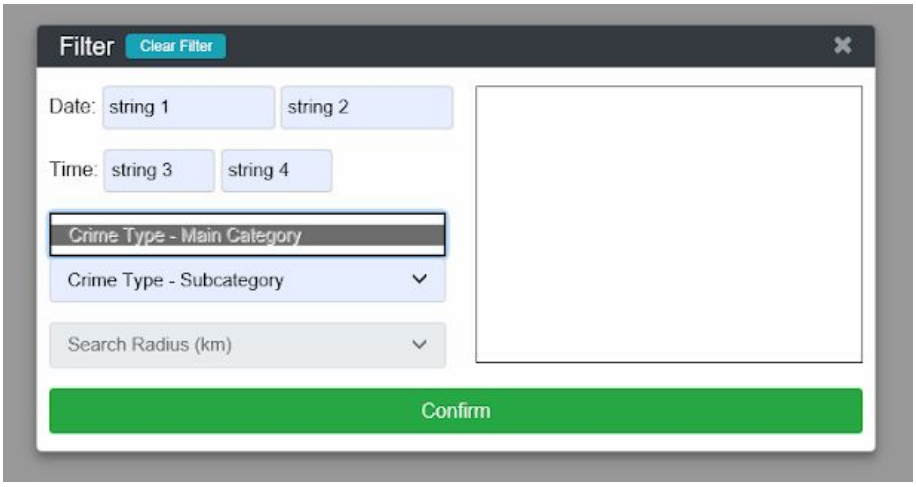


Figure 89: Unfunctional ‘Filter Crime’ popup window when using IE11

Whilst all effort has been made to santisie and handle malicious input, the ability to enter a string value into a field for which it was previously thought that only a date value could be entered (as a result of the inherent validation of the field in other browsers as seen in Figure 83, Figure 85 and Figure 87), is troubling from a security perspective. Although procedures are in place, the best thing to do is to disable all functionality for this browser and to display an instructional message to use another browser instead:

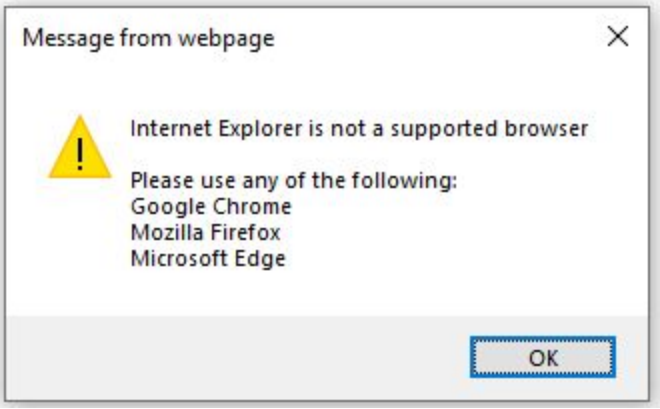


Figure 90: IE11 Compatibility warning message

5.2 Resolution/Responsive Testing

To further test the responsiveness achieved using the BootStrap framework, the solution was next tested on devices of varying screen resolution.

5.2.1 (1920 x 1080) - PASS

Tested prior with for the ‘Results’ section and as the constant resolution used for cross-browser testing.

5.2.2 (1366 x 768) - FAIL

The ‘Add Crime’, ‘Edit Crime’ and ‘Filter Crime’ popup windows are all misaligned horizontally (off center using the center of the search bar as reference):

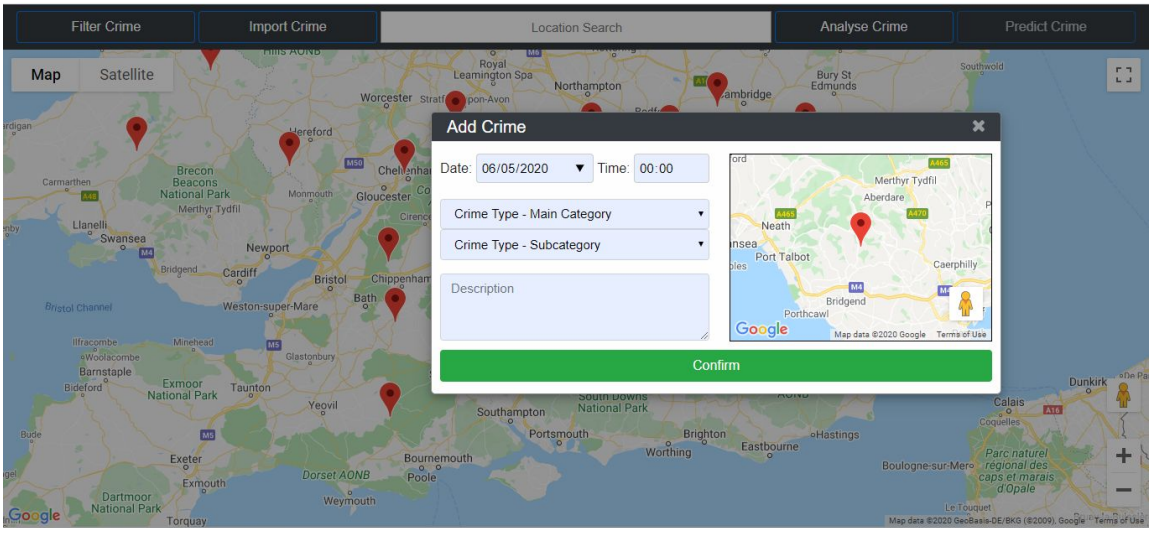


Figure 91: Misaligned ‘Add Crime’ popup window using 1366 x 768 resolution

5.2.3 (1024 x 600) - FAIL

The same problem applies with this resolution but to a greater extent and also in terms of vertical alignment (the page becomes scrollable), with even less of the modal being visible:

