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

April 2019

Literature Review

Introduction

An important point to note is that the following review is purposefully limited to researching literature concerning the use of computer systems for crime mapping (and its related processes) and therefore excludes or will not directly explore physical implementations of crime mapping. Their importance may still however be briefly described perhaps as context for how crime mapping has evolved or as justification for the need for computer systems as the way forward in crime mapping. Equally important to note is that the scope of the review is intended to extend beyond just crime mapping and will proceed to focus on three key concepts, **crime mapping** - an initial component of crime analysis involving the process of identifying where a crime has taken place in order to then map and visualise that crime, **crime analysis**, the process of evaluating crime data in an attempt to identify the existence of both short and long term patterns in crime, as well as **crime prediction/prevention**, the process of identifying and then preemptively putting a stop to potential crime in the future. 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.
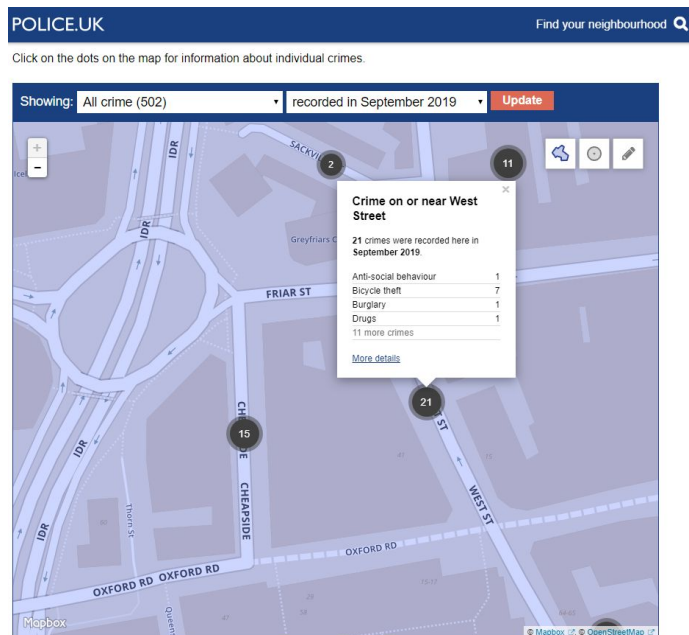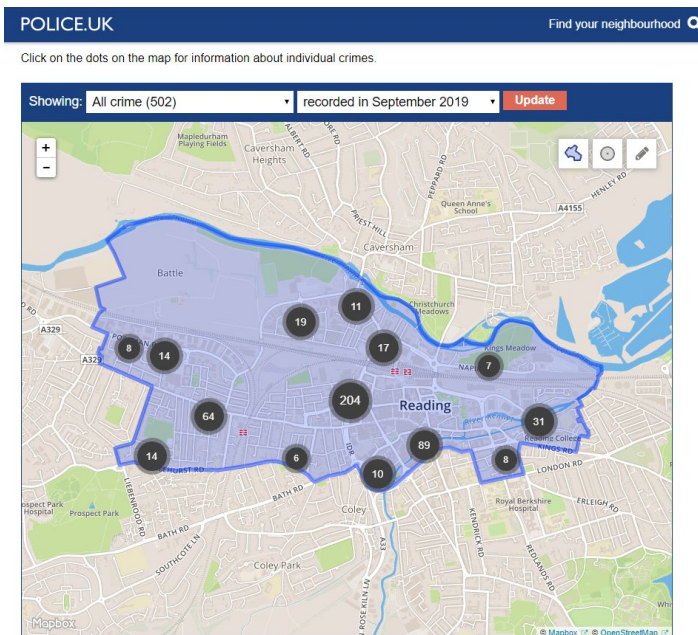
Shortly after the police crime map was launched, a newspaper article questioned its practicality. The writer proposed there was a degree of doubt amongst third party developers as to the reliability of the data made available and went as far as including a viewpoint suggesting the implementation had 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)
- Only textual representations of the types of crime in the form of information panels

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).

Aims and Objectives

Aims

Objectives

Solution Approach

Now knowing, from research, the approaches existing solutions have made and thus where the proposed solution could become established within the field and in order to best meet the aims and objectives defined in the previous section, a handful of proposed solutions can be devised with a final decision being made as to which approach is most suited to be carried forward as the proposed solution (approach).

Potential Solution (1) - Desktop Java Application

A crime mapping solution could be developed as a standalone desktop application developed using Java along with a mapping platform to provide the mapping functionality. As identified in the literature review, two such mapping platforms are Google Maps and Mapbox, but with no native APIs for the Google Maps mapping platform when developing for Java, the alternative platform, Mapbox, would be required. The Mapbox mapping platform provides a Java SDK that would allow for access to static map images and when used in conjunction with the JavaFX library, these maps could be embedded as part of the interface along with any necessary GUI elements or components. With no inbuilt functionality to add markers as part of the map, a custom overlay would have to be developed and added to the map in order to visualise crime and similarly being limited to static map images would mean complete map coverage is made inherently more difficult than having a map which is interactive and that dynamically updates. To overcome this problem, the design of the potential solution involves a collection of independent static maps for different regions (that is several static maps at predefined locations which are switched between as opposed to one dynamic map).

This proposed solution by the way of features could involve basic interaction with these static maps with markers or pointers being shown at the location where the map is clicked to map crime locations and once finished adding crime points, a snapshot or print of the resultant map could be saved for later viewing providing for retroactive visualisation of the crime. The proposed solution fills the role of a crime mapper which allows an individual user to perform quick and easy mapping and visualisation of crime but an important result of the design to note is that without designing any way to permanently store any user input or information, once the solution is closed, any changes made (i.e pointers placed) would not be shown when the solution is opened again and furthermore users would not be able to see the actions of other users, just their own.

