BTech Project - HazApp

The Geospatial Hazard Management System

Rowan Carmichael
University of Auckland
June 2015

Abstract

This paper will be following my progress as a fourth year BTech student working with the software development team of Opus International Consultants Limited to solve a company-wide problem of on-site hazard reporting and management. Everything from the highest level of planning for the problem, to the lowest level programming will be documented, as well as research and recommendations for the technologies I see best fit for any design problems that arise, and any testing that has been to help us complete this project to the highest of quality.

Contents

Figures

Tables

Acknowledgements

A special thanks to Mano for giving me this wonderful opportunity. Also a massive thank you to the Opus team: Kodie Wixon, Taylor Carnell, Sulo Shanmuganathan, Roquito Lim and Mitchel Bennett, who have all made this process as equally enjoyable as it is fulfilling.

Chapter 1 - Planning

1. Project Introduction

The HazApp system is a proposed geospatial hazard management system to be used by Opus employees and contractors. It will consist of both a mobile app for on-site hazard reporting and real-time hazard alerts, and a desktop app for statistical analysis and management of hazards. The original proposal was an outcome of Opus' Big Ideas Competition [1] as an improvement to Opus' current paper-based hazard reporting system.

1.1 The Company

Opus International Consultants Limited is a multi-disciplinary international consultancy company consisting of over 3,000 engineers, designers, planners, researchers and advisors, situated across 5 countries (New Zealand, Australia, Canada, America, and the United Kingdom). Their work services include transport asset development, building design, water, and other infrastructure. Because of the nature of their work fields, many Opus employees and contractors working for Opus are very often found on work sites (rather than in the office).

Some of Opus' more recent projects include, but are not limited to, the Newmarket Rail Station redevelopment, Ngatamariki Geothermal Power Plant construction, the Waikato Expressway, and the Carterton Events Centre. All of these projects are of massive scale and as such are relatively prone to on-site hazards. HazApp hope to minimise the time spent towards reporting and managing on-site hazards so that work can continue on the more important aspects of planning and construction.

1.2 The Problem

Offering professional consultancy services in asset development and management often requires Opus employees to be working on-site where if any hazards occur they must be reported and stored for management, however the current hazard reporting system is a paper-based form (See Appendix Item 1 for a sample hazard reporting sheet) which is tedious to complete, takes time to be transferred into a database, and does not offer any advice or alerts to the reporter or anyone else working on the same site.

As well as having an inefficient paper-based hazard reporting system, each different Opus office (both nationally and internationally) has a different means of reporting and storing the hazard data. This lack of connectivity has misaligned Opus' safety and business practices and is continuing to promote an absence of interconnectivity within the company.

1.3 The Project

HazApp was created to rectify these problems by both moving the hazard reporting away from paper towards utilisation of smartphones and tablets, and realigning databases to make the best use of the reported hazard data. Using a mobile map-based hazard reporting not only allows for real-time hazard reporting and management, it also has room to offer immediate advice and mitigation devices to best handle reported hazards. Using a desktop management system with a single global database allows for powerful statistical analysis.

2. Technologies

Due to the general complexities and cross-platform nature of the project, HazApp will be heavily reliant on technologies if it is to be a success. The discussion following will outline key decisions defining how the project is to be made and how it will function. The discussion and ultimate decisions will be based on a review of current technologies that are able to fit the required function specifications. These will range from development languages, technical application program interfaces (APIs), user interface (UI) frameworks, and database systems.

I will also note that in many of these cases, decisions will not be made until development starts, and any decisions that are made here may be subject to change when actual development does commence.

2.1 Programming Language

Seeing as this project requires both a mobile (smartphone and tablet) and desktop component it is crucial to decide on a language (or languages) that accommodate the platforms and functionality of the application. The primary programming languages that I will be considering for this project are; native app for mobile (consisting of some/all of Android, iOS, and Windows Phone), native application for desktop computer (likely in either C#.net or Java), a web based application (HTML5) for both mobile and desktop, or an HTML-built application utilising PhoneGap.

Native Mobile App

Perhaps the most obvious solution for the mobile side of this project, native mobile apps allow for very powerful functionalities and generally faster speeds for functionalities such as use of GPS tracking and photo capturing (when compared to web based applications). Another attractive benefit for developing a native mobile app is the ease of use for either offline work or in areas that have sporadic network connectivity. This is going to be something that may not be a primary decision factor now, but will definitely need to be considered for the completion of this project.

One big drawback for both forms of native applications, in comparison to a web application, is that if the project is ever updated (which will most definitely occur), the native applications will have to be manually updated. This is not the case for the web application as all of the updates will occur server-side so whenever a user goes to the web page it will be showing the most updated version. Along with having to update the application where necessary, both native apps require disk space on the device being used. While this may not be an issue for a desktop computer, it is something we have to keep in mind for mobile devices with limited storage space.

Unfortunately another major drawback of creating a native mobile app is that it would likely like more than one language codebase (as some combination of Android, iOS, and Windows Phone) for the mobile side alone. Taking into considerations the limitations of time, expertise, and money we have decided that a native mobile app approach would not suit what we hope to achieve.

Native Desktop App

Similar to native mobile apps, a native desktop application can offer increased speed and functionality in comparison to a web application. However it also shares the same drawback of requiring separate code bases for both the mobile and desktop sides. With this in mind it is clear that the two options we could take in terms of coding languages are either a native mobile app and a native desktop app, or a web app for both mobile and desktop.

Web App

The limitations of web applications compared to native mobile and desktop applications somewhat numerous, however due to the relatively simple functionality of the project, we feel as though a HTML5 web application using JavaScript would more than suffice the primary needed functionality of mapping, geolocation tracking, photo/video capturing, and connectivity to a server side database.

As mentioned earlier, having some functionality for offline use or intermittent internet connectivity needs to be considered. Fortunately there are options utilising HTML5 and JavaScript which allow for such functionalities. The possible solutions to this problem will be assessed later.