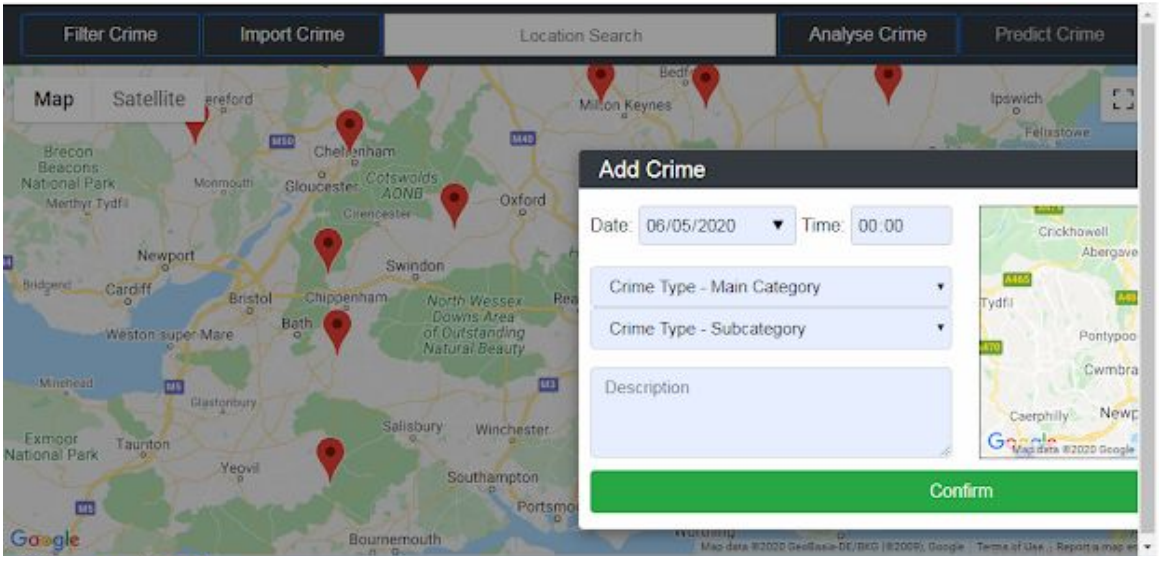


Figure 92: Misaligned 'Add Crime' popup window using 1024 x 600 resolution

5.2.4 Solution/Fix

To address this problem, media queries can be implemented allowing for the position of the modals to be changed for different screen resolutions. The top property and left property of the modal can be used to alter the vertical alignment and horizontal alignment respectively with different values being assigned for different ranges of screen resolution:

```
#modal_add_dialog, #modal_edit_dialog, #modal_filter_dialog { /* All resolutions */
    min-width:1700px;
}

@media (min-width:1200px) and (max-width:1370px) { /* e.g 1366 x 768 pixels */
  #modal_add_dialog, #modal_edit_dialog, #modal_filter_dialog {
    left:-160px;
    top:-25px;
  }
}

@media (max-width:1200px) { /* Even smaller resolutions */
  #modal_add_dialog, #modal_edit_dialog, #modal_filter_dialog {
    left:-340px;
    top:-35px;
  }
}
```

Figure 93: Code snippet of media queries used to conditionally position popup windows

5.3 Performance Testing

The performance of the solution can be explored with the time to complete certain tasks or operations under different scenarios with varying amounts of markers being recorded. Whilst most operations happen with no noticeable delay, the two components of loading and importing of markers can take a considerable amount of time to complete and so therefore the majority of attention will be directed towards testing and potentially optimising the associated or related modules of code.

5.3.1 Loading Markers

When the solution is first opened or after a successful import the currently recorded markers, the duration to do this can be me measured, which are shown in Table 2:

Table 2: The average duration to load markers (before optimisation)

Number of Markers	Average Duration (ms)
0	0.230
10	12.9
100	86.8
250	152
500	279
1000	503
2500	1110
5000	1960
10000	3850
25000	8290
50000	16300
100000	-

A form of stress testing was also inadvertently performed as with 100,000 markers, the map would no longer show, even when given several minutes to load. Whether this remains the case after completion of code optimisation is something to be seen, but in the meantime to try to decrease the average duration taken to load the markers (such as over 16 seconds to load 50000 markers), ways in which this module of code or surrounding modules of code could be optimised, were explored.

The potential optimisation which could be implemented, concerns how the InfoWindows for markers are created and shown. As it stands, the InfoWindow (which is made up of numerous HTML elements) of a marker, for every marker, is always created after that marker has been loaded and placed on the map, with it being opened when the marker is clicked on. The InfoWindows for all markers are created preemptively even if they may never be requested to be opened. The suggested optimisation however proposes that the InfoWindows are only created on a per request basis where the InfoWindow for a marker is only ever created when requested (that marker is clicked on). This change can be implemented simply by moving the creation of the InfoWindow objects for markers inside a click event listener for markers.

Table 3 shows the average durations after this optimisation (with the results being taken and recorded using the same method adopted for collecting the results before the optimisation):

Table 3: The average duration to load markers (after optimisation)

Number of Markers	Average Duration (ms)
0	0.167
10	5.45
100	19.1
250	27.4
500	46.1
1000	80.6
2500	148
5000	258
10000	402
25000	887
50000	1600
100000	-

The average duration to load markers has significantly reduced as a result of the implementation of the optimisation (e.g with 50,000 markers, on average, being reduced from 16.3 seconds to 1.6 seconds)
.

5.3.2 Resource Utilisation

To briefly summarise how the solution performs by the way of system resources it consumes throughout its use, during operations such as analysing (clustering) a large number of markers or navigating the map around areas that are heavily populated with markers the CPU usage would spike with it being that on lower-end systems, the browser would hang for a short period of time. Memory usage remains almost stable throughout use with only small increases in usage as more markers begin to be displayed on the map.

[Blank Page]

5.4 Unit Testing

The two main individual components of the solution that can be individually tested are the components frequently referred to as ‘Load Crime’ and ‘Add Crime’ (the other components depend on these components or external factors and cannot be tested using unit testing and will instead be tested with integration testing).

5.4.1 Load Crime

Unit testing for this component should not test using information retrieved from the database as this is an external factor which is to be tested with integration testing. Instead, a test using mockup information with arrays is written, one of these arrays is empty (to simulate an empty database) and the other contains a specified number of rows of information (to simulate a populated database). If the number of placed markers increases by the amount of rows of information in these arrays, and in the case of the array that isn’t empty, the placed markers properties match with the values in these arrays, it can be said that the module has passed the unit test (with a log being written to the console depending on the outcome of the test):

```
780 for( i = 0; i < unit_testing_markers.length; i++ ) {
781     if (MarkerArray[last_added-i].ID != unit_testing_markers[(unit_testing_markers.length-1) - i][0]) {
782         load_matchingValues = false;
783     }
784     if (MarkerArray[last_added-i].Crime_Type != unit_testing_markers[(unit_testing_markers.length-1) - i][1]) {
785         load_matchingValues = false;
786     }
787     // ... Other properties
```

Figure 94: Code snippet of ‘Load Crime’ unit test

5.4.2 Add Crime

For this component, similar checks as those for the previous component must be made, this time comparing the entered values with the placed or added marker’s property (therefore meaning the testing must be completed manually) and also checking the number of placed markers increases by 1. The different categories for the crime types that can be chosen as well as being able to adjust the location with the smaller map must also be tested. For this type of testing, it is not yet tested whether the entered values match with those added to the database (as that is integration testing):

```
1199 console.log("/// Add Crime ///");
1200 var add_length_before = MarkerArray.length;
1201
1202 placeMarker(result,Crime_Type,Crime_Date,Crime_Time,Description,FirstLocation,map);
1203
1204 // Unit Testing //
1205 var add_length_after = MarkerArray.length;
1206
1207 if (add_length_after == (add_length_before + 1)) {
1208     console.log("Count: PASS")
1209 }
1210 else {
1211     console.log("Count: FAIL");
1212 }
```

Figure 95: Code snippet of ‘Add Crime’ unit test

5.5 Integration Testing

The best way to test some components (such as those which interact with the database) is to observe and verify the output in practice using test cases.

