The University of Sheffield

Faculty of Engineering

Department of Computer Science



A Time Traveller’s Map

Oliver Rahman

Supervised by Dr Siobhan North

COM3600

May 2018

Signed Declaration

All sentences or passages quoted in this report from other people’s work have been speciﬁcally acknowledged by clear cross-referencing to author, work and page(s). Any illustrations which are not the work of the author of this report have been used with the explicit permission of the originator and are speciﬁcally acknowledged. I understand that failure to do this amount to plagiarism and will be considered grounds for failure in this project and the degree examination as a whole.

Name: Oliver Rahman

Signature:

Date:

**Abstract**

**100 Words**

Contents

[Chapter 1 Introduction](#_Toc5903258)

[1.1 Project Description 6](#_Toc5903259)

[1.2 General Analysis 6](#_Toc5903260)

[1.3 Project Risks 6](#_Toc5903261)

[1.4 Report Structure 6](#_Toc5903262)

[Chapter 2 Review](#_Toc5903263)

[2.1 An Introduction to GIS 8](#_Toc5903264)

[2.2 Existing Mapping Systems 9](#_Toc5903265)

[2.1.1 Map Storage and Search 9](#_Toc5903266)

[2.1.2 Mapping Functions 10](#_Toc5903267)

[Chapter 3 Requirements](#_Toc5903268)

[3.1 System Overview 12](#_Toc5903269)

[3.2 Requirements 13](#_Toc5903270)

[3.2.1 Requirements Overview 13](#_Toc5903271)

[3.3 System Architecture Decisions 14](#_Toc5903272)

[3.3.1 Platform 14](#_Toc5903273)

[3.3.2 Programming Languages and Tools 15](#_Toc5903274)

[3.3.3 Mapping Software 17](#_Toc5903275)

[3.3.4 Database 17](#_Toc5903276)

[3.3.5 DBMS and Mapping Data 18](#_Toc5903277)

[3.4 Anticipated Development Problems 19](#_Toc5903278)

[3.5 Ethical Considerations and Pre-Implementation Changes 19](#_Toc5903279)

[3.6 Conclusion 20](#_Toc5903280)

[Chapter 4 Design](#_Toc5903281)

[4.1 User Interface Design 22](#_Toc5903282)

[4.1.1 General Layout 22](#_Toc5903283)

[4.1.2 World Events 23](#_Toc5903284)

[4.1.3 Timeline 23](#_Toc5903285)

[4.1.5 Map Annotation 24](#_Toc5903286)

[4.2 Database Design 25](#_Toc5903287)

[Chapter 5 Implementation](#_Toc5903288)

[5.1 Software Lifecycle and Development Paradigm 26](#_Toc5903289)

[5.2 Data Storage 26](#_Toc5903290)

[5.3 Development Tools and Project Structure 27](#_Toc5903291)

[5.4 Validation 28](#_Toc5903292)

[5.4.1 Validation 28](#_Toc5903293)

[5.5 Server Security 29](#_Toc5903294)

[5.6 Timeline Development 30](#_Toc5903295)

[5.6.1 Timeline Granularity 30](#_Toc5903296)

[5.6.2 Handling BC Dates 31](#_Toc5903297)

[5.6.3 Global Timeline 32](#_Toc5903298)

[5.7 Map Development 32](#_Toc5903299)

[5.8 User Tools Development 32](#_Toc5903300)

[Chapter 6 Testing](#_Toc5903301)

[Chapter 7 Evaluation](#_Toc5903302)

[Chapter 8 Appendix](#_Toc5903303)

[9.1 Specific Requirements 37](#_Toc5903304)

[Chapter 9 References](#_Toc5903305)

# Chapter 1

Introduction and Overview

## 1.1 Project Description

This project – A Time Traveller’s Map – is an endeavour to build a piece of useable, expansive and adaptive mapping software for a wide range of users, but primarily geared for educational use. The final product will display detailed map data, in chronological order, and designed in a way as to be easily navigable and customisable. Users should be able to share their historical data from any time period, at any level of granularity and with as much additional information as they wish to display, all while being pleasant to looks at and easy to use. Users will be able to share their maps with other users, who should be able to easily access these newly created timelines.

An example of the intended use of this software would be to link a series of maps showing the expansion and contraction of the Roman nation between the years 510BC and 1453AD. Users should be able to view individual turning points – The Punitive War, The Final War of the Roman Republic, etc - and should be able to ‘zoom in’ on individual events, seeing battle maps, troops deployments and other related information. This transition between time periods should be facilitated with a slider bar as well as any other suitable methods deemed contributory to ease of use during the Literature Survey.

The first two chapters of this report will analyse the existing marketplace for similar technology, examine and compile the best aspects of these systems, before breaking the potential application into its component requirements in preparation for design.

## 1.2 General Analysis

- To be done on completion –

## 1.3 Project Risks

In general, the authors inexperience working in with any type of GIS or mapping software will need to be considered throughout this project. An expectation of underestimations for work required to complete the project should pre-emptively counteracted where possible and enough time will need to be allocated to learn new libraries, techniques and standards. As a continuation, this field of software engineering is relatively unexplored by most developers, and expectation of lightly documented code and minimal online support can be assumed.

As well as this inexperience, the constant, looming threat of time pressures will need to be carefully managed. Time keeping will be critical to the success of this project and some way of measuring effective working will need to be implemented.

On a more practical note, bugs and errors can be expected for the full course of this project. Regular testing, refactoring and defragmentation of the project structure will need to take place to ensure efficient working continues throughout.

## 1.4 Report Structure

**Chapter 2** will be an analysis of existing software solutions to problems similar to that proposed here. Effective findings will be discussed and considered for adaption into this project.

**Chapter 3** will discuss implicit requirements and priorities for the project. A discussion about tools and libraries to be used in development will also take place.

**Chapter 4** Designs for the project will be collected here, database schemas will be drawn up in preparation for development.

**Chapter 5** will discuss the implementation and how it took place. This chapter will also discuss any unexpected problems encountered in the development process.

**Chapter 6** will be where testing strategy and results are to be collated. Requirements will also be reflected upon and future additions to the project will considered here.

**Chapter 7** will contain the post-project discussions and a general analysis of the project’s successes.

# Chapter 2

Literature Survey

This project will largely be a software development undertaking and so a significant part of the literature review will be examining existing software solutions. Investigating current applications will mostly involve compiling a list of features that work well, features that could be improved, and features that do not work well. As this analysis is completed, special attention will need to be paid towards finding a niche in functionality that the potential system can work within or a better way to gather and display information that results in a more user-friendly system.

Note that for ease of reading, the Literature Survey has been divided into two chapters, Chapter Two will primarily be an analysis of existing mapping solutions and approaches in order to get a wide understanding of current standards. Chapter Three will discuss appropriate software, development techniques and will provide explicit requirement definitions of the project.

## 2.1 An Introduction to GIS

The Esri institute defines GIS as “A framework for gathering, managing, and analysing data, rooted in the science of geography.” (Esri, 2017) GIS arose from the need to convert paper maps into a digital form for easy manipulation and access. “The sheer volume of information meant that areas that are large with respect to the map scale could only be represented by a number of map sheets” (Burrough, 1998) If someone wanted to display their data in full, it would require the creation of multiple copies of the same map with different data sets on each, often with accompanying graphs. This approach is very time consuming and exorbitantly expensive, meaning it was close to impossible for independent groups to display graphical data on maps. GIS arose as an attempt to smooth out these difficulties and to allow easy data display at approachable cost.

GIS is centred around the use of…

## 2.2 Existing Mapping Systems

### 2.1.1 Map Storage and Search

Old Maps Online allows the user to search and navigate through a collection of historical maps. (Klokan Technologies GmbH 2013) The user inputs a specific location of interest by either typing into a text box or by pressing on the appropriate area on a blank world map. The website then displays maps relevant to that location. The returned maps do not appear to be in any specific order and original production date information is often absent. This comes across as a big problem for the system and it makes the website difficult to use as you are unsure of what will be returned. The final system will need to return maps in some sensible order to avoid this problem. Although the site has problems, the act of pressing on the map to select a region is a very interesting UI choice and greatly improves the user experience. The idea would likely be straightforward to implement in the project solution, allowing a user to find a map series in a specific location. Rather than navigating through a menu.