+ No movement of data between the application and any server (all data is local and no delays waiting on back-end functionality)
+ Little in the way of technical complexity means deliverables could be produced within a short timeframe, allowing for users to be hands-on with the solution and provide valuable feedback sooner in the software development life cycle
- Only static map images being availables means mapping would be segmented and unintuitive adding unnecessary complexity to development and hindering the user experience crime
- Any mapped crimes are not saved once the solution is closed and would be lost
- Being a desktop application, the solution would need to be downloaded and installed in order to be used, posing as a barrier of entry to some users

Potential Solution (2) - Android Application

The solution could also alternatively be developed as an Android application for mobile devices with the mapping platform being provided through the use of the Google Maps SDK for Android. A map based on Google Map data could be added as the main component of the application and along with native functionality to add markers to plot crime locations and to navigate and update the map using map gestures and automatic handling of information from Google Maps servers respectively. Beyond devising a different development environment and platform or using a different mapping platform, this proposed solution is made to differ from the previous solution through a proposed feature to save the current state of the map so it can then be opened again at a later date (saves the actual map and all information added to it). This would be done by requesting a unique link for a map's current state and saving and opening this link as and when is needed as opposed to relying upon an external data store.

+ Improved freedom in design with access to mobile specific features such as touch gestures and development frameworks such as user preferences
+ Mapping platform chosen (Google Maps) is commonplace online or in other applications ensuring a familiar user interface to use as the mapping component
+ Mapped crimes are made semi-permanent with the ability to load maps containing markers indicating crime from previous use of the solution
- Developing an application and deploying it to be available for every mobile platform (e.g Android, iOS, Windows) increases development time
- Smaller screen sizes of target devices may add complexity to how or the amount of information which can be visualised at once, map navigation would similarly be affected
- Users will still need to download and install the app either directly from a package or from a digital distribution platform which may act as a barrier of entry to some users

Chosen Solution - Website

The other two proposed solutions due to the nature of their proposed development or because of the platform that they are targeted for, require to be downloaded and installed in order to be used. This barrier of entry which was identified as a disadvantage of both solutions can however be circumvented or at least significantly minimised by adopting a web-based approach and so therefore the chosen solution approach involves the development of a website which can be used simply with access to a web browser and will use the same mapping platform as proposed in the second potential solution, Google Maps.

With all users visiting the same website as opposed to running an executable or application locally, the aim of developing a collaborative crime mapping solution becomes more easily feasible, where users can see the additions made to the map by other users. The additions made such as markers on the map should be permanently stored using a database with the map's contents being updated and shown on the website (and therefore to every user) using the information currently stored in the database. With the feature to permanently store instances of crime mapping, features such as editing and deleting these crime additions should also be implemented with different levels of privileges for different types of users using an account system.

[IDE used, scripts or libraries used (Bootstrap, JQuery, Google Maps JavaScript API)]
[Coding languages]
[Why Bootstrap? Compare to other frameworks]

Design

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, not to mention how it affects more central or core functionality of crime mapping, such as efficiently moving around the map (navigation) and the visibility of information through data visualisation.

Context Menu (Initial Design)

An early design for the proposed solution aimed to overcome this problem was by not relying upon static elements and instead making use of a dynamically created context menu which would appear only when the map is clicked on and thus meaning the map could occupy the entire screen space when the menu is not currently open (the entire premise of making such a decision being to maximise the efficiency and effectiveness of using the map). The options found in this context menu would then open a popup window (designed in the next sub-section) temporarily overlaying the map titled and structured according to which option had been selected (to clarify these popup windows are designed to appear only temporarily, until closed, as opposed to being static and permanently apart of the main user interface). Frequently used options in this context menu would be highlighted in a different colour to the rest of the options, that is to say the colour acts as a signifier to the perceived, relative importance of the option. The process began with producing a low-fidelity prototype, simply as a way of brain-storming and recording ideas, before moving on to a high fidelity prototype (a computer based, interactive prototype which simply displays the name/text of the option the user selects), with the latter being a possible/crucial way to gauge and test the interface's effectiveness when interacted with by a user:

[Low-fidelity prototype, hand drawn mockup]
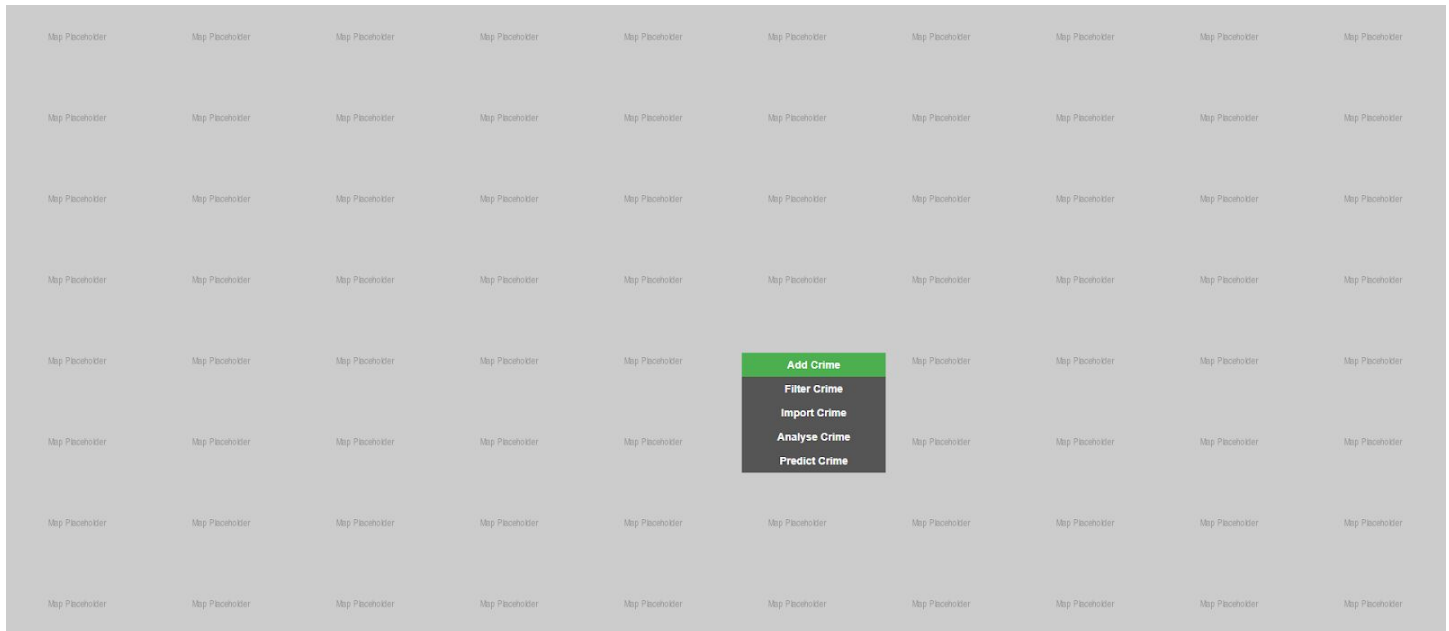
Whilst the context menu seemed for all intents and purposes suitable initially, when the high-fidelity prototype underwent usability testing, the problem of users accidently choosing the wrong option was frequently reported and along with it uncertainty as to what the precise nature of this problem was. After some clarification from the testers, contrary to what was first thought, this wasn't an issue of choosing an option but the functionality being unintended (a mistake error) such as clicking the 'Add Crime' button but the implementation displaying that crimes will now be filtered, instead it was the 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 to click). 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.

