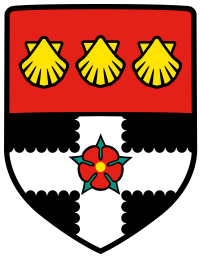


University of Reading

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

5) Identify applicable crime attributes to isolate specific occurrences of crime and classify them as descriptive or numeric, discrete or continuous (as they are being implemented)
6) Provide a process to specify crime attribute values for all applicable attributes 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, selecting the most commonly used format to be implemented
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 within 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.

1.6 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

Introduction

As an early indication, as the review was conducted, it was apparent that there was a high availability of literature (and existing solutions) for the initial concept of crime mapping but less availability for the concept of crime analysis and an even lesser degree of availability for material in relation to crime prediction or prevention, which indirectly highlights how well established and researched crime mapping is in contrast to how crime analysis, and to a greater extent crime prediction and prevention, seem to be in their early stages of adoption or have yet to have been fully explored (which provides an initial understanding as to the gaps of knowledge currently present in the area of research).

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 (*U.S Department of Justice, 1999*), 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 (*Jerry H.Ratcliffe, 2001*), 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, address or plots of land to go by (*Chris Overall, 2008*).

A consideration alluded to when using GIS applications for crime mapping is the degree of abstraction present in a map (*U.S Department of Justice, 1999*), the extent to which some information of the map is removed in order to better present its important information, is one of these considerations 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 (Google, 2005).

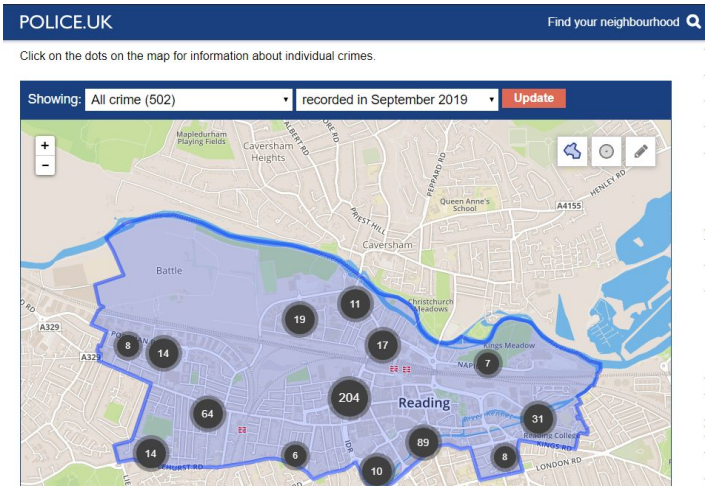
The way in which information is visually communicated through mapping can also vary. The types of maps frequently used in mapping crime include: **Point/Dot** - which as the name suggests involves using either a point or dot as a symbol on a map, **Statistical** - the use of charts (e.g bar and pie charts) as symbols placed in subdivisions of the map or **Area/Choropleth** - the shading of areas or regions to express information.

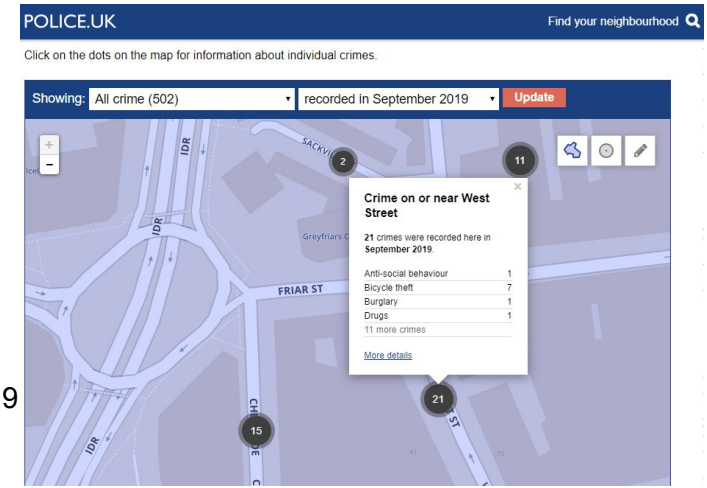
It is made clear that statistical maps are prone to overcrowding or overlapping of data points whilst also showing how area maps can be easily misinterpreted, highlighting the importance of considering the amount of data the solution to be developed may be required to accommodate and show (as well as the clairity and perceived accuracy of the patterns or information it shows) and such is why, to help manage this risk, the next section will involve the analysis of an existing implementation, which also gives a better impression of what has already been done in terms of crime mapping.

Police Crime Map

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 (*Home Office, 2010*). The information was released on the official crime and policing website 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 (*Home Office, 2011*) 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. These large marker icons dynamically change in size and divide into smaller marker icons when clicked on or when a different level of zoom is selected for the map in order to show more detailed levels of information. When the lowest level of marker icon is clicked on, a small information panel about the crime(s) is shown with a navigation link to an external webpage for the crime(s) in question. 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.





Shortly after the police crime map was launched, a newspaper article questioned its practicality. The writer proposed there was a degree of doubt amongst third party developers as to the reliability of the data made available and went as far as including a viewpoint suggesting the implementation had hidden political motivations and did not benefit society in the ways that it claimed to do so. Despite a somewhat obvious conflict of interest (involving the police providing and presenting data which reflects highly upon themselves and their own performance), the knowledge that unsolved crimes, crimes sensitive in nature or crimes in which the legal process is still ongoing seems a better justification as to why a false impression in terms of the reliability of data may be put across. This consideration highlights the challenge of balancing privacy and accuracy in crime mapping. Another voiced concern was how only one month of data could be shown at any one time but this has since been updated so that a filter can now be used to show crime during different months in the past, a significant feature allowing for identification and analysis of crime trends.

Observations

+ Hierarchy of information with both high level and more detailed representations of crime
- 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)

Effect on solution

Being limited to viewing information for only a single neighbourhood is restrictive and hinders the user experience which is why the intended solution would enable users to view (and map) crime across different geographical locations more intuitively and in terms of filtering, a better approach would be a checklist or allowing a range of values to be chosen. The analysed solution also highlights the presence of robust mapping platforms, APIs and libraries, which the intended solution could make use of in order to streamline the development process.

Crime Analysis

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 (Lalitha *Saroja Thota, 2017*). 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.

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 (*Zhang, 2010*) 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 (*B. Sivanagaleela, 2019*).

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.

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 (*Astari Retnowardhani, 2017*). 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.

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) (*Spatial Dynamics of Crime*). 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 (*Risk Terrain Modelling, RTM*) 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" (*Javed Anjum Sheikh*). 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 (*Xifan Zheng*).

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 prediction/prevention techniques for the intended solution if it wants to stand out in the field. Whether 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 Requirements Specification

Building on the aims and objectives outlined in the ‘Introduction’ chapter, the software requirements of the solution are next to be defined.

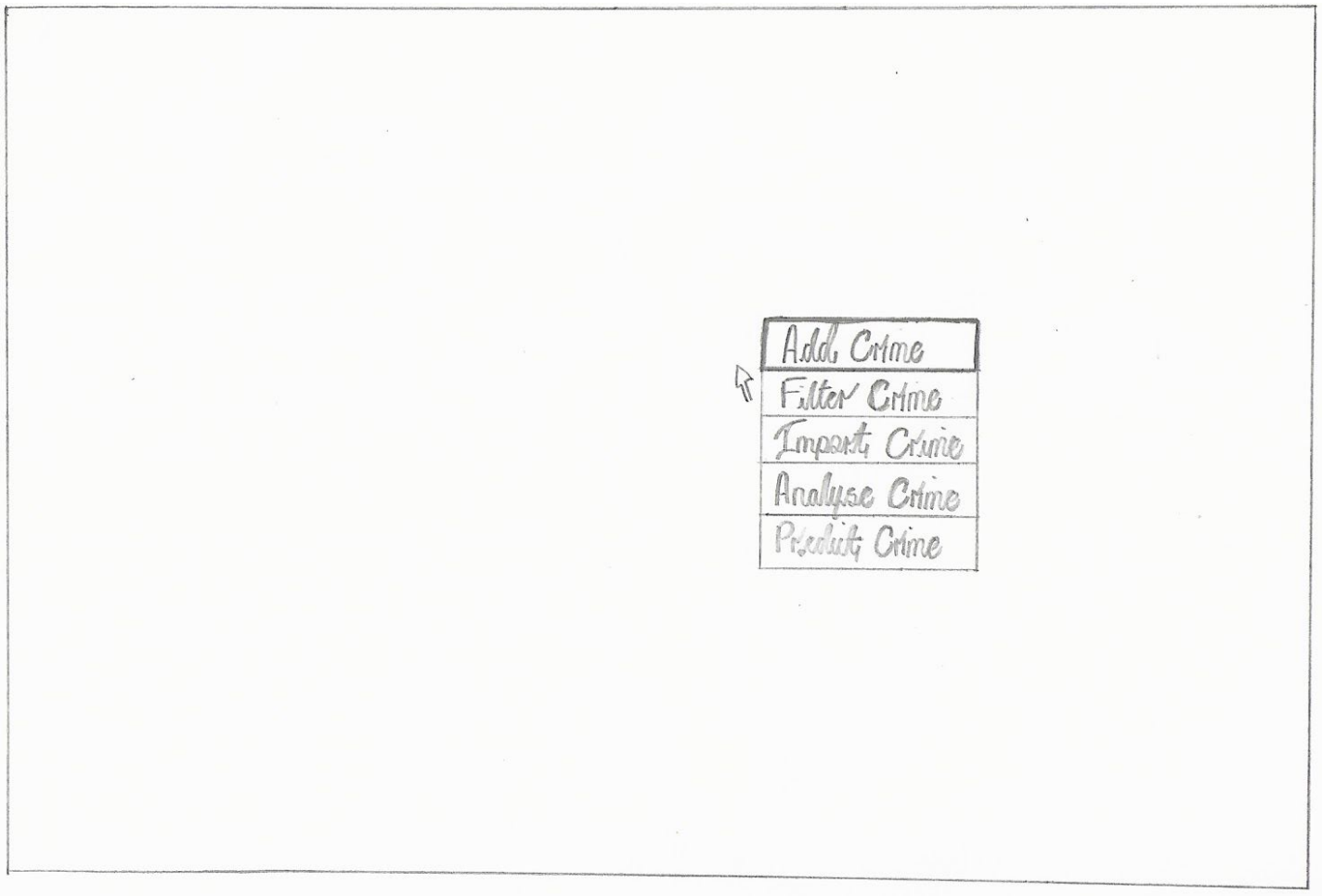
3.2 Design

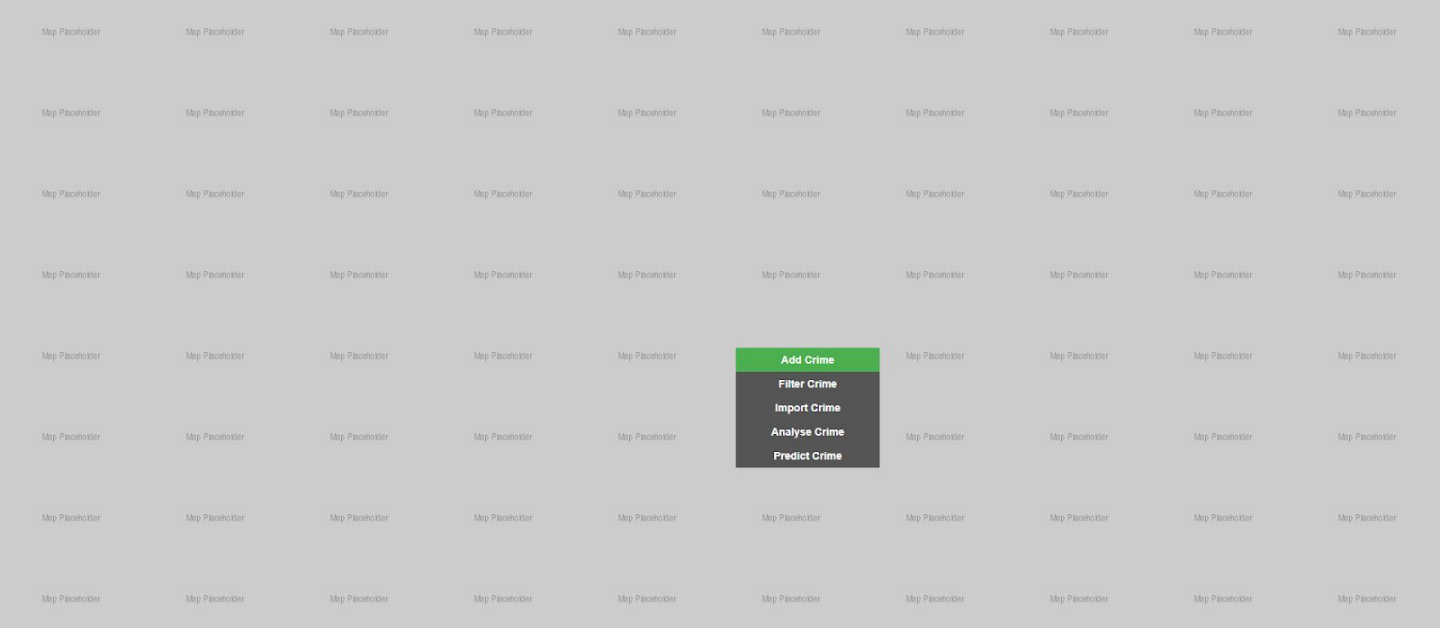
3.2.1 User Interface (UI)

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.

3.2.1.1 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 next produced:



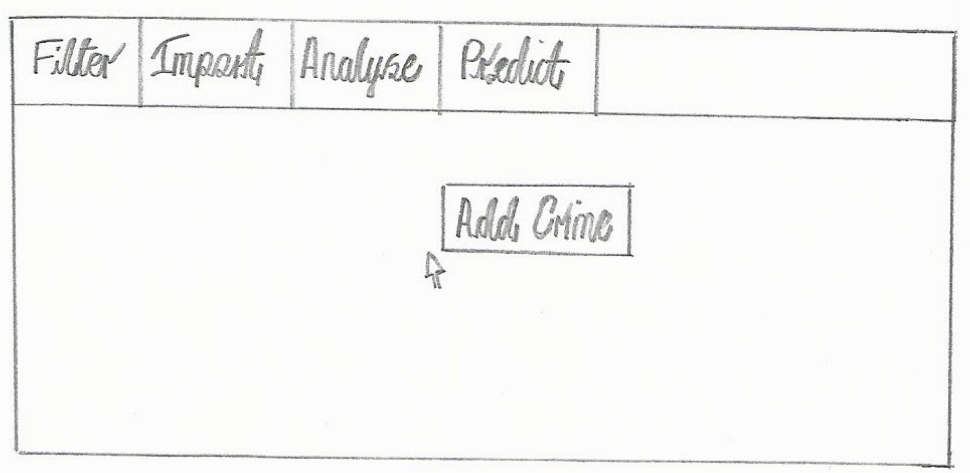


Whilst the context menu seemed for all intents and purposes suitable initially, when the high-fidelity prototype underwent usability testing, the problem of users accidently choosing the wrong option was frequently reported and along with it uncertainty as to what the precise nature of this problem was. 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. Whilst increasing the universal size for the options seemed like an easy fix for the design, after follow up testing, although made less common the problem was still arising, and after due thought, this highlighted the importance of consideration towards the different screen sizes the user may be using. And so a solution to this particular design setback could have been to increase the size of the options in addition to scaling it to the size of the display but this would have inadvertently caused a tradeoff with travel time (the time taken to navigate to an option) as larger sized options would mean the last options in the context menu are further away from the top of the context menu (where the menu was requested). 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' Law). 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.

Toolbar and Context Menu (Revised Design)

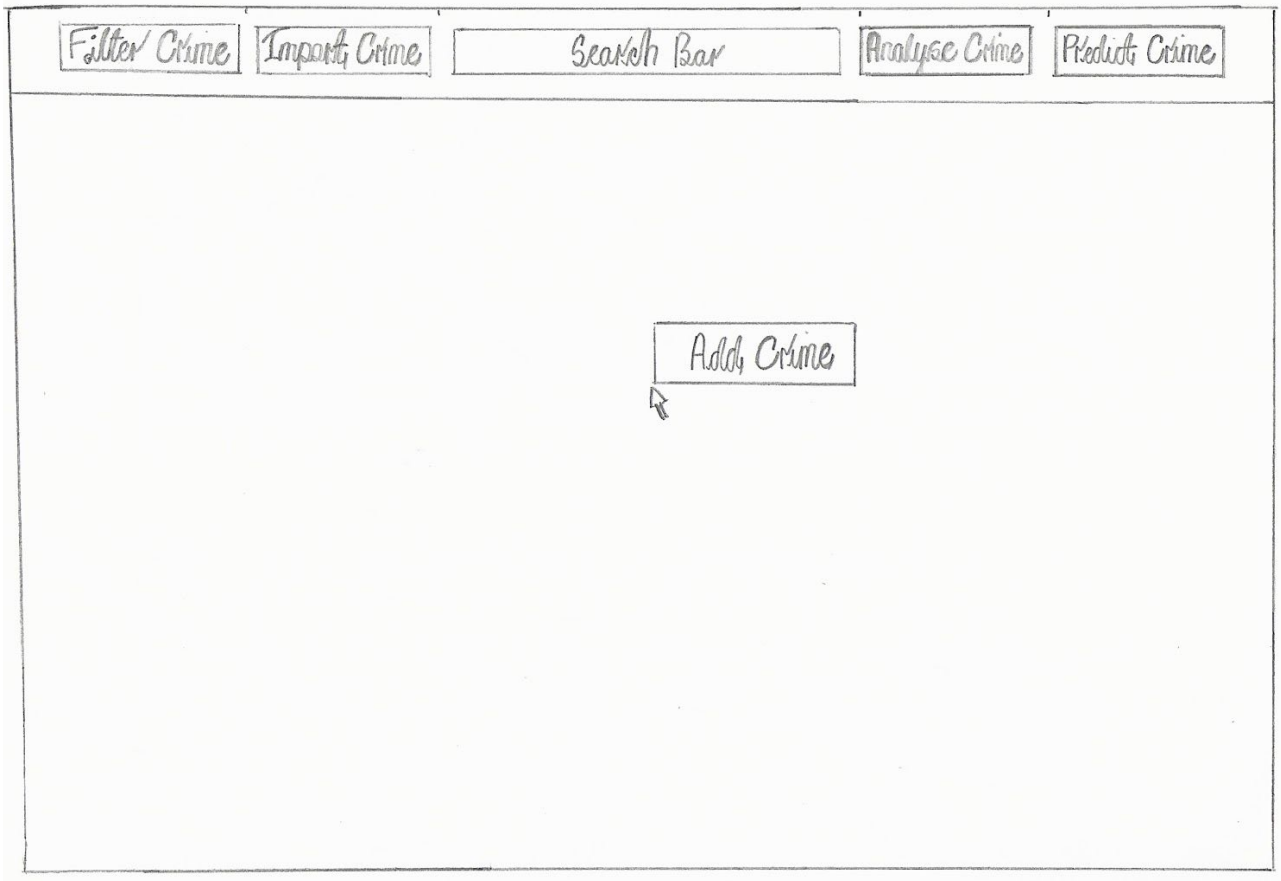
With this in mind, the next revised design made a small compromise in terms of the screen space for the map by adding a single navigation bar or toolbar across the top of the screen and migrating all but the ‘Add Crime’ option to it (leaving just the ‘Add Crime’ option as part of the map context menu):



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.