Old Maps Online uses a very heavily modified version of Open Street Map with Open Layers.

Old-Maps is a service like Old Maps Online but with more of an emphasis on selling historical maps. Unlike Old Maps Online, Old-Maps displays maps in a chronological series in a navigation menu on one side of the screen that makes maps very easy to navigate through in order. This design choice is really very nice – having a navigation side menu makes the entire process very easy to understand. Navigating around and zooming into an individual map in not an easy task and the website often takes some time to transition between zoom levels. The website doesn’t make it clear when it’s loading and at first it appeared as though the website had crashed – something to avoid. As a whole, this results in a displeasing user experience. The website has taken a series of questionable design choices that will need to be avoided, but an approach like the side navigation menu could well be something to investigate implementing. Unlike Old Maps Online, Old-Maps appears to use bespoke software built on top of ordinance survey data.

The National Library of Scotland also maintains a collection of historical maps. In a similar way to Old-Maps, the NLS has a clickable map that displays a series of relevant maps in a side view box. The NLS approach is different however, using pre-defined highlighted boxes. This approach is not as good as Old-Maps from an aesthetically pleasing point of view and results in a cluttered display, but it may – arguably – make the website a little easier to navigate through. The actual map display is very nice in comparison to the earlier software, selected maps are displayed in a zoomable sub page that allows for easy navigation. Moving between maps is a problem, the only way to navigate back or forward through maps is to move back to the original map and search from the start again. This is a slow process that could easily be improved upon by importing the side box into the map view page. The NLS, like Old Maps Online also uses a version of Open Street Map.

MyHistoricalMap is another website that tries to fill the demand for viewing historic maps, though this one is probably the worst from several standpoints. The website is clunky, difficult to navigate and the map display is not at all clear. The website serves more as an example of what not to do. After some additional investigation, it appears that the service lost development support towards the end of 2018, again highlighting the fact that only aesthetically pleasing and easy to use services are successful. Like Old-Maps, MHM uses data from the ordnance survey

As is clear to see, all the discussed websites are run using at least a large part of their own bespoke software for displaying maps, even if it is just heavily adapted existing software.

### 2.1.2 Mapping Functions

Moving away from map storage and search,

Liveuamap (Live Universal Awareness Map co 2014) is a web-based live map for tracking conflicts in areas of unrest. The display has a series of coloured polyhedrons differentiating who controls which area of land. The system uses ‘buttons’ overlaid on the map which expand when pressed on to allow the user to very quickly and easily examine a detailed description of the event that took place there without making the display overly complicated. The system is heavily geared to the ‘here and now’ and viewing events from previous days is not a straightforward task, requiring navigation of a series of menus to accomplish. This approach is vastly different from what has been examined previously and it is clear that the creators prioritised usability and aesthetic appeal over anything else. The system relies on a small number of administrators to update the display and users are unable to make their own changes or add their own maps. Further, by making the maps ‘live’ and constantly searching for updates, the website draws an unusually large amount of bandwidth and processor power. The website was nigh on unusable on mobile. This is something to avoid in the final design. The user interface is very pleasing to look at and the interactivity is extremely well done. The use of button overlays is very effective at minimising the amount of information on screen at any one time. Similar to Old Maps Online and the NLS, Liveuamap uses a heavily modified version of Open Street Map. It appears that Open Street Map is something of an industry standard in this field.

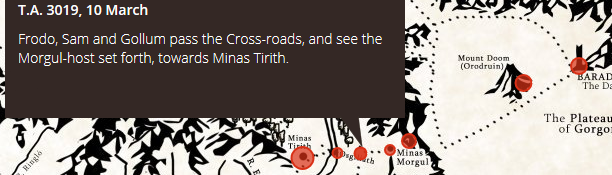
Google’s “My Maps” service (Google 2007 / 2014) allows users to create their own maps by placing layers on top of a blank world map. The user can create these layers by selecting a series of points on the map, saving them to a database and then filling the area within those points with colour. This appears to work in a similar way to the JavaScript SVG Polygon function and seems to be comparable to how Liveuamaps handles its layers. This technique appears to be a standard in many online mapping tools and for good reason, it would be relatively simple to implement such a function. My Maps also allows you to place markers on the map, allowing the developer to hide details until the user requests them and keeping the UI clear – again, like how Liveuamap works. My Maps has no way of connecting a series of maps and so is limited in that capacity.

Looking once more at Liveuamap, the use of map markers on this site is something else to consider in a more focused way. Liveuamap presents its event descriptions in an innovate way, compared to the other examples that have been examined so far.



Each actor taking part in a conflict is assigned a colour and each event is assigned an image from a small collection. This combination of a unique colour and a descriptive image allows a user to very quickly analyse the whole map in significant detail. If the user wishes for a more detailed breakdown, clicking on one of the markers causes additional information to be displayed in a text box and a second box appears on one side of the display offering still further information and links to sources. This minimalist approach is extremely effective at minimising screen clutter and allows only the most important information to be seen at load. Of all the software that has been examined so far, this approach is by far the most effective and will certainly be something to consider in the final design.

Continuing this, the Lord of the Rings Project (Emil Johansson, 2012) offers a similar approach to displaying map markers.



Here, map markers are larger if multiple events take place there. When clicked on, markers expand with an information box giving a breakdown of the details associated with it. This approach is far less effective than liveuamaps approach as there is no fast way to analyse the map, each marker must be clicked on individually to understand the story. This website did offer one useful insight – it uses open layers. The Lotr Project is in many ways similar to the end goal of this project – navigation around arbitrary maps…

# Chapter 3

**System Requirements and Analysis**

## 3.1 System Overview

After the analysis of current trends and standards in the mapping software world, it is time to define the proposed project.

(For a quick glance at the project requirements, view the explicit list on page 15)

‘A Time Traveller’s Map’ will function primarily as an educational aid for children both at home and at school. The software will need to display maps in an aesthetically pleasing and easy to use way and should allow the user to navigate through a series of maps that have been deemed relevant to a given request i.e. a specific location or an event. This navigation should be achieved through a slider and a date and location input box.

The user should be able to upload their own maps and define key features about those maps. They should also be able to connect a series of maps so that they can search look them in order. This map and series should be shareable between users and should be accessible to the creator on multiple devices.

Creators should be able to place markers at important locations by pressing on appropriate locations on the map. They should be able to define information about these markers. The markers should be expandable on click to show this information. The markers should be removable and moveable.

Users should be able to define regions by selecting a series of map points and then give information about these regions. Pressing on a region should display the additional information about it.

Both markers and regions should be minimizable in order to keep the display as clutter free as possible.

Users should be able to upload world events that contain a series of maps with attached markers that make up a ‘story’. These maps should be sharable and should appear automatically in a side scroll box when viewing maps relevant to the events.

Consideration should be paid to user accounts and groups during the design phase in order to maximise the ease of sharing maps and series.

## 3.2 Requirements

### 3.2.1 Requirements Overview

The following section outlines the seven key tenets that will permeate the entire project. These requirements are what should always be referred to in order to ensure the system is staying true to the original aims.

**Ease of Use**

Considering the main target audience of this application is children, ease of use is perhaps the most important concern. Options, instructions, actions and notifications should be easy to understand and intuitive to use. Tooltips and text boxes should be in simple English. Users are not expected to have any expert knowledge to use the service.

**Aesthetically Pleasing**

One of the key points taken from the literature review was that the most successful mapping software was pleasing to look at. Further research into this topic showed that “users can judge a web site's credibility in as little as 3.42 seconds merely on the basis of its aesthetic appeal”[[1]](#endnote-1) (Farah Alsudani 2009) An inescapable fact is that good applications look good and a significant amount of time will need to be spent ensuring that the project is aesthetically pleasing.

**Performance**

As the literature survey has shown, mapping software can be very processor intensive. The system should be geared towards minimising computational complexity and memory use where possible. This is further reinforced by the desire to keep the system as accessible as possible. Important operations should be fast to execute and, in cases where some wait time is expected, a notification should appear alerting the user that the program is still running. Tests dedicated to maximising performance will need to be run during the Testing and Evaluation phase.