5.5.1 Import Crime (Database)

*Test Case **LC1** - Loading no or zero records -* **PASS**

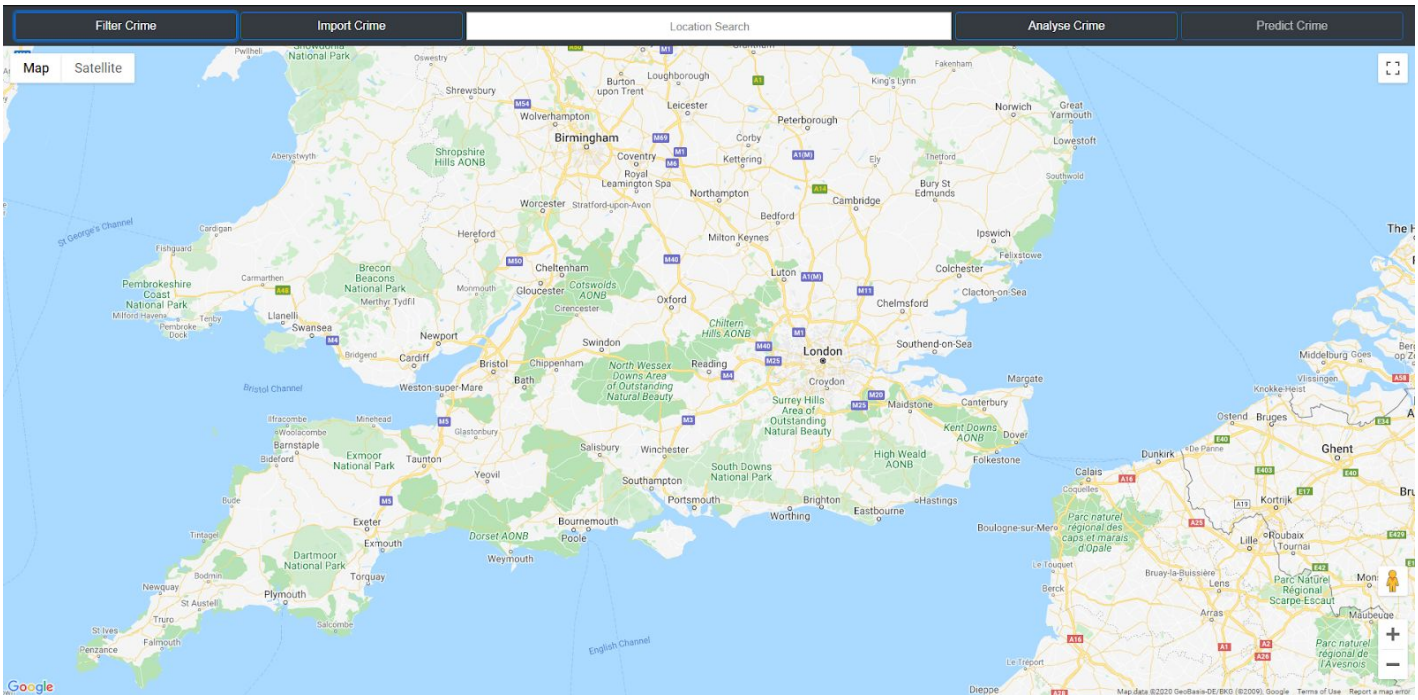


Figure 96: LC1 Test Case, observed output

*Test Case **LC2** - Loading a single (1) record -* **PASS**

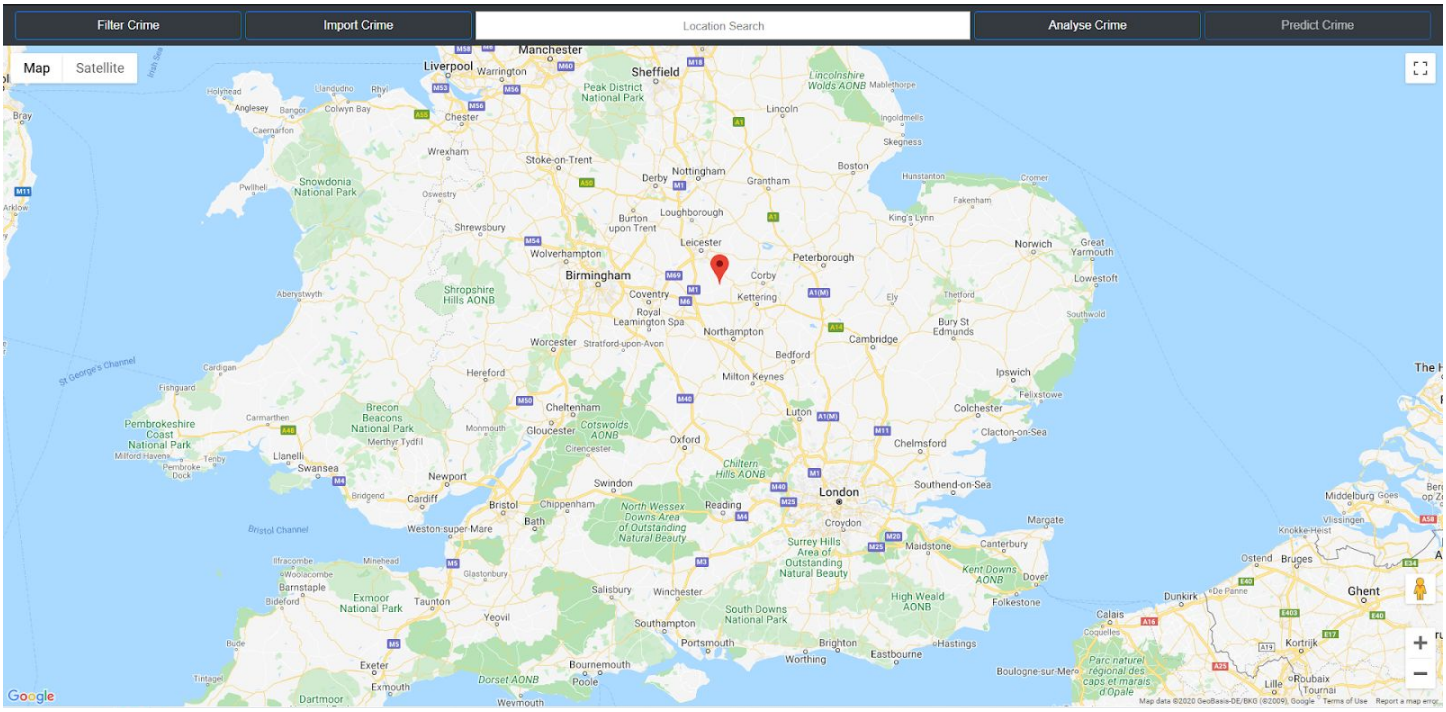


Figure 97: LC2 Test Case, observed output

*Test Case **LC3** - Loading multiple (10) records -* **PASS**

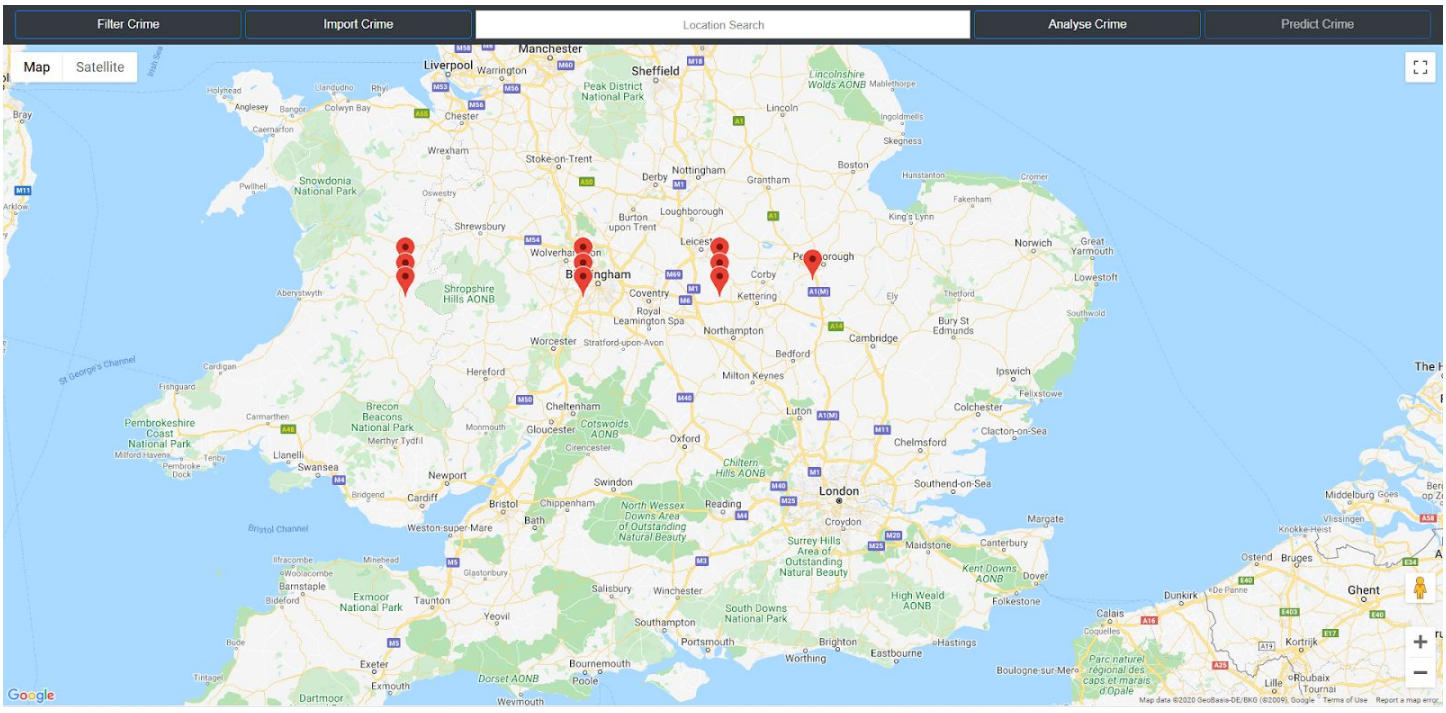