Finally, using a HTML5 web based application will allow for a lot of code sharing between the mobile and desktop application. This is an incredibly attractive trait of web applications and is a main contributor to why we have ultimately chosen to develop a web application in HTML5, utilising JavaScript APIs, and a PHP database.

PhoneGap

PhoneGap is a special case that we will be also looking into specifically for the mobile and tablet side of our project. In terms of development it should be exactly the same as a web application using HTML5, CSS, and JavaScript. However it differs in how it is deployed, instead of being a web based application PhoneGap would allow for out HTML/CSS/JavaScript codebase to be converted into native mobile applications for Android, iOS, and Windows Phone. This would eliminate the problem of having multiple code bases for each different device type and could also allow for better local storage on the device for offline or poor-connection usage. While in theory this sounds great it may not be as easy to include such functionalities. [2]

Although this seems like a good middle ground decision it does not come without its own faults. The most glaring problem that would most likely arise is decreased performance compared to a regular native application or a web based application. From what has been suggested from some of the other Opus developers, PhoneGap may not be the best choice if we are interested in having decent performance speeds from out application. From what I have gathered PhoneGap’s conversion comes with a cost, and seeing as it is important for our app to function fast enough as to not have a negative impact on the user’s view of the app, PhoneGap may not be the most obvious choice.

Decision

Due to the very small size of our development team and the large size of the project itself we have decided that we will be developing using HTML5 for both the desktop and mobile/tablet side of the application. As mentioned earlier HTML5 comes with more than enough functionalities to accommodate the project requirements and by ch0osing HTML5 we will be able to share a decent amount of code between the two sides of the application.

2.2 Database Management Language

One of the key inspirations for this project was to connect the entire Opus community through a global database system, as such the decision for the database management language is very important to the longevity of the project. As discussed earlier we are going to be creating a HTML5 based web application and as such we will be using PHP to connect client and server databases. In terms of database implementations we will be looking into two of the most popular options being a regular SQL database and a NoSQL database.

SQL

SQL (structured query language) databases have been around for many years, with their first appearance in 1974 and initial release in 1986 and as such have been the dominating framework for databases up to present day. The biggest difference between SQL and NoSQL is that SQL databases are primarily relational databases utilising tables containing data fitted into predefined categories. While SQL has been around significantly longer than NoSQL it does come with its limitations. The major limitations to note are scalability and complexity. As SQL uses relational databases scaling the size of a database is an expensive and difficult task which requires powerful servers. The other major drawback as compared to NoSQL databases is the complexity of relationships within the database. SQL requires a network of tables all connected through some means of relationship strings. An implication of this is that altering the design of the database structure can be very complex and can downright break your database (especially in the case of deleting data/tables).

One benefit that regular SQL databases have over NoSQL databases is that, for complex queries, SQL offers standard interfaces aiding in working with such queries. In general SQL databases are best fit for heavy duty transactional type applications, the reason being is that they offer more stability and promise atomicity as well as integrity of the data. This is emphasised through SQL's ACID properties (Atomicity, Consistency, Isolation and Durability). The HazApp project will be including some form of transactions (most likely on the management side of the application) so the benefit of stability and atomicity will be kept in mind.

Finally the last advantage to note that SQL has over NoSQL is that, as it has been around for so much longer, SQL offers excellent support for their databases from vendors. Whereas NoSQL largely has to rely on community support. [3]

NoSQL

NoSQL databases have surged in popularity since their release in 1998. The aim of NoSQL was to move away from the idea of concrete relational tables for a more flexible framework. NoSQL databases focus more on key-value pairs, no longer requiring fixed table schemas and relational join operations. They have traded off the ACID properties for Brewer's CAP (Consistency, Availability, Partition tolerance) theorem.

One of the bigger positives NoSQL has over has over regular SQL is that no schema are required. That is to say data can be inserted into a database without having to define a rigid database schema. This also allows the format of the data being inserted to be changed at any time without application disruption, leading to massive application flexibility. In general NoSQL databases process data faster than relational databases as their data models are more flexible and often simpler. [4]

While both SQL and NoSQL databases have their own differences and benefits, we are yet to make a decision on what database type we will be implementing. This decision will be made closer to development start.

2.3 JavaScript Frameworks

Selecting appropriate JavaScript frameworks can greatly reduce the need for tedious manual calculations. Essentially what we are looking for in a JavaScript framework is functionalities which will aid help with manipulation of the webpage's data through things such as functions and bindings. In this case we will not necessarily be settling for a single JavaScript framework, but instead we may use several in different areas which we see fit. For general JavaScript we will be considering Knockout.js, Angular.js and Backbone.js. This decision will likely be influenced by how the JavaScript framework combined with the web app framework. For offline storage functionalities we will be looking at Lawnchair and Local Forage.

Knockout

While Knockout, AngularJS, and Backbone all offer some of the same very useful functionalities that regular JavaScript does not naturally support (such as data binding and DOM templating of code into smaller maintainable pieces), Knockout is different as it is primarily a lightweight data-binding library. Unlike AngularJS it has explicitly put work towards focusing on unobtrusive code, which could be important for our project. While at times each of these three frameworks may outperform the others in terms of performance speed, Knockout has a stronger focus on speed and should offer better performance than the other two for common tasks that we will be implementing. [5]

Angular

Angular is different to both Knockout and Backbone as it is a full-fledged framework (rather than a lightweight one). It has be built from testability and as such of this can clear project organisation more effectively that the other two alternatives. It is the "heaviest" of the three frameworks, and because of this it can offer more luxury functionalities such as custom elements. It is difficult to say whether or not these extra functionalities will be of any benefit without having started development yet. [6]

Backbone

More similar to Knockout, Backbone is a lightweight JavaScript framework and as such in general it will also perform better that AngularJS in terms of speed. Unfortunately this comes at a cost; while Backbone excels in simple applications, it may fall behind when dealing with heavy built-in data interactivity or extensive scaling. As mentioned earlier, it is difficult to tell which of these frameworks will suit out problem, although in terms of the mobile side of the application we are going to try and make it as simple as possible so both Knockout and Backbone may have the slight edge at the moment. It is going to be our job when we start developing to identify and handle the balancing act between these frameworks. [7]

With all of this in mind I will conclude that all of these frameworks essentially are solving the same problems. There are small differences between the functionalities and syntax between the three but ultimately any of these frameworks seem as though they would adequately work for our project. While this decision is still undecided, it will probably be influenced by more specific problems we encounter while actually developing. If any one framework can manage a specific problem better than the others, it will most likely be the one we use, however that will be judged on a case-by-case basis.