**Stability**

The system is expected to be usable at any time. It will need to protect against crash causing errors and invalid inputs. The system should be usable on launch to avoid unnecessary database downtime.

**Platform Independence**

Ideally, the system should be able to run on any machine running any operating system or browser with no noticeable difference between the distribution. This may prove to be unfeasible, but an effort will need to be spent to make the system as cross-platform compatible as is possible. At a minimum, the system should run on Linux, MacOS and Windows operating systems as well as Chrome, Firefox and Edge browsers.

**System Security**

Although the project is highly unlikely to be the target of an attack, it is good practice to be prepared. The system should be resistant to common types of attacks. In 2013, “the average prevalence rate of injectable URL of IPv4 was 5.55%”[[2]](#endnote-2) (Ying - Chiang) so SQL injections should be defended against with the use of sterilised user input boxes and explicit parameterisation. Cross Site Scripting attacks should be resisted where possible. Explicit attention should be paid towards the file upload system. Effort should be spent ensuring that only ‘safe’ file types are uploadable.

**Data Protection**

In order to comply with the European GDPR guidelines (General Data Protection Regulation 2016/2018), the system will need to ensure any data taken from a user is stored safely and in a format that limits vulnerability. Stored user data will need to be minimised, only essential information should be taken to identify the user.

## 3.3 System Architecture Decisions

### 3.3.1 Platform

Throughout this project, the focus has repeatedly returned to the initial decision of wanting to keep the software as easy to access and interact with as possible, and many of the choices made through the course have been with this in mind. To this end, the decision of which platform to base my system on was a relatively straightforward one.

Desktop systems have one major advantage over any other type of platform – no data storage cost the author. This was a significant consideration when deciding on a platform but was eventually discounted for several key reasons.

The first and most important is that each new user would need to download a separate application to their own machine. This might not be a problem if this project was just aimed towards personal use, but that is not the case for this system. Maps are meant to be shared between users and for this to be achieved, the system would need to download copies of every map used onto every PC and the user would need to manually conduct this update. This problem will only grow as the database gets larger. All these problems exist even before considering copyright issues associated with storing protected maps on personal devices.

As a continuation of this problem, it would be very difficult to transfer save states between a user’s devices. Making maps on a computer and then transferring that map series to a laptop would require significant time and effort.

From a more technological perspective, desktop applications are harder to develop platform independent software for - requiring careful use of standards. From a long-term standpoint, if the project were to be expanded to require a GPU to display maps, the system would largely be limited to Java development on the JDK and – as COM2002 taught the year group – developing graphical software in Java can be an excruciating task when aesthetic appeal is a key concern.

A desktop application also suffers the problem of difficulty transferring to a mobile platform. A web application, on the other hand, will be relatively simple to convert to mobile and visa-versa allowing for even greater system reach.

Finally, the problem of bug fixing. If the system ships with a large undetected bug, it would be a huge amount of work to fix. It would also require some sort of software updating ability, something that would require significant research and work to build safely and legally. A web-based bug could be discovered, fixed and updated without the user’s knowledge.

With all the listed reasons, the decision of which platform to use in the initial deployment can be narrowed to mobile vs web-based.

Both options allow for far greater accessibility than a desktop-based alternative and it is difficult to set them apart using only this metric. Because of this, reference back to the core tenets is required to distinguish them. As outlined previously, the most important concern is ease of use and in this respect, a web application is far superior. A mobile application doesn’t offer the finger room to navigate or interact in a highly detailed way and would limit the use of complex functions.

A mobile application would also be breaking the third tenet. Processor requirements to view, load and update maps are high and only good phones will be able to handle this. Testing some of the previously discussed mapping software on an iPhone 6s was not an easy task and forced the user to deal with constant jittering and freezes. It would not be acceptable to demand the newest phones to access the application, especially considering the target audience.

Finally, consider common successful mapping systems like Google Maps, web-based mapping software is very much the industry standard for casual map viewing. That’s not to say mapping software does not exist on desktop or mobile, in fact, there is a great deal of desktop-based mapping software, industrial standard GIS for instance, but these are mainly geared towards professional usage due to the very high cost of such systems. To make a successful, free mapping service in the modern era, it must be web-based.

Because of this, the only acceptable platform would be web-based. In the future, attention could be turned towards creating a limited, processer friendly mobile version.

### 3.3.2 Programming Languages and Tools

Considering the system will be a web-based application, the choice of which programming language to use is limited. The standard front-end development languages are HTML, CSS and JavaScript. These languages are very versatile with powerful libraries and compatible software, and so will be used as a front end.

A typical full stack development project needs some way of communicating with a database and managing a server. Basic JavaScript is incapable of this, so an alternative server-side management tool will need to be found. The four big options in this field are PHP, Node.js, Ruby and Python. All four options are generally interchangeable with each other in most domains, but some particularities exist between them.

As discussed in 3.2.1, performance is a major concern throughout this project and mapping software is especially prone to issues in this area. To counter this, efficient use of server time will need to be maintained and server queries will need to be handled quickly.

There are two commonly used ways of handling server requests: -

* Concurrent Threading – The process of assigning multiple master threads to a single client. In the event of a slow query response, a second thread takes over as master and continues as before. Moderately reliant on manually developing switching rules.
* Asynchronous Processing – Very similar to concurrency but allowing the language to handle threading automatically. Typically, asynchronous languages use one ‘event’ thread and N ‘worker’ threads. The worker threads are responsible for I/O tasks and managing callbacks, while the event thread handles everything else. Asynchronous event threads that are given a callback can halt their current task, execute the callback and then return to the original task. This typically results in a reduced number of active threads[[3]](#endnote-3) (Node documentation)

Hypothetically, if “Fast I/O and a limited number of connections”[[4]](#endnote-4) (Mansnun, 2017) can be relied upon, manual multithreading should result in slightly increased query execution efficiency. Unfortunately, this is not something that can be guaranteed and so it is preferable to plan for the worst-case scenario – low server responsiveness and numerous active connections. Asynchronous is heavily favoured in this type of scenario.

Another significant threat to website responsiveness is the occurrence of blocking – blocking is when a thread is forced to wait for a task to be executed before it can carry on to the next task. This is a significant problem in websites and is a common cause of jittering and freezes – as can be seen in some websites in Chapter 2. Asynchronous architecture is more resilient to this than multithreading and should result in a smoother experience, allowing the client use of most site features without noticing a delayed server response or any occurrences of freezing.

Because of the advantages discussed above, asynchronous architecture should exist in the choice of server-side tool. This decision discounts PHP from the running - the language doesn’t fully support asynchronous programming, instead ‘relying on external dependencies like Spatie’[[5]](#endnote-5) to simulate an asynchronous nature. Ruby can also be discounted - the language does offer some limited asynchronous support, but it is still relatively new to the platform, has limited functionality compared to the other options and requires additional dependencies like event machine [[6]](#endnote-6) to function correctly.

This leaves Python and Node.JS as the remaining candidates. Both tools are excellent and well documented but there are some significant differences between them. A big advantage of Node over Python is that Node was built from the ground up to function asynchronously – this is not the case for Python, which requires additional processing to convert to asynchronous. This difference is most clearly visible when considering benchmarks – Node out-performs python by 3-10x in almost all use cases[[7]](#endnote-7) (Debian benchmarks, 2018). As well as this, Node.JS has one other notable advantage over Python – Language Standardisation. Both the front and back end of the project can be developed in JavaScript, significantly simplifying the project structure and allowing both ends to be developed in the same IDE, which should accelerate development time. There is one downside to Node - it is widely regarded as having a steep learning curve. Hopefully as this project advances, the problem should diminish as the author’s experience increases. If this difficulty can be overcome, and some effort can be put in to avoid the fairly common problem of “callback hell”[[8]](#endnote-8), development in Node should be fairly smooth after the initial learning process.

The standard and by far most documented and used Node.js server tool is Express – the project will use Express for this reason.