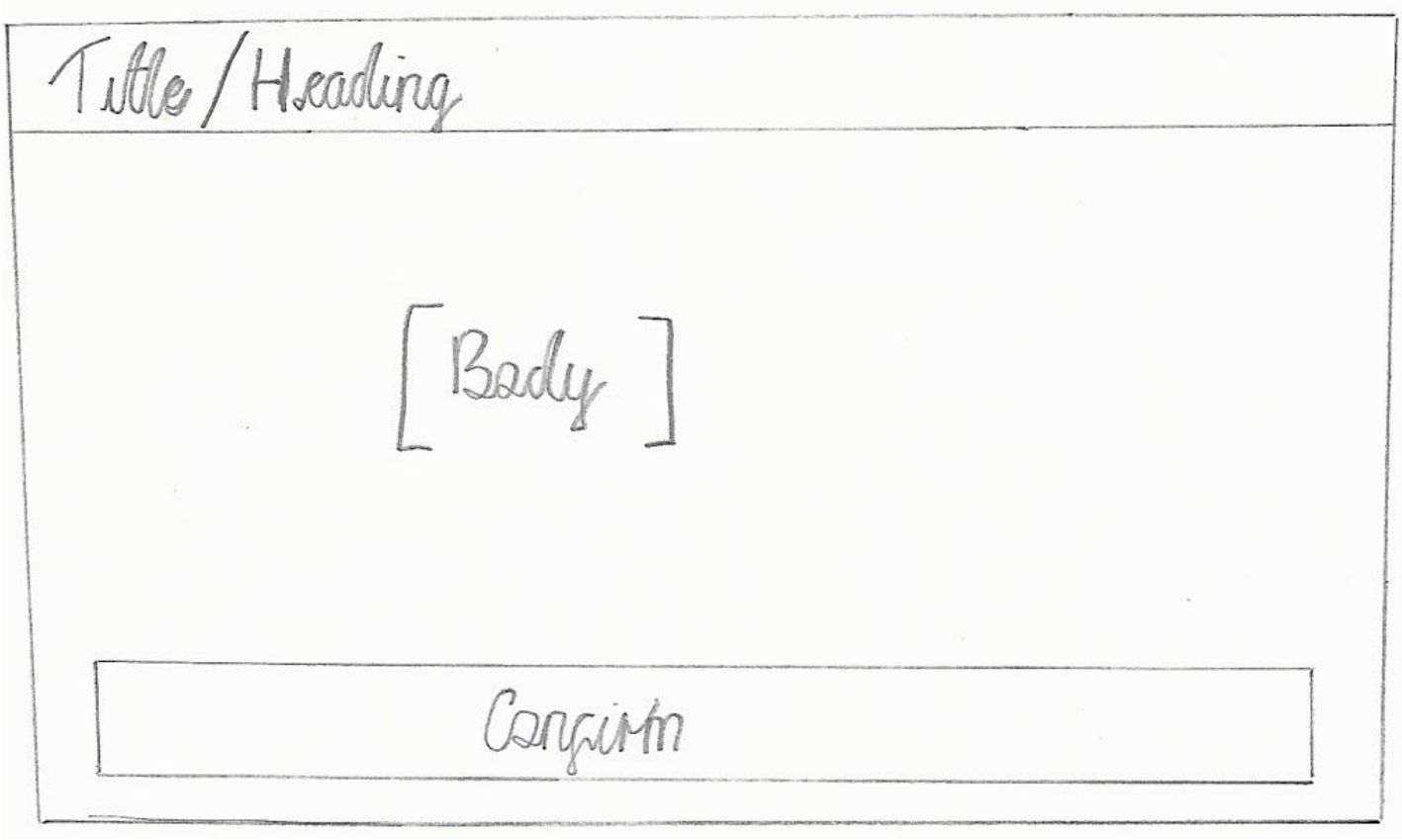
Toolbar with Search and Context Menu (Final Design)

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



Popup Window - Base Design

Before designing the interface for each individual popup window which are now designed to open either by the options in the toolbar (in the case of ‘Filter Crime’, and ‘Import Crime’ but NOT ‘Analyse Crime’ or ‘Predict Crime’ as these actions will not require any further input beyond simply requesting the operation) 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 usability (Jacob Nielsen’s usability principles). To further ensure consistency of the interface the designed colour for the headers of these popup windows is chosen to be the same colour as the toolbar. Similarly, to ensure another usability principle of user freedom, the popup window is designed to be closed at any time using an appropriately shaped close button in the top right of the window, the only other time the popup window would close is when the confirmation button (coloured green which semantically indicates progression of a successful action) is clicked to prompt its functionality:



Being just a base design, the title would evidently change based on the functionality the individual popup windows provide as would the content of the body and it may even be the text of the confirmation button also varies across the different popup windows.

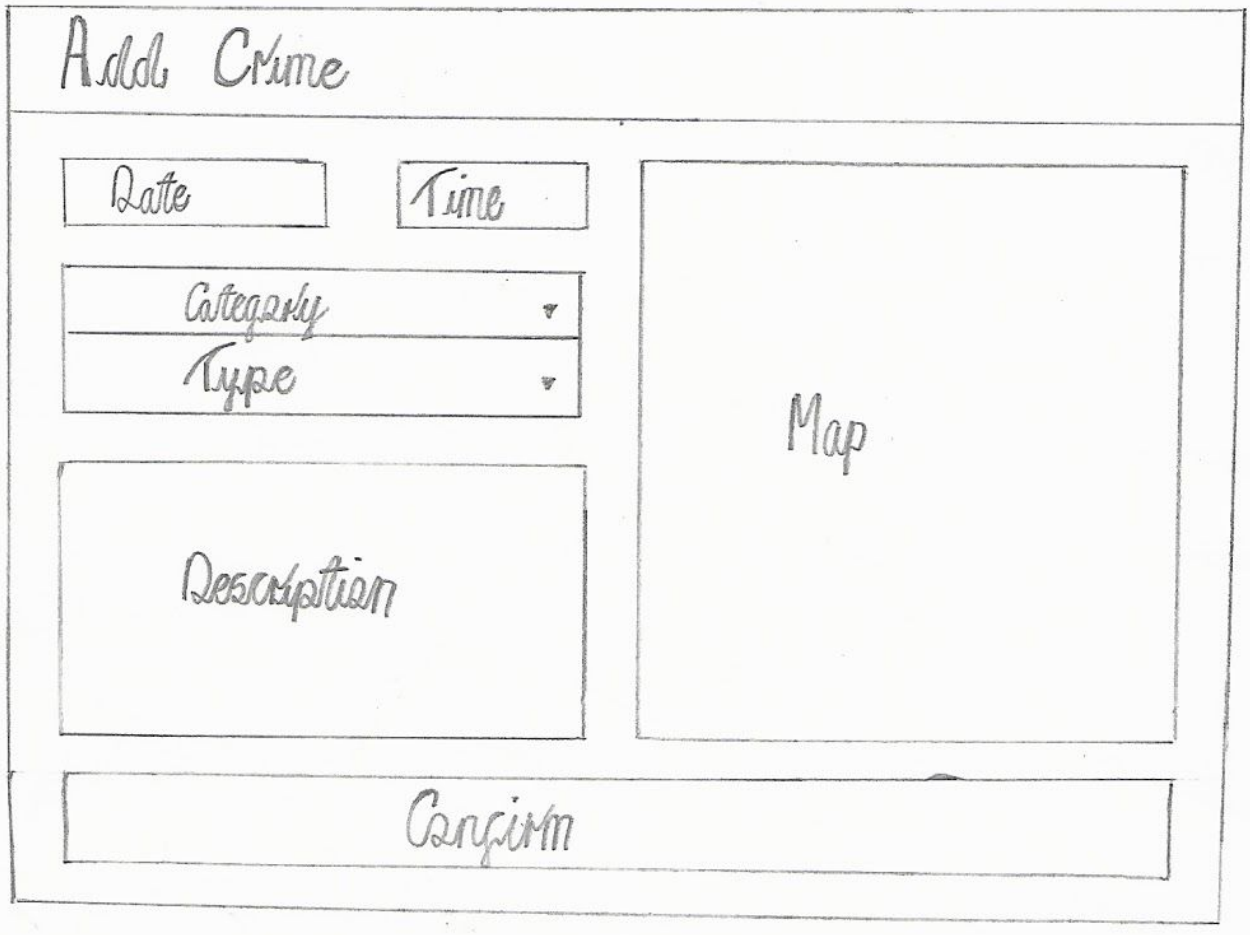
The size of the title and confirmation button should be kept constant for all the windows but the overall size of the window can change and be as large as the body requires it to be (but no larger than necessary).

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 are designed to provide the processes described in some of these objectives), now is a good time to mention what they attributes have been identified to be used throughout the solution:

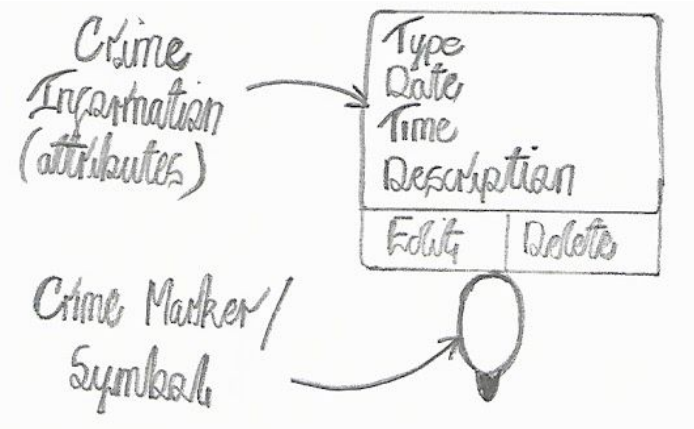
Type - the name given to the crime or type of offence committed (that most suitable describes the crime)
Date - the date of when the crime took place (or failing that when the crime was recorded)
Time - the time of when the crime took place (or failing that when the crime was recorded)
Location - where the crime took place or is believed to have taken place
Description - any further relevant details of the crime

Popup Window - 'Add Crime' Design



Mapping and Viewing Crime

The process of mapping a crime is 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 on the map is not yet designed and so the following design drawing conveys what is shown to display a crime as well as how the crime attributes describing it are shown:

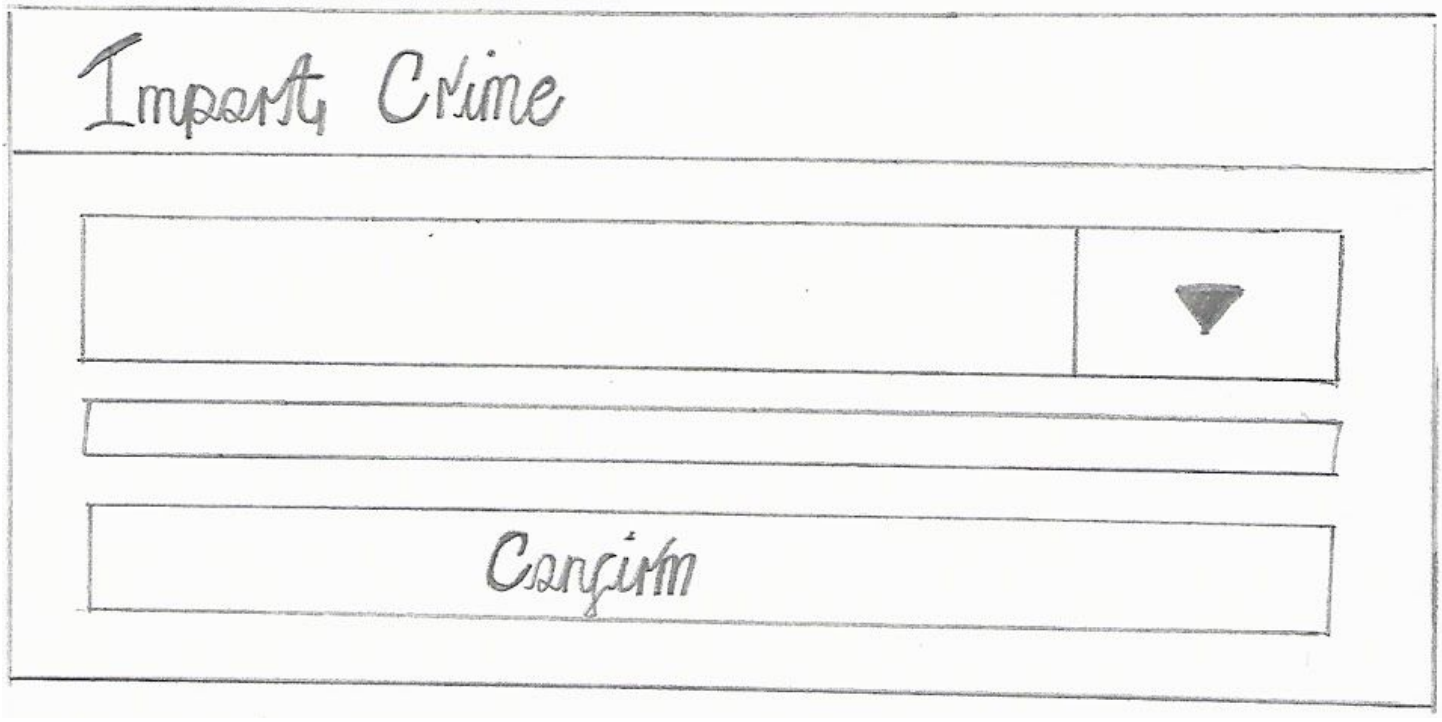


This design introduces two additional options, ‘Edit Crime’ which is designed to open another individual popup window targeted directly at addressing the fourth objective of the project as well as a ‘Delete Crime’ option which as the name suggests will be designed to remove the symbol or marker shown to represent a crime.

Popup Window - ‘Edit Crime’ Design

The popup window for this newly introduced option is designed to match with the ‘Add Crime’ popup window as the functionality they are designed to provide is very similar (simply amending existing crime requires the same input and therefore the same body of the window as simply defining new crime attributes).

Popup Window - ‘Import Crime’ Design



3.2.2 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 would achieve 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 to include.

Conceptual Design

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 is given the name *'markers'*.

Logical Design

In terms of the columns present in this single table, they should reflect the information that is entered by the user when they are adding a crime (both in their name and as later explained in the physical model, their data type), along with any additional columns required to ensure a reliable structure for the database such as a primary key column which should uniquely identify each record (in other words each row of the table or each instance of crime). Referring back to the user interface design related to providing the functionality of adding a crime and the crime attributes, the designed columns for the table are *'Crime_ID'*, *'Crime_Date'*, *'Crime_Time'*, *'Crime_Type'*, 'Latitude', *'Longitude'* and *'Description'*.

Important or notable design choices include 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 is to avoid predicted complications with reserved keywords (either in SQL when creating the table or when later using other programming languages for querying/referencing the table). If these predicted complications can however be managed during the implementation stage of the project's development, the implemented column names can however be changed.

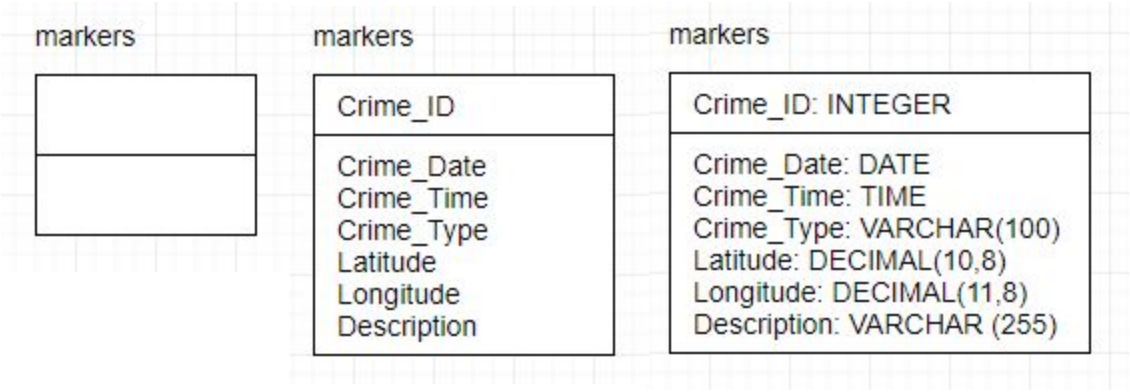
Physical Design

As stated previously, the '*Crime_ID*' column should be enforced as the primary key being of an integer data type and importantly with 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 date and 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 indeed left unchanged with the date column having a 'date' data type and the time column having a 'time' data type.

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 **[1]**, the decimal data type is 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 should not be. There is a small difference in terms of the range of values they represent 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 influencing 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, being nominal values and of variable length, the varchar data type should be employed, 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 entered to these lengths that the database can store would be good practice (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).



Database Management System (DBMS) and Administration

With the solution being web-based, MySQL will be the database management system of choice and for administration of the database, phpMyAdmin is to be used. Using phpMyAdmin, a database should be first created with the name ‘crime_mapper’, and the ‘marker’ table designed should be added to it.

3.2.3 Website Structure/Information Architecture

The website will be hosted and managed using cPanel with the aforementioned tools provided through its control panel being used to host the database along with the website (pages) on a server. Any information or data sent to the server such as user input is to be sent using the client-server transmission method of AJAX.

3.2.3.1 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 the 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 x.x) 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 is to be used to implement an interactive Google Map as the mapping component for this page. This webpage will also include back-end components to facilitate a connection to the designed database (Section x.x). In order to connect to the MySQL server this database will be hosted on, a scripting language will be required. The scripting language used will be 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:

```
Include 'dbConfig.php'
Include 'layout.css'

Display Mapping Component (Google Maps)

Query all database information
Locally store database information
Place markers on mapping component using information

Toolbar 'Filter Crime' button clicked
    Open 'Filter Crime' popup window

Toolbar 'Import Crime' button clicked
    Open 'Import Crime' popup window

Context Menu 'Add Crime' option clicked
    Open 'Add Crime' popup window
```

3.2.3.2 Page Styling (layout.css)

In order to control how elements are displayed for this front-facing webpage, CSS will need to be used which can take the form of either inline styling, where the styling is applied uniquely for each individual element, internal styling, where the styling is defined uniquely for (and within) each webpage or external styling, where all CSS is defined in a separate file and referenced or linked to within each page it needs to be applied for. 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 seems 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 is designed to be used, which will be given the filename *'layout.css'* along with some inline styling if absolutely required.