Lawnchair

Again Lawnchair and Local Forage share the same roll of maintaining data offline. Although we are still in a very early stage of planning and offline data management is rather low on our priority list I still thought it would be beneficial to review two of the most popular options for local HTML5 storage using json.

Lawnchair has been designed with mobile in mind, which is great to hear as the mobile hazard reporting side of our project is where we will be wanting offline support. It has a few very simple but powerful functionalities which cover the basis of offline data storage. These include mapping key-value pairs, saving them to a local “store”, and accessing them later. Unlike Local Forage, Lawnchair has stopped releasing new builds and has been “finished” as a project. This means that as time goes on it most likely will fall further and further behind the regularly updated Local Forage. [8]

Local Forage

Mozilla’s Local Forage seems to be a more complete solution to the problem of local storage for HTML5. It shares a similar methodology of saving and retrieving data as Lawnchair, but it also offers built in error handling. This essentially means it is slightly more complex but covers lightly more functionality than Lawnchair. As mentioned earlier it is continually updated and from what I have gathered has far more extensive documentation and support. [9]

At this time in our research, offering offline storage functionalities has one of the lowest priorities and will probably not be mentioned again until we are nearing our final release.

2.4 Mobile Web App Frameworks

The web app frameworks we will be deciding to use will primarily be based on user interface and ease of use. With this in mind we will only be looking at a select few (although there are a numerous amount of potential contenders) that we deem most likely to fit our needs. We will consider JQuery Mobile, Ionic Review, and Bootstrap.

JQuery Mobile

JQuery Mobile is a very well know and very popular choice for HTML5/CSS/JavaScript development for smart phones and tablets. It is a very easy to use framework that does a lot of useful work for you (especially in terms of automatically generated user interfaces). It is such a popular choice for smart phones and tablets as it includes a very clever built in scaling system so that you program can easily be transferable between many different screen sizes. In essence JQuery Mobile is a minimalistic upgrade to JQuery designed for responsive web pages and platform independent applications.

Another great benefit of JQuery Mobile is that seeing as it is so simple, it is incredibly easy to extend further JavaScript libraries. As such it should be able to fit well with any of the JavaScript frameworks discussed above. As well as working great for mobile devices JQuery Mobile also offers smart designs and implementations on desktop applications. This may come into our decision making process as it is important to have coherency between out mobile and desktop application which can be boosted by having a similar user interface style for both.

The final benefit that JQuery Mobile which is very attractive for our project is the fact that it offers a lot of mobile-specific function handling such as swipe-events, page transitions and touch-friendly components. However while it does offer a lot of useful functions it can be have very slow performance, especially if the application is not designed properly. [10]

Ionic

The Ionic framework is the most recently created web app framework we will be considering, with the alpha release in November 2013. Similar to JQuery Mobile, Ionic Review primarily focusses on the user interface, however it differs in the fact that it is built on top of Google's AngularJS framework. This pairing is a necessity for Ionic to function to its fullest potential so if we were to choose it we would also have to be working with AngularJS.

Another similarity Ionic shares with JQuery Mobile is having a strong focus on responsive web design, which is a big plus. We will be wanting to have our application provide optimal viewing and interaction experience, as well as quick and easy navigation, and the ability to function on a wide variety of devices. By utilising a responsive web design to its maximum potential, we should be able to share a lot of code between out desktop and mobile applications. [11]

Bootstrap

Bootstrap is a front-end framework which also offers a number of great user interface components such as dropdowns, breadcrumb navigations, and button groups. Unlike JQuery Mobile, it has not been designed to primarily focus on mobile applications and as such seems to have the appearance of a desktop application (even when on a smartphone screen). To fix this, custom code would be necessary. As it is less dependent of JQuery, it generally will exhibit better performance. [12]

While we will be developing for both mobile and desktop we are yet to decide if we will use any of these frameworks for both sides of the application or if we will divide our application by using different frameworks for the two sizes (for example JQuery Mobile for the mobile/tablet side, and Bootstrap for the desktop side).

2.5 Stylesheet Languages

While not a major priority for the project, utilising an effective stylesheet language that can be compiled into CSS, can make the CSS code easier to understand and simpler to create. The two big names in this area that we will be considering are Less and Sass. It should be noted that while we will be considering both, we may end up not using either and just stick to regular CSS.

Both Less and Sass share a lot of syntax and functionalities, and are essentially attempting to solve the same problem of decreasing the amount of code needed for stylesheets through added functionalities. These include but are not limited to:

- Mixins: which allow embedding of properties of a class into other classes, creating a soft of variable which can be repeatedly used.
- Parametric Mixins: act as functions by allowing passing of parameters
- Nesting: similar to a nesting in a language like Java, cuts down on repetitive code
- Functions and Operators: allows for mathematical equations within your CSS code (for example taking a colour variable and making it slightly darker by adding to the RGB value)

- Namespaces: which are groups of styles that can be called by references (rather than requiring several CSS files)

Less

While both Less and Sass are pre-processors for CSS, Less has been greatly influenced by Sass. This is very apparent in the shared functionalities of the two. One difference between the two is that Less is a JavaScript library and is processed client-side. Being a JavaScript library it is incredibly easy to incorporate into a web based application. All that is needed is two extra lines of code in the HTML file, one referring to the .less file and one referring the less.js file. [13]

Sass

As stated earlier the one significant difference between Less and Sass is that Sass is not a JavaScript library, it instead uses Ruby. However it seems as though this is not a big deal at all as if I were to develop using either of these frameworks I wold have to be learning new syntax anyway. [14]

It seems as though if we do decide to use either of these frameworks it will most likely come down to personal preference as the differences between the two seem to be minimal. However we will probably be ignoring these for the start of our development as we will be wanting to only focus on the most necessary functionalities.