For the complete full stack experience, a templating engine should be used to minimise the amount of repeated code used. The differences between the various options in limited and most of the candidates work perfectly well with Node and Express. The two most popular templating engines are Embedded Java Script (EJS) and Handlebars. One of the biggest advantages of Handlebars over EJS is that it essentially acts as an extension to the default engine – Mustache, using all of Mustache’s original functions and making for an easy transition for experienced users. Unfortunately, the author has no experience with *any* engine, so this doesn’t help narrow down the options.

One of the few particularities between EJS and Handlebars is the format partials are stored in – “EJS is different in that it expects your templates to be in individual files, and you then pass the filename into EJS.” (Creativebloq, 2015)[[9]](#endnote-9) This is not an enormous advantage, but it should aid in keeping the project structure in a clear and useable state and should accelerate development speed. As well as this, EJS has no overlap with Angular, React or any other common frameworks[[10]](#endnote-10) (Raj, 2015). This is not of any great importance at this present time but may prove beneficial if the site is to expand in the future. For these reasons, EJS will be the templating engine of choice.

Finally, it would be near heresy to discuss website development without discussing jQuery and frameworks to some extent. jQuery is a monolithic, all-encompassing JavaScript library that offers many quick and easy solutions to common scripting problems. As discussed above, performance is a key concern of this site and so the number of dependencies used should generally be kept to a minimum. Additionally, the number of use cases for frameworks in the project is minimal – mostly just ajax server calls and DOM manipulation. It makes limited sense to import and parse a 100-250kb library for such a small number of functions. Frameworks are typically easy to implement at any stage of development, and in the event the website needs to make use of them at some point in the future, a discussion can be had as to which is most appropriate without worry of significant additional implementation time.

In summary, the front end will be handled with HTML, CS and JavaScript. The server will be managed by Express and Node.JS. Templating will be handled by EJS.

### 3.3.3 Mapping Software

**Rewriting this section.**

### 

### 3.3.4 Database

Considering this application is going to be web-based, some thought must be put into what type of data storage to use. The main decision to make is whether to use a relational SQL style database or NoSQL.

Data structure is a key consideration when deciding on a database structure. NoSQL style databases are designed for data with little structure that is prone to regular change, whereas SQL databases are designed for rigid data sets. The data layout in this project is very unlikely to change soon, so the structure will be static.

Performance is also something to consider. According to Kumar (Comparative analysis of NoSQL 2015), NoSQL style databases perform queries approximately 40% faster than their SQL counterparts. Despite this, in the same paper Kumar outlines that this effect is only noticeable when considering thousands of queries at once and the speed difference will not be felt on my application. This may change at some point in the future if the software expands and demands a greater number of queries.

Ease of development is another concern. SQL style databases have a single de facto standard language that is well documented. NoSQL style databases do not have this privilege, instead relying on varying competing standards with limited documentation and tools. Yishan Li (A performance comparison of Databases 2014)[[11]](#endnote-11) and Lewitt agree saying “there is a lack of support tools to help.” Due to the limited time frame available to develop this system, attention needs to be paid to development speed and a well-documented style will result in far faster development.

Experience is a lesser concern, but it is still something to consider. The author has a deal of experience developing applications using SQL style databases and that experience will transfer into rapid development. It would be fair to expect NoSQL based development to be considerably slower, requiring significant time to learn a new system.

Considering that the key advantages of NoSQL style databases, big data scalability, frequent structure changes and performance will have very limited effects on the system, the only real differentiating factors are ease of development and experience. It is likely that development will be far faster using an SQL style database. Because of this, the system will be built with an SQL backend.

### 

### 3.3.5 DBMS and Mapping Data

One thing to consider when choosing a database flavor is whetherthe chosen service can store and use data in an efficient way. The project, as mapping software, will rely heavily on longitude and latitude values and because of this, a database that can store and with libraries to manipulate data in the form [(X, Y)] will be heavily preferred.

This requirement immediately discounts the use of MySQL, as the service offers no support for geospatial data. MySQL is also not appropriate because it only offers library support for calculations based in Euclidian geometry.[[12]](#endnote-12) (Openlife.cc, 2016) This problem could lead to accuracy inconsistencies close to a planet’s poles and between data sources. OpenSQL and SQLite both suffer from a similar problem, being unable to store and use longitude and latitude with accuracy.

Because of this, one of the few remaining popular SQL style databases to consider is PostgreSQL. This style is different in that it offers two very popular libraries for Geospatial storage and manipulation – Earthdistance and PostGIS which, combined offered a huge array of functions. Both libraries are well documented and have significant user bases, allowing for fast bug fixing and development rates.

Before settling on a final database type, there are two final things that need to be considered – format of date data types and of BC dates. These two concepts need to be easily translatable into a JavaScript usable format and visa-versa. According to the PostgreSQL documentation[[13]](#endnote-13), dates are stored in the standard format yyyy-mm-dd. This is preferable, but there is one problem – PostgreSQL stores months 1-indexed where JavaScript is 0-indexed. This is not a huge concern, but it will require additional conversion code to be created for Selection from, and Insertion to the database. Secondly, BC date formats are stored “…with the suffix ‘ BC’” attached to the end of a string being inserted into the database. This will require additional handling code as JavaScript handles negative dates in a -00yyyy-mm-dd format (Discussed in detail in 5.6.2).

As an aside, the ‘year’ 0 is something that should be considered before implementation begins. The ‘year’ 0 does not exist in the Gregorian Calendar, but JavaScript is capable of handling both +0 and -0 as inputs into its new Date() function - this will cause a crash in PostgreSQL. Generally, the new Date function does not handle negative dates very well, and this problem is compounded with 0 inputs Unfortunately, neither of these inputs produce usable outputs: -0 → 2000AD and +0 → 0000. It will likely be a good idea to make error handling code to ensure these inputs are never entered, but caution should be paid in general around 1BC/1AD to ensure the phantom ‘year’ 0 has no effect on the normal functions of the website and the additional year does not create an offsetting factor into calculations around this date range.

Because of the discussion above, the system will rely on PostgreSQL to handle data storage, though additional attention will need to be paid when using negative dates and around the ‘year’ 0.

## 3.4 Anticipated Development Problems

Considering the specificity of this project, tools and libraries have a very limited number of users and code bases that are relatively untested when compared to standard and commonly used libraries. Having conducted some research, it is fairly standard for some mapping libraries to have less than ideal documentation and the implementation may require some significant level of experimentation to get these tools to work correctly. This problem is likely to be compounded by the authors inexperience working in this field. Productivity will need to be measured and a strict time table will need to be adhered to. In the event of a continuous stubborn roadblock, it may become necessary to remove or delay features in order to stick to completion schedules.

Continuing this point, testing may also be a problem. Specialised libraries often aren’t completely compatible with standard testing tools. Care will need to be taken during testing to ensure particularities in the code base are not causing unexpected - or worse - unreportederrors.

For a less software-based consideration, copyright is going to be something to pay attention too. Many maps are owned by the map makers and are not licensed for public dissemination. The website will need to do what it can in order to avoid legal trouble, or at a minimum, pass the legal burden to the user.

In general, effort to minimise the number of bugs in the code makes for standard good coding practice but, as always, some bugs will slip through the cracks. To combat this, regular test processes should be run, followed by a large final test to ensure the system ships without any known bugs. Outlines for this testing can be found in section [**Testing Section**]

## 3.5 Ethical Considerations and Pre-Implementation Changes

Originally, this project was intended as an educational aid for teachers and children. This is still a key focus, but after some advice and consideration, some aspects have had to change. The amount of data that was to be stored on the system raised some important ethical concerns. Storing large collections of data from children brings additional unnecessary risks and is something to be extremely wary of in general. Because of this, the required information has been slimmed to the bare minimum - no delicate personal information will be stored at all. The only details to be requested on sign up will be an email address, username and suitable password. Usernames will be the only public information available to users.

Further, cyberbullying and abuse are things to consider. The number of young people reporting instances of cyberbullying have doubled since 2011[[14]](#endnote-14) (Comparitech – 2018) It is conceivable that users could upload offensive images in place of maps. To combat this, users should only be able to share maps with people whose usernames they know and have whitelisted. This should prevent unsolicited images being shared to a wide audience. Users will also be able to un-whitelist someone with ease.