Grouped 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). At the time, a decision made to group some of these options into categories was made (a decision which was ultimately later reversed):

[Low-fidelity prototype, hand drawn mockup]
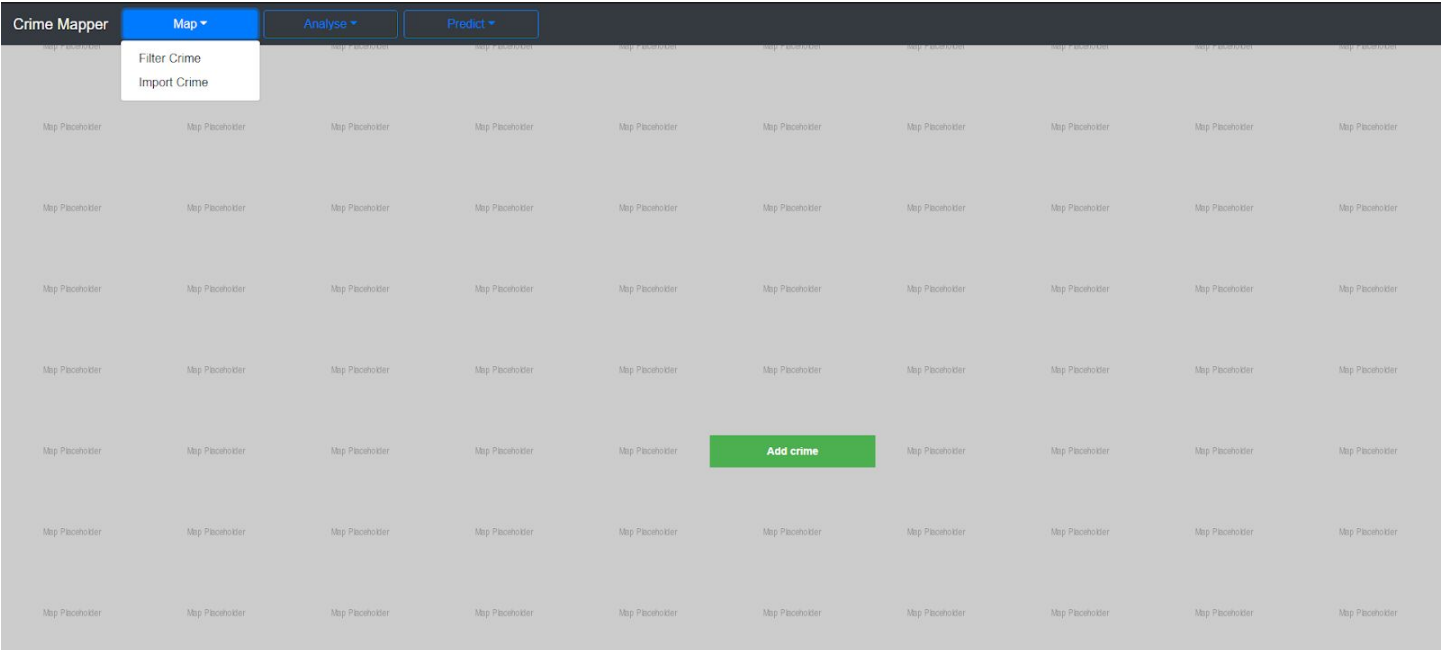
But the user feedback at this stage resembled how the user interface looked incomplete/unfinished, probably due to the large amount of unused space as well as some of the dropdown menus only including one option (there is little to no benefit in grouping a single option, the dropdown should just be directly replaced with the option.

Toolbar with search and Context Menu (Final Design)

The final iteration of design involved ungrouping the options and increasing their size along and making use of toolbar's empty space ny including a search bar (used to enter location names allowing for faster navigation of the map):