2.6 Mapping API

It is of immense importance to get the mapping technology that will best suit our project. The primary properties we will need from our mapping API is an attractive interface, capabilities to effectively send user input to, and retrieve and map data from a server's database. We would also really like to have easy and flexible movement options for the user (in particular moving the map location and zooming), and a fast overall performance. Although the mapping API ArcGIS has already been chosen by Opus for this project (as for all mapping-based projects within Opus ArcGIS has been used) I will still review this API in hope that I will gain a better understanding on how I will end up developing with it.

ArcGIS

The ArcGIS engine allows for adding dynamic mapping and geographic information system (GIS) capabilities to both existing applications and custom built mapping applications. Some very useful features include creating custom and prebuilt drawing/graphics features, such as points, lines, and polygons. These graphical features are not just for show, ArcGIS offers powerful manipulation and geographic operations on these shapes, such as calculating differences, finding intersections, and even assigning points on a map to database objects. This is exactly the kind of functionality we are looking for in our HazApp project. As well as the interfacing side of the mapping technology, ArcGIS also offers network analysis which is another crucial part to the success of our project. All this is very well documented and there are many demos on their site which will most definitely speed up my learning process as so far I have had no exposure to ArcGIS. [15]

In terms of the visuals of the mapping, ArcGIS seems to offer an abundance of choices. Again this is a massive positive as we can customise how the map looks to best represent the data, and maximise the ease of use for users. I will be reviewing some of the more specific visualisation options later in the planning phase.

3. Planning and Design

Now that the majority of decisions have been made on the technologies we will be using to construct HazApp we now have the opportunity to move onto planning specifics of the project. Again, seeing as we are at an early stage in development, it is very important to create a sound foundation before development starts. Our planning and design phase will cover the underlying database design, a basic user interface design, and an initial plan for our development phase.

It should be noted that while we are still only in the planning and design phase any decisions made here are subject to change.

3.1 Database Design

The decision has been made for this project to be implemented using a PostgreSQL database [16]. The reason we have chosen to go with a more typical SQL database rather than a NoSQL database is due to the relational nature of the data we will be storing. The figure below shows our first ERD (Entity Relationship Diagram) including the fundamental tables, attributes, and relations.

To begin the planning we started with mapping the most important entity in our database: the Hazards. From this we expanded the database outwards, keeping in mind the relationships that would follow the new tables. The most important tables that we identified were: Hazards, Projects, Users, Lookups, Attachments, User_history, and Sync_history. While the Hazards, Projects, and Users tables are fairly self-explanatory, I will be briefly covering a few of the details of the less obvious tables. To start, the Lookups table is to help define and categorise specific hazards. Also closely related to the Hazards table is the Attachments table, this is to be used for linking photos/videos with reported hazards. And finally, User_history and Sync_history are there to offer some form of documentation on users and to help synchronising the database (for example in times of a lack of internet connectivity).

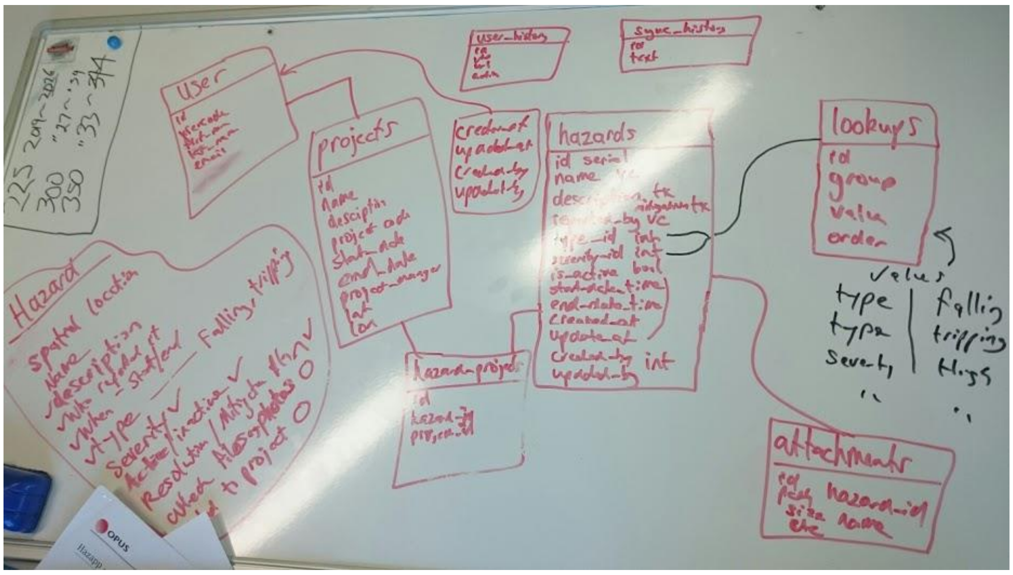

Figure 1 - First ERD draft

As this was our very first draft of the database design there was a lot of room for change. Through several iterations of reviewing the ERD we finalised the design (as shown below) with several changes. These include but are not limited to the addition of attribute variables, further defining the relationships (as well as the relationship types), and the addition of new relational tables (in this case the table User_project which is used to resolve the many-to-many relationship between the User and Project tables).