Figure 98: LC3 Test Case, observed output

5.5.2 Add Crime (Database)

*Test Case **AC1** - Old Date and No Description -* **PASS**

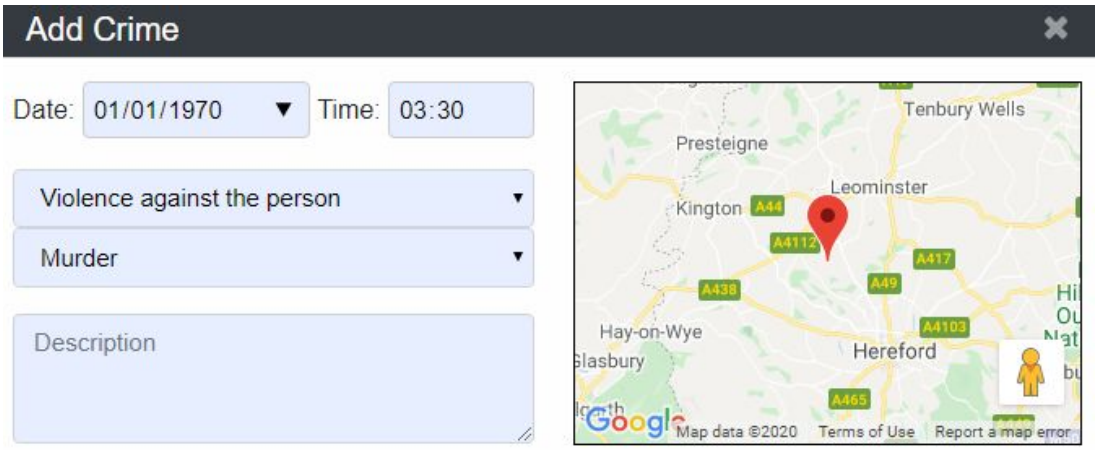


Figure 99: AC1 Test Case, test step

ID	Crime_Type	Crime_Date	Crime_Time	Description	Latitude	Longitude
47	Murder	1970-01-01	03:30:00		52.14773194	-2.82636937

Figure 100: AC1 Test Case, observed output

*Test Case **AC2** - Adjusted position and long description (<500 characters) -* **PASS**