[Low-fidelity prototype, hand drawn mockup]
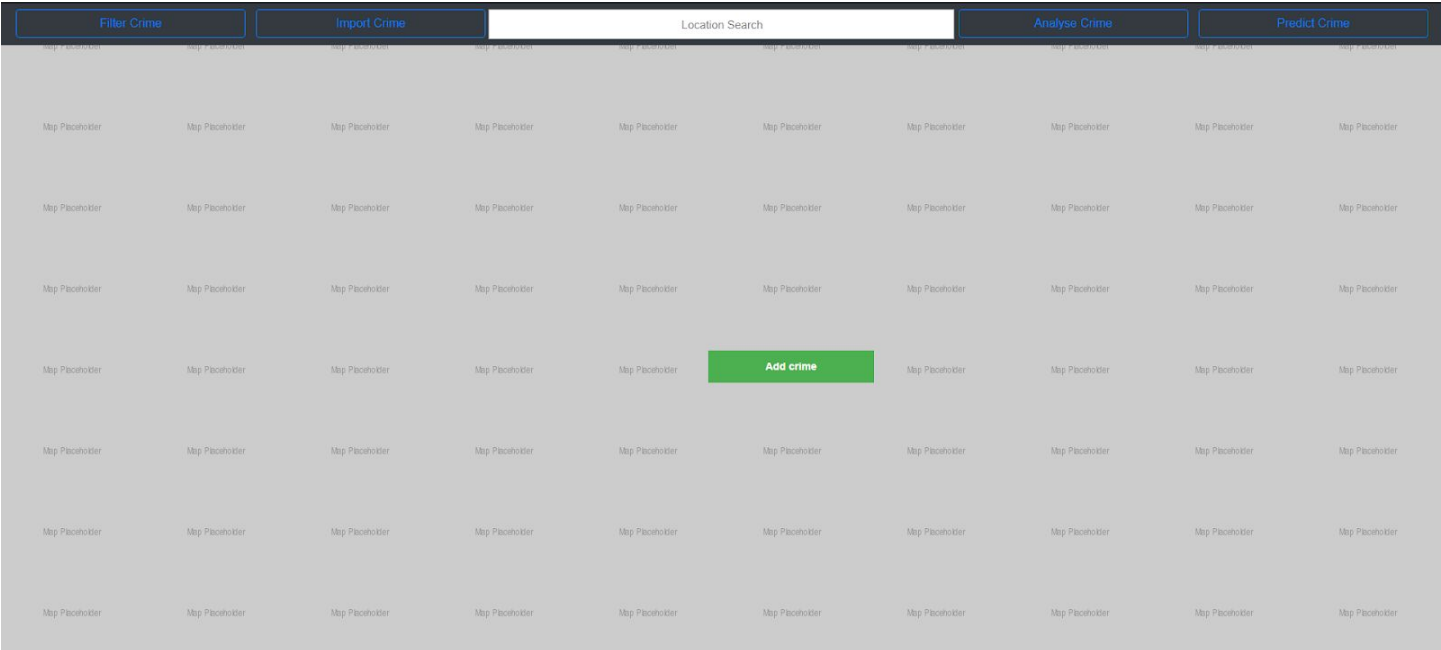
Popup Window - Base Design

Instead of designing the interface for each individual popup window, a base design was devised (in the form of a medium-fidelity prototype, with no functionality to provide there was no benefit in producing a high-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 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 indicate progression of a successful action) is clicked to prompt its functionality:

Title

[Low fidelity prototype, hand-drawn mockup]

[Body]

Confirm

Popup Window - Individual Designs

Building upon the base design, the individual designs below detail the inputs or other elements that will be needed to occupy the main body of these popup windows to provide their intended functionality.

Popup Window - 'Add Crime' Design

An input element to choose a date (labelled 'Date')

An input element to choose a time (labelled 'Time')

A dropdown list to choose a crime type from a list (labelled 'Type')

A smaller map to make smaller adjustments to the location (labelled 'Adjust Location')

A textarea to write a short description about a crime (labelled 'Description')

Popup Window - 'Filter Crime' Design

An input element to choose a date to filter by (labelled 'Date')

An input element to choose a time to filter by (labelled 'Time')

A dropdown list to choose a crime type to filter by (labelled 'Type')

Popup Window - 'Import Crime' Design

An input element to choose a local file to import (labelled 'Choose File')

A progress bar to show the progress of the import

Popup Window - 'Analyse Crime' Design

An input element to choose the analytical method

A smaller map to choose the location to apply the method to

Popup Window - 'Predict Crime' Design

An input element to choose the analytical method

A smaller map to choose the location to apply the method to

Database Design

Database Design - Conceptual

In order for the crimes being added to the crime mapper to persist, any related
data to the entity of the crime needs to be part of this database so that it can
be permanently stored and retrieved at a later date and with only one entity to
collect and store data for, only a single table is required, making for a simple conceptual model of the
database. This table is named *'markers'*.

Database Design - Logical

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, 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 ought to be made to differ slightly from those shown in the
above concept.

Database Design - Physical

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 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).
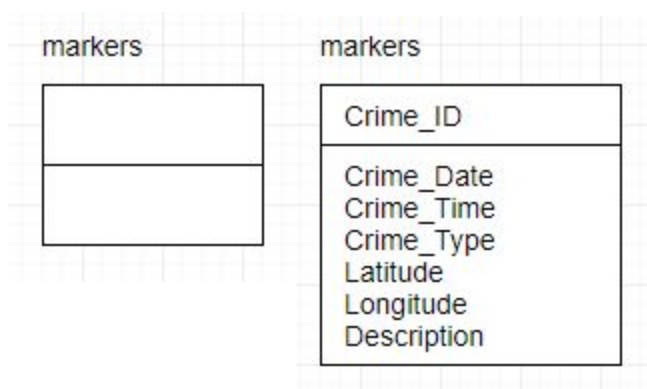
Database Management System (DBMS) and Administration
With the solution being web-based, MySQL will be the database management system of choice and for administration of the database, phpMyAdmin is to be used. Using phpMyAdmin, a database should be first created with the name 'crime_mapper', and the 'marker' table (designed above) should be added to it.

[Design what crime analysis methods will be used, how they will be implemented (clustering is one)]
[What will the markers look like on the map?]
[What is the user workflow to edit a marker or delete it?]
[Design of simply viewing information about a crime]

[May include more low and high-fidelity prototypes]

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 (to be processed by any PHP page) is to be sent using AJAX such as user input being stored to variables in JS and then being sent to the server and processed by PHP pages in order to interact with the database hosted on the server.

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 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 design created in the previous section 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 and the Google Maps JavaScript API to implement an interactive Google Map as the mapping platform. This webpage will also include back-end components to facilitate a connection to the also previously designed database. In order to connect to the MySQL server this database will be hosted on, a scripting language is required and such is why it is proposed the PHP language is to be used and thus meaning the filename of this main landing page is to be *index.php*. The back-end functionality of this particular page is designed to be limited to a connection to the database and the retrieval of all information about crime markers from the database ready for them to be placed on the map when the solution is opened. Keeping a local copy of the information retrieved would be good practice so that it can be referred to at any time if needed for any other functionality.