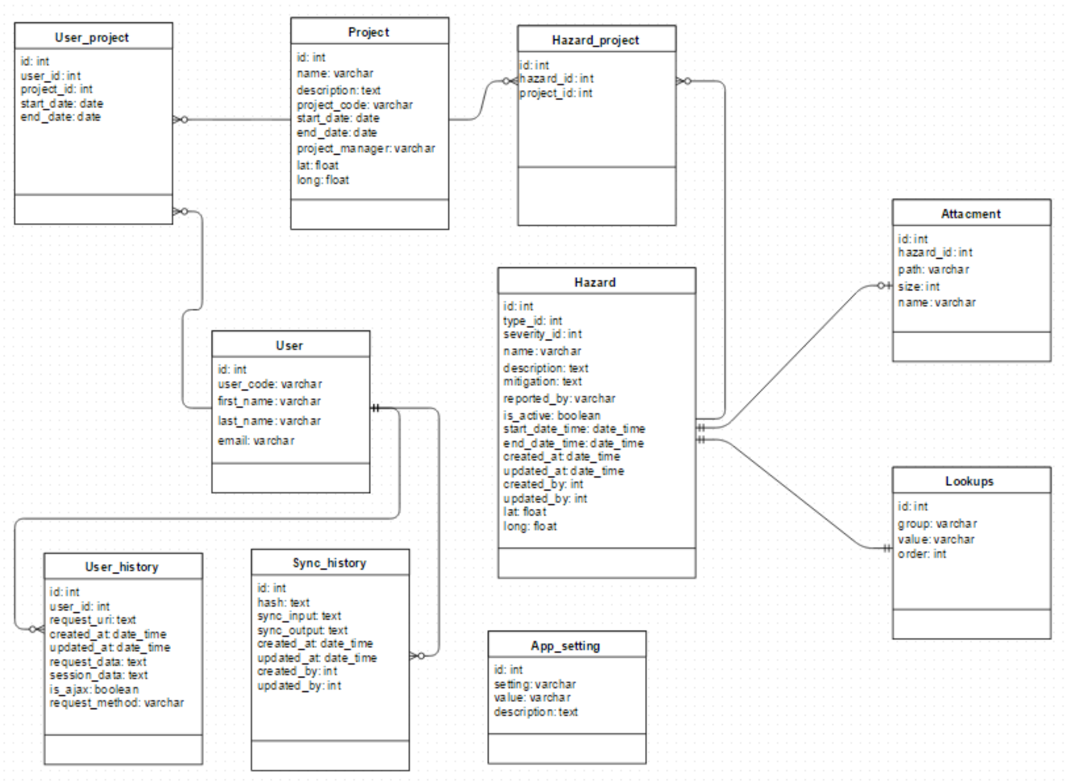

Figure 2 - First ERD

This ERD represents the basis of our database for the first implementation of our project. After creating it and going through several iterations (using draw.io [17]), we have now converted this design into a PostgreSQL database using pgAdmin 3 [18]. This allows us to quickly and easily edit the database through a simple interface, and it also has allowed me to set up a local version of the database on my machine (for testing purposes).

3.2 User Interface Design

The user interface is not the highest priority at the moment, however it is important to create an easy to use interface that the first testers (Opus employees) will be able to understand and use effectively. For our first prototype not much time will be allocated into making the user interface look "nice." This time will instead be put towards the primary functionalities of the application. With that being said, the first prototype will ultimately be used by a group of Opus employees, and as such we don't want the user interface to have any negative connotation due to the way it looks. It should be as simple and clean as possible.

The following figures are mock ups created at the very first idea proposal stage and were used as a tool to aid HazApp's application process. While they are slightly dated and a lot has changed since my involvement in the project, I will be using these as a very rough template for the user interface design I will be working on.

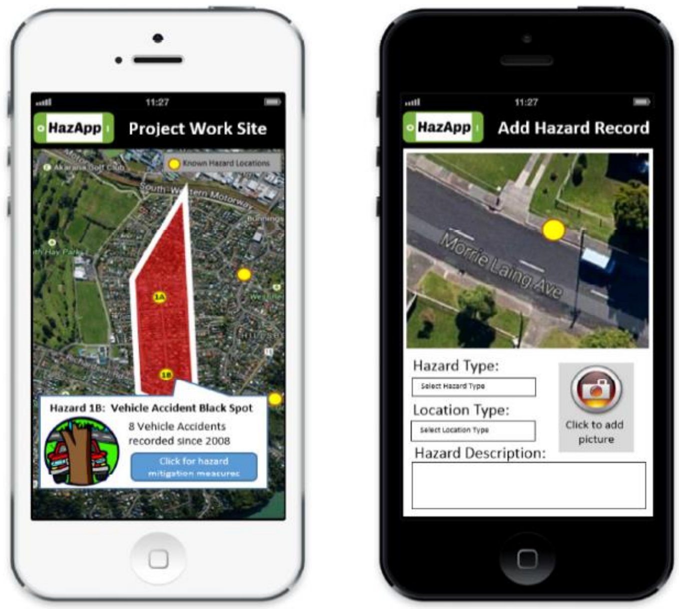

Figure 3 - Mobile user interface prototype [1]

I really like the idea of having a full screen map view that showing the local area's project sites and hazards. Due to the limited screen sizes of mobiles, maximising the efficiency of space by minimising the amount of unneeded clutter is a must. Similar to the above figure I will also include a small hazard form popup that will take up about half a screen when someone wishes to view or log a hazard. This should still leave enough room for the user to accurately select the correct area on the map.

3.3 Progress Plan

With a more concrete game plan forming we are now working on a scheduled plan for the first release of the mobile and desktop application. The table below is our finalised development plan for this stage. It should be noted that this plan covers all aspects project and there will be some areas that I will not be directly involved in (mainly the desktop side of the application). This is primarily to allow me to focus purely on the mobile side of the application. In terms of general development the areas that are of critical importance to me are the database design (which will be implemented as a PostgreSQL database), accessing the database and performing simple CRUD (Create, Read, Update and Delete) operations using PHP, the ability to add hazards to the database using a mobile interface I will be creating (this should also incorporate the mapping technology ArcGIS), and the ability to retrieve all local hazards from the database and display them via the ArcGIS map on the mobile application.