3.2.3.3 Database Configuration (dbConfig.php)

Any remaining back-end functionality related to the database is to be 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) can be 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
```

3.2.3.4 Add Crime (SaveMarkers.php)

Requesting to add a crime to the map would require for a new record to be made in the database in order for it to be permanently stored. The related information about the crime that has been entered by the user in the 'Add Crime' popup window would need 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 is designed to be sent back from the server (returned) and included in the local copy of the database's information (along with the other information of the crime). That way, if in the future, a particular added crime wishes to be edited or deleted, it is known which record in the database must also be edited or deleted to reflect these actions (a concept further explained in the next two subsections):

```
(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
```

3.2.3.5 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 deleting the record and inserting a new record with the updated information, the updated information about the crime or input entered by the user in the 'Edit Crime' popup window should 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, all of which will be handled or processed by a file named '*EditMarkers.php'*:

```
(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
```

3.2.3.6 Delete Crime (DeleteMarkers.php)

Similarly, when requesting to delete an added crime that would also mean a record for that a crime already exists in the database and so that record of the crime should be designed to be deleted. To know which record to delete, the local copy of the ID for that crime should be sent to the server and this ID can then be used as part of a SQL DELETE statement to permanently delete the record in the database. As with every other one of these additional PHP files this would all be handled by a file named, '*DeleteMarkers.php'*, reflecting the action it processes and performs:

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

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

3.2.3.7 Import Crime (ImportMarkers.php)

Within design for adding and record a single crime with '*SaveMarkers.php*', importing crime (multiple or a large number of crimes at once) can be 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 can now be said as having been identified as text files. With the interface of the relevant 'Import Crime' popup window including an input to select a file, the file to be imported can 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 should be 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

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

3.2.3.8 Filter Crime

The remaining options or actions that filter, analyse and predict crime, which are reminded as not opening their own popup windows, also don't have their own separate PHP files associated to them as their functionality is designed to be implemented as part of the main index.php file, making use of the local copy of crime markers and not requiring any interaction with the database. For the filtering of crimes, the information entered into the 'Edit Crime' popup window can be compared against this local copy with any crimes not matching the criteria being temporarily hidden from view (and not the database as it would be both strongly unintuitive and redundant to add/remove information as and when it is filtered).

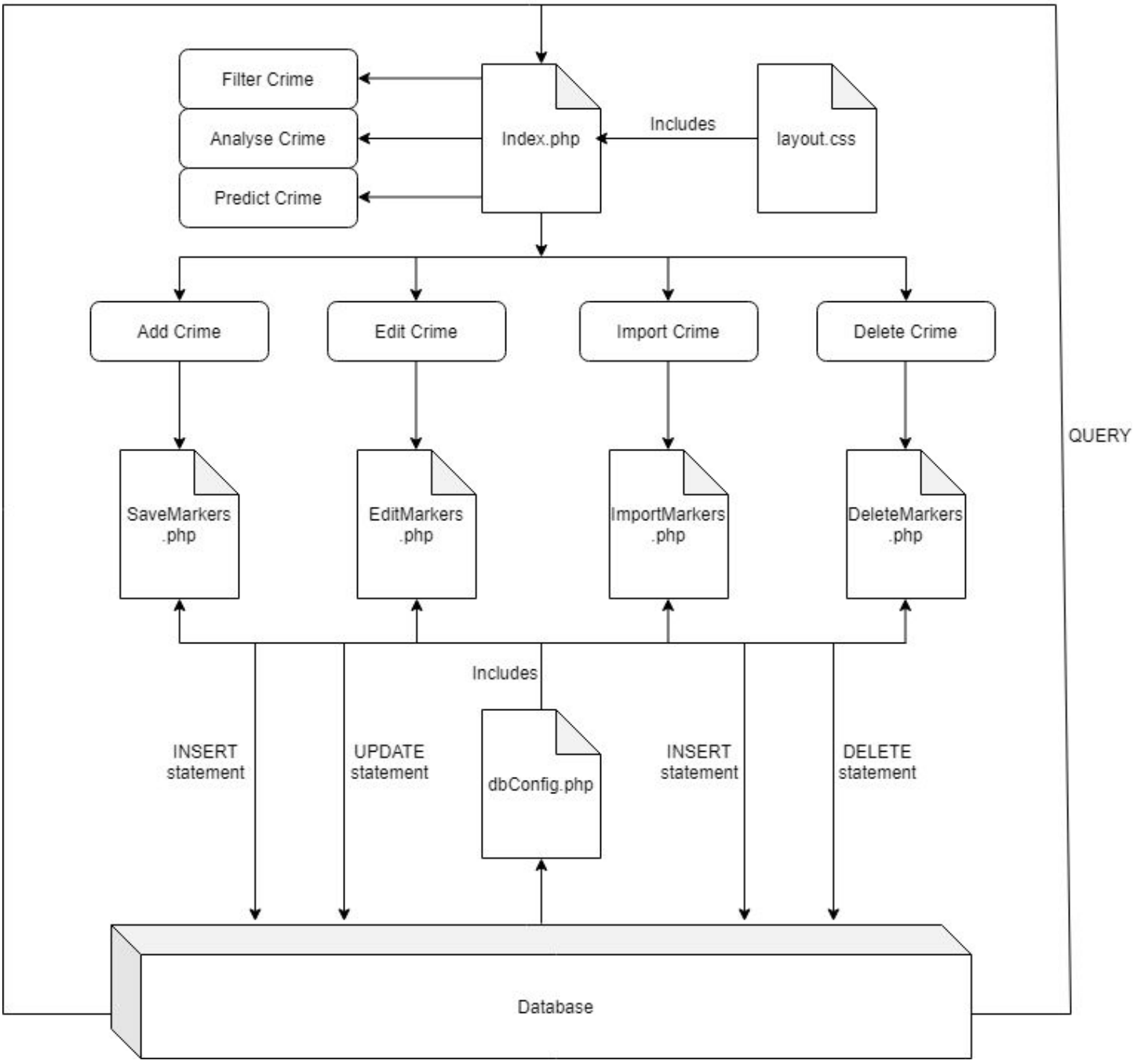
```
'Edit Crime' popup window confirmation button is clicked
    Record filter criteria

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

3.2.3.9 Analyse Crime and Predict Crime

The chosen and designed analysis method for crimes is clustering but as opposed to designing and implementing an algorithm, the Google Maps MarkerClustererPlus library can be used and applied to the crimes plotted or added to the map instead. The analysis provided using the library can then 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 plotted as any other crime is designed to be but with the important detail of the information of the crime not being added to the database, as these predicted crimes should only be created and shown when requested (when the 'Predict Crime' toolbar button is pressed) and not by default.

3.2.4 Website Structure Diagram

To summarise the web pages designed and how they are all interconnected to form the structure of the website as well as the database is interacted with, 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):



3.3 Implementation

3.3.1 Preliminaries

To build the structure of the website, the webpages outlined in the design section were first created and renamed as needed, with it being made sure that all these webpages were kept collectively under the same directory (within a folder). The only of these webpages that is designed to have external dependencies is the main index.php file, but before referencing the file locations or web addresses of any of these dependencies, some setup for this page is advisable in order to help ensure maintainable code is produced (which uses a consistent use of style) and the correct displaying and scaling of content.

3.3.1.1 Declaration

As the first line of the page, the version of HTML the page uses or is written with is 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.

3.3.1.2 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 is used. This acts as the root element with any and all elements of the webpage being contained within its opening tag and closing tag.

3.3.1.3 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, is created using a <head> element. Any applicable metadata tags or elements (three are to follow) should be placed within its opening tag and closing tag.

The <title> tag, which as the name suggests defines the title of the web page displayed at the top of the browser window and shown in any browser tabs for the page, is an example of metadata implemented. As per the project title, the title of the webpage was set to 'Crime Mapper'

A coding convention of web development applied across most websites is the setting of the viewport, instructing the browser as to how to control the area for which web content can be seen, which will of course vary across different devices with mobile devices usually having smaller screen sizes. The viewport for the solution is set by including attributes that match the page width with the screen width and that set the initial zoom of the web page to a default value

The style sheets used to control the layout and formatting of elements within the webpage are also added to this container with one of these stylesheets linked to being the CSS provided as part of the Bootstrap framework (which was stated to be used in the Section x.x) and another being the unique style sheet (designed in Section x.x and earlier created, the file named *'layout.css'*). An absolute location is used to reference the web hosted CSS of BootStrap whereas a relative location is used to reference to the layout.css file (as the index.php file and layout.css file are within the same directory).

Content

The content of the web page, or the <body> containers:

```
<!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>
```

Database

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

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

#	Name	Type	Collation	Attributes	Null	Default	Comments	Extra
1	**ID** 🔑	int(5)			No	*None*		AUTO_INCREMENT
2	**Crime_Type**	varchar(100)	utf8mb4_general_ci		No	*None*		
3	**Crime_Date**	date			No	*None*		
4	**Crime_Time**	time			No	*None*		
5	**Description**	varchar(500)	utf8mb4_general_ci		No	*None*		
6	**Latitude**	decimal(10,8)			No	*None*		
7	**Longitude**	decimal(11,8)			No	*None*		

3.3.2 Mapping Component

With the required API now able to be accessed, the mapping component decided upon in the design can be implemented for this newly created and setup main web page. To begin with an area needs 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 is assigned and used, appropriately being set as 'map'. These percentage values are subject to change however, as some space will need to be reserved for when the static toolbar at the top of the page is to be implemented.

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

With the map being critical in providing the website's functionality, it's 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 is made to the script that loads the API.

A callback feature is added, a feature which allows for a function of code to be called once the API is ready. Therefore, a function is developed, which is made to include the JavaScript code to create the map and is added as part of this callback feature:

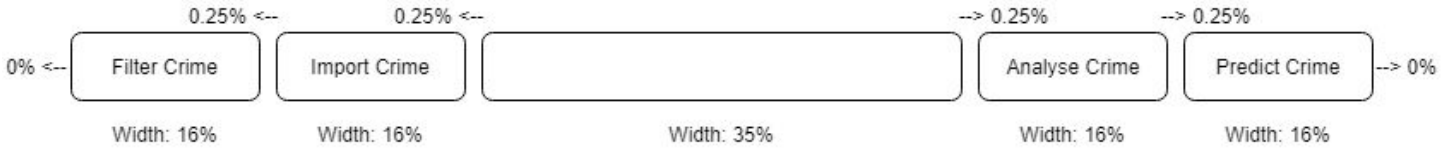
```
17  <script>
18  function initMap() {
19      var Reading_loc = {lat: 51.454266, lng: -0.978130};
20      var map = new google.maps.Map(document.getElementById("map"),
21                                    {zoom: 8, center: Reading_loc});
22  }
23  </script>
24
25  <script src="https://maps.googleapis.com/maps/api/js?key=?&callback=initMap"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

3.3.3 Toolbar

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

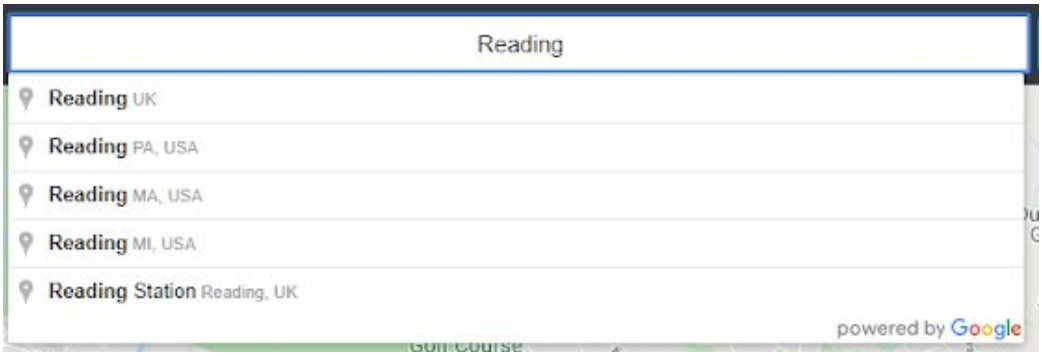
The search bar is also added to the toolbar using an autocomplete widget, namely the SearchBox class (of the Google Maps JavaScript API). In order to construct an instance of the SearchBox class, a text input is required to be made, acting as the location for user input (a search) and for where any results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

Before implementing the functionality of the search bar, the formatting and enforcing the layout of these elements within the navbar should first be implemented. The intended design exhibits the four navigational buttons being of equal width and the search bar being of a considerably larger width as compared to these buttons 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 the margin-left or margin-right modifiers), resulting in the layout (where the widths and spaces add up to the entire screen's width of 100%) as seen in the below visualisation:



Before implementing the functionality of the search bar, the height of the mapping component must be updated to accommodate for this static toolbar, otherwise the map will 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) is 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):



But the functionality to update the map's location based on the selection of one of these predictions must still be implemented. This is 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
    });

}
```

[Caption needs to indicate this particular module of code is adapted from: https://developers.google.com/maps/documentation/javascript/places-autocomplete]

3.3.4 Placing Markers

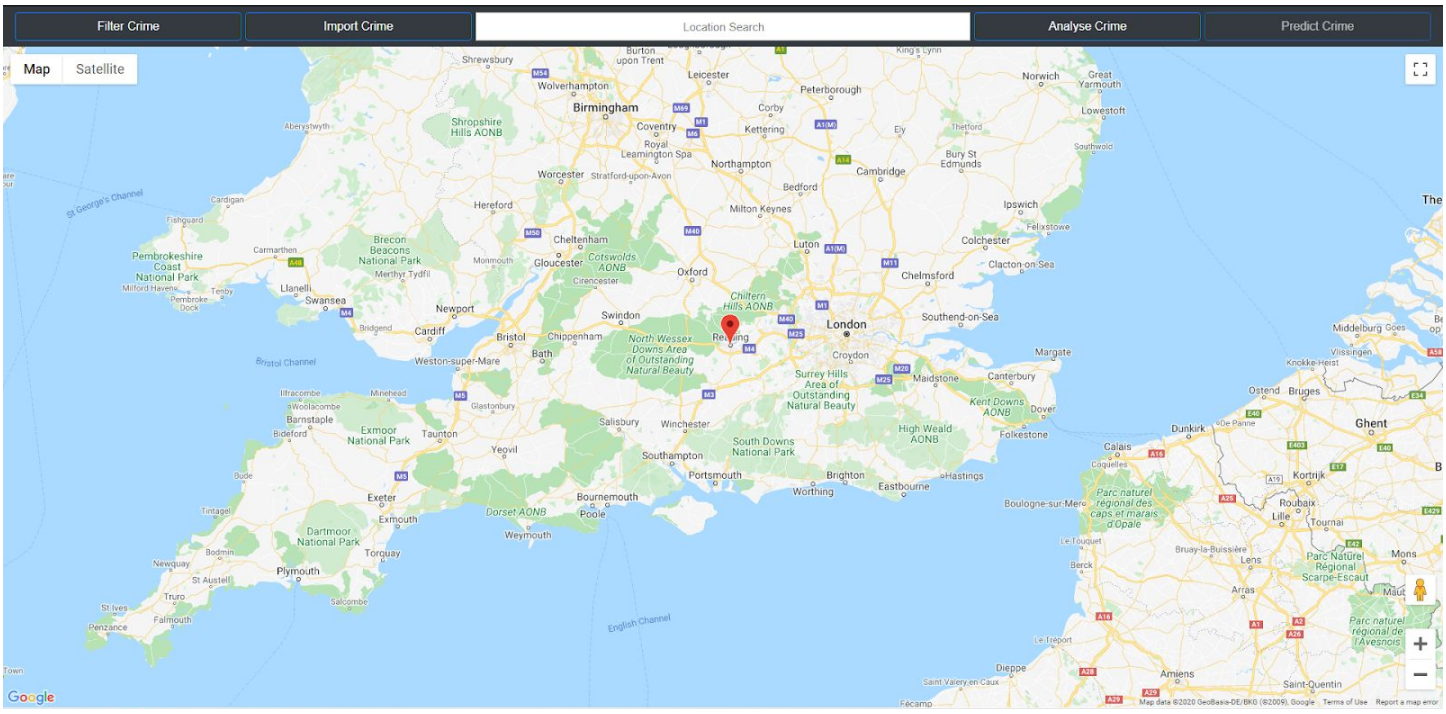
The Google Maps JavaScript API has native Marker objects which can be constructed and added to the map component, which resemble the design in Figure x.x. To reach the designed workflow of being able to place a marker at a specified location on the map with a mouse click (and a context menu being opened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

3.3.4.1 Predefined Marker

The first step in this approach can be to simply add a marker along with the map when it is created. This is 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 is 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 can be re-used as the location for this marker. For the marker to be shown with the map when it loads, this definition of the marker is 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>
```

This addition results in the strived for output for this step, of a marker placed at the center of the map as seen below (the toolbar implemented in the previous section can also be seen for the first time):



3.3.4.2 Marker at map click location

The next incremental step can be 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 can be 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  });
```

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 can be used 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  });
```

3.3.4.3 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 x.x) should next be implemented as opening with a right click on the map. The HTML (elements) needed to create this context menu should first be developed. The entire context menu can be derived with a div element, whilst the sole option of this context menu can be derived by nesting an additional div element inside this element (both of which are styled as required in the external CSS file):

With the context menu only designed as appearing dynamically (it is not designed to be shown all of the time) these elements should importantly not be made initially visible and so to begin with they are hidden using CSS (the display property of the elements) 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 can be shown by unhiding the elements and moving their 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  });
```

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:

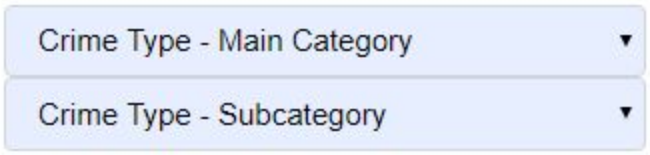
```
81  const add_btn = document.getElementById("btn_add"); // 'Add crime' button (in context menu)
82      add_btn.addEventListener('click', event => {
83          placeMarker(Location, map); // Place a marker with latLng object
84      });
```

3.3.4.4 Pop-up Window

The penultimate step for this module of code, involves 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 act as an additional form of confirmation.

Using BootStrap’s JavaScript modal plugin, this pop-up window can be 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 x.x) 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).

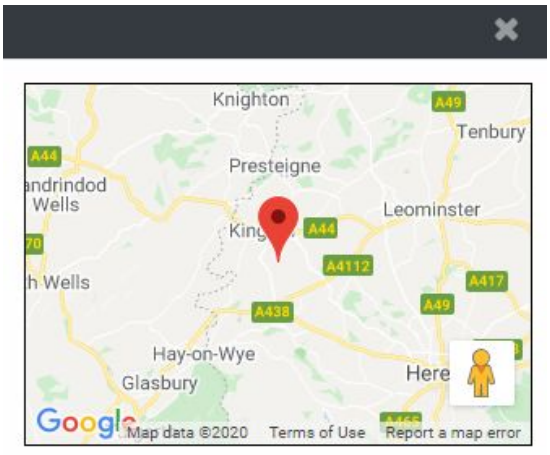
In terms of the two select elements, with the selection of the category of crime envisioned as narrowing down the possible types of crime that can be selected (categorising them), the first select element used for selecting the category of crime should be implemented as dynamically changing the selectable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



This involved first 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 are 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") { // ...
```

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.



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:

```
var map2 = new google.maps.Map(document.getElementById("map2"), SmallMapOptions); // Show smaller map

var Draggable_marker = new google.maps.Marker({ // Add a single draggable marker to smaller map
    position: Location, // At location of click location
    draggable: true,
    map: map2 // On this smaller map, not the main map
});
```

All the popup window's inputs are 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 for now, after repurposing the click event of the context menu to now just open the popup window (and no longer place a marker), a marker is 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 was made):

```
google.maps.event.addListener(Draggable_marker, 'dragend', function (evt) {
    Location = evt.latLng; // Adjust (update) position
});

$("#add_submit_form").submit(function(e) { // Confirmation button is clicked
    placeMarker(Location,map); // Place a marker on the main map
}
```

However, as of yet, this implementation does not bundle the other information (the crime attributes) with the marker and so to associate information with marker objects, some properties can be assigned to them when they are created. These properties can then be set to the values specified in the popup window. With the only way to create marker objects being to use the placeMarker() function (Section x.x), this function needs to be added to, in order to support markers having properties:

```
function placeMarker(Crime_Type,Crime_Date,Crime_Time,Description,Location,map) {
    var marker = new google.maps.Marker({
        Crime_Type: Crime_Type,
        Crime_Date: Crime_Date,
        Crime_Time: Crime_Time,
        Description: Description,
        position: Location,
        map: map
        });
}
```

The values of the inputs in the popup window, when the confirmation button is pressed, which are trivial to obtain using JQuery or even standard JavaScript), can then be assigned as these properties, using this newly updated function.

3.3.4.5 Storing Markers

The last stage for this module of code, of adding crime to the mapper, is 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 must also now be stored to the database. The data to be sent can be 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 can then be 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  }
```

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  ?>
```

Some final implementation changes for this module include to ensure 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 can be implemented by working within the success function of the implemented AJAX call, 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 is 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    });
```

3.3.6 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, should next be implemented. This is implemented using PHP and a SQL query from within the main webpage (a separate PHP file is not required). The query involves 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    ];
```

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     }
```

3.3.7 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 are currently not shown or displayed in any way. Therefore, development moved to implementing a way of presenting the properties for a marker when they are requested to be viewed.