As mentioned previously, with only one web page containing HTML elements, the benefit of having an external CSS file and applying it across multiple web pages in order to control how elements are displayed (their layout), is not applicable for this solution. Despite this however, an external CSS file was chosen over inline styling because as can be seen with the interface designs, there are multiple identical or similar elements in need of having the same styling and was chosen over internal styling because there is still possibly the benefit of once the user has visited the solution at least once, their browser will cache the external file, improving load time the next time they visit 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. This external CSS file will have the filename *layout.css*.

Any remaining back-end functionality related to the database is to be implemented in additional PHP files. With each of these 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 each of these files, a dedicated file which could be included by these other files should be made, namely *dbConfig.php*. This file would contain information relating to the database host, the database name, ('crime_mapper'), as well as information for a universal user which can not only query information but also insert to or update and delete information found in the table of this database (actions which each have their own PHP file as designed below).

If a crime were to be requested to be deleted that would mean a record for that a crime already exists in the database and so that record of the crime should 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 DELETE statement to permanently delete the record in the database. Reflecting the actions performed, this statement and accompanying scripting would be included as part of a file named **DeleteMarkers.php**.
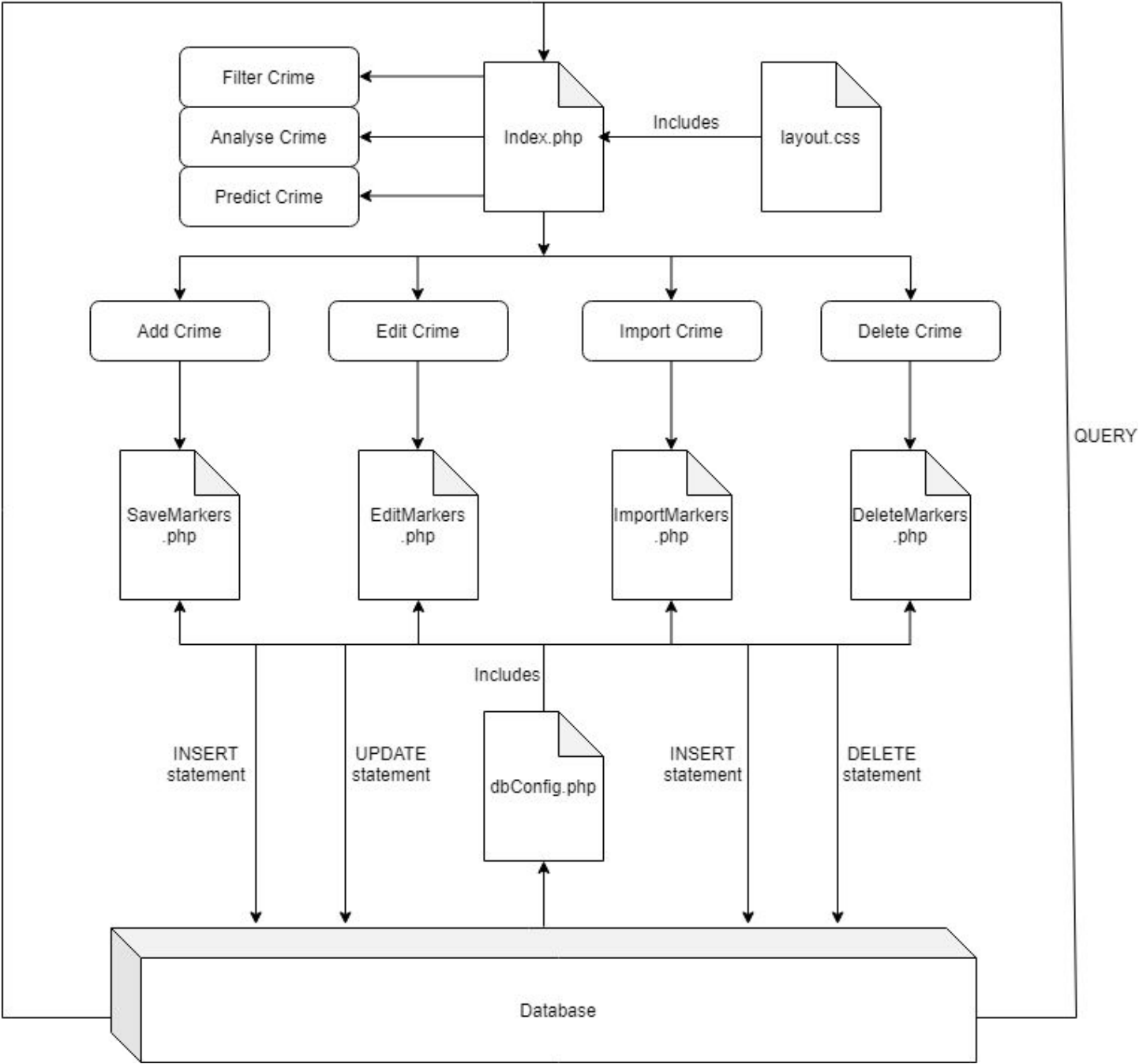
If a crime were to be requested to be edited that would similarly mean a record for that crime is already present in the database. As opposed to deleting the record and inserting a new record with the updated information, the updated information about the crime (the user input entered into elements of the popup window designed) should be sent to the server along with the relevant ID for the record in need of being updated. This bundle of information can then be used as part of an UPDATE statement to make the requested changes as part of a file named **EditMarkers.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 entered by the user in the popup window would need to be sent to the server and then be added to the database by the way of an INSERT statement. Additionally with the functionality of deleting a crime requiring a local reference of the ID that has been used in the database, the ID used when inserting this record should be sent back and kept track of in the local copy of crimes (not doing this would leave the crime without an ID locally until the solution is loaded again after which the ID can be determined when it is read from the database as part of loading crimes from the database in the index.php file, meaning the crime would not be able to be deleted soon after if required). All these actions should be included as part of a file named **SaveMarkers.php**.

