

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 amounts 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

Geographic Information systems (GIS) are a rapidly growing field of software engineering with many new and exciting systems entering the market each year. Education has also seen extensive migration towards an online platform, with innovative interactive learning platforms being launching regularly.

This project sought to combine these two domains into a single, usable system for teaching History and Geography to students and curious members of the public with an interest in historical map viewing and creation.

The final system offers extensive tools for collaboratively creating in-depth historical maps in an intuitive and powerful way. Maps can display a range of information, from big-picture outlines of a country's history to microscale battle maps and troop deployments. Users can construct large-scale map series together and then share their creations with others. The website also facilitates substantial user-to-user interaction, designed in a way that prioritises user safety.

Contents

List of Figures	v
Chapter 1 – Introduction and Overview	1
1.1 Project Description	1
1.2 Report Structure	1
Chapter 2 – Literature Survey	2
2.1 Chapter Introduction	2
2.2 An Introduction to GIS	2
2.3 Existing Mapping Systems	3
2.3.1 Map Storage and Search	3
2.3.2 Embedding Information	4
2.4 Chapter Conclusion	6
Chapter 3 – System Requirements and Analysis	7
3.1 Chapter Introduction	7
3.2 System Overview	7
3.3 Requirements	8
3.3.1 Requirements Overview	8
3.4 System Architecture Decisions	9
3.4.1 Platform	9
3.4.2 Programming Languages and Tools	10
3.4.3 Mapping Software	12
3.4.4 Database	12
3.4.5 DBMS and Mapping Data	13
3.5 Ethical Considerations and Pre-Implementation Changes	14
3.6 Chapter Conclusion	14
Chapter 4 – System Design	15
4.1 Chapter Introduction	15
4.2 User Interface Design	15
4.2.1 General Layout	15
4.2.2 Display Boxes	16
4.2.3 Timeline	16
4.2.4 Map Annotation	17
4.3 Database Design	18
4.4 Chapter Conclusion	19
Chapter 5 – Implementation	20
5.1 Chapter Introduction	20
5.2 Software Lifecycle and Development Paradigm	20
5.3 Project Structure	20
5.4 Server Security	21
5.5 Timeline Development	22
5.5.1 Timeline Granularity	22
5.5.2 Handling BC Dates	23
5.5.3 Global Timeline	24
5.6 Map Tools	25
5.6.1 Tool Summary	25
5.6.2 Polygon Functions	26

5.6.3 Creating New Series, Maps and Events	28
5.6.4 Embedded Historical Information	29
5.7 User Tools	29
5.8 Administrator Tools	30
5.9 Chapter Conclusion	30
Chapter 6 – Testing	31
6.1 Chapter Introduction	31
6.2 Generic Testing	31
6.2.1 Compatibility Testing	31
6.2.2 Performance Testing	32
6.2.3 Security Testing	32
6.3 End-to-End / Acceptance Testing	34
6.4 Chapter Conclusion	35
Chapter 7 – Evaluation And Conclusion	36
7.1 Chapter Introduction	36
7.2 Implementation Evaluation	36
7.3 Project Limitations	37
7.4 Future Development	38
7.5 Final Conclusion	38
References	39
Appendix	42

List of Figures

Figure 2.1 Old Maps Online results box.....	3
Figure 2.2 Old-Maps side bar	3
Figure 2.3 National Library of Scotland map overlay boxes	4
Figure 2.4 Liveuamap markers and information boxes.....	5
Figure 2.5 Lord of the Rings project map markers.	5
Figure 4.1 Initial user interface design mock-up.....	15
Figure 4.2 Initial event box design mock-up.	16
Figure 4.3 Timeline design mock-up.....	16
Figure 4.4 Initial database design.....	18
Figure 4.5 Final database design.....	19
Figure 5.1 Extended timeline design	22
Figure 5.2 Simplified timeline design	22
Figure 5.3 Simplified timeline 1BC/1AD	22
Figure 5.4 Timeline transition BC - AD.....	23
Figure 5.5 Map Tools overview	25
Figure 5.6 Polygon marker overview.....	26
Figure 5.7 Naive map drawing algorithm	26
Figure 5.8 Advanced map drawing algorithm.....	27
Figure 5.9 Limited polygon function.....	27
Figure 5.10 Expanded Polygon function	27
Figure 5.11 Create new Series tab.....	28
Figure 5.12 Select Series tab	28
Figure 5.13 Embedded information markers in maps.....	29
Figure 6.1 Opera testing error	31
Figure 6.2 SQL injection example	32
Figure 6.3 A basic URL manipulation attack example	33

Chapter 1

Introduction and Overview

1.1 Project Description

Geographic Information systems (GIS) are a rapidly growing field of software engineering with many new and exciting systems entering the market each year. Education has also seen extensive migration towards an online platform, with applications like MyMaths and Code Combat being moved to the forefront of their respective classrooms. There are still several large gaps in the education domain where no system has yet risen to dominance, with Geography and History being prime examples.