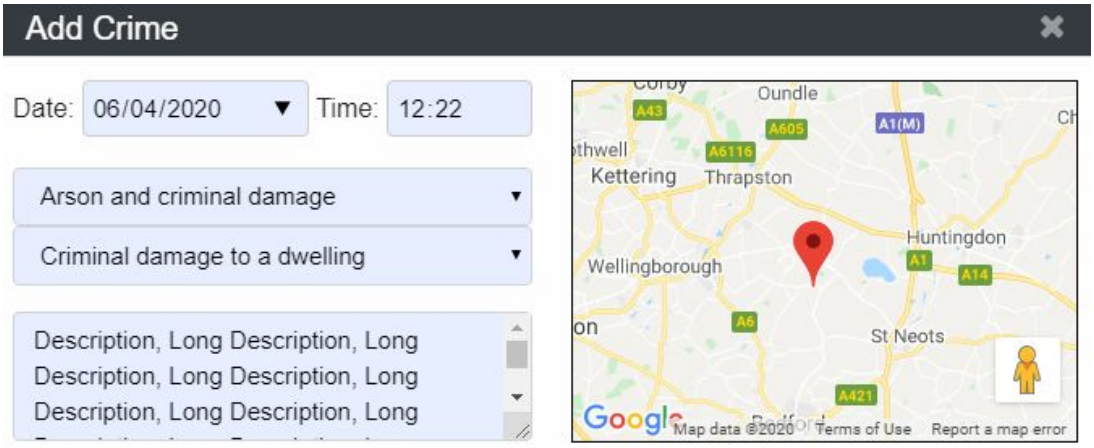


Figure 101: AC2 Test Case, test step

ID	Crime_Type	Crime_Date	Crime_Time	Description	Latitude	Longitude
48	Criminal damage to a dwelling	2020-04-06	12:22:00	Adjusted position and long description, Adjusted p...	52.88976634	-0.32148656

Figure 102: AC2 Test Case, observed output

*Test Case **AC3** - Long description (>500 characters) -* **PASS**

(**NOTE**: Something not mentioned until now is that in the implementation, a small deviation from the design for the database domain length of the description was made by increasing it from 255 to 500 characters):

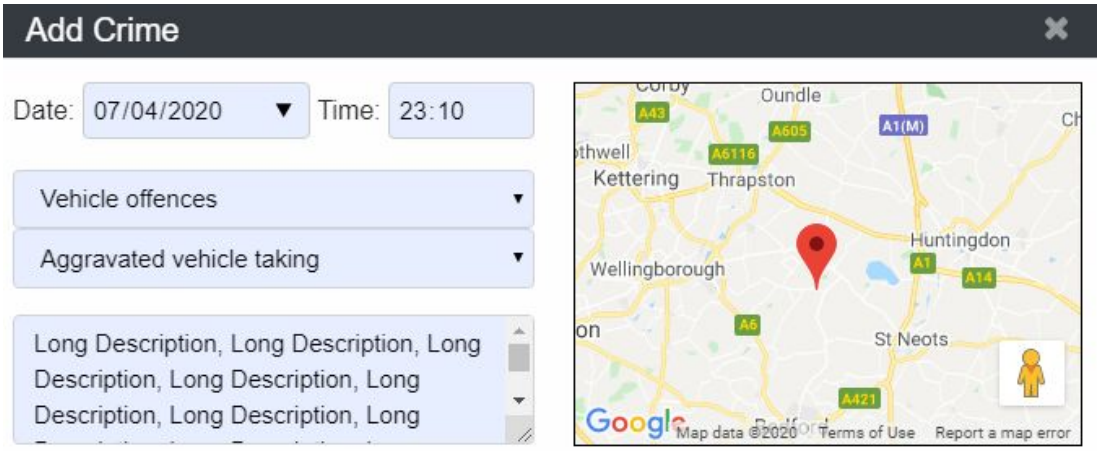


Figure 103: AC3 Test Case, test step

Figure 104: AC3 Test Case, observed output

ID	Crime_Type	Crime_Date	Crime_Time	Description	Latitude	Longitude

Figure 105: AC3 Test Case, observed output

*Test Case **AC4** - No Crime Type(s) specified -* **FAIL**

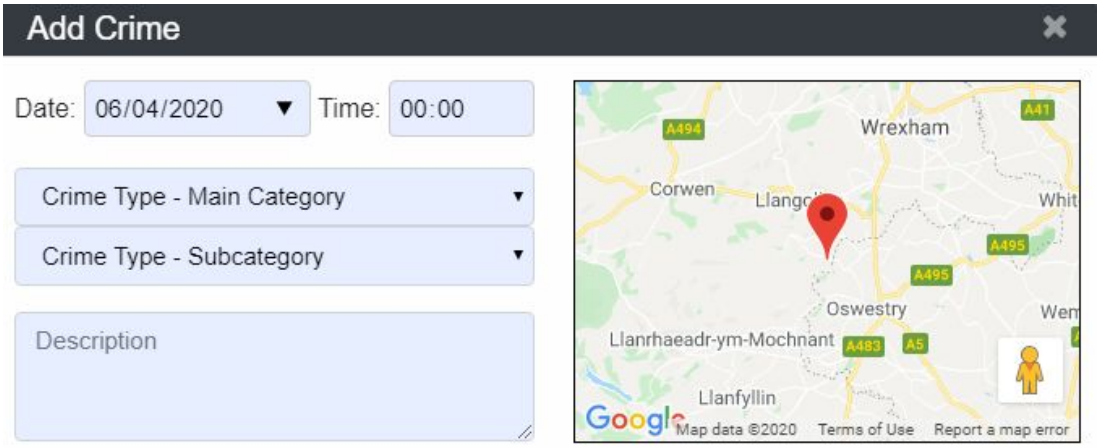


Figure 106: AC4 Test Case, test step

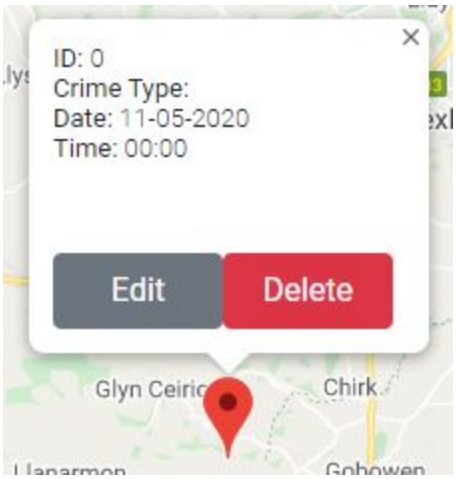


Figure 107: AC4 Test Case, observed output

ID	Crime_Type	Crime_Date	Crime_Time	Description	Latitude	Longitude

Figure 108: AC4 Test Case, observed output

5.5.2.1 Solution/Fix

To remedy this unintended behaviour, a check for any unspecified crime type (the fields not changed from their default hidden values) was implemented, that when met prevents any addition being made to the database and shows a similar alert to the alert shown in Figure 104.