This is 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 is 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     });
```

Whilst most of these properties could be mapped directly and displayed as they are, 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 have already been made in other parts of the solution and to ensure consistent standards, this later format should be the format shown with the conversion being achieved using an external library (moment.js) which has 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     }
```

Whilst, the InfoWindows are created for markers when they are placed, in order to be shown these InfoWindows need to be actually opened. This is 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 in question is opened:

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

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

This is achieved by assigning this 'Crime Type' property as the value for the marker's 'title' property, a reserved property for a marker (which has the functionality as just described and shown):

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

3.3.8 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 using the process described in (Section x.x, the ‘Loading Markers’ section). 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 is 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
```

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.

3.3.8.1 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.

3.3.8.2 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.

3.3.8.3 Database

Deleting the record of the marker in the database is 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 (or sent) ID.

The deletion process is 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     }
```

The code to delete a marker at this database level is 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 ?>
```

3.3.9 Editing Markers

The InfoWindows are 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 (refer back to Deleting Markers section if these terms are not familiar), once the popup window's confirmation button is clicked.

3.3.9.1 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);
```

But for one of the select elements, this approach is 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
```

3.3.9.2 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) and the only other differences being a change in the destination file (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.

3.3.10 Filtering Markers

The local array of markers can be consulted when implementing the functionality to filter markers as no changes to or interaction with the database is required (filtering is the temporary hiding of markers from view by search criteria) with a modal that has inputs facilitating filtering by a minimum date, maximum date, minimum time, maximum time and by crime type being implemented.

Search Criteria and Validation

With the date and time elements initially being set as having empty values and the crime type element being set to a value of ‘ALL’, determining which of these inputs had been changed from their initial values when the confirmation button (to filter) is pressed and which had been left unchanged is a good starting point. Boolean values initially set to true for the date and time inputs were created and in the case of an input being empty, the corresponding boolean value was set to false. Another boolean value is used for the crime type and is initially set to true, but in the case of it being changed is set to false. In the case of the value of an input changing from the initial value (a value being entered), the value is recorded to be later used in filtering.

Checking different combinations of these boolean values (which indicate the presence of any changes made to the inputs) is then used to perform basic validation checks on the entered search criteria with a boolean value being used to flag if the search criteria is invalid. This is implemented to flag in the following scenarios:

- If values are entered in both the minimum date and maximum date fields, but the minimum date is a date after the maximum date
- If values are entered in both the minimum time and maximum time fields, but the minimum time is a time after the maximum time
- If a value is entered in only one of the two time fields

(**Note:** Although having just one date value to filter by is valid with a minimum date filtering by all dates after this date and a maximum date filtering by all dates before this date, having just one time value is considered invalid because although having one value could signify any values before or after midnight enforcing both time fields ensures the range of times intended can be directly specified, avoiding unnecessary confusion).

Marker Visibility (Filtering)

If all the search criteria have been determined as valid, the visibility of the markers can then be altered based on the search criteria. Firstly, any previous filters which may be currently applied to the markers are removed by looping through the local array of markers and ensuring that every marker is made visible (unfiltered). The crime type, date and time properties for every marker in the array are then compared with the entered/recorded crime type, date and time values. A comparison by crime type is only needed if the crime type input was changed from its initial value of ‘ALL’. If a marker does not meet the criteria in any aspect, that marker is hidden.

3.3.11 Importing Markers

A small modal is implemented as appearing when the ‘Import Crime’ button from the main toolbar is selected. This modal includes a file input element, two button elements (one of which will be used to download a template import file and another to confirm to import the file chosen) as well as two progress bars (one to indicate the progress of the file being uploaded to the server and another to show the progress of markers/records being added to or imported into the database).

File Selection and Validation

Before sending any user selected file to the server, there should be implementation for validating the type and contents of the file chosen. A design decision was made to only accept .csv files when importing markers and so therefore the file input implemented to appear in the modal is set to only show .csv files when browsing for files by default, however, there is nothing to prevent this option being changed to show all files and an alternative file format being chosen, and such is why additional checks which examine the extension found in the filename as well as the file type are also made.

If the file is of the specified file format, the file is then opened and read client side (using JavaScript) with the next validation being to check for the correct column headers. This is achieved by declaring extensive lists of accepted column headers using arrays and comparing the values in the first row of the file (typically the column headers) against the values in these arrays. When a column header in the file matches with any of these accepted column headers, their position in the first line (their column index) is recorded, thus allowing for the column headers in the file to be of any order. The column headers expected to be found in the file correspond to the implemented properties of markers or the fields of input used when adding a marker that have been previously implemented (those being Crime Type, Date, Time, Latitude, Longitude and a Description) but the implemented approach means that not all of these column headers (or similarly named column headers) are needed in order for the file’s information to be imported. Whilst the Latitude and Longitude columns are deemed to be essential or required column headers (as how can a crime record be mapped without a specified location?) and, if missing, result in the file being flagged as invalid and subsequently not being sent to the server, it was decided and implemented that the remaining column values can be either inferred or be set as a default value if the relevant column header was not found within the file. For instance, if an acceptable column header for a column of date values is not present in the file, instead of outright rejecting the file and refusing to import any information, the option to use the current date is presented instead to the user. When a required or optional column is missing from the file, an alert message is displayed to show any expected columns which have been identified as missing but if both the required headers (of Latitude and Longitude) are present in the file, the file is sent to the server using an AJAX call with the progress of this transmission being displayed using one of the progress bars.

With the file being sent to the server to be handled by the ‘ImportMarkers.php’ file, the previously outlined step of identifying the index of each of the columns is performed again server side (using PHP). Although not ideal, and whilst the index of columns could have been sent to the server using the AJAX call, due to complications with bundling these values along with the file as the data to be sent, it was decided that the task of column identification could just be translated and repeated using the different language (now PHP instead of JS). The column indexes this time are identified with the csv file being converted and stored into an array and the first row of this array (the expected column headers) being checked against accepted values (that are yet again specified using arrays).

Import

With the column indexes identified, the remaining rows in the array (every line after the column headers) are then read. For every line and for each of the optional crime attributes, the value begins as a default value (for instance, for every line, the date recorded begins by default as the current date for the Date attribute). Then, if the relevant column header was specified and found in the file, a search is made using the relevant column index and for the current row/line in the array (of the recently converted csv file) with this two dimensional index (position) in the array first being checked to see if it exists as there may have been incomplete lines in the file with missing values. If a value is present at this index in the array, the found value then undergoes validation. Values for the dates and times of crimes found below column headers for ‘Date’ and ‘Time’ are validated to ensure they are a valid date and time respectively (using the PHP strtotime() function) and values for the types of crime and their descriptions (‘Crime Type’ and ‘Description’) are checked they are of string type and contain at least one alphanumeric character (i.e are not missing or blank values). If the searched for and found value passes validation, then the recorded value for the current line and for the current crime attribute can be changed from the specified default value to this new value.

The required crime attributes (of ‘Latitude’ and ‘Longitude’) are handled similarly but with the validation being to ensure the values are numeric and fall within the previously specified ranges of values (refer to the design section and the ‘Database - Conceptual’ sub-section). Boolean values are used to keep track of whether the found values for both of these attributes for the current line are valid. With these required attributes being searched for and validated first, if either of the boolean values are flagged to indicate a valid latitude or longitude value for this line could not be found, the rest of the line needn’t be checked as the remaining attribute values of this line have no effect on whether the values of the current line (record) can be imported into the database (as a a required attribute is missing). Only if both of these boolean values remain unflagged, can the current line and the recorded values for each of the crime attributes be added to the database using an INSERT SQL statement. Regardless of whether the current line was added to the database (i.e after each line has been handled), a counter is incremented to record the progress of processing of the file. This count value in proportion to the total number of rows in the file (excluding one row for the column headers) represents such progress (in the form of a percentage).

Ideally this percentage would be sent directly back periodically to be used locally to update the progress shown by the relevant progress bar (client-side) but as of the current implementation so far there isn’t currently a feasible way of communicating this percentage routinely when it is needed, the only currently possible transmission method is the return function of the AJAX call but this only occurs in the timeframe the file has successfully been uploaded and reached the server not when the certain stages in the execution of the PHP file (‘ImportMarkers.php’) which handle this uploaded file have been reached.

Therefore, an alternative approach of writing the current percentage of records that have been processed to a seperate file and routinely reading the contents of this file was implemented instead. This implementation involved after an arbitrary amount of rows had been processed (the total number of rows in the file divided by 20 and rounded up to the nearest integer), the progress being reported to the file, and every second after the file had reached the server, this progress being retrieved and used to update the progress bar.

3.3.12 SQL Injection

Problem

Due to user input being used in several of the SQL statements employed in the solution (such as just recently outlined with user entered values being read from an external file and although being validated to some extent being imported into the database with little consideration to how these values could be malicious), 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 with which malicious SQL statements could be entered into the input fields and be unknowingly executed in the database, was next on the agenda.

It's worth noting that all information stored within the database is shown through the markers and their InfoWindows and so an attacker attempting to dump the contents of the database would be in vain (there is no sensitive information such as passwords being stored in the database) and similarly using the technique for deleting or updating a single record in the database would be of little use considering this functionality is made openly available as intended features of the solution. Whilst not absolutely critical to prevent, with the following overhaul, effort should be made to ensure these operations are confined to being performed through the provided functionality as opposed to through unintended and malicious methods. The main rationale for the overhaul or where the real danger arises from is an exploit or payload which compromises the back-end database infrastructure (whether that be by deleting the table storing these records, the database containing this table or other means) which would not only mean that currently stored records would be lost but also render the solution unable to store records in the future, in turn breaking a significant amount of functionality the solution intends to provide.

Before implementing the overhaul, identifying where the solution is currently vulnerable (the locations which would benefit from the overhaul) followed by an explanation of a possible SQL Injection that the overhaul must ensure is prevented is to be conducted first to ascertain a better understanding of both the possible attack vectors and the possible methods or approaches, the findings from which can be used to inform and shape the overhaul that is to be implemented.

The SQL statements in the solution identified as being critically vulnerable to SQL Injection are the following:
The DELETE statement used to delete a single record/marker (*DeleteMarker.php*)
The UPDATE statement used to edit a single record/marker (*EditMarkers.php*)
The INSERT statement used to add a single record/marker (*SaveMarkers.php*) and which is executed multiple times in quick succession to add the records/markers found within an external file (*ImportMarkers.php*)

(**NOTE:** Whilst there is no indication that the SELECT statement used to return the currently stored records/markers *(index.php*) is vulnerable to direct SQL Injection (as it does not make any direct use of any user input), the risk of second order SQL Injection, involving user input being stored to the database by other queries and then being used unsafely for this query, may well still be a risk posed and so therefore with no foreseen disadvantage in the overhaul encompassing this statement, this SQL statement was also identified as being potentially vulnerable to SQL Injection).

Although just one of many queries implemented in the solution which has been identified as and is currently vulnerable to SQL Injection, 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):

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

Whilst strong typing and validation is enforced for the majority of input values (for instance, the date input field only allows for dates to be entered and the latitude and longitude values are only derived from a marker on a map as opposed to being directly entered through user input), any string input can be entered into the 'Description' field. For instance, input which involves entering arbitrary and valid values for all fields up to this point but then includes crafted input for the 'Description' field, facilitates SQL Injection:

Input	**Text', '50', '50');drop table markers;--**
Query	**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');**

The crafted input populates the INSERT statement with the remaining required values and completes it with a closing bracket and semicolon but a subsequent statement to drop the specified table then immediately follows this (with a comment following on after this to ensure the rest of the query is ignored by being commented out). Put simply, the query now 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).

SQL Injection - 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 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 treated as data as the SQL query is already compiled and any data which is parameterized is never 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 and such is why the benefit observed will be more substantial when prepared statements are implemented for the addition of a larger number of records to the database such as when markers are imported from an external file as opposed to when implemented for handling the addition of a single marker to the database through the provided functionality ('Importing Markers' as opposed to 'Storing Markers' respectively).

SQL Injection - Implementation (of solution)

With the overhaul using prepared statements, beyond the query itself being changed, associated statements binding the parameters to the query involving the data type and value to be used for each parameter are also in need of being specified. The data types for parameters are specified with single characters, 's' denotes the string data type (which is used for the majority of parameters in the statements) but 'd' which denotes the decimal data type (used for the parameters involving latitude and longitude values) and 'i' which denotes the integer data type (used for the parameter involving ID values) are also used. After parameterisation, only then and is the query executed.

INSERT statement (*SaveMarkers.php* and *ImportMarkers.php*)

```
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()
```

UPDATE statement (*EditMarkers.php*)

```
UPDATE markers SET Crime_Type = '$crime_type', Crime_Date =  '$date', Crime_Time = '$time', Description = '$description', Latitude = '$latitude', Longitude = '$longitude' WHERE ID = '$MarkerID'
UPDATE markers SET Crime_Type = ?, Crime_Date = ?, Crime_Time = ?, Description = ?, Latitude = ?, Longitude = ? WHERE ID = ?
```

DELETE statement (*DeleteMarker.php*)

```
DELETE FROM markers WHERE ID = '$MarkerID'
DELETE FROM markers WHERE ID = ?
```

SELECT statement (*index.php*)