Additionally, data management is something else to consider. Minimising stored data in the way described above has the added benefit of lowering system scaling costs as there is less data to secure and handle. Further, the process of database setup and management is streamlined significantly and should lead to quicker initial preparation, bug fixing and general database maintenance.

## 3.6 Conclusion

This chapter has largely been dedicated to defining the problem at hand, breaking it down in detail and then looking for tools, methods and systems to help produce a solution.

With the system requirements discussed (Section 9.1 for a specific breakdown), and tools decided upon, attention will be turned towards designing the final product. Chapter 4 will consider each of the major component features and will draft designs for them in a way that adheres to the previously discussed requirements in Section 3.2.1.

# 

# Chapter 4

**System Design**

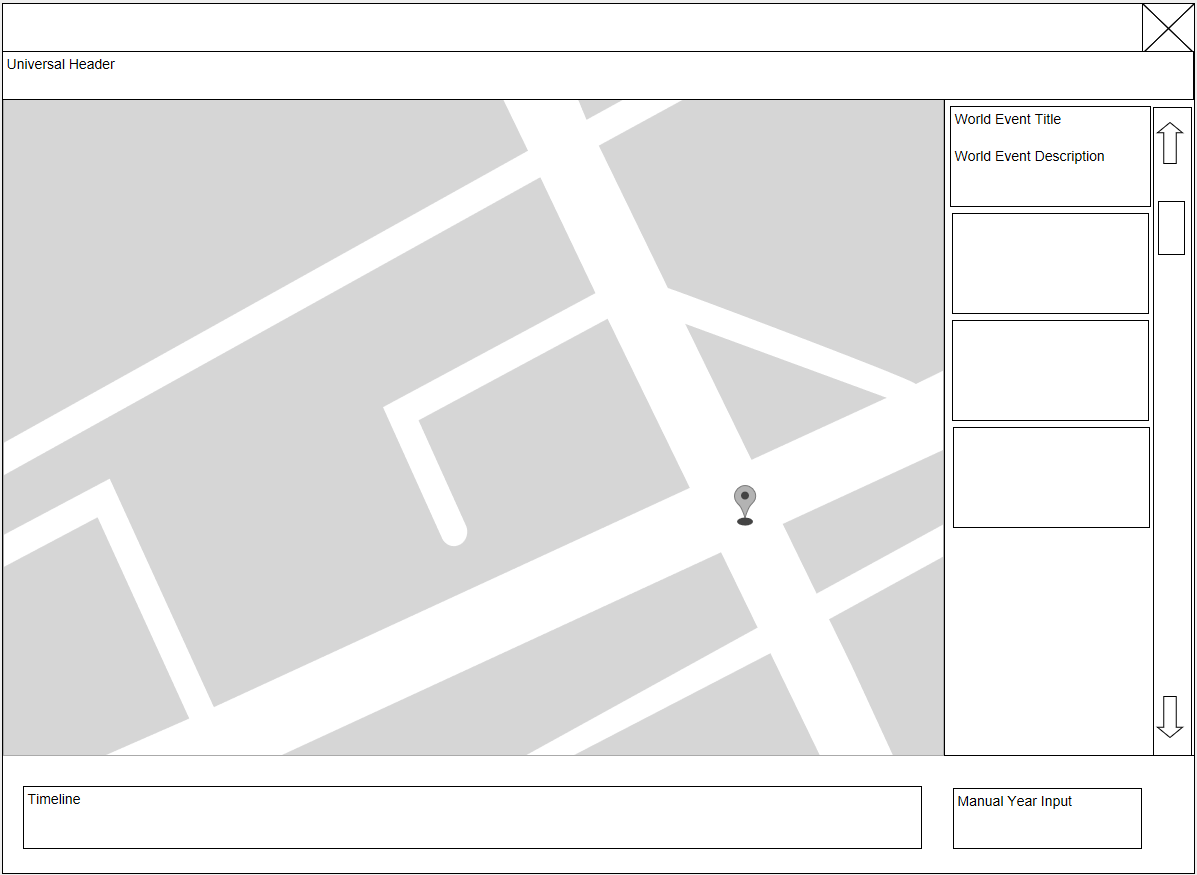
From the start, the plan for this project was to minimize the amount of user interface and to make the display as streamlined and minimalistic as possible. To achieve this, a dynamic approach to design will need to be taken during development as discoveries of ‘what works’ and ‘what doesn’t’ become clear through extended use and testing. To this end, the amount of ‘physical’ designs in this section is also minimal with just a wire frame outline to most of the features and an expectation of dramatic change throughout the course of development.

The basic user interface designs will be discussed first, followed by a breakdown of the database design choices and a discussion of how the design changed before development began.

# 4.1 User Interface Design

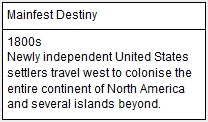
### 4.1.1 General Layout

During the Literature Review, it was clear to see that the best designs were minimalist with only the most important features visible at the start, with more complex and messy features accessible through menus. To that end, the initial design for the general front page of the website is as bare bones as possible. The map takes up most of the screen, like both LOTR-project and LiveuaMap.

****

### 4.1.2 World Events

World Events are user created events that will have a series of maps connected to them. By clicking on them, a different user will have the option of exploring the topic in detail or finding relevant maps in the database. This is an example of a World Event.

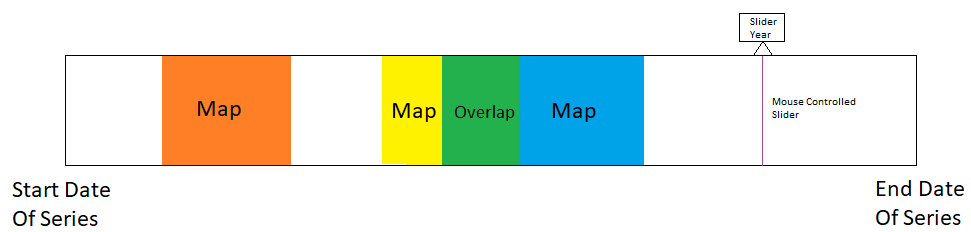


World events will have four key features attached to them, a start date, an end date, a location where the event took place and a type which will be used to decide on which field in the appropriate database table to look through, i.e. ‘Global’, ‘National’, ‘Local’ etc. With this information, relevant, new maps will be retrievable from the for the event created to see. It may be possible to make a number of trusted events public so that everyone can see events that are taking place in other areas of the world.

World events that are close to the current time-period that the user is looking at should appear in the right-hand scroll box. For example, if the user is looking at a series of maps from the British conquest of India, they will also be prompted to view a map series of the Seven Years War on mainland Europe, if such an event has been created before.

### 4.1.3 Timeline

The timeline design will focus on the principles stated above – minimalist but with maximum information transmission. With this in mind,



This design allows users to quickly see relevant areas of the timeline and easily allows them to navigate to those areas. The adaptive start and end date design should have the added effect of making maximal use of time line space – a potential problem with long series that constitute many maps – and minimises the amount of unused white space at both ends.

If users want to go to a specific date, a popup above the slider will make this very easy.

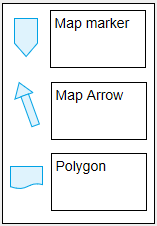
When multiple maps overlap, a special display on the timeline should appear, and when selected, the map will enter a unique state, allowing users to flick between them.

### 4.1.5 Map Annotation

Each layer may have several map events attached to it. A map event is a clickable occurrence on the map that will expand to give additional information.