Importing crime is simply the process of adding a record to the database as designed for the SaveMarkers.php file but being applied numerous times and to information found in an external file not information entered to a popup window. The interface of the relevant popup window for this functionality has been designed to include a file selector and so as a first step this PHP file, named **ImportMarkers.php**, would need to check this file is of appropriate type with the design choice that only .csv files are valid being made. The next check would be for appropriately named headers for the content of the file and only then should establishing a way of reading this file should then be conducted with the file first being seperated by each new line (indicating a new record) and then by the delimiter used (indicating each attribute of the record). It's important to note that when adding a single crime, effort to validate the user input is inherently made through the design of the elements used and the values they accept, enforcing validation of the contents of the file is however more troublesome. Erroneous values or missing values should be handled by skipping that line in the file but crucially from a security and stability perspective, all contents of the file should be treated as being malicious unless otherwise determined (a concept which should also be applied to any input in the popup windows where direct validation cannot be enforced). The values of each line found under the appropriately named headings when determined as being safe to be added to the database through use of an INSERT statement and just as before the ID of every record added should be returned in the order for which they were added.

Other options such as the options that filter, analyse and predict crime are designed not to have their own separate PHP files associated to them and their functionality is designed to be implemented as part of the main index.php file (they can make use of the local copy of crime markers and don't require any interaction with the database as part of their functionality).

Website Structure Diagram

Filter Crime

Analyse Crime

Predict Crime

Index.php

Includes

layout.css

Add Crime

Edit Crime

Import Crime

Delete Crime

SaveMarkers.php

EditMarkers.php

ImportMarkers.php

DeleteMarkers.php

INSERT statement

UPDATE statement

Includes

dbConfig.php

INSERT statement

DELETE statement

QUERY

Database

Implementation

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 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.

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 simply by adding to the tag as such:
<!DOCTYPE html>
[https://www.keycdn.com/blog/html-vs-html5]

Root Element

Whilst the version of HTML has been instructed to the browser 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 and closing tag:
<html> </html>

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 head element (three of which are described below) should be placed within its opening and closing tag:
<head> </head>

An example of a tag or element which must be placed within this head container is 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 as the web browser's tab for the page. As per the project title, the title of the webpage was set with the following:
<title>Crime Mapper</title>

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. A <meta> tag is used within the head container to set the viewport 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:
<meta name="viewport" content="width=device-width, initial-scale=1">

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 and following this, the style sheet created (that is the file named layout.css), is also linked to from this main index file simply using a link tag with the attributes specifying that the linked document is a stylesheet as well as the file location. With the files being within the same directory, the relative path simply needs to be the filename but if the location of the stylesheet were to ever change the file location would need to be the relative path from this main file. But as it stands at this stage in development, the link tag is as described:
<link rel="stylesheet" href="layout.css">

Content
Another container made to be within the root container is a container for the content of the webpage, by the way of a <body> container. With the preliminaries of the head container unlikely to drastically change as development continues, the majority if not all of the remaining implementation will be placed within the opening and closing tag of this container:
<body> </body>

The aforementioned external dependencies are added as scripts and are placed as a matter of best practice towards the end (still within) this newly created container of content (the document's body). These include three scripts needed by components of the Bootstrap framework (jQuery, Popper.js and additional Javascript plugins) as well as a script loading the Google Maps JavaScript API from a URL.

```
1   <!DOCTYPE html>
2   <html>
3
4   <!-- Metadata -->
5   <head>
6   <title>Crime Mapper</title>
7   <meta name="viewport" content="width=device-width, initial-scale=1">
8   <link rel="stylesheet" href="layout.css">
9   </head>
10
11  <body>
12
13
14  <!-- Content -->
15
16
17  <!-- External Dependencies -->
18  <script src="https://maps.googleapis.com/maps/api/js?key=AIzaSyDDpgBmZOTCzsVewLlzsx77Y5bDUVS_MZg" async defer></script>
19  <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.4.1/jquery.min.js"></script>
20  <script src="https://cdn.jsdelivr.net/npm/popper.js@1.16.0/dist/umd/popper.min.js"></script>
21  <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/js/bootstrap.min.js"></script>
22  </body>
23
24  </html>
```

Toolbar

The first action of any development is to implement the static toolbar along the top of the webpage. The provided navigation header of the BootStrap framework, navbar is adopted to achieve this. Within this navigation bar, there are to be four buttons and a search bar. The order in which these elements are implemented and will therefore appear in is two buttons then the search bar being positioned centrally followed by the remaining two buttons.

[Symmetrical show margin-left and margin-right percentage on a displayed scale
Two buttons left two buttons right, search bar in middle
Buttons same width, search bar occupies rest]

Navigation Buttons

(with their display text set as 'Filter Crime' and 'Import Crime')
(that are set with the labels 'Analyse Crime' and 'Predict Crime')

Search Bar

The search bar is implemented using a Google Maps SearchBoxshould allow for locations to be searched with the map being focused or centered to any location selected.

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, firstly by reserving an area for the map (within the body container) by adding 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 of this div in the linked to style sheet (referencing the div involves an id selector with the id attribute of the div appropriately being set as "map"). Importantly with these percentage values being a percentage in relation to their containing elements (namely body and above even that html), similar CSS was also applied to those elements. However these values are subject to change, likely when the static toolbar at the top of the page is to be implemented.

```css
1  html, body {
2      margin: 0;
3      height: 100%;
4      width: 100%;
5  }
```

```css
7   #map {
8       position: absolute;
9       height: 100%;
10      width: 100%;
11  }
```

At this point in time, using HTML and CSS, the created div element is simply an unapparent and empty block. This is because the map needs to be added inside the div through using JavaScript (thus requiring opening and closing script tags **<script> </script>**) and by creating a new map object and specifying this div as the element to hold it or be its containing element in this script (with it being that in this case, referencing the div involved using the document.getElementByID() method with document being a JavaScript object that is the parent to the webpage's HTML nodes). Along with specifying the container for the map, other required initialisation parameters included 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. In order to reference this map itself elsewhere in the codebase for future development, when created, the map object was also assigned to a local variable with this also appropriately being named as map.

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 would the map (be able to) be displayed. To ensure this happens, a callback feature can be added to the script used to load the API that involves a function being executed once the API is ready, with this function creating the map as outlined. Therefore the previously written JS code related to creating the map needed to be placed inside a function named initMap() and the URL specified for the script which loads the API be appended with the callback parameter alluding to this function as such:
&callback=initMap()

Placing Markers

The Google Maps JavaScript API has native Marker objects which can be constructed and added to the map component and so using a divide and conquer approach with an end goal for this module of code being to place a marker at the location for which the map was right clicked on, with a context menu being opened to confirm and bundle information along with the crime location, the first step taken can be to simply add a marker along with the map when it is created.