```
SELECT * FROM markers
```

Chapter 4 - Results

4.1 Website Address (URL)

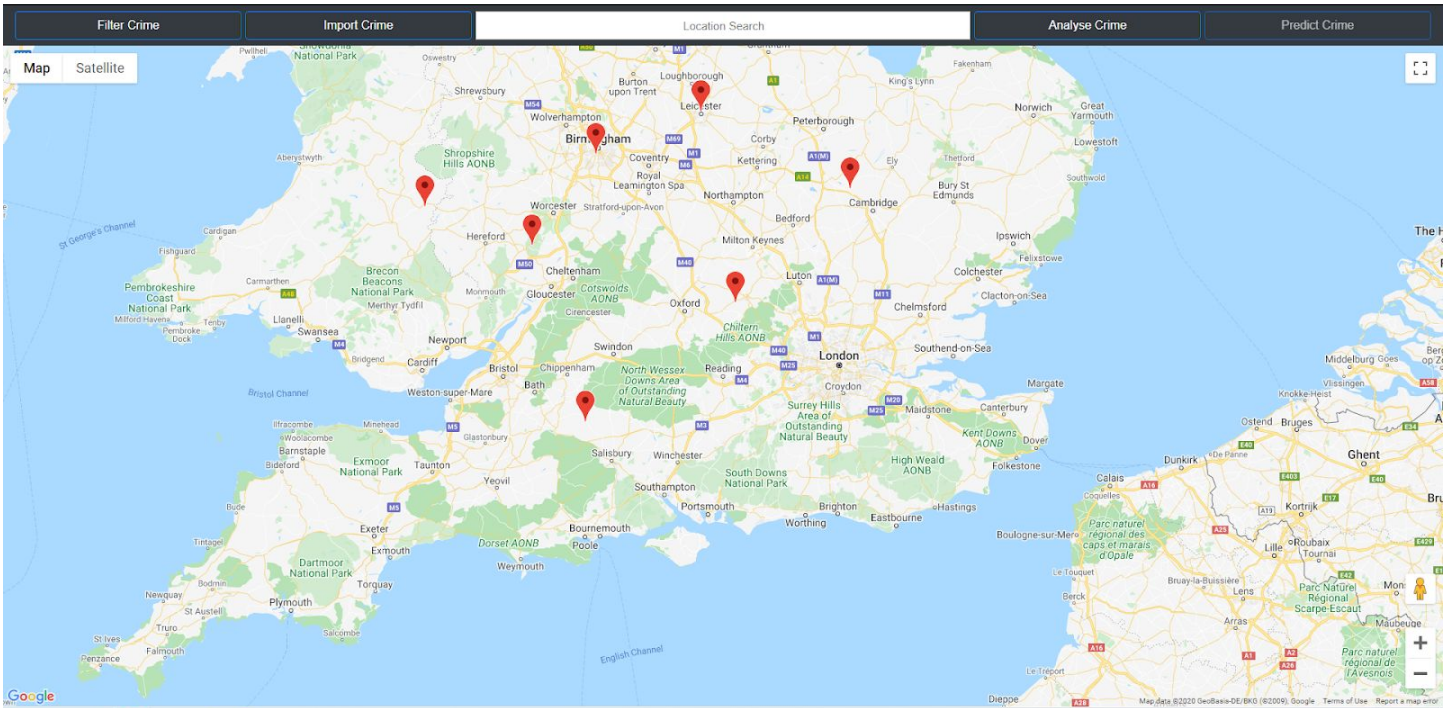
The developed web-based solution (website) is hosted and accessible using the URL:
https://hq017496.webs.act.reading.ac.uk/

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:



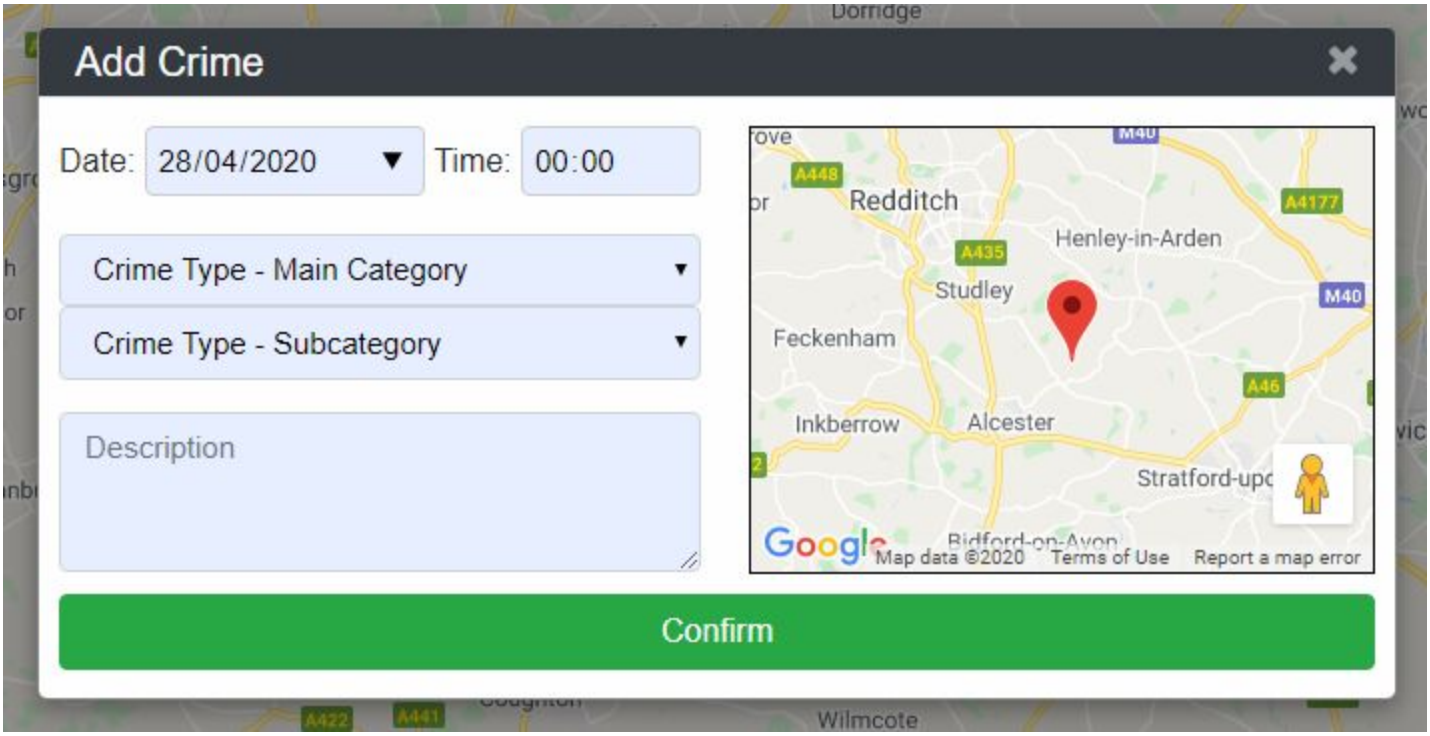
4.2.1 Search Bar

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

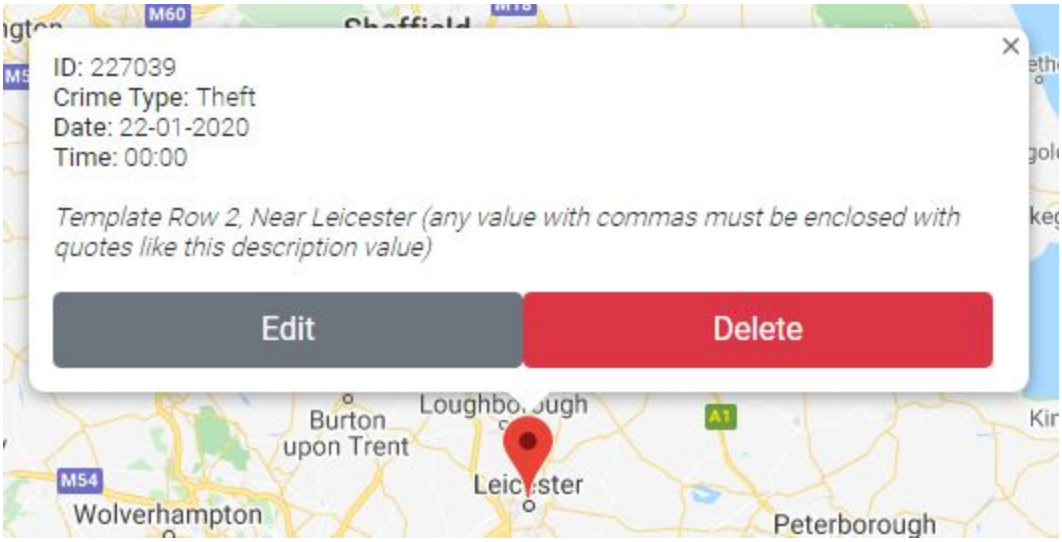
A mouse right click anywhere on the mapping component will open a one item context menu at the click location. Selecting this option in the context menu will open the ‘Add Crime’ popup window (modal):

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:



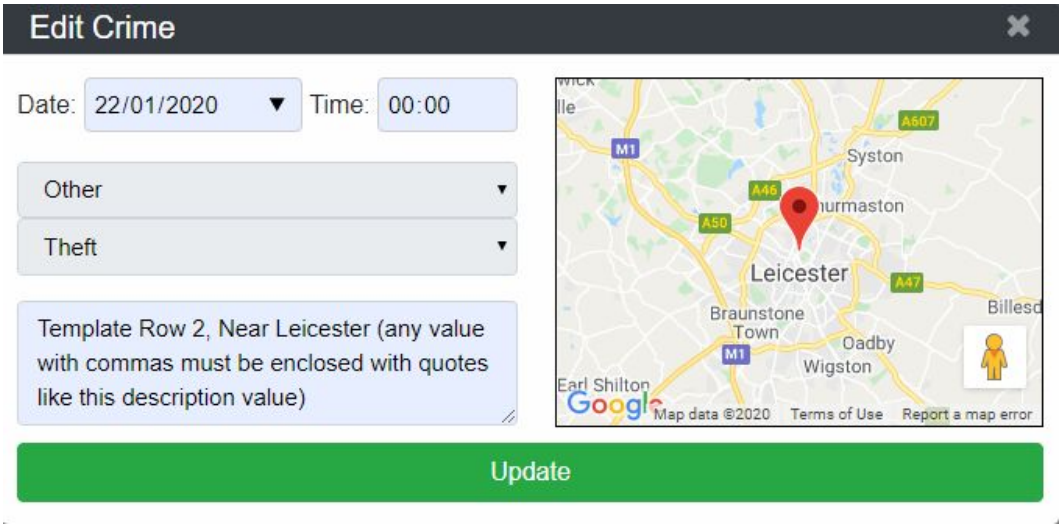
4.4 Viewing Crime (InfoWindows)

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:



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:



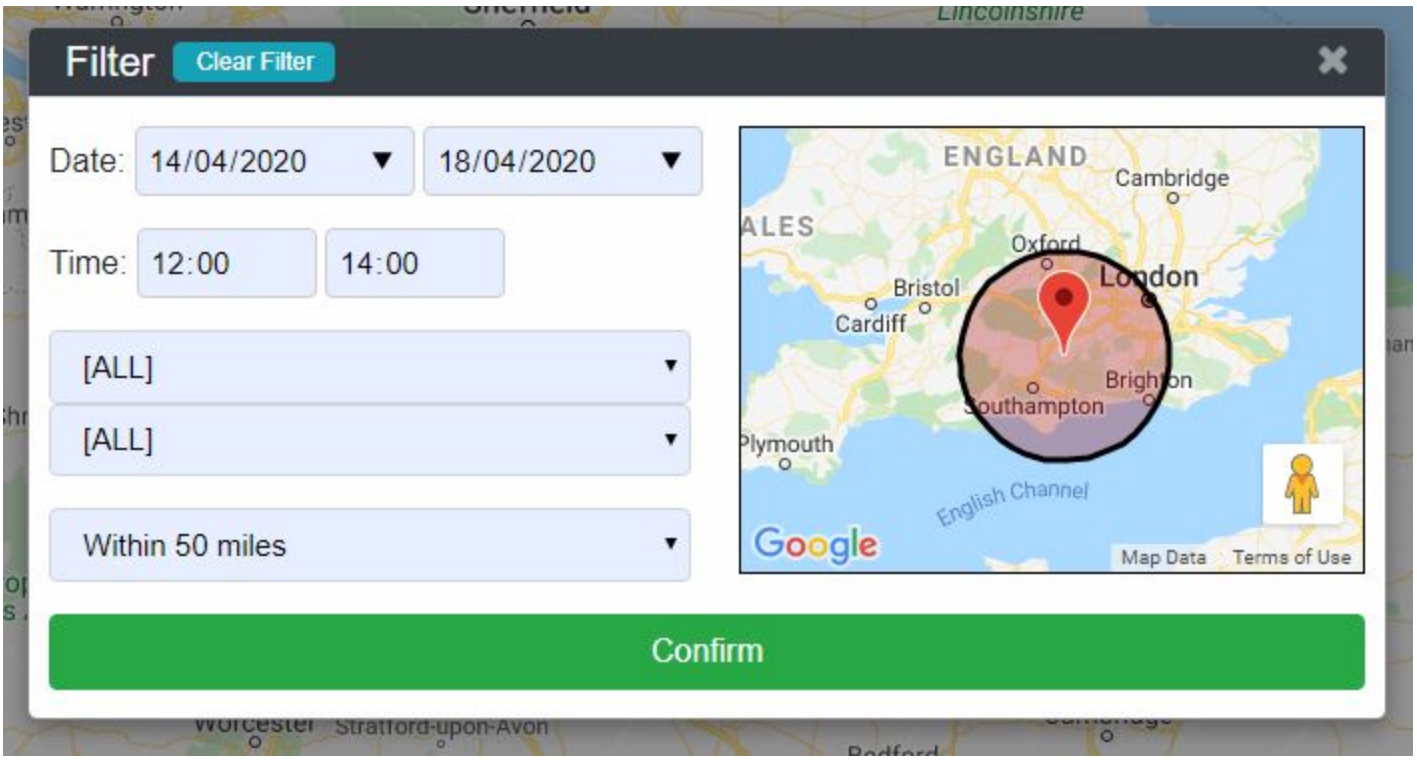
Whilst every property can be changed for markers created using the 'Add Crime' popup window or functionality, if the marker was imported from an external file, the crime type properties of this marker can not be edited (the fields are disabled). This is because the crime type imported can be different from the lists of crimes available to be chosen within these windows and because it is not wished for any unique imported crime types to be added to these lists.

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

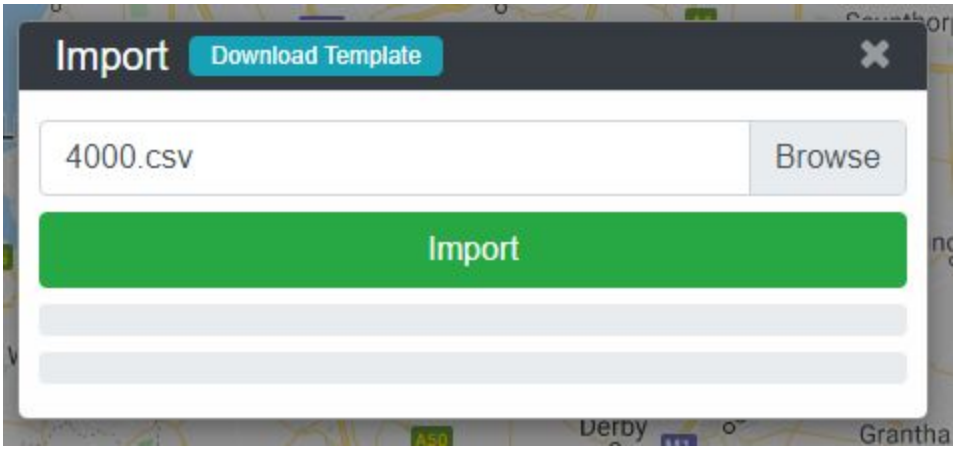
The ‘Filter Crime’ button of the toolbar will open the ‘Filter Crime’ popup window (modal). The markers or crimes can be filtered by the values entered into this modal with the ability to select a range of dates, a range of times, only crimes belonging to a specified main category or subcategory as well as filter by location by specifying a position on the smaller map provided and selecting a distance for the search radius:



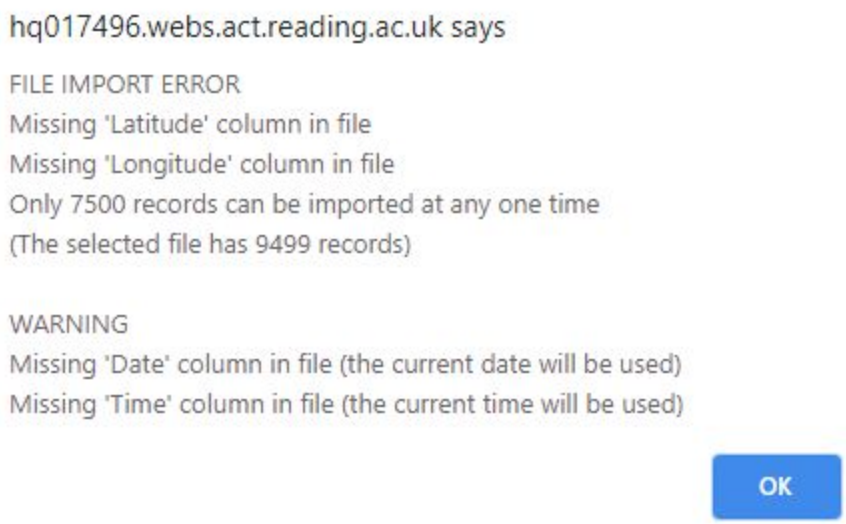
Unlike other popup windows in the solution, the values of this window (the filter criteria) are not automatically reset as similar filters with minor differences may wish to be made quickly after one another whereas the adding of a crime that has attributes identical to that of previously added crimes is unlikely to occur as frequently. Despite this decision being made, a small button next to the heading of the popup window, labelled as ‘Clear Filter’, can be clicked to reset all fields to their default values.

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). In the same way a small button is provided to clear any current filters for the ‘Filter Crime’ popup window, a small button beside the heading of this window, labelled ‘Download Template’ is provided which when clicked will download a template.csv file containing valid column headers and two example records, one of which demonstrates the importance of enclosing string values that contain one or more commas within double quotes (the rational and technicality behind this being that with commas being used as the delimiter in the file, this allows commas intended to be within a value to be distinguishable from those which separate values). When a file has been selected, the filename appears as the text of the input:



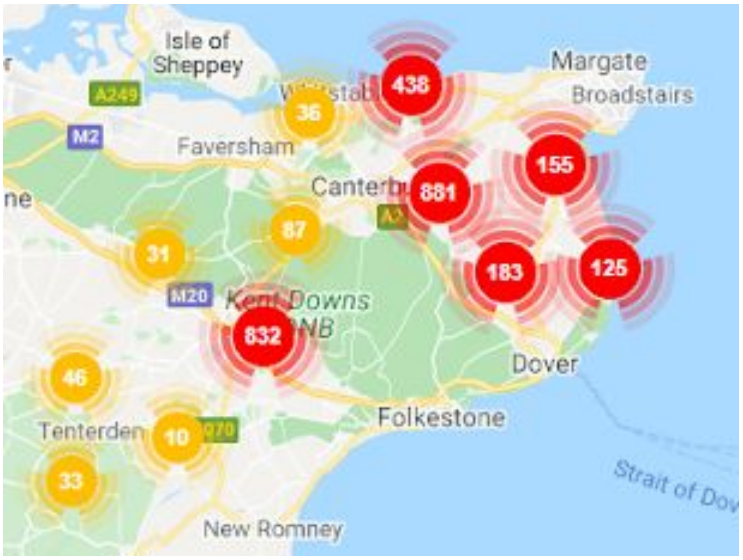
The file can then be confirmed to be imported by pressing the ‘Import’ button, if the file and its contents are not considered valid, an alert message is displayed stating a reason why the file is unable to be imported and/or what must be changed with the file in order to be valid:

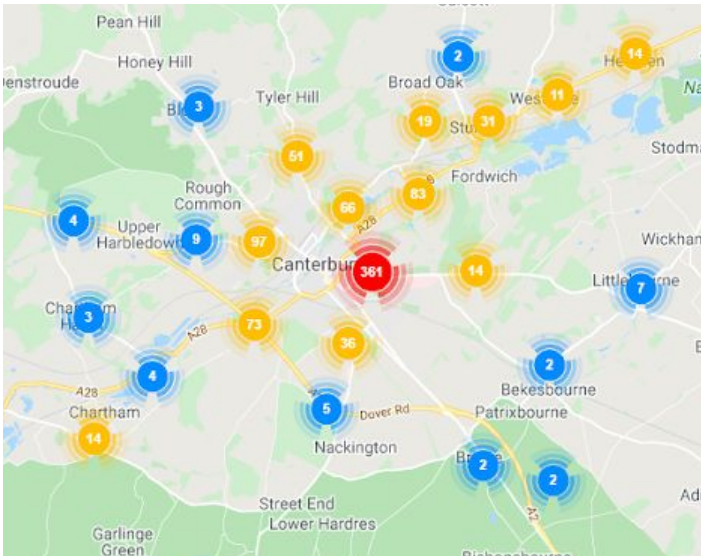