These three UI map events will be a part of the initial minimum viable product:

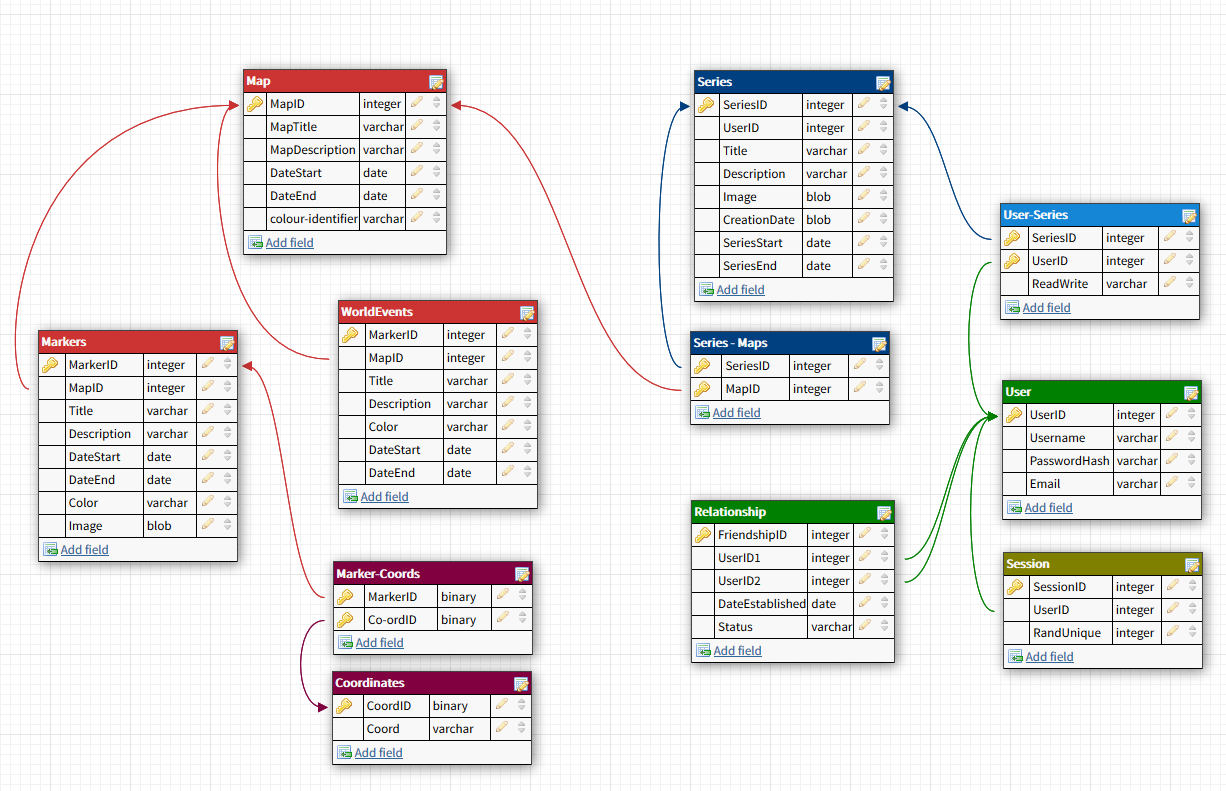
* A *Polygon* – Polygons are shaded, multi point events that show an area.
  + The territory held by a group
  + An area that has been flooded
* An *Arrow* – An Arrow is an occurrence that takes place between two points on the map and represents movement.
  + The movement of an army.
  + The spread direction of floodwaters.
* A *Marker* – A Marker is a single point on the map, a location of interest.
  + The site of a battle.
  + The location of an important meeting.

As discussed earlier, markers will be designated a colour and an icon either from a predefined list or custom made by the user. 

To do this, a drop-down menu like this will be used.

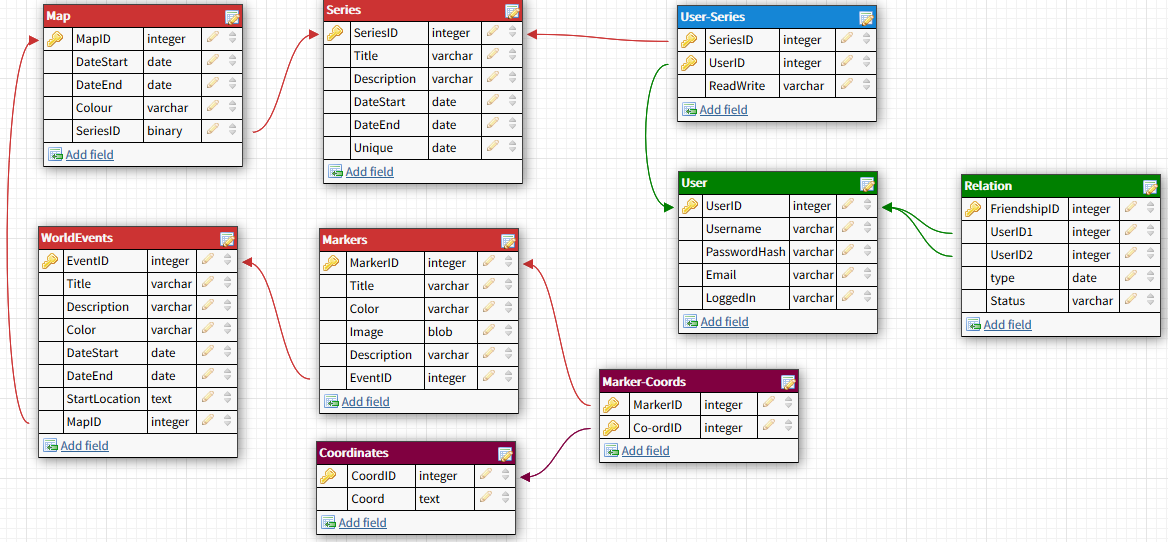
## 4.2 Database Design

The original design for the database was as follows: (Arrows indicate One → Many)

****

However, after some consideration, this design was changed to simplify the development process and to allow for some additional features to be added. These changes resulted in this design: -

(Note - text fields substitute a [longitude, latitude] data type)



This new design simplifies the previous by reducing the number of linker tables by one and removing the need for another full table entirely. The removal of the “Session” table can be justified by placing a new ‘LoggedIn’ field into the user table. The function of this field, and that of the new ‘Unique’ field in “Series” is discussed in detail in Section 5.5.

A further advantage is that the direct link between “Markers” and “Maps” is removed - this can be justified by having Map areas treated in the same way Events are. This is discussed in Section 5.7.

This design decision has one final effect – it simplifies the data access route to a straight line – there is less of a need to store multiple copies of the same variable in multiple tables.

# Chapter 5

**Implementation**

Having decided on the specific tools and libraries that will be used throughout the completion of this project and following the completion of initial user interface designs, it is time to begin development. This Chapter will begin with the creation of core data storage structures, then focusing on discussion of individual feature development and problems encountered.

## 5.1 Software Lifecycle and Development Paradigm

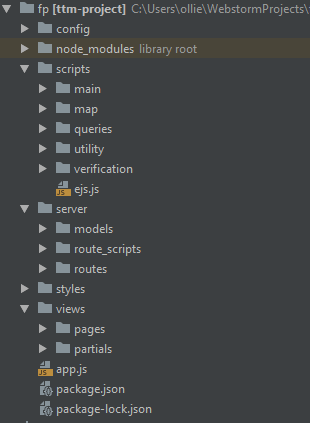
Throughout the course of this stage of development, the author will attempt to follow a highly agile design methodology, focusing on a lean development model and using a Kanban board to aid with progress visualisation. The Agile Manifesto[[15]](#endnote-15) prioritises “rapid production of functional prototypes, frequently changing requirements and continuous delivery of working systems”. These principles are critical in a development situation like this, as time constraints are by far the most significant limiting factor and it is unlikely the entire project will be seen to completion. An unrefined but working product is better than a refined product that does not work.

Throughout this project, tasks completed on the Kanban board will be recorded and tracked on a spreadsheet. This will be used to monitor productivity and to ensure the author stays focused on the task. The effectiveness of this strategy is discussed in Section 7.

## 5.2 Data Storage

## 5.3 Development Tools and Project Structure

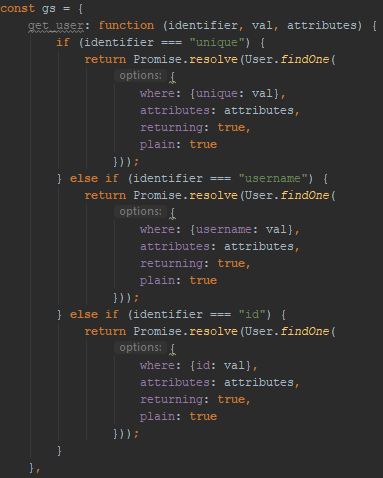
The project will be developed using InteljJ’s WebStorm IDE and the project will follow this file structure: -



……

## 5.4 Server Security

The author chose to use Sequelize to handle SQL query creation and sterilization.