Type	**Feature**	**Description**	**Iteration (Week)**	**Due Date**
Design Stage	Database	Design database (ERD)	0	28th April
Project Initiation	Planning	Low level task assignment and plan	0	29th April
First Release Dev	Database	Setup dev, uat databases with Postgis	0	6th May
First Release Dev	Database	Implement database	1	13th May

First Release Dev	Base Application	Base CI app with required libraries etc	1	13th May
First Release Dev	Base Application	Mapping requirements	2	20th May
First Release Dev	Base Application	Theme, styling, look / feel. mock pages	2	20th May
First Release Dev	Adding a Hazard on Desktop	CRUD for hazards	3	27th May
First Release Dev	Hazard List	Prepare a list of the hazards in the system, with an ability to search and query	3	27th May
First Release Dev	Base Application	Configure to authenticate with AD	4	3rd June
First Release Dev	Base Mobile Application	Base app setup for the HTML5 WebApps	4	3rd June
First Release Dev	Mobile App Add Hazards	Ability to add hazards on the mobile interface	5	10th June
First Release Dev	Mobile App View Hazards	Ability to view all hazards on the mobile device (using GPS as well)	5	10th June
First Release Dev	Hazard Map	Map view of all hazards with the additional of other spatial queries	6	17th June
First Release Admin	UAT Feedback	To prepare users for UAT and any adjustments before commencing full pilot	7	24th June

Table 1 – Progress plan for first release

As seen in the table above, the final due date we are aiming for (at least for our initial prototype) is the 24th of June. At this point we hope to have a working desktop and mobile application that can be tested in a real world environment by a small group of Opus employees. As it is the first prototype we will only be focussing on the most important aspects of the application that we deem necessary. This means we will not be implementing functions such as offline use or statistical visualisations until a later release. Assuming all goes well the first prototype will be the most important part of the project so far. It will enable us to truly see if the idea is viable, and it will definitely highlight aspects that are weak in the design.

Chapter 2 - Development

4. First Prototype

After the successful completion of our planning phases we will now be beginning the development on our first prototype. As mentioned in the project plan, I will primarily be focussing my development on the mobile side of the project, and as such this chapter will be following development from my point of view. Throughout this chapter I will be making reference to the desktop side of the project however my involvement in that part of the project may be limited for the time being.

The first working prototype will be due at the end of June. By then I hope to have the mobile application accessing the database servers and preforming simple CRUD operations for hazards, a basic form for adding hazards, and utilisation of the ArcGIS mapping technology which will show localised hazards and have some functionality for reporting of hazards through a geospatial navigation interface.

As this is only the first prototype and we have a limited time frame before the first iteration of user testing commences, several “luxuries” will be left out depending on time. As mentioned earlier we will not be bothering applying any functionality for offline testing at this stage and I will most likely not be spending a lot of time on the user interface and other things like fancy transitions between screens. As such I will probably be ignoring the mobile web app frameworks, stylesheet languages, and will only be using JavaScript frameworks where absolutely necessary.

It should be noted that due to a differential between the submission time for this report and the time for our first application testing phase, several of the above features may not be completed by the time this report is completed. With this in mind the final area in this report will summarise what will be completed in the coming days (leading towards the first deployment).

4.1 Accessing Database using PHP

While I have had some experience with HTML, JavaScript and CSS, this is my first exposure to PHP programming. Fortunately I have also had some experience with SQL databases and the functions of such databases. Because of this PHP for database management felt very familiar, and no real difficulties were encountered when trying out the basic CRUD operations. One minor setback I did encounter was when trying to access the database tables. At first I did not realise that when using PostgreSQL queries in PHP, all table names in single quotations were automatically converted to lower case. This essentially meant my PHP code was not recognising any of the tables in my queries (as they were defined in the database with a capital first letter, for example ‘Hazards’). After some minor confusion I contacted one of the PHP experts at Opus who informed me of the problem, and a few solutions. One being changing the table names in the database to be all lower case, and the other to slightly alter my queries to include an extra set of quotations around the table name so that it keeps its case.

I have included below the two most important aspects of using PHP to access the database; creating a connection to the database, and defining a query (in this case a simply lookup). [19]

```
$db = pg_connect("host=localhost port=5432 dbname=postgres
user=postgres password=password")
```

Figure 4 - Connecting to the PostgreSQL database using PHP

If you notice in the above code I have set the host to localhost, I have done this as while we are yet to deploy our first version, I have been testing my code using a localised copied version of the database. To access the database through the localhost on my computer I have used the free database connection program XAMPP [20] which opens specific connection modules such as Apache or MySQL. When we release the first version of our applications to some Opus testing staff I will instead be connecting to a server database created by the company.

```
$query = 'SELECT * FROM hazards;
$result = pg_query($query)
```

Figure 5 - A simple SELECT PHP query

This PHP query is the most simple of queries, all it does is get all of the entries in the ‘hazards’ table and stores them in a result. For testing purposes I continued this followed this code with a few simple echo statements to view the data.

For purposes of error handling and safety I have also included catches to handle and display any errors that may occur while either connecting to the database or querying the database. Again this is very simple to do but it holds massive benefits in terms of resolving errors in the early development stages. This is done within a single line that follows either an attempted connection or query (as seen below):

```
or die("Error: ".pg_last_error());
```

Figure 6 - Basic PHP error handling

4.2 Creating the Mobile Side Using HTML5

The second piece of the mobile prototype is the actual HTML5 application, it will essentially be the interface between the user, the map, and the database. Due to its complex nature I have separated this task into 4 smaller subtasks. These are; the ArcGIS map, the HTML5 form, adding hazards to the database via the HTML5 form and a PHP connection, and retrieving and displaying local hazards from the database onto the map.