The two progress bars below the ‘Import Button’ will show progress of the import, if allowed to go ahead, the first showing the progress of the file being uploaded (to the server) and the second showing the progress of processing and mapping each record in the file. These progress bars themselves can 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:

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 (blue being the lowest for single digit counts, yellow for two digit counts and red, being the highest for three or more digit counts) and that has white text showing this number of markers. These groups are dynamically created and change with different zoom levels, showing the spatial distribution or density of crimes at different geographic levels:



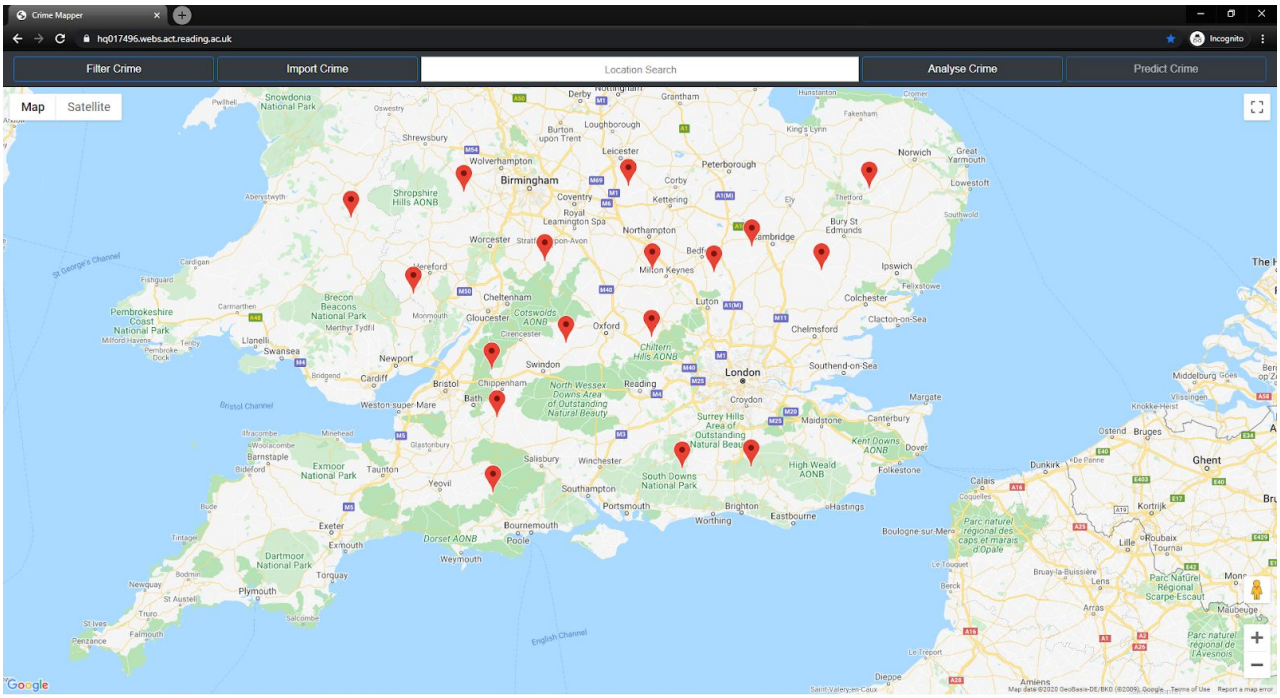


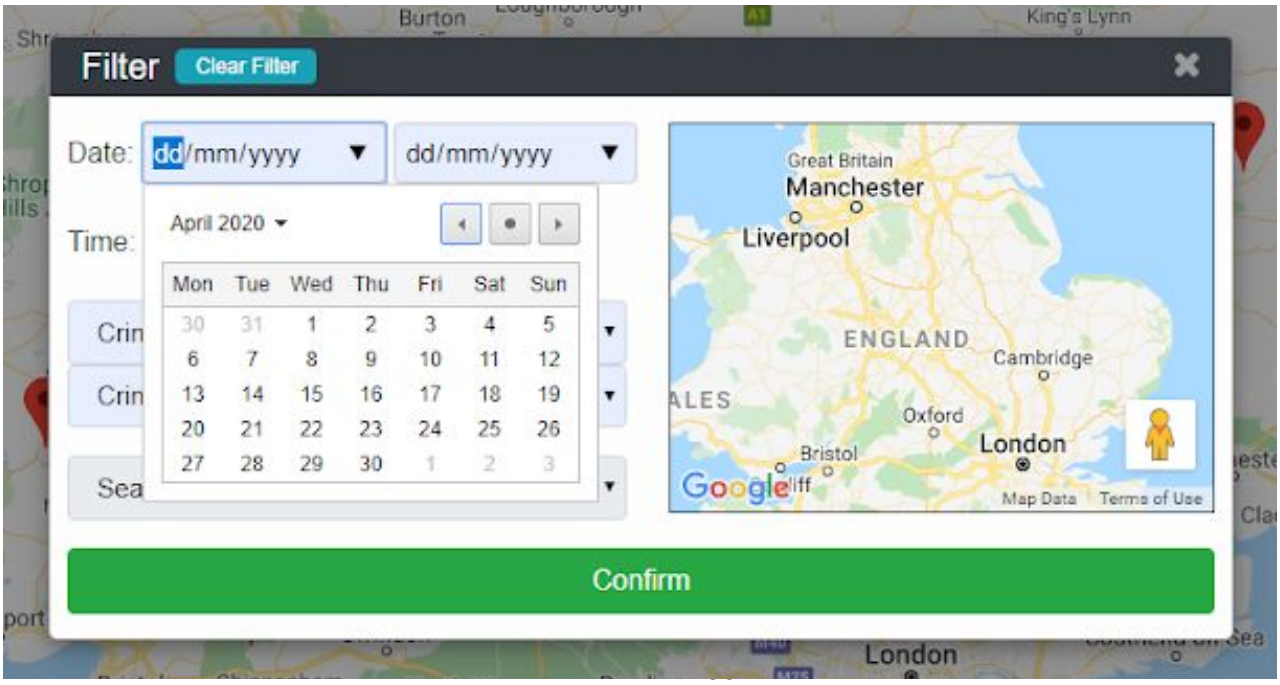
Chapter 5 - Testing and Validation

5.1 Cross Browser Testing

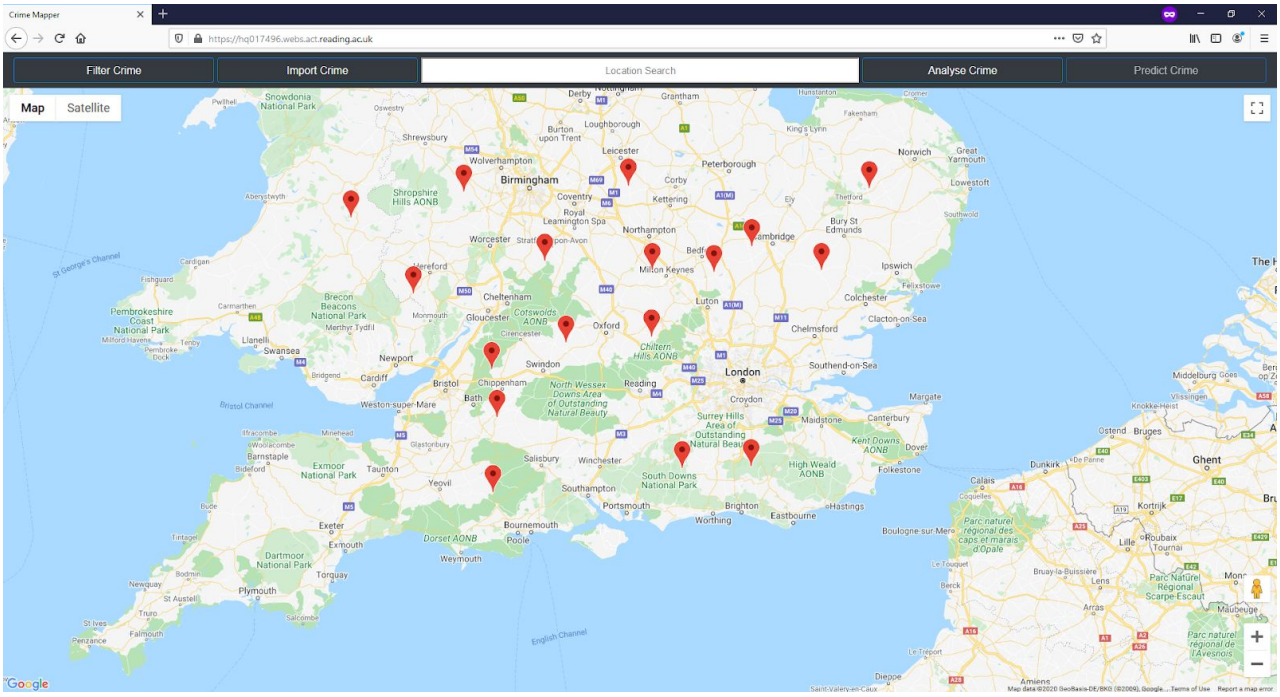
The solution was next tested using various different web browsers with the latest versions (at the time of testing) of four common browsers along with a constant screen resolution of 1920 x 1080 being used. With the functionality or features of the solution already exhaustively tested, all consideration was directed towards the layout of user interface elements (UI) and the website responsiveness of the solution. If during this testing, the solution does not behave or appear as expected, then the compatibility issue will be highlighted, and if possible addressed, otherwise, evidence or images (one of the map and one of a modal) of the testing under the different environments will be shown.

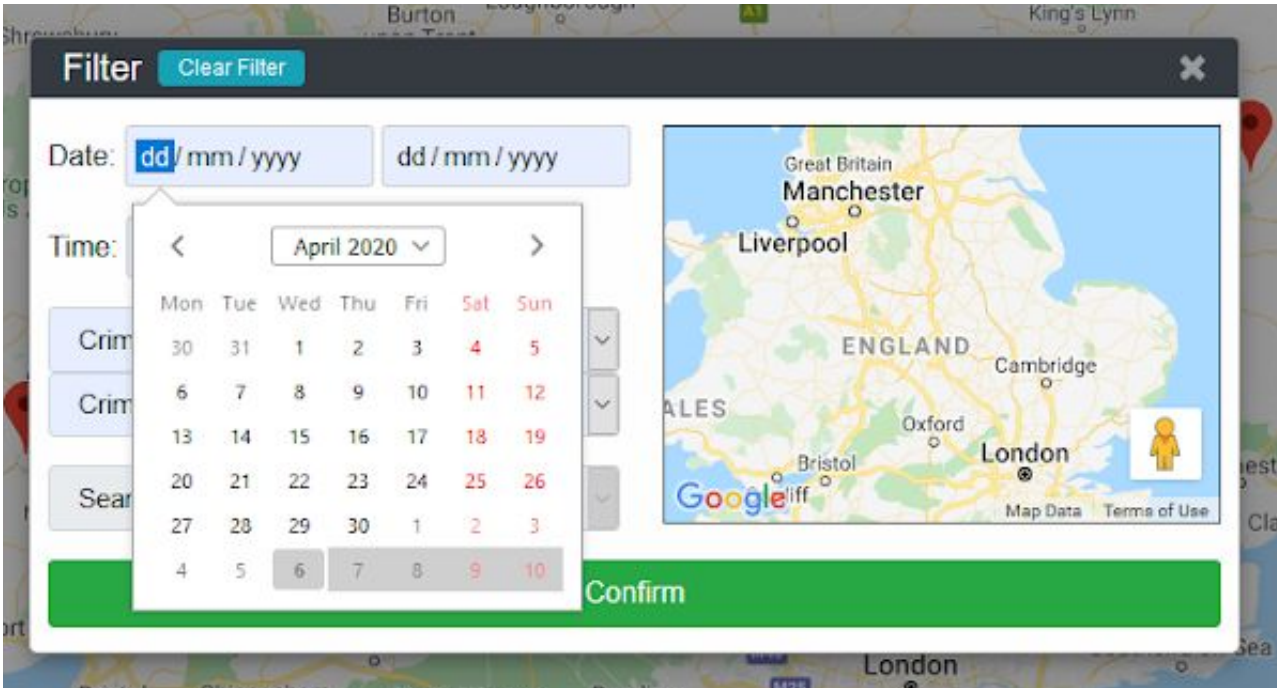
5.1.1 Chrome (v81.0.4044.129) - **PASS**



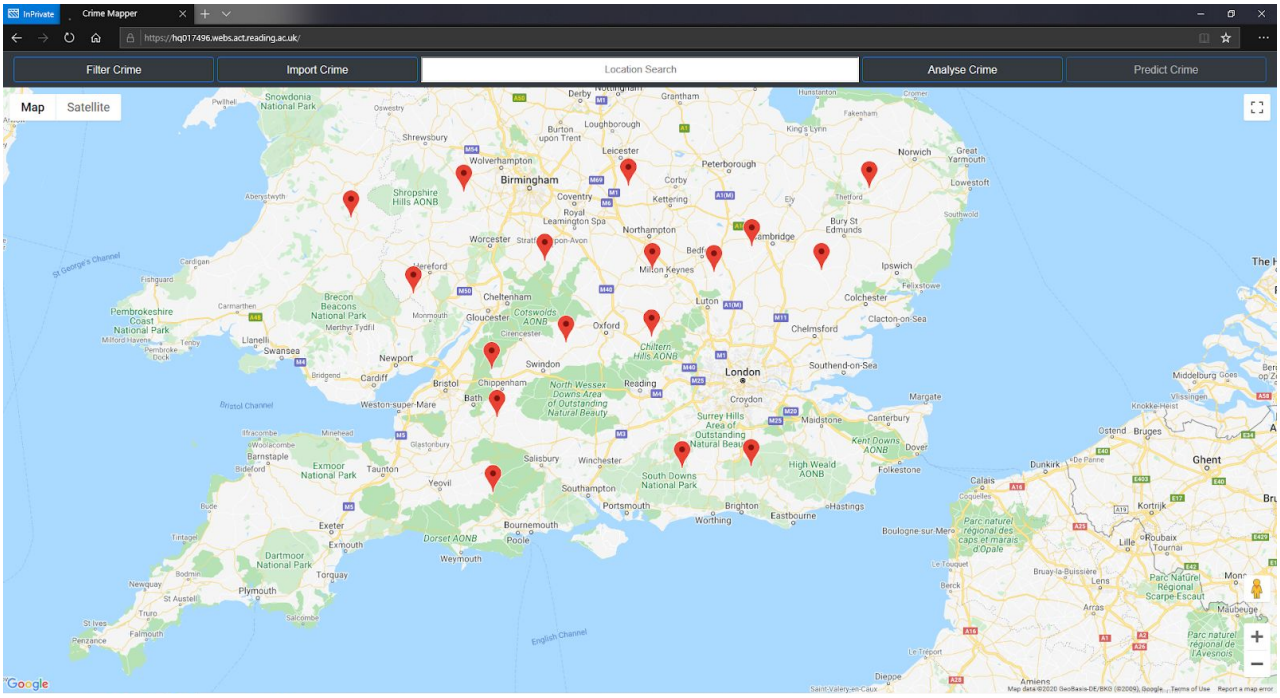


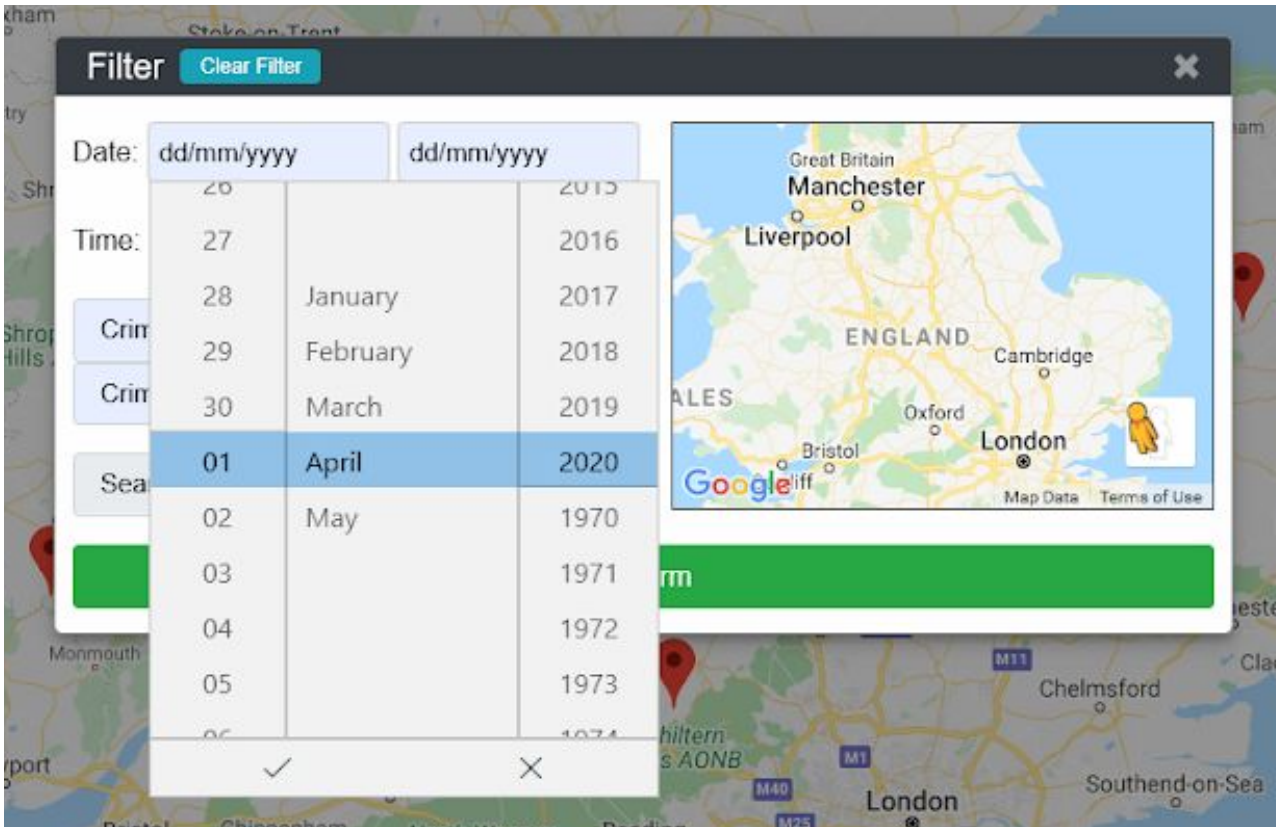
5.1.2 Firefox (v75.0) - **PASS**





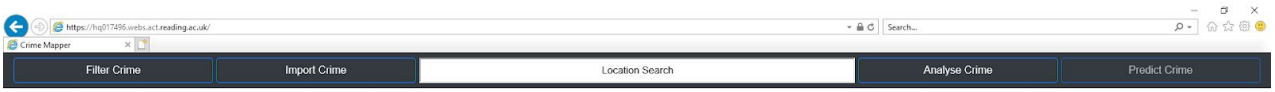
5.1.3 Edge (v44.18362.449.0) - **PASS**



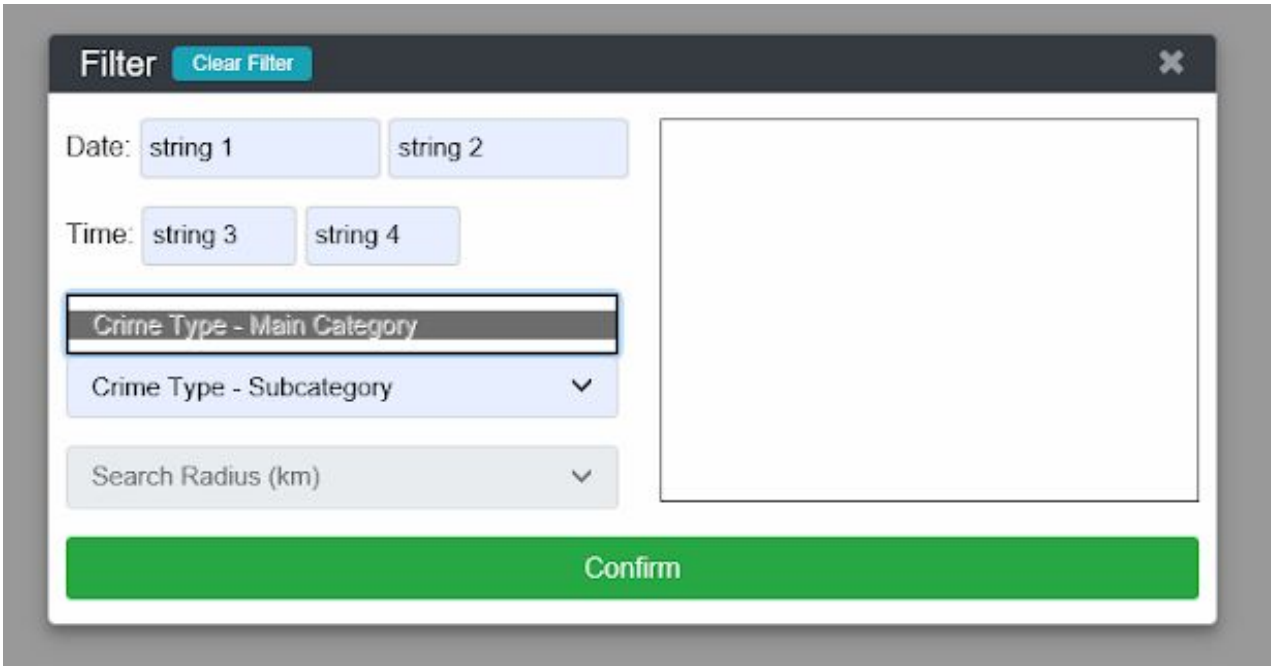


5.1.4 Internet Explorer (v11.778.18362.0) - FAIL

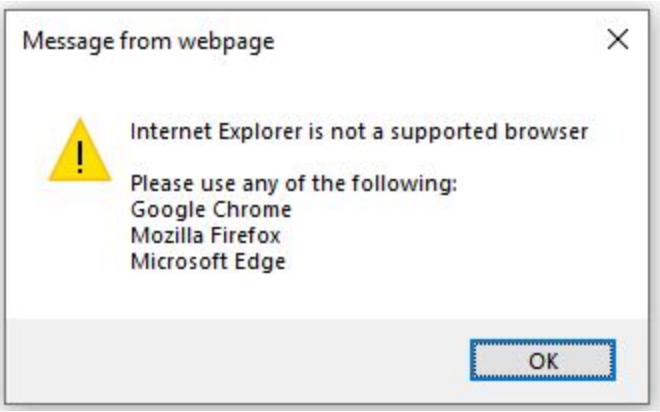
The mapping component does not load (or isn't displayed) with this browser and only the toolbar is shown. The toolbar buttons do open their relevant popup windows and it may be that their functionality does work but the results of which cannot be seen:



The inputs of these 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:



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 as seen in other browsers), is troubling from a security perspective. Although procedures are in place (refer to SQL Injection), as it stands with the solution not being useable in IE11 and simply acting as an unnecessary vector of approach for malice, the safe approach is to disable all functionality for this browser and to display an instructional message to use another browser instead:



5.2 Resolution/Responsive Testing

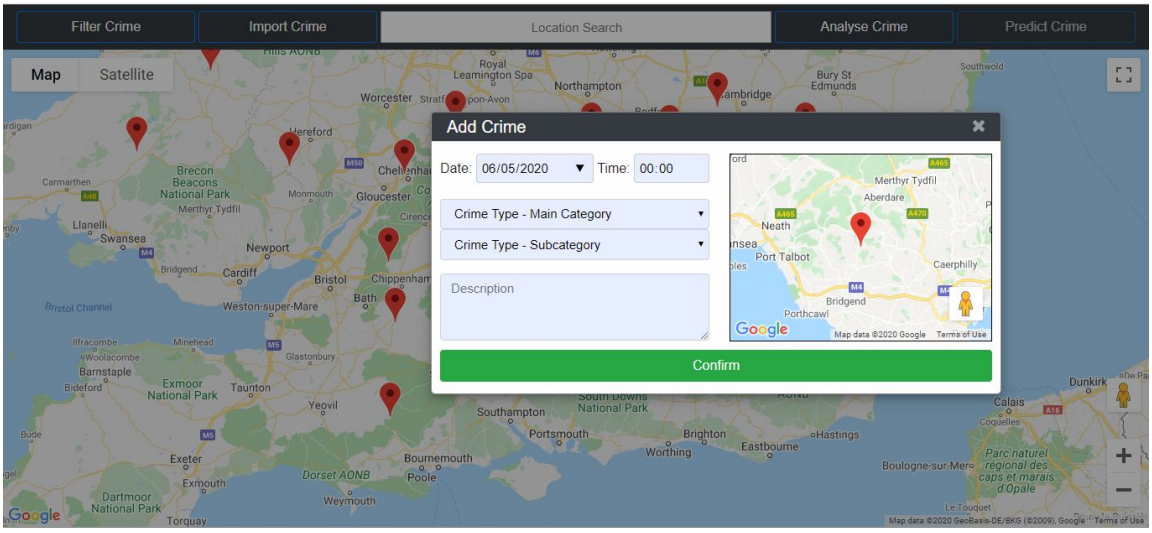
With a solution objective being to ensure the website is usable with all common desktop resolutions, and to further test the responsiveness achieved using the BootStrap framework, the solution was next tested on or with devices of varying screen resolution.

5.2.1 (1920 x 1080) - **PASS**

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

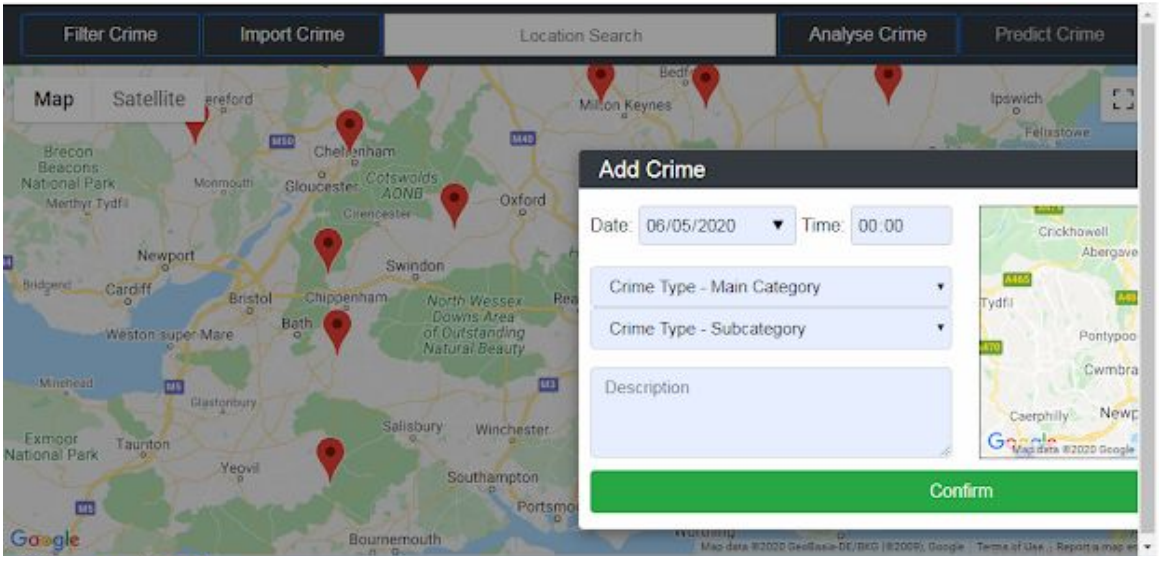
5.2.2 (1366 x 768) - **FAIL**

For this resolution, the ‘Add Crime’, ‘Edit Crime’ and ‘Filter Crime’ popup windows are all misaligned horizontally with the map and remaining right side of the modal not being shown:



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:



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;
  }
}
```