*Test Case **AC4** - No Crime Type(s) specified -* **PASS**

Figure 109: AC4 Test Case, observed output

5.5.3 View Crime (and Add Crime)

A marker must be placed or added before its information can be viewed in an InfoWindow. To test that this 'View Crime' component works correctly along with the 'Add Crime' component, a test checking that the information displayed in the InfoWindow matches with the marker's properties (which have been previously tested as matching the the entered values way back when a marker is added), could be written. But with the marker's properties being directly assigned as the content of the InfoWindow, testing is better directed towards checking if the InfoWindow correctly opens and its content remains the same before and after being opened (without an edit being made):

```
296         if (typeof(marker.info) === "undefined") { // InfoWindow not created
297             console.log("Show InfoWindow: FAIL");
298         }
299         else {
300             console.log("Show InfoWindow: PASS");
301             var content_before = marker.info.getContent();
302 
303             marker.info.open(map,marker);
304 
305             var content_after = marker.info.getContent();
306 
307             if (content_before == content_after) { // Matching content
308                 console.log("Content: PASS");
309             }
310             else {
311                 console.log("Content: FAIL");
312             }
313         }
```

Figure 110: Code snippet of 'View Crime' (and 'Add Crime') integration test

5.5.4 Edit Crime (and Add Crime)

The integration test for this component closely resembles the unit test for the ‘Load Crime’ unit (Figure 94), but slightly differently tests whether the entered values in the ‘Edit Crime’ popup window match with the newly updated properties of the relevant marker:

```
549   var edit_matchingValues = true;
550
551 if (MarkerToEdit.Crime_Type != Crime_Type) {
552     edit_matchingValues = false;
553   }
554 if (MarkerToEdit.Crime_Date.substring(0,5) != Crime_Date.substring(0,5)) {
555     edit_matchingValues = false;
556 }
```

Figure 111: Code snippet of ‘Edit Crime’ (and ‘Add Crime’) integration test

5.5.5 Edit Crime (Database)

The interaction of this component with the database can be tested with an integration test using test cases adapted from the integration tests used for the ‘Add Crime’ unit (Section 5.5.2).

5.5.6 Delete Crime (and Add Crime)

For a marker to be deleted, it must evidently first be added in the first place, hence why these two modules are tested together. The integration test involves checking if the number of stored records (in the local array) decreases by 1 after the deletion:

```
582         var delete_length_before = MarkerArray.length;
583
584         if (index !== -1) MarkerArray.splice(index, 1); // Remove marker from array
585
586         var delete_length_after = MarkerArray.length;
587
588         console.log("/// Delete Crime ///")
589         if (delete_length_after == (delete_length_before-1)) {
590             console.log("Count: PASS");
591         }
592         else {
593             console.log("COUNT: FAIL");
594         }
```

Figure 112: Code snippet of ‘Delete Crime’ (and ‘Add Crime’) integration test

5.5.7 Delete Crime (Database)

*Test Case **DC1** - Delete an added marker -* **PASS**

ID	Crime_Type	Crime_Date	Crime_Time	Description	Latitude	Longitude
51	Theft from vehicle	2020-04-07	02:04:00	Phone stolen from car	52.15565467	-3.35667004

Figure 113: DC1 Test Case, prerequisite step

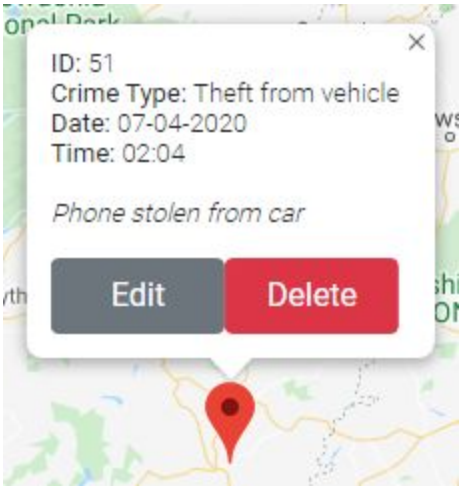


Figure 114: DC1 Test Case, prerequisite step

ID	Crime_Type	Crime_Date	Crime_Time	Description	Latitude	Longitude

Figure 115: DC1 Test Case, observed output

5.5.8 Analyse Crime (and Add Crime)

The test cases for this component are designed to ensure the different coloured icons for groups of markers are correctly displayed. Due to the time consuming process of manually adding 100 or more markers, the testing of the third level of icon was left until testing the integration of this component and imported crime (an assumption was made that if the integration of those components function correctly, it is unlikely the integration would differ for these components):

*Test Case **AnC1** - Cluster 5 markers (similar locations) -* **PASS**

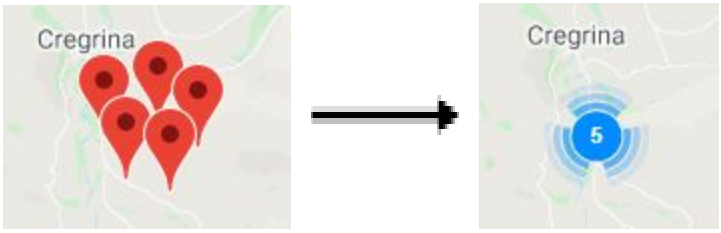


Figure 116: AnC1 Test Case, observed output of second (blue) level icon clustering markers

*Test Case **AnC2** - Cluster 15 markers (similar location) -* **PASS**

Figure 117: AnC2 Test Case, observed output of second (blue) level icon clustering markers

5.5.9 Filter Crime (and Add Crime)

The following test cases were used to thoroughly test the different input combinations for filtering crime which include testing the thresholds of the different fields as well as leaving some fields as their default values:

*Test Case **FC1** - No values specified (filter cleared) -* **PASS**

*Test Case **FC2** - Minimum date and time the same date and time as test marker -* **PASS**

*Test Case **FC3** - Maximum date and time the same date and time as test marker -* **PASS**