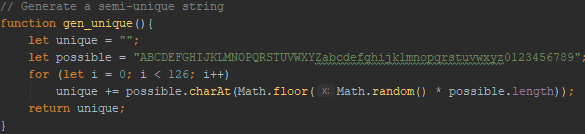
Typically, there are two ways to uniquely identify an entity in the database defined in 4.2 – the entity unique identifier ‘ID’ and a random low-collision string ‘unique’.

“When the user logs in, generate a large (128 to 256 bit) random token. Add that to a database table which maps the token to the user ID, and then send it to the client in a cookie.”[[16]](#endnote-16) (ircmaxwell, 2016)

ID can only be accessed by a single query which uses ‘unique’ as the identifier. This means that no ID value is ever exposed to the client, but the user can still access all the queries that having the ID would normally offer.

Even in the very unlikely event ‘unique’ was compromised, it would only be useful for a short period of time – until the next login, logout or tab close.

This is the function used to generate the low collision string:



User is slightly different to the other tables in the database in that it has 3 unique identifiers – ‘ID’, ‘Unique’ and ‘Username’.

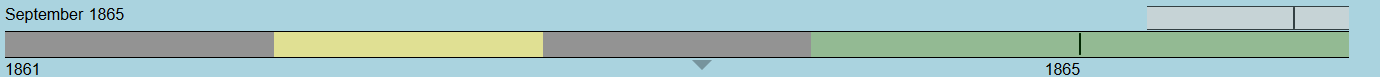
In this instance, unique is used to keep a user logged in between page moves and to ensure that the user has valid access to a requested html route. If not, they are redirected to a page they have access to.

‘Username’ acts in a similar way to ‘Unique’ in that it is only ever used to access the ‘ID’, this simplifies the sharing of maps, as the user doesn’t need to remember the 128-character unique identifier and the ID still doesn’t need to be exposed to the client-side.

## 5.5 Timeline Development

### 5.5.1 Timeline Granularity

The original timeline design was discussed in section 4.1.3.



As is clear to see, some alterations had to be made to make this design compatible with screen limitations. The key limitation that required these alterations was that longer map series couldn’t be accurately represented in the limited number of horizontal pixels available on the screen. In some situations, one pixel was required to represent multiple date values. This is unfortunate, as dividing a year into twelve-month blocks is far more user friendly and simply looks better. For instance, the grey blocks represent years with multiple maps – making the timeline difficult to read in detail.

To compensate for this, the system will use an adaptive design. The site can represent roughly 800 periods in the timeline without data loss, allowing one twelfth of that as includable years – roughly 67. This calculation was made on a 1920-pixel width monitor and so, to account for monitors with lower pixel densities, a series with a length lower than 30 years will use the simplified timeline and series with more than 30 years will use the separated month timeline design shown above. This design choice should allow a significant number of series use of the simplified timeline, shown here.



The same data as above, but information is far clearer in appearance.

An additional benefit of this design is that it allows the coloured map display blocks to also use month by month granularity, greatly increasing usability and providing a more information-dense appearance without compromising on aesthetics or simplicity.

This design works perfectly with both BC dates and around the year range 1BC/1AD.



Further time will need to be spent considering how to include days or even hours into the time line – though this level of granularity is likely going to be limited by time constraints.

The only downside to this design is that it requires one additional input value on the far right of the slider. Validation code had to be created to avoid database query errors.

Unfortunately, due to time constrains and a moderate priority, it was decided that the extended timeline would have to be delayed until after launch so that other, more critical requirements could be focused on. This is a regrettable decision, but one that still allows many timelines to be displayed in the compressed and more aesthetically pleasing display.

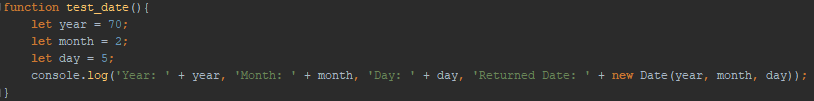
Additionally, Information retrieved in generating this design is reusable - it shows the number of maps in each month segment. This information could be stored in temporary browser memory and repurposed to generate the tool tip pop-out with minimal computation and zero server time cost – just a local O(1) complexity dictionary lookup. The result appears like this.

### 5.5.2 Handling BC Dates

As discussed in 3.3.5**,** PostgreSQL handles dates slightly differently to JavaScript, this was one the biggest concerns when choosing this combination of tools and is a problem that required a significant amount of time to solve completely.

The PostgreSQL documentation [[17]](#endnote-17) lists the appropriate way of handling BC as appending – “BC, bc, AD or ad – Era indicators” to dates. This, in and of itself, is not a problem if converting directly into a date format on the server side. Unfortunately, this is not the case most of the time.

Conversely, JavaScript uses an archaic date format, relying on the ISO date method from the original Java port[[18]](#endnote-18). This results in unexpected results from some inputs.





As can been seen above, there are two problems with the standard “new Date” function, the first is that any year value less than 99 is treated as 1900 + that value, second is that JavaScript treats months differently from PostgreSQL, starting its count from 00 as opposed to PGs 01. PG will reject any date sent to it with month 00 and will cause a site crash.

This problem is compounded in two situations found relatively frequently in this project. BC dates, and when calculations need to be made on dates. BC dates in particular are troublesome when parsing.

To solve this problem, there is a relatively unknown input to the Date constructor that allows easy use of BC dates. [[19]](#endnote-19) (“…a negative date and **two leading zeros before the four-digit representation of the year**.”) Strangely, inputting a negative date in the format -00yyyy-mm-dd into the ISO date constructor produces an easy to use and manipulate date object. This solution works in all major browsers. As much as the author wishes to say it was planned, this method unintentionally worked almost immediately with the code written to handle AD dates, requiring only minimal alterations. Care must still be taken adding 1 to months of dates retrieved from the database and subtracting 1 from those being inserted.

Despite this, there is still one edge case left to consider – the year 0. As discussed in 4.1.3, this was a problem seen coming and accounted for. A small alteration was required to ensure the year 0 wasn’t counted when calculating map block locations and offsets, but otherwise, there were no significant problems. It was decided that a check was a required frequently enough to warrant an additional Boolean session cookie if the year 0 ‘existed’ in a series.

The slider output simply moves from December 1BC to January 1 in this fashion: -





5.5.3 Global Timeline

In addition to the granularity problem discussed earlier, the need for a second global timeline was also identified. Instead of maps, this timeline was to hold series information, allowing users to navigate between the various series they have access to.

There are 3 types of series a user will have access to: -

1. User-made – Series made by the user viewing the timeline
2. Friend-made – Series made by friends of the user that the user has read/write access to.
3. Admin-made – Globally viewable model series made by admins.

The intention was for this bar to display all three of these types, but complications with the timeline – like those in 5.6.1 – made this impossible until the more fundamental problem of displaying fine granularity is solved first. These problems are unlikely to be solved in the limited time before release, so the idea of a global timeline will be put on hold for the near future. To facilitate the user browsing these maps without the timeline, all three types will be displayed in the right-hand bar with the option of hiding any of the groups from a tick box in the user tools menu.

## 5.6 Map Development

**Rewrite this.**

## 5.7 User Tools Development

Early on in development, the decision to use a single page to handle most functions was made. The real meat of this project will be in developing functions in the back end with the UI being more of a sideshow. To facilitate this, all the user’s functions can be accessed in a single popup partial that can be created by accessing the drop-down menu in the index. This decision vastly simplifies the development process and allows the author to focus away from writing HTML and on to more demanding problems. If a need to expand the number of HTML pages appears, it should be straight forward to implement them, requiring only a new route in app.js and the creation of the HTML. This decision also preserves the goals laid out in 3.2.1, and arguably increases usability, as all the functions are present on the front page, just hidden until required.