Predefined Marker

Creating a new marker object and providing the map component as its map property when it is constructed will place the marker on the map with the only other property required being a location for where on the map it is placed and just as with the center location of the map this is required to be a pair of a latitude value and longitude value. With an object already holding the required values representing the location of Reading, England, this object was used again as part of the marker's options, therefore meaning a marker is to be placed at the center location of the map when it is initialised:

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

[Large picture of full screen map with marker at Reading, England]

Location of Mouse Click (on map)

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 the click location can be recorded. A marker can then be placed by configuring the marker's location option to this recorded event location but instead of explicitly defining a new marker object each time a marker needs to be placed, a placeMarker() function with the two main options as parameters can be defined and called where and as needed:

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

```
26  function placeMarker(location, map) {
27      var marker = new google.maps.Marker({
28          position: location,
29          map: map
30      });
31  }
```

```
26  map.addListener('click', function(e) {
27      placeMarker(e.latLng, map);
28  });
```

Context Menu

In order to prevent undesired addition of markers (made by accidental clicks or slips onto the map), the concept that instead of immediately placing a marker, a right click context menu is opened with an option to place a marker, can be implemented, first involving the HTML elements that build up this context menu being created. The entire context menu can be derived with a div element, whilst the option of this context menu can be derived by nesting an additional div element inside this element. Importantly, these elements should not be initially visible and so to begin with the elements are hidden using CSS whilst also being positioned off the page as a precautionary measure. But when the context menu is requested with a right click on the map, the context menu can be shown by moving the position of the elements to the position on the screen clicked, unhiding the elements using CSS, as well as the click location on the map, in terms of latitude and longitude, being recorded in order to place a marker, if and only if the option of the context menu goes on to be selected:

Context Menu + Pop-up Window

Building upon the development so far, now the designed 'Add Crime' pop-up window can be made to appear when the context menu option is selected as a way to bundle any information entered along with the marker.

Using BootStrap's JavaScript modal plugin this modal (pop-up window) can be created with the inputs that make up this window being placed in the body of the modal. These include two different types (date and time) of input elements, select elements (the first of which determines the selectable values of the following element), a text area element, a new div element holding a smaller map and a button. These inputs should be nested inside a form element to accommodate for this information being sent to the database as well as their possible values being enforced with validation where possible. This smaller map is initialized to have a draggable marker at and be centered to where the map was clicked on originally to open the context menu (and subsequently the pop-up window). With the modal acting as additional confirmation, the direct functionality to add a marker through calling the placeMarker() function can therefore be migrated to the click event of the implemented modal's confirmation button making sure to place a marker on the main map at the same location as where the draggable marker in this modal is located, with the click event of the context menu option being repurposed to now only just showing this modal (shown left).

Storing Markers

The next stage for this module of code is to associate the information entered into the inputs of the modal with the relevant marker and to store these markers both locally and to the database. When the confirmation button of the modal is clicked, the values of the modal's elements (including the locational information of the draggable marker which may have been adjusted from the original location) are recorded. These set of values are to be used to locally store the marker.

Ideally these set of values would also be the format of data to be sent to the server (to be stored to the database) but with the decided upon client-server transmission method being an AJAX call, an encoded query string is required to be sent instead. Whilst each individual variable can be converted to this required format and appended together, a simpler method is to generate the majority of this query string by serializing the form (which surrounds all the inputs) and only applying this conversion to the latitude and longitude of the marker (due to the map not being included in this form), appending their values to this string. This complete bundle of data can then be sent using an AJAX call targeted at being handled by the 'SaveMarkers.php' file.
[SQL Injection]

In this 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 all values being received successfully (an additional check determining whether the form was successfully received as a whole along with the latitude and longitude values is made) a query is made to the connected database with this query including an INSERT SQL statement using these values. If this query successfully executes, the ID used when inserting this row into the database is returned so that it is made available to be stored locally, otherwise a value indicating an unsuccessful query is returned:

```php
34 □if(isset($_POST['submit']) && isset($_POST['Latitude']) && isset($_POST['Longitude'])) {
35        // Insert information into database
36 □    $sql = "INSERT INTO markers (Crime_Type, Crime_Date, Crime_Time, Description, Latitude, Longitude)
37            VALUES ('$crime_type', '$date', '$time', '$description', '$latitude', '$longitude')";
38        $query = $db->query($sql);
39    }
40 □else {
41        $query = false;
42    }
43
44 □if($query) { // Query successful
45        $id= mysqli_insert_id($db);
46        echo $id;
47    }
48 □else {
49        echo -1;
50    }
```

Back in the main file, the markers can be placed on the main map so long as the associated query was successful (i.e that marker is now stored in the database) which is determined by checking the result of the AJAX call (which will be whatever value is echoed). The function to place markers is updated to store the markers and their details locally with this being implemented by using properties for the markers and changing the placeMarker() function to accommodate for creating markers with these new properties and adding these created markers to an array (the function is now implemented as requiring a number of parameters which can be the markers' details allowing these details to be assigned as the markers' properties):

```javascript
761        /* Also send to database */
762        var formData = $("#add_submit_form").serialize();
763
764        var Vars = {Latitude: Latitude, Longitude: Longitude};
765        var varsData = $.param(Vars);
766
767        var data = formData + '&' + varsData;
768
769 □    $.ajax({
770            url: 'SaveMarkers.php',
771            type: 'POST',
772            data: data,
773            success: function(result)
774 □        {
775 □            if(result != -1) {
776 □                if (SmallMarkerMoved == true) {
777                        placeMarker(result,Crime_Type,Crime_Date,Crime_Time,Description,SecondLocation,map);
778                    }
779 □                else {
780                        placeMarker(result,Crime_Type,Crime_Date,Crime_Time,Description,FirstLocation,map); /
781                    }
782                }
783                SmallMarkerMoved = false;
784                $("#modal_add").modal('hide');
785            }
786
787        });
```