5.2.4 Solution/Fix

The three previously misaligned modals now appear centrally and can be used as intended. The implemented media queries using ranges of screen width should apply for all desktop resolutions. Smaller screen sizes such as those on mobile devices, although untested, are likely to not be fully supported.

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 (where the solution is loaded again), the currently recorded markers are loaded and placed on the map. The average durations (over three instances) from the start of this process to when all markers have been loaded are recorded by the time elapsed from the start to end of the LoadMarkers() function. The amount of markers are scaled up with the same marker to ensure the accuracy and reliability of the results which are shown in the table below:

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 and surrounding modules of code could be optimised were explored (code optimisation).

One 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 a marker's InfoWindow from within the placeMarker() function (where/when a marker is placed on the map) to within the click event for a marker.

Whether the change has had a beneficial effect on the solution can be determined first by ensuring the functionality remains working as expected and then also once again recording the average duration taken to load a varying amount of markers (as a means of comparison):

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 with an example of the benefit it provides being the time taken to load 50,000 markers, on average, being reduced from 16.3 seconds to 1.6 seconds. Although a significant improvement which also benefited other areas of the solution such as how smoothly the mapping component can be navigated, to further improve the solution, it was additionally decided that a loading symbol was to be implemented as displaying at times when knowledge of the system status would be beneficial to the user experience such as from the solution being opened to all markers having been loaded.

5.3.2 Importing Markers

Whilst the previous optimisation improves the duration taken for the solution to load markers after an import has finished, the duration taken to complete the import process itself remained unchanged. An optimisation that could improve performance for this module of code is concerned with the order in which the column values for the records are validated. Currently, the order is to check for the presence and validate the Date followed by the Time, Crime Type, Description and then lastly the Latitude and Longitude. But as stated previously, with the Latitude and Longitude column values being required values for every record that can determine whether a particular record is imported or skipped over and the remaining column values only being optional (replaced with default values if not present for a record), a better approach would be to check for these required column values first, so that if one of them is missing, the other column values aren't in need of being validated (because the entire record won't be imported if it has a missing or invalid Latitude or Longitude value).

Before implementation of this optimisation, the average duration taken to import a file with a varying amount of records (not including an empty file without any records and only up to the maximum allowed limit of 7500 records) where half of the records are missing a Latitude value, was investigated (see Appendix). The optimisation was then implemented in code first by moving the checks for these required column values as the first checks to complete (Latitude followed by Longitude). If the Latitude value for a record (first column validated) is missing or invalid, the entire row is skipped over, otherwise the validation proceeds to the Longitude value. If this value is missing or invalid, likewise the entire row is skipped over but if not it means that the remaining column values of that row can be derived ready to be imported. As with the previous optimisation after first checking the import functionality still works as expected (as before the optimisation), the average durations were recorded again to see whether the optimisation had improved the time taken to complete the import process (see Appendix).

The improvement in average duration was minimal and was only prevalent for a larger amount of markers. For some amounts of markers, the average duration actually worsened as a result of the optimisation. Despite this, a decision was made to keep the optimisation, because if the technical limitation of 7,500 markers being imported at once were ever addressed in the future, the benefits of the optimisation would likely begin to outweigh the small downturn in performance for certain scenarios and become more pronounced.

5.3.3 Resource Utilisation

To briefly summarise how the solution performs by the way of system resources it uses 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.

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. 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:

```
688         var last_added = (MarkerArray.length-1);
689         var load_matchingValues = true;
690 
691         for( i = 0; i < unit_testing_markers.length; i++ ) {
692             if (MarkerArray[last_added-i].ID != unit_testing_markers[(unit_testing_markers.length-1) - i][0]) {
693                 load_matchingValues = false;
694             }
695             if (MarkerArray[last_added-i].Crime_Type != unit_testing_markers[(unit_testing_markers.length-1) - i][1]) {
696                 load_matchingValues = false;
697             }
```

A log is written to the console depending on the outcome of the test:

```
718         if (load_matchingValues == true) {
719             console.log("Matching Values: PASS");
720         }
721         else {
722             console.log("Matching Values: FAIL");
723         }
```