*Test Case **FC4** - All Main Crime Types -* **PASS**

*Test Case **FC5** - All Sub Crime Types -* **PASS**

*Test Case **FC6** - Only Search Radius -* **PASS**

5.5.10 Edit Crime and View Crime

*Test Case **EC1** - Edit an added crime and check InfoWindow -* **PASS**

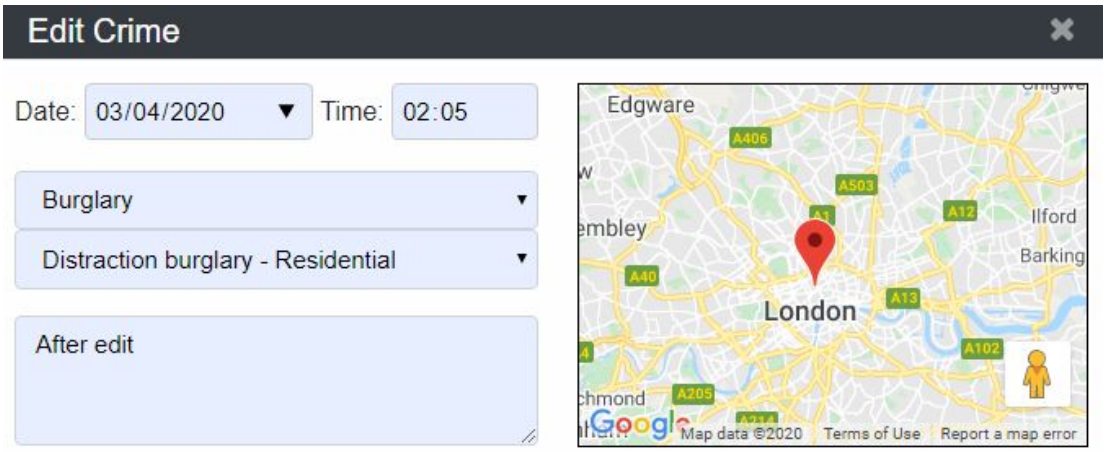


Figure 118: EC1 Test Case, test step

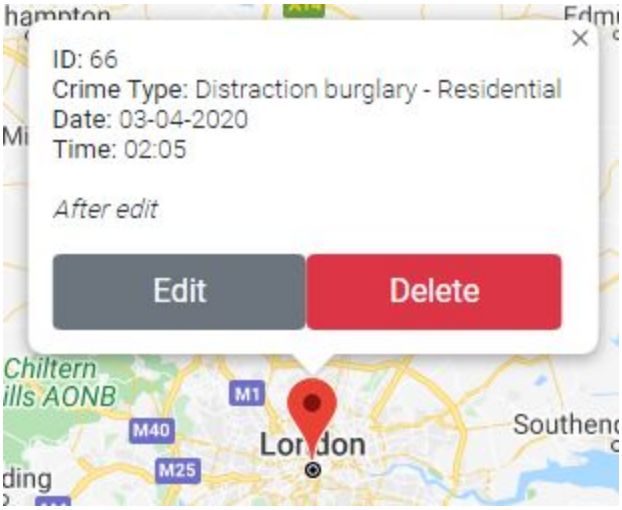


Figure 119: EC1 Test Case, observed output

5.5.11 Other Testing

Further manual integration tests testing the remaining, less frequently occuring combinations of components were also performed and all passed (Appendix, Table 6).

5.5.12 Testing Log

```
/// Load Crime ///
0 records: PASS
3 records: PASS
Matching Values: PASS
/// Add Crime ///
Count: PASS
Matching Values: PASS
/// Add Crime - Adjusted ///
Count: PASS
Matching Values: PASS
```

```
/// View Crime ///
Show InfoWindow: PASS
Content: PASS
/// Edit Crime ///
Matching Values: PASS
/// View Crime ///
Show InfoWindow: PASS
Content: PASS
/// Delete Crime ///
Count: PASS
```

Figure 120: Testing log showing console output after unit and integration tests

5.6 Limitations

Despite code optimisation and testing, the limit on the number of markers which can be displayed on the map at once was not resolved, therefore beginning to suggest this may be the limit of the API in terms of how many markers can be overlayed on a single map. At this stage in development, instead of redesigning and changing large portions of the codebase to try to increase the marker capacity, a limit of 50,000 markers is enforced. Similarly, importing external files with more than 7,500 records is not able to be correctly handled by the solution.

Another key limitation is the inability to filter imported crimes by their crime type, the crimes can only be filtered by the main and sub categories of crime that are available to be chosen from the provided dropdown lists. Lastly, although the currently filtered markers can all be deleted with one action and there is the ability to filter by a search radius from a specified location, the solution does not provide a way to delete markers within a uniquely specified area they each individually have to be clicked on and deleted, which is unintuitive when deleting a large number of markers. The problem becomes apparent after importing a large number of records and then perhaps wanting to delete all these newly added markers (essentially trying to undo the action).

Chapter 6 - Conclusions and Future Work

With the solution now developed, it can be said that the most significant achievement made is the enabling of the support for crime data dumps from official sources to be imported and displayed in the mapper. The researched existing solutions and an extensive search of other crime mappers has revealed that a feature or process to allow a user to themselves import crime directly from within the solution, is not yet offered. The closest currently offered is official crime data being imported into a database from the back-end by privileged users. The significance of this unique feature is also strengthened by being implemented as part of a web-based solution, which is inherently highly accessible

And with the implementation acting as a mapping platform, the first future work which can be taken up is to implement a greater set of analysis techniques. The only currently implemented analysis technique is a basic clustering of the markers showing (albeit weakly) the spatial distribution of crimes. These future analysis techniques can be entirely independent from the location of crimes and instead use the other implemented crime attributes to show different forms of analysis.

With the marker capacity of the solution currently being around 50,000, future work could also be directed at increasing this capacity. This may be in the form of completely redesigning the objects used to map the instances of crimes (implementing an alternative to marker objects) or instead redesigning how the map handles these objects, potentially using overlays to show additional markers to circumvent the limit on the number of API calls which can be made at once.

Lastly, the crime mapper in its current state could also be picked up and quickly modified to map instances unrelated to crime. At the time of writing this report, where the infection and death instances of a spreading disease are located, being mapped, came to mind.

Chapter 7 - Reflection

The most prominent learning experience taken over the process of developing and documenting this project has been the revelation of how critical the design stage of the software development process is. Being so eager to start the implementation to be finished with this stage earlier was actually counterintuitive. Rushing the design process meant that what needed to be done and efficient ways of providing implementation was not fully grasped and it is likely that more time was actually spent continually iterating with different implementations than would have been spent with a length design process and then successfully completing the information first time.