ArcGIS Mapping

While I have had some experience using enterprise mapping API’s (in particular Google Maps), this is my first experience using Esri’s ArcGIS mapping technologies. As such I have had to put in quite a lot of time towards learning the API as there are a lot of different functionalities that I will have to be utilising to effectively solve the problems. To start, I simply tried to get a map displayed within my testing browser. This was done purely using JavaScript and some of Esri's libraries [15]. The code snippet I have included below is the most basic of maps which is located above Auckland city.

```
require(["esri/map", "dojo/domReady!"], function(Map) {
  var map = new Map("map", {
    center: [-174.7400, 36.8406],
    zoom: 8,
    basemap: "topo"
  });
});
```

Figure 7 - Basic ArcGIS JavaScript code

The most important aspect to using ArcGIS through JavaScript is the first line of the above code. The '`require`' keyword defines what libraries are to be downloaded from Esri as well as the functions that will operate on the given map.

The first functionality that I have decided to add is the ability to zoom to the current device's location (using Geolocation to get a latitude and longitude value). The reason I have prioritised this functionality so highly is because even for this first prototype, the users will be wanting to log hazards using their mobile devices while on work-sites. By allowing them to automatically zoom the map to their current location by the click of a button, Opus employees will be able to quickly locate the area where they wish to report a hazard and can easily view all hazards in the local area (to be completed later).

In terms of the map style I want to have something that is clean and not overly complicated, but also shows enough detail of streets, building groups, and environmental areas to not only be visually appealing, but also practically useful.

Below are a few of the alternative map styles I have considered. While there are many more map styles available (both created by Esri and created open source), these were the four that I found to best fit the design requirements.

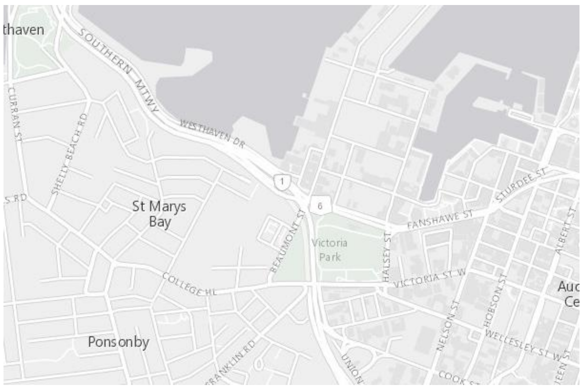

Figure 8 - Gray view

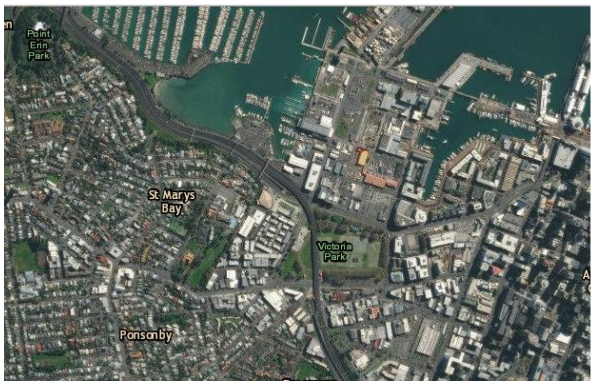

Figure 9 - Hybrid view

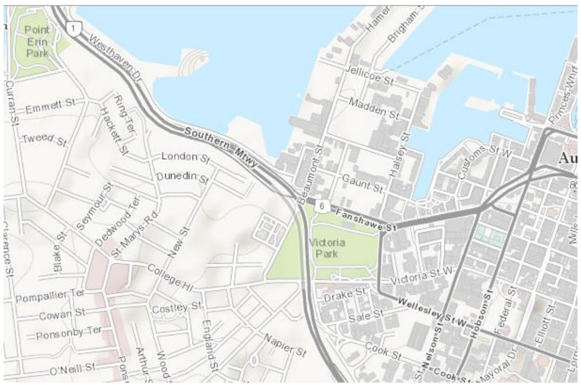
Figure 10 - Topo view

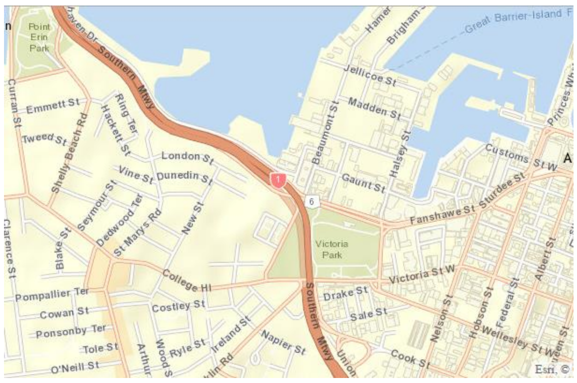
Figure 11 - Streets view

My decision was narrowed down to either the ‘streets’ view or the ‘topo’ view. Both ultimately function very similar but have slight differences that I have used in my decision making process. The ‘streets’ view offers more detail of roads; it has several different shade to categorise the different types of roads (for example the darkest road in the above image is State Highway 1). Whereas the ‘topo’ view does not focus so heavily on the roads (although it does show them in adequate detail), instead it offers better visualisations of building areas. In the above image it is clear what area is more residential (the left half) and what area has a higher density of larger commercial buildings (the right half). While both the ‘streets’ view and the ‘topo’ view have only minor differences, I have decided to go with the ‘topo’ view as I believe it gives enough detail of roads, it is clean and easy to look at, and the building density shading system will be massively beneficial to users when locating particular work sites.