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 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    }
```

5.5 Integration Testing

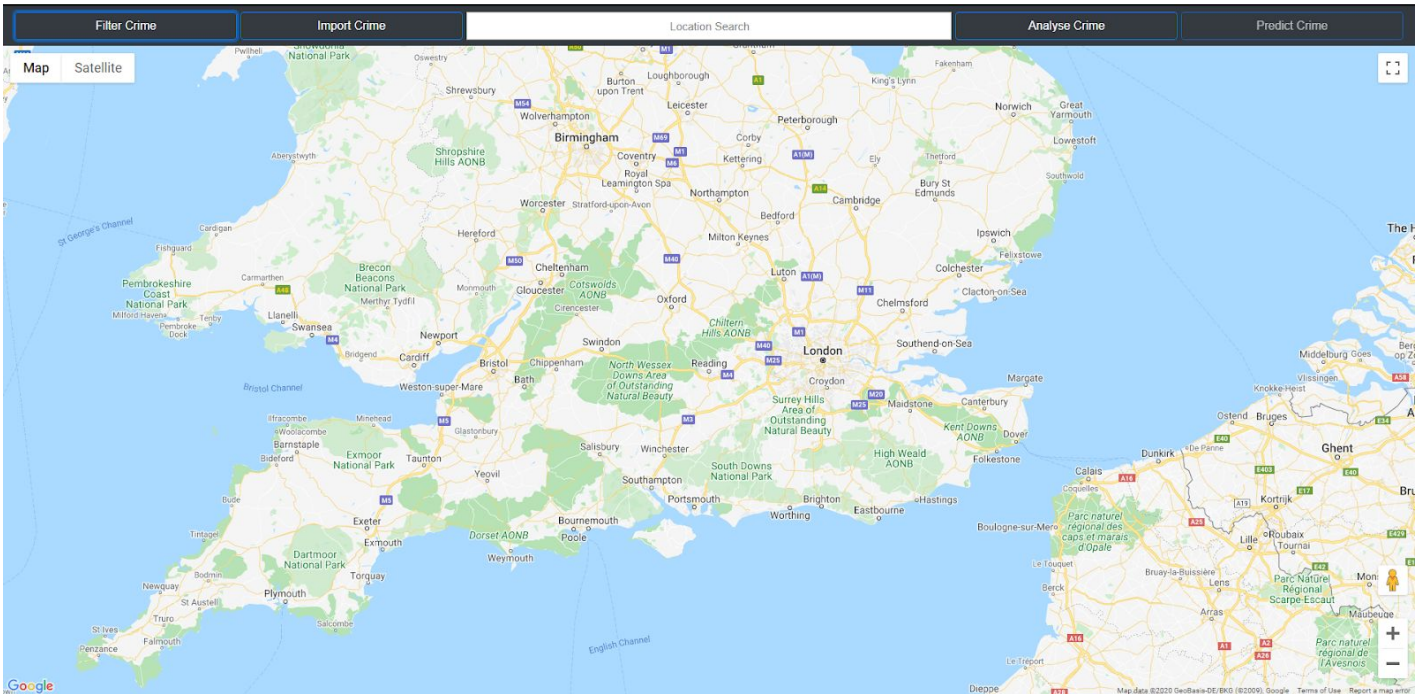
The individual components tested in unit testing can now be combined with other components of the solution (those that couldn't be tested with unit testing) both as a way to test these other components but also to test that all components work as expected when they depend or connect to other components.

The number of different combinations or sequences of components being used together is extensive and so only the most frequent or important are shown (with test cases in the Appendix) with other combinations (although were tested and are listed) only being shown if they arise as being problematic or erroneous.

5.5.1 Import Crime (Database)

As opposed to (or in addition to) producing and checking logs, 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:

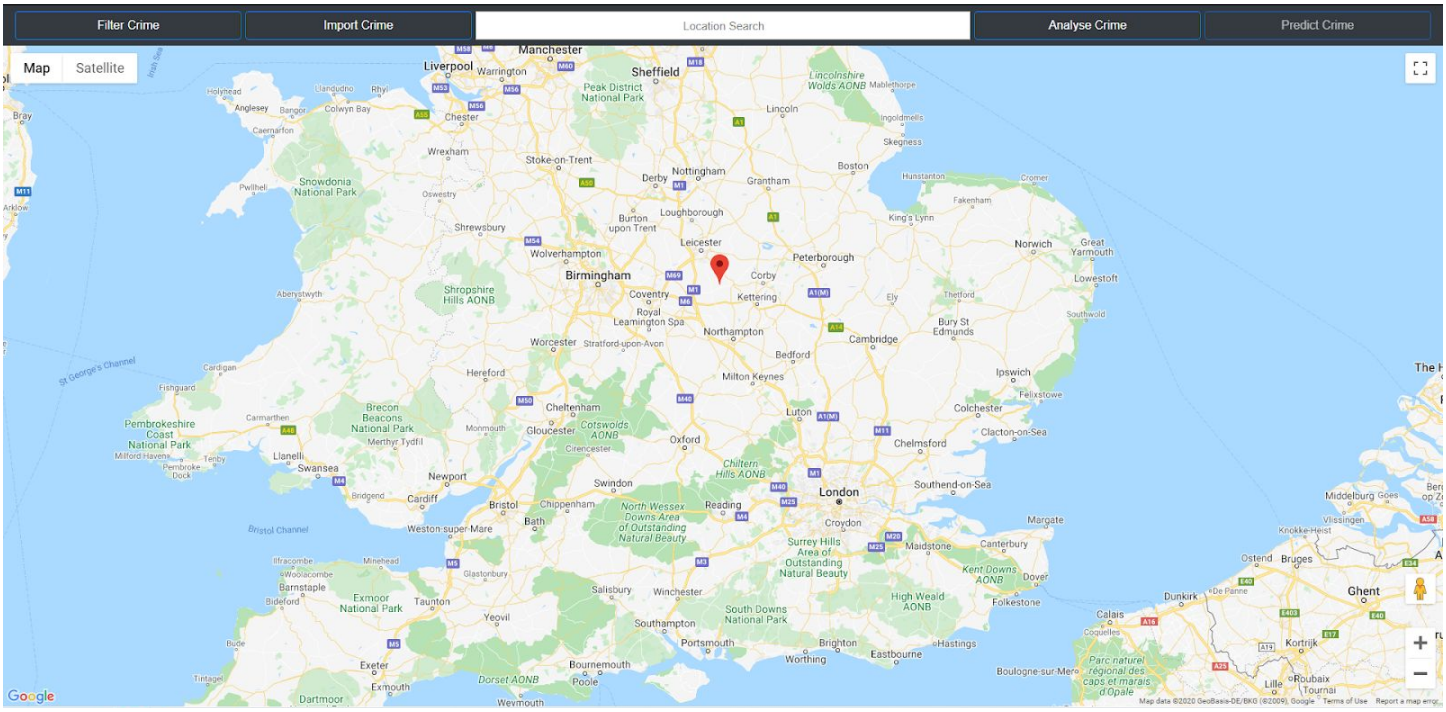
Test Case LC1 - Loading no or zero records - **PASS**



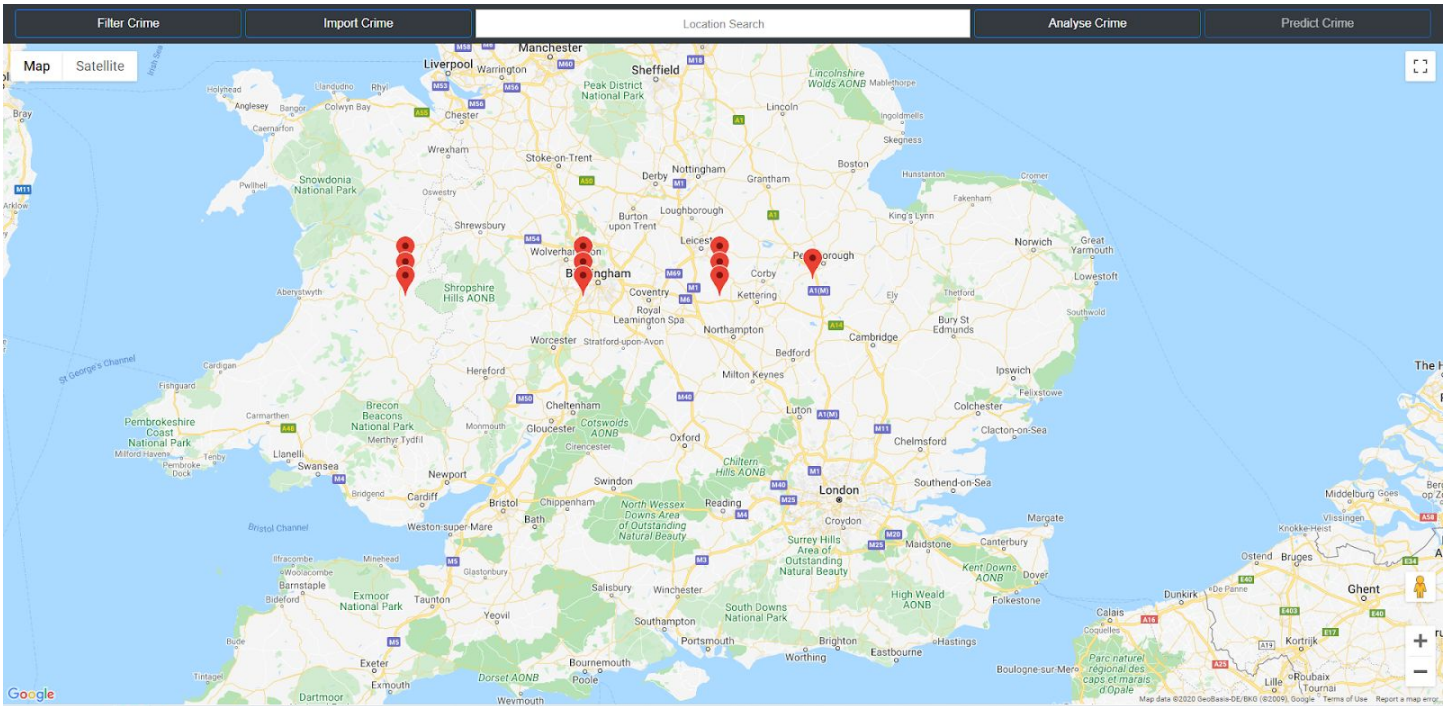
Testing an erroneous record - **PASS**

Although a distinct test case was not devised to test for erroneous records, it was observed that the prerequisite step of adding an erroneous record to the database was correctly prevented in almost all cases (by means of the database column domains) in effect passing the brief test. The only erroneous information which could be added to the database directly was a longitude value outside the typical range in which the expected outcome was the marker would not be placed. The actual outcome however was that a marker was placed with the value of the remainder of inserted value after the range limit. Whilst not the expected outcome, no action was taken as the marker was able to be handled in a way to not affect the functionality, and additionally, it has been tested that such a value can only be inserted into the database directly using the DBMS tool (and not any internal functionality of the solution).

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

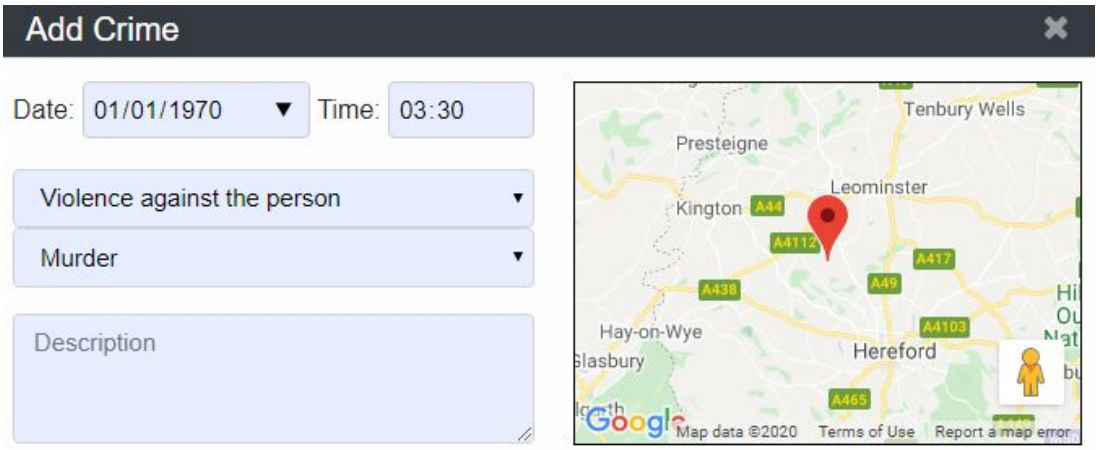


Test Case LC3 - Loading multiple (10) records - **PASS**



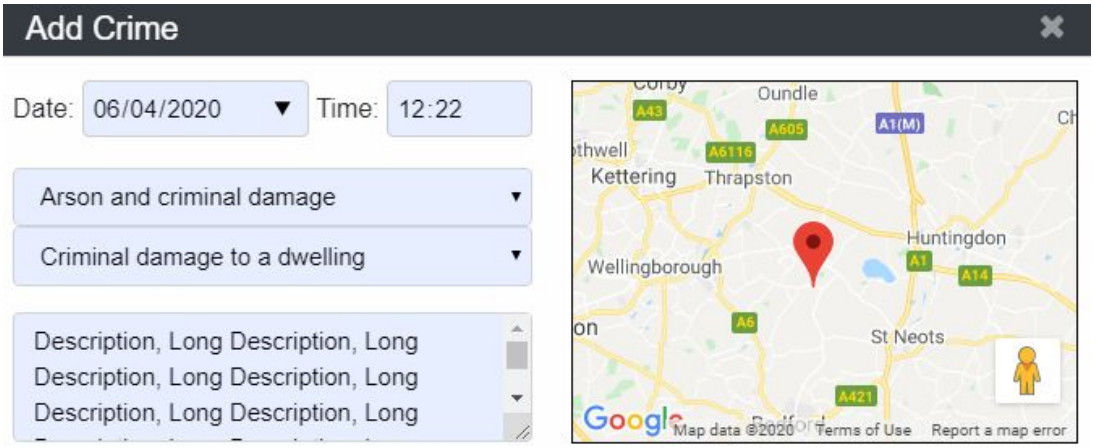
5.5.2 Add Crime (Database)

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



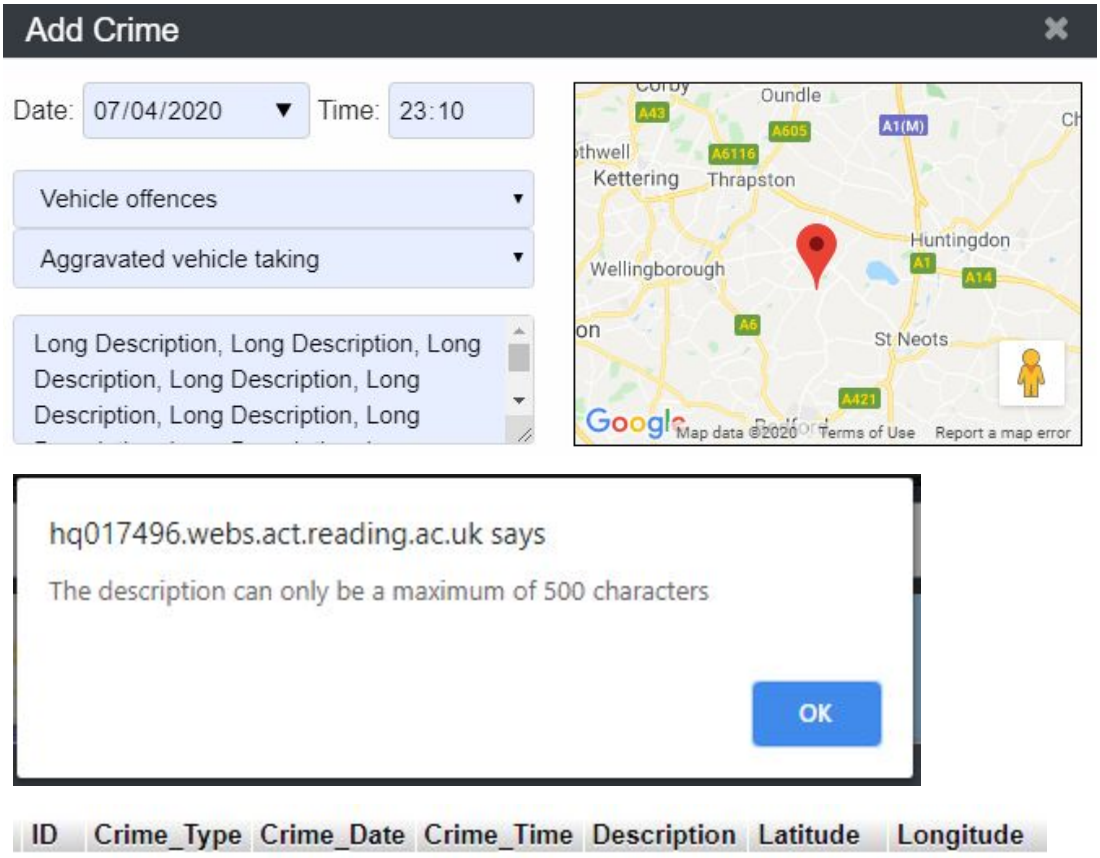
ID	Crime_Type	Crime_Date	Crime_Time	Description	Latitude	Longitude
47	Murder	1970-01-01	03:30:00		52.14773194	-2.82636937

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



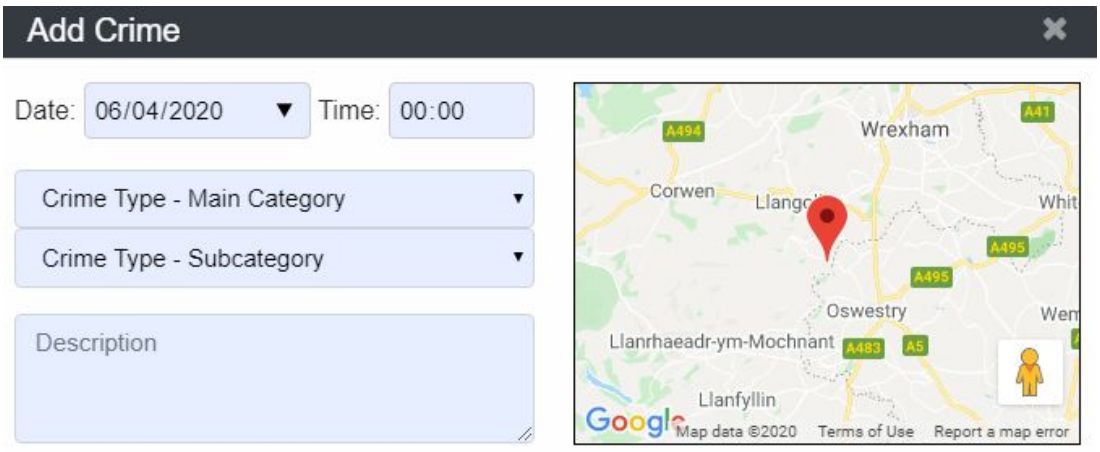
ID	Crime_Type	Crime_Date	Crime_Time	Description	Latitude	Longitude
48	Criminal damage to a dwelling	2020-04-06	12:22:00	Adjusted position and long description, Adjusted p...	52.88976634	-0.32148656

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



ID	Crime_Type	Crime_Date	Crime_Time	Description	Latitude	Longitude

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



ID	Crime_Type	Crime_Date	Crime_Time	Description	Latitude	Longitude

5.5.2.1 Solution/Fix

To remedy this, a check for any unspecified crime type (the fields not changed from their default hidden values) was implemented in which case preventing any addition being made to the database and a similar alert to that highlighting too long a description (above) was implemented to be shown.

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         }
```

5.5.4 Edit Crime (and Add Crime)

The integration test for this component closely resembles the unit test for the 'Add Crime' unit, testing whether the entered values match with the newly updated properties of the relevant marker.

5.5.5 Edit Crime (Database)

The interaction with the database was tested similarly to how the interaction of the 'Add Crime' component was tested (using similar test cases), all of which was completed without highlighting any issues.

5.5.6 Delete Crime (and Add Crime)

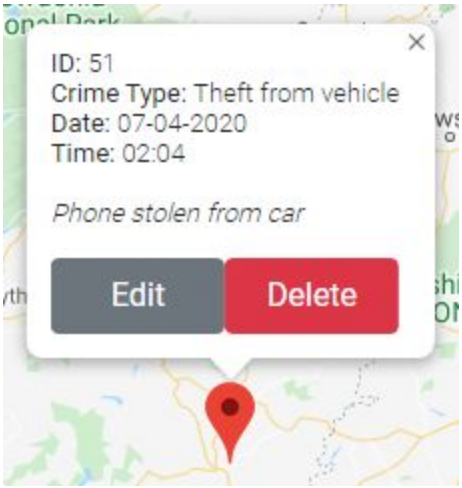
For a marker to be deleted, it must evidently first be added in the first place, hence why these two modules are tested together. The integration test would involve checking if the number of stored records (in the local array) decreases by 1 after the deletion:

```
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         }
```

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

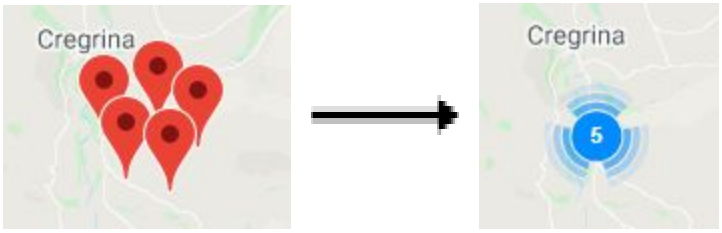


ID	Crime_Type	Crime_Date	Crime_Time	Description	Latitude	Longitude

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 (refer to ‘Results - Analyse Crime’). 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**



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

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 (threshold test) - **PASS**

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

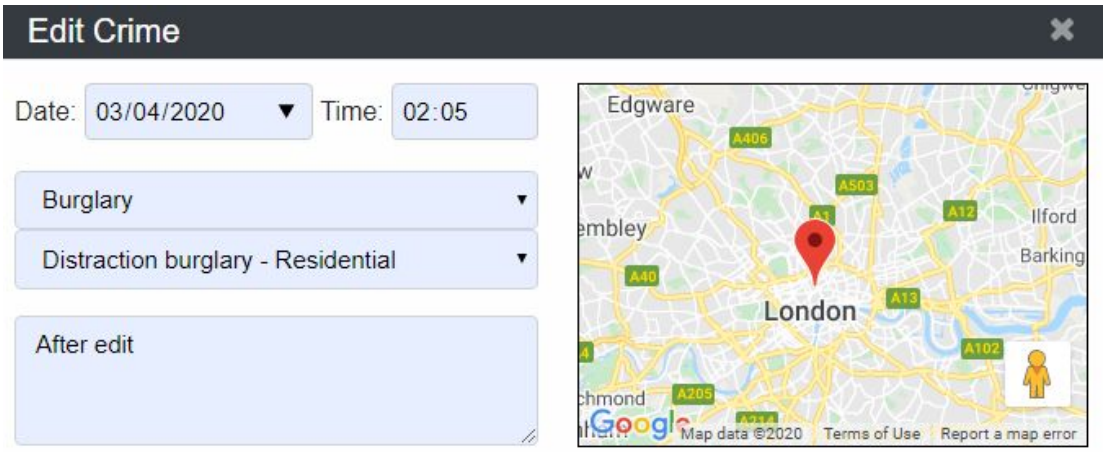
Test Case FC4 - All Main Crime Types - **PASS**

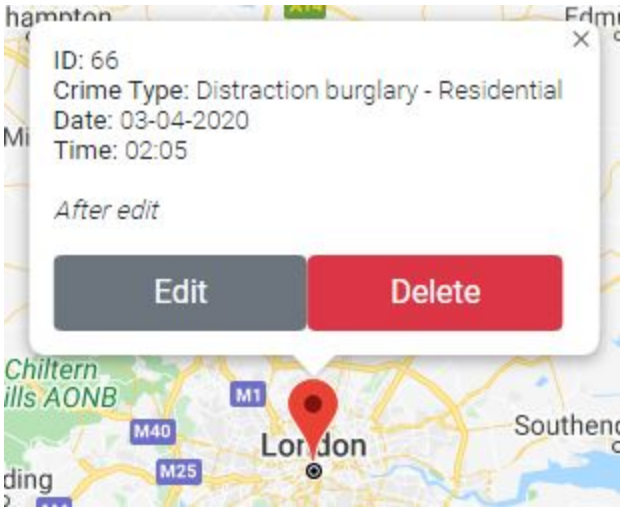
Test Case FC5 - All Sub Crime Types - **PASS**

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

5.5.10 Edit Crime and View Crime

Test Case EC1 - Edit an added crime - **PASS**





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

5.5.11 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
```

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. With there being two ways to add markers (‘Add Crime’ and ‘Import Crime’ functionality), both these methods must be prevented if the limit is at risk of being exceeded. Similarly, importing external files with more than 7,500 records is not able to be correctly handled by the solution and also couldn’t be addressed other than enforcing this limit, which is especially limiting considering that a large number of official crime data dumps (for larger constabularies or geographic areas) easily exceed this number.

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 and because user specified values from external files are not added as options within the 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 (perhaps a bounding shape made by clicking and dragging the mouse) or by selecting multiple markers at once, 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 - Conclusion and Future Work

6.1 Conclusions

6.2 Future Work

Initial plans for the project involved implementing crime prediction techniques

Chapter 7 - Reflection

Identified that were no current solutions for crime prevention , tried to differentiate by attempting tom implement, it, too much to implement (not feasible)

References

U.S Department of Justice (1999, December). *Mapping Crime: Principle and Practice.* Retrieved from: https://www.ncjrs.gov/pdffiles1/nij/178919.pdf

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

Chris Overall. (2008, May/June). *Crime mapping and analysis: filling the gaps.* Retrieved from: https://www.ee.co.za/wp-content/uploads/legacy/GisT-Crime%20mapping.pdf

Google. (2005, February 8) *Google Maps.* Retrieved from: https://maps.google.com/

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

Home Office. (2011, February 1). *Police.uk*. Retrieved from: https://www.police.uk/

Lalitha *Saroja Thota.* (2017, April 24). *Cluster based zoning of crime info.* Retrieved from: *https://ieeexplore.ieee.org/document/7905269*

Xiang Zhang (2010, September 9). *Detecting and mapping crime hotspots based on improved attribute oriented induce clustering.* Retrieved from: https://ieeexplore.ieee.org/document/5568075

B. Sivanagaleela (2019, October 10). *Crime Analysis and Prediction Using Fuzzy C-Means Algorithm*. Retrieved from: https://ieeexplore.ieee.org/document/8862691

Astari Retnowardhani (*2017, June 19). Classify interval range of crime forecasting for crime prevention decision making*. Retrieved from: *https://ieeexplore.ieee.org/document/7951409*

Spatial Dynamics of Crime. Retrieved from: http://www.riskterrainmodeling.com/overview.html

Risk Terrain Modelling, RTM. Retrieved from: *http://www.geog.leeds.ac.uk/courses/other/crime/risk-terrain/index.html*

Javed Anjum Sheikh (2017, 12 October). *IST: Role of GIS in crime mapping and analysis.* Retrieved from: *https://ieeexplore.ieee.org/document/8065761*

Xifan Zheng (2011, 12 August). *A mathematical modelling approach for geographic profiling and crime prediction.* Retrieved from: https://ieeexplore.ieee.org/document/5982362

[1] How many decimal digits for storing longitude and latitude?, Link: https://rapidlasso.com/2019/05/06/how-many-decimal-digits-for-storing-longitude-latitude/

Appendix

Import Average Duration (before optimisation)

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

Import Average Duration (after optimisation)

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

Test Case LC1

Test Case Description	Load the solution without any records in the database
Prerequisite steps	Empty database
Test steps	Load solution website
Expected outcome	Toolbar and empty map (map without any markers)
Actual outcome	As expected
Status	**PASS**

Test Case LC2

Test Case Description	Load the solution with 1 record in the database
Prerequisite steps	Database populated with 1 record (**directly using the DBMS tool**)
Test steps	Load solution website
Expected outcome	Toolbar and map with 1 marker
Actual outcome	As expected
Status	**PASS**

Test Case LC3

Test Case Description	Load the solution with 10 records in the database
Prerequisite steps	Database populated with 10 records (**directly using the DBMS tool**) with each of the records having a different Latitude and Longitude value
Test steps	Load solution website
Expected outcome	Toolbar and map with 10 markers
Actual outcome	As expected
Status	**PASS**

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**