Logging in is discussed in 5.5 and creating an account is straight forward – The user enters a username, email and password. Email and username are checked against the database and an alert appears if those values already exist.

As discussed in 5.6.3, the user is then given access to all admin made maps in the database. They have the option of disabling these maps appearing in the user panel.

**Put creating series / maps in here.**

If the user has a number of maps or series they are working on or viewing at any time, they are able to right click the event box in the right-hand menu to have a shortcut appear at the top of the main page.

[**ADD IMAGE**]

When a user wishes to add a friend to their network, block a user or to share a map with someone else, the process is straightforward. The user simply enters a username into the input box, selects an action from the drop-down menu and presses accept.

[**ADD IMAGE**]

This action then updates the “Relations” table (4.2) with both user’s names and the type of relation “friend” or “block”. If the user is sending a friend request, status is set to “pending” and a notification is sent to the other user. This appears as a red circle next to their user name in the menu.

[**ADD IMAGE**]

A block is carried out immediately and the blocked user is not informed. They are no longer able to share maps or send friend requests to the blocking user, though it appears to them as if they can.

## 5.8 Conclusion

This chapter has been a discussion about the implementation of the project – what went right, what went wrong and what required some change to make work effectively.

As is somewhat clear to see, the authors relative inexperience in this field of development led to a difficulty underestimation for some of the requirements. This underestimation combined with the inherent time constraint resulted in a slight scale back of several requirements - particularly those cantered around the map display. Most of these problems are not beyond solving and with a little more time, it is likely the author will be able to find solutions for a significant proportion.

Despite this scale back, the author was able to produce many effective functions and fulfilled a significant proportion of the requirements. The specific successes will be discussed in detail in section 7.

The following chapter will discuss the testing process that took place during and after development.

# Chapter 6

**Testing**

**1500 Words**

# Chapter 7

**Evaluation and Conclusion**

**1500 Words**

<http://whc.unesco.org/en/list/85>

https://www.postgresql.org/docs/9.1/datatype-datetime.html

# Chapter 8

**Appendix**

Additional relevant information can be found here, along with code snippets and work that might be useful to students embarking on similar projects in the future.

## 9.1 Specific Requirements

Below is an explicit list of requirements for the system - a checklist for development. Requirements are ranked by priority – ‘Maximum’ is necessary for a viable minimum product.

**M. Map Requirements**

|  |  |  |  |
| --- | --- | --- | --- |
| Identifier | Requirement | Priority | Exp. Difficulty |
| M1 | User can upload permanent maps | Maximum | Low |
| M2 | User can view selected maps | Maximum | Low |
| M3 | User can delete maps | Maximum | Low |
| M4 | User can edit map details | Maximum | Low |
| M6 | User can define a location for their maps | High | Low |
| M7 | User can define other details for their maps | High | Low |
| M8 | User can view other people’s maps | Medium | Medium |
| M9 | Relevant maps should be displayed on start up | Low | Very High |
| M10 | User can zoom the map in and out. | Low | Very High |
| M11 | User can search for locations by clicking on the map | Low | High |
| M12 | User can add maps directly from the internet | Low | Unknown |

**E. Map Events**

|  |  |  |  |
| --- | --- | --- | --- |
| Identifier | Requirement | Priority | Exp. Difficulty |
| E1 | User can add events to a map with the mouse | Maximum | High |
| E2 | User can view events on a map | Maximum | High |
| E3 | User can remove events from a map | Maximum | Medium |
| E4 | User can define a name for an event | Maximum | Low |
| E5 | User can define an event description | High | Low |
| E6 | User can define and start and end point to events | High | Medium |
| E7 | User can view an expanded details box for an event | Medium | Medium |
| E8 | User can view the details box with a mouse click | Medium | Medium |
| E9 | User can toggle event appearance on the map | Low | Low |
| E10 | User can add custom map events | Low | Medium |

**W. World Events**

|  |  |  |  |
| --- | --- | --- | --- |
| Identifier | Requirement | Priority | Exp. Difficulty |
| W1 | User can view a collection of world events | High | Medium |
| W2 | User can create world events | High | Low |
| W3 | User can delete world events | High | Low |
| W4 | User can add start and end points to world events | High | Low |
| W5 | User can edit world event details | High | Low |
| W6 | User can select a world event and get relevant maps | Low | Medium |
| W7 | User can include links to additional information | Low | Medium |

**T. Time Line**

|  |  |  |  |
| --- | --- | --- | --- |
| Identifier | Requirement | Priority | Exp. Difficulty |
| T1 | User can view the time line | Maximum | Low |
| T2 | User can select required year with the timeline | Maximum | Medium |
| T3 | Current year is visible somewhere on screen | Maximum | Low |
| T4 | User can navigate by defined time periods blocks | High | Medium |
| T5 | User can navigate time periods with a text input box | High | Medium |
| T6 | Upcoming and recent world events are visible | High | Medium |
| T7 | Upcoming and recent map transitions are visible | High | High |
| T8 | Upcoming and recent map events are visible | Low | Very high |

**U. User**

|  |  |  |  |
| --- | --- | --- | --- |
| Identifier | Requirement | Priority | Exp. Difficulty |
| U1 | User can create an account | Optional | High |
| U2 | User can sign in and out of their account | Optional | High |
| U3 | User can authorise their account through email | Optional | Medium |
| U4 | User can create a map linked to their account | Optional | Low |
| U5 | User can create a map series | Optional | Low |
| U6 | User can link a map to their account | Optional | Low |
| U7 | User can delete maps from their account | Optional | Low |
| U8 | User can share their maps with other users | Optional | Unknown |
| U9 | User can whitelist other users | Optional | Unknown |

# Chapter 9

**References**

1. https://www.bcs.org/upload/pdf/ewic\_hci09\_paper66.pdf [↑](#endnote-ref-1)
2. https://www.researchgate.net/publication/260396729\_Vulnerability\_Assessment\_of\_IPv6\_Websites\_to\_SQL\_Injection\_and\_Other\_Application\_Level\_Attacks [↑](#endnote-ref-2)
3. https://nodejs.org/es/docs/guides/dont-block-the-event-loop/ [↑](#endnote-ref-3)
4. <http://masnun.rocks/2016/10/06/async-python-the-different-forms-of-concurrency/> [↑](#endnote-ref-4)
5. https://stitcher.io/blog/asynchronous-php [↑](#endnote-ref-5)
6. https://codereview.stackexchange.com/questions/84793/asynchronous-execution [↑](#endnote-ref-6)
7. https://benchmarksgame-team.pages.debian.net/benchmarksgame/faster/node-python3.html [↑](#endnote-ref-7)
8. https://www.altexsoft.com/blog/engineering/the-good-and-the-bad-of-node-js-web-app-development/ [↑](#endnote-ref-8)
9. https://www.creativebloq.com/web-design/templating-engines-9134396 [↑](#endnote-ref-9)
10. https://www.quora.com/Whats-the-best-view-engine-for-node-js-ejs-jade-or-handlebars [↑](#endnote-ref-10)
11. https://www.researchgate.net/profile/Yishan\_Li/publication/261079289\_A\_performance\_comparison\_of\_SQL\_and\_NoSQL\_databases/links/564fbcf708aeafc2aab3ff73.pdf [↑](#endnote-ref-11)
12. https://openlife.cc/blogs/2012/august/opengis-standard [↑](#endnote-ref-12)
13. https://www.postgresql.org/docs/8.2/functions-formatting.html [↑](#endnote-ref-13)
14. https://www.comparitech.com/internet-providers/cyberbullying-statistics/ [↑](#endnote-ref-14)
15. https://agilemanifesto.org/principles.html [↑](#endnote-ref-15)
16. <https://stackoverflow.com/questions/1354999/keep-me-logged-in-the-best-approach> [↑](#endnote-ref-16)
17. https://www.postgresql.org/docs/9.1/functions-formatting.html [↑](#endnote-ref-17)
18. https://www.digitalocean.com/community/tutorials/understanding-date-and-time-in-javascript [↑](#endnote-ref-18)
19. https://scholarslab.lib.virginia.edu/blog/parsing-bc-dates-with-javascript/ [↑](#endnote-ref-19)