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 now be implemented. This can be achieved by using PHP and a SQL query from within the main webpage (a separate PHP file is not required). This query involves everything from the database table being retrieved using the SQL * operator. The result of this query is then fetched row by row with each row being read into an intermediary array. The values at every index for each of these intermediary arrays are then combined into a single line (separated by commas) and added to an all encompassing JS array that can be used locally. Markers can be then placed using the values of each row in this new array.

```
497   function LoadMarkers() {
498       var markers = [
499           <?php
500           $result = $db->query("SELECT * FROM markers");
501           if($result->num_rows > 0){
502               while($row = $result->fetch_assoc()){
503                   echo '['.$row['ID'].',"'.$row['Crime_Type'].'","'.$row['Crime_Date'].'","'.$row['Crime_Time'].'",
504                       "'.$row['Description'].'",'.$row['Latitude'].','.$row['Longitude'].'],';
505               }
506           }
507           ?>
508       ];
509
510       for( i = 0; i < markers.length; i++ ) {
511           var ID = markers[i][0];
512           var Crime_Type = markers[i][1];
513           var Crime_Date = markers[i][2];
514           var Crime_Time = markers[i][3];
515           var Description = markers[i][4];
516           var Point = new google.maps.LatLng(markers[i][5], markers[i][6]);
517           placeMarker(ID,Crime_Type,Crime_Date,Crime_Time,Description,Point,map);
518       }
519
520   }
```

InfoWindows

All markers, whether that be markers loaded from the database or newly added markers, now have properties associated with them (which detail the crime at the marker's location). 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 that when a marker is clicked on, these properties would be displayed. This is achieved by creating an InfoWindow object (a native object of the Google Maps library) for a marker as it is being placed. The content of these InfoWindows are simply the properties of the marker along with sub-headings that name these properties (detail what these properties are).

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 pop-up window the date is in the format 'DD-MM-YYYY' and so in order to reflect the design choices that have already been made in other parts of the solution, this is the format which should be shown and is enforced by using an external library (moment.js) which has inbuilt functions to convert dates between formats. 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 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.

Whilst, the InfoWindows are created for markers when they are placed, in order to be shown these InfoWindows need to be opened. By using the native event listener for a click on a marker, when the action has been heard, the InfoWindow for the marker in question (the marker has been clicked on) can be opened and shown. As a last addition to this module of code, although the property detailing the type of crime the marker represents is shown in the InfoWindow, it is also intuitively implemented as showing when the marker is hovered over by assigning this property as the value for the marker's 'title' property, a reserved property for a marker which has the functionality as just described.

Deleting Markers

The functionality of these recently implemented InfoWindow objects was extended by adding a button to delete the marker inside these InfoWindows. The implementation would involve that when this button is pressed, the associated marker (the marker clicked on to open the InfoWindow) would be deleted.

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 the 'Loading Markers' section. The marker, even without the webpage being reopened, would also still be present in the local array storing markers which is intended to be used to ease the implementation of future functionality. 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 array and even more importantly be deleted from the database

These different levels of deletion are implemented within a DeleteMarker() function which is called when the delete button within the InfoWindow is pressed and has a parameter providing a reference to the marker that is to be deleted, in the form of the ID (property) of the marker. Using the reference provided as the parameter to the function, the actual marker object is retrieved by looping through the local marker array and checking to see whether the current marker's ID matches with this reference, in which case this is the marker to be kept track of and is the marker to delete.

View

The marker to delete can be hidden from view using the native setVisible() method for marker objects by calling this method on the marker 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 be also hidden by being closed.

Array

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

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. This statement deletes the entire row that starts with the unique ID which had been sent:

```php
9    $sql = "DELETE FROM markers WHERE id = $MarkerID";
10   $db->query($sql);
11   ?>
```

Editing Markers

The InfoWindows are further extended still to include functionality to edit markers by implementing an edit button inside these InfoWindows. The implementation would involve that when this button is pressed, a modal designed similarly to the modal that shows when adding a crime is opened with its inputs being initially set to the marker's current values and being able to be edited and updated with these updates being reflected locally and in the database. The similarity between this functionality and the adding of crime markers (storing markers) occurs not only with the implementation of the relevant modals but also with the nature of the client-server transmission of data using AJAX calls.

Setting current values

After a similarly constructed modal (with the only differences being an adjusted title for the modal and text of the confirmation button) to the previous 'Add Crime' modal had been created, instead of these inputs beginning with having empty values, their values should be the properties of the marker as they are currently. For instance, the date input allowing for the marker's date to be edited should initially be assigned the value of the date property of this marker. A simple assignment of these input's values to their relevant properties suffices for most of the inputs but for one of the select elements, this approach is not entirely applicable. Whilst the subtype of the crime is a property of a marker and can be assigned directly in this way, the above select element of the main category of crime type is not and so the set value for this input must be somehow derived from the subcategory. 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 for as many options as their are in this array, there are the same number of subsequent arrays holding 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.

Updating values

When the confirmation button of the implemented 'Edit Crime' modal is pressed, a similar AJAX call to that used for 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). An UPDATE SQL statement is used to set the values for the record (where the record's ID matches the ID sent) to the newly entered values:

```
42  $sql = "UPDATE markers
43         SET Crime_Type = '$crime_type', Crime_Date = '$date', Crime_Time = '$time',
44         Description = '$description', Latitude = '$latitude', Longitude = '$longitude'
45         WHERE ID = '$id';";
46
47  $db->query($sql);
48  ?>
```

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.

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 examines the extension found in the filename (the string after the last '.') and the direct type of the file 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.

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, an example SQL Injection in need of being prevented (or protected against) can be explored to ascertain a better understanding of the possible attack vectors. 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. In a scenario where a user enters arbitrary and valid values for all fields up to this point but then enters the following crafted input for the 'Description' field:

Text', '50', '50');drop table markers;--

The query executed would be the following:

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

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 anything after this point in the query is ignored). 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 described in the 'Importing Markers' section) as opposed to when implemented for handling the addition of a single record or marker to the database through the provided functionality (as described in the 'Storing Markers' section).

SQL Injection - Implementation

Results and Testing

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/