HTML5 Form Creation

Crucial to reporting hazards is the form detailing the specifics of a given hazard. I will be reducing the possible inputs to only those I consider a necessity. First and foremost is a hazard categorisation input, which will give the option to either chose a pre-set hazard type or input a custom hazard site. This will be followed by the location (latitude and longitude) of the current hazard, which will either take the values from the area selected on the map or another value directly input by the user. The third most important aspect of the hazard reporting form is a short description detailing the specifics of the hazard. This will just be a simple editable text box.

Some optional inputs which may or may not be implemented in the first prototype based on time are the ability to attach a photo or video, the ability to link a hazard to an existing project site in the database, the ability to add any additional notes such as mitigation techniques used, as well as input verification.

Development in the Coming Weeks

At this stage I am still well on track to complete the mobile side of the application by the intended date (end of July). I will now be continuing the form creation, after which I will be focussing on the ArcGIS map to allow users to select a specific point or area, so that the location may be recorded and used in the hazard form. Once this is finished I will then move on to joining my PHP code with my HTML5 so that when a user successfully fills in a hazard form, the details may be recorded and saved in our database. After this I will be working on the last most important area of the first release, which is displaying local hazards (and potentially projects) on the map based on the user’s current location.

If I manage to finish all of the above with time to spare I will most probably move onto areas such as allowing capturing and attaching of photos/videos onto hazards, improving the user interface and overall flow of the application, and incorporating some basic form of logging hazard updates to a history list.

Overall Progress

Looking at the greater schemes of things, I believe we will have no problem completing this project to entirety by the end of the year. In terms of big ideas for the mobile side we will need to implement offline storage of data, some form of user login, and a hazard alert system. In terms of the desktop side we will need to incorporate more management systems (for example for issuing alerts to employees), some form of statistical analysis to hopefully help Opus identify and minimise common hazards and alter training for mitigation and resolution techniques. Also we will most likely be adding functionalities to view and edit a particular hazard based on the level of accessibility (either the creator of the hazard, or a management admin).

Concluding Thoughts

So far I have thoroughly enjoyed this project and I am very much looking forward to completing the first prototype and seeing my work in a real-world environment. The experience so far has been great and I have been trying my best to learn as much as I can (not only for the project itself, but also experience of working in a real software development team, albeit rather small). Due to the complex nature of the entire project I’m sure that as our development progresses we will run into obstacles or road blocks of some kind, but I intend in making the most of any difficulties that arise, as I believe that is the time we learn most.

Appendix

Item 1 - Simple Hazard Reporting Template

Area/Locality of hazard	Date

Name...

(Name of person preparing report)

DESCRIPTION OF HAZARD (Include area and task involved, any equipment, tools, people involved. Use sketches if necessary.)

POSSIBLE REMEDIES (List any suggestions you may have for reducing or eliminating the problem, e.g. re-design mechanical devices, procedures, training, maintenance work, etc.)

To be submitted to the Manager

Signed…………………………………….

ACTION TAKEN

Date…………………………………….

Manager…………………………………….

CONTROL IMPLEMENTED & EVALUTATED

Date…………………………………….

Manager…………………………………….

Bibliography

[1] Opus International Consultants Limited, “HazApp: The Opus Geospatial Hazard Management System,” Auckland, 2015.

[2] Adobe, “PhoneGap,” 2015. [Online]. Available: http://phonegap.com/. [Accessed 2 May 2015].

[3] Wikipedia, “SQL,” May 2015. [Online]. Available: http://en.wikipedia.org/wiki/SQL. [Accessed 10 May 2015].

[4] Wikipedia, “NoSQL,” May 2015. [Online]. Available: http://en.wikipedia.org/wiki/NoSQL. [Accessed 10th May 2015].

[5] Knockout, “Knockout JS,” May 2015. [Online]. Available: http://knockoutjs.com/. [Accessed 10 May 2015].

[6] Google, “AngularJS,” May 2015. [Online]. Available: https://angularjs.org/. [Accessed 10 May 2015].

[7] Backbone, “Backbone JS,” April 2015. [Online]. Available: http://backbonejs.org/. [Accessed 14 May 2015].

[8] B. LeRoux, “Lawnchair simple json storage,” 10 March 2015. [Online]. Available: http://brian.io/lawnchair/. [Accessed 14 May 2015].

[9] Mozilla, “Local Forage,” April 2015. [Online]. Available: https://mozilla.github.io/localForage/. [Accessed 14 May 2015].

[10] The JQuery Foundation, “JQuery Mobile,” 2015. [Online]. Available: https://jquerymobile.com/. [Accessed 20 May 2015].

[11] Drifty, “Ionic,” 2015. [Online]. Available: http://ionicframework.com/. [Accessed 22 May 2015].

[12] Bootstrap, “Bootstrap,” 2015. [Online]. Available: http://getbootstrap.com/. [Accessed 24 May 2015].

[13] Less, “Less - Getting started,” 2015. [Online]. Available: http://lesscss.org/. [Accessed 26 May 2015].

[14] W. N. E. C. Catlin H., “Sass - CSS with superpowers,” 2015. [Online]. Available: http://sass-lang.com/. [Accessed 28 May 2015].

[15] Esri, “ArcGIS,” May 2015. [Online]. Available: http://www.arcgis.com/. [Accessed 28 May 2015].

[16] The PostgreSQL Global Development Group, “PostgreSQL,” 2015. [Online]. Available: http://www.postgresql.org/. [Accessed 20 May 2015].

[17] JGraph Limited, “draw.io,” 2015. [Online]. Available: https://www.draw.io/. [Accessed 24 May 2015].

[18] pgAdmin, “pgAdmin - PostgreSQL Tools,” 12 December 2014. [Online]. Available: http://www.pgadmin.org/. [Accessed 28 May 2015].

[19] The PHP Group, “PHP,” 14 May 2015. [Online]. Available: http://php.net/. [Accessed 1 June 2015].

[20] Apache Friends, “XAMPP Apache + MySQL + PHP + Perl,” 2015. [Online]. Available: https://www.apachefriends.org/index.html. [Accessed 2 June 2015].