References

[1] U.S Department of Justice. (December 1999). *Mapping Crime: Principle and Practice.* [online] Available at: https://www.ncjrs.gov/pdffiles1/nij/178919.pdf

[2] Jerry H.Ratcliffe. (October 5 2001). *Crime mapping and its implications in the real world.* [online] Available at: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.491.8519&rep=rep1&type=pdf

[3] Chris Overall. (May 2008). *Crime mapping and analysis: filling the gaps.* [online] Available at: https://www.ee.co.za/wp-content/uploads/legacy/GisT-Crime%20mapping.pdf

[4] Google. (February 8 2005). *Google Maps.* [online] Available at: https://maps.google.com/

[5] Emily Thomas. (September 11 2016). *Screen-Shot-2016-09-07-at-11.36.58-AM-1024x562.png* [online] Available at: https://1dzi3m2noa4tyufv442laiv9-wpengine.netdna-ssl.com/wp-content/uploads/2016/09/Screen-Shot-2016-09-07-at-11.36.58-AM-1024x562.png

[6] Plumplot. (April 2019). *London-vehicle-crime-rate-comparison-map.png* [online] Available at: https://i.plumplot.co.uk/London-vehicle-crime-rate-comparison-map.png

[7] Lalitha Saroja Thota. (April 24 2017). *Cluster based zoning of crime info.* [online] Available at: https://ieeexplore.ieee.org/document/7905269

[8] Xiang Zhang. (September 9 2010). *Detecting and mapping crime hotspots based on improved attribute oriented induce clustering.* [online] Available at: https://ieeexplore.ieee.org/document/5568075

[9] B. Sivanagaleela. (October 10 2019). *Crime Analysis and Prediction Using Fuzzy C-Means Algorithm*. [online] Available at: https://ieeexplore.ieee.org/document/8862691

[10] Astari Retnowardhani. (June 19 2017*). Classify interval range of crime forecasting for crime prevention decision making*. [online] Available at: https://ieeexplore.ieee.org/document/7951409

[11] Risk Terrain Modelling. (2020). *Spatial Dynamics of Crime.* [online] Available at: http://www.riskterrainmodeling.com/overview.html

[12] Leeds University. (2010). *Risk Terrain Modelling (RTM).* [online] Available at: http://www.geog.leeds.ac.uk/courses/other/crime/risk-terrain/index.html

[13] Javed Anjum Sheikh. (2 October 2017). *IST: Role of GIS in crime mapping and analysis.* [online] Available at: https://ieeexplore.ieee.org/document/8065761

[14] Xifan Zheng. (12 August 2011). *A mathematical modelling approach for geographic profiling and crime prediction.* [online] Available at: https://ieeexplore.ieee.org/document/5982362

[15] Home Office. (2010). *Policing in the 21st Century: Reconnecting police and the people*. [online] Available at: https://assets.publishing.service.gov.uk/government/uploads/system/uploads/attachment_data/file/118241/policing-21st-full-pdf.pdf

[16] Home Office. (February 1 2011). *Police.uk*. [online] Available at: https://www.police.uk/

[17] CrimeMapping. (2020). *CrimeMapping.com*. [online] Available at: https://www.crimemapping.com/

[18] Interaction Design Foundation. (2020). *Fitts's Law | The Glossary of Human Computer Interaction*. [online] Available at: https://www.interaction-design.org/literature/book/the-glossary-of-human-computer-interaction/fitts-s-law
[19] Jakob Nielsen. (April 24 1994). *10 Heuristics for User Interface Design: Article by Jakob Nielsen*. [online] Available at: https://www.nngroup.com/articles/ten-usability-heuristics/
[20] Wikipedia. (April 27 2020). *Latitude*. [online] Available at: https://en.wikipedia.org/wiki/Latitude
[21] Wikipedia. (May 5 2020). *Longitude*. [online] Available at: https://en.wikipedia.org/wiki/Longitude
[22] rapidlasso GmbH. (May 6 2019). *How many decimal digits for storing longitude and latitude?* [online] Available at: https://rapidlasso.com/2019/05/06/how-many-decimal-digits-for-storing-longitude-latitude/
[23] DBQuest, Inc. (2020). *Top Reasons to use MySQL.* [online] Available at: http://www.databasequest.com/index.php/product-service/mysql-dbquest
[24] GitHub - MarkerClustererPlus for Google Maps V3 (2020). [online] Available at: https://github.com/googlemaps/v3-utility-library/tree/master/packages/markerclustererplus
[25] Google Developers. (2020). *Overview | Maps JavaScript API.* [online] Available at: https://developers.google.com/maps/documentation/javascript/tutorial
[26] Google Developers. (2020). *Autocomplete for Addresses and Search Terms.* [online] Available at: https://developers.google.com/maps/documentation/javascript/places-autocomplete
[27] Google Developers. (2020). *Markers | Maps JavaScript API.* [online] Available at: https://developers.google.com/maps/documentation/javascript/markers
[28] Google Developers. (2020). *Info Windows.* [online] Available at: https://developers.google.com/maps/documentation/javascript/infowindows
[29] Moment.js. (2020). [online] Available at: https://momentjs.com/downloads/moment.js
[30] Google Developers. (2020). *Marker Clustering | Maps JavaScript API.* [online] Available at: https://developers.google.com/maps/documentation/javascript/marker-clustering
[31] PortSwigger. (2020). *SQL Injection in Different Statement Types.* [online] Available at: https://portswigger.net/support/sql-injection-in-different-statement-types

Appendix

The source code repository for the implemented solution can be accessed by following the link (may require a suitable GitLab account in order to be accessed):
https://csgitlab.reading.ac.uk/hq017496/Crime_Mapper.git

Table 4: The average duration to import markers (before optimisation)

Number of Markers	Average Duration (ms)
10	6050
100	6050
250	6050
500	6050
1000	7040
2500	9740
5000	13000
7500	15100

Table 5: The average duration to import markers (before optimisation)

Number of Markers	Average Duration (ms)
10	6040
100	6050
250	6040
500	6050
1000	7450
2500	9040
5000	12700
7500	15000

Table 6: Additional integration tests

Integration Tests	Status
Import Crime + View Crime	**PASS**
Import Crime + Edit Crime	**PASS**
Import Crime + Delete Crime	**PASS**
Import Crime + Filter Crime	**PASS**
Import Crime + Analyse Crime	**PASS**
Load Crime + View Crime	**PASS**
Load Crime + Edit Crime	**PASS**
Load Crime + Delete Crime	**PASS**
Load Crime + Filter Crime	**PASS**
Load Crime + Analyse Crime	**PASS**
Edit Crime + Filter Crime	**PASS**
Edit Crime + Analyse Crime	**PASS**
Edit Crime + Delete Crime	**PASS**