This project – A Time Traveller’s Map – built a piece of useable and expansive mapping software for a wide range of users, but primarily geared towards Geographical and Historical educational use. The final product displays detailed map data, in chronological order, and designed in a way that is easy to navigate and customise. Users can create historical maps from any period, at any level of granularity and with as much additional information as they wish to display. Users can then share these maps with others, allowing for group collaboration in map creation. At present, no free historical map making systems that are geared towards collaborative creation exist, and this project attempted to fill that gap.

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 can initially view a brief description and historical overview of the series, as well as individual turning points – The Punic War, The Final War of the Roman Republic, etc at a macro scale. They can then zoom in on individual events, seeing battle maps, troops deployments and other related microscale historical information. This transition between time periods is facilitated with a slider bar allowing for easy manoeuvre between the various date periods and event locations.

1.2 Report Structure

Chapter 2 is an analysis of existing software solutions to problems like that described above. Effective findings are discussed.

Chapter 3 devises requirements and priorities for the project. A discussion about tools and libraries to be used in development also takes place.

Chapter 4 holds general designs for the project. Various database schema are also recorded.

Chapter 5 discusses the implementation and how it was carried out. This chapter also discusses unexpected problems that were encountered during the development process.

Chapter 6 records the testing strategy and the results to those tests. Requirements are reflected on.

Chapter 7 contains the post-project discussions and a general analysis of the project’s successes, limitations and future.

Chapter 2

Literature Survey

2.1 Chapter Introduction

This project was largely a software development undertaking and so a portion of the task was spent examining existing software solutions that are available to the public. Investigating current applications mostly involved compiling a list of which features work well and which do not. As discussed in Chapter 1, this project is aimed primarily at an audience looking for education and so features that improve inclusive usability and information transmission were highly desirable.

Note that for ease of reading, the Literature Survey has been divided into two chapters, Chapter 2 contains a brief analysis of existing mapping solutions. Chapter 3 discusses appropriate software, development techniques and provides explicit requirement definitions.

2.2 An Introduction to GIS

The Esri institute defines Geographic Information Systems (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 format that provided easier manipulation and wider accessibility. “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 and McDonnell, 1998) Before the introduction of widespread GIS, an analyst looking to display their data in full would need to commission a cartographer to create multiple copies of the same map, each with different data sets overlaid. This approach was extremely 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.

There have been countless ground-breaking developments around GIS in recent years, be it unprecedented accuracy in mapping our planet (NASA, 2019), helping urban planners design our future cities (Xhafa, 2015) or even helping rescuers save lives during natural disasters. (USC, 2018) GIS is a quickly maturing and rapidly expanding field that it is likely to bring a huge number of exciting new developments in the next few years.

2.3 Existing Mapping Systems

The following systems are a cross section of map providing websites found during the course this project. The list is not exhaustive and is largely composed of systems that presented particularly interesting ideas. URLs to each of them are available in the references section of this report.

2.3.1 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 did not appear to be in any specific order and original production date information is often absent. This came across as a major problem and it made the website difficult to use as what is returned is unpredictable.



Figure 2.1 Old Maps Online results box.

Although the site had problems, the act of pressing on the map to select a region is a very interesting UI choice and greatly improves the user experience. Old Maps Online uses a very heavily modified version of Open Street Map (OSM Foundation, 2004) with Open Layers (MetaCarta, 2006). Both services and other relevant map providing software are discussed in 3.4.3.

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. This design choice was effective and easy to use.

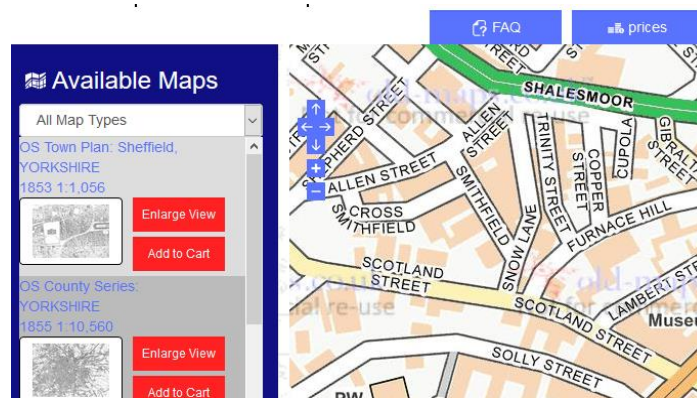


Figure 2.1 Old-Maps side bar

Chapter 2. Literature Survey

Navigating around and zooming in to an individual map was not an easy task and the website often took some time to transition between zoom levels. The system does not make it clear when it is loading and at first it appeared the website had crashed. This resulted in a displeasing user experience. The website has taken a series of questionable design choices, but an approach like the side navigation menu was extremely effective. Unlike Old Maps Online, Old-Maps appears to use bespoke software built on top of ordinance survey data.

The National Library of Scotland (NLS) also maintain 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 region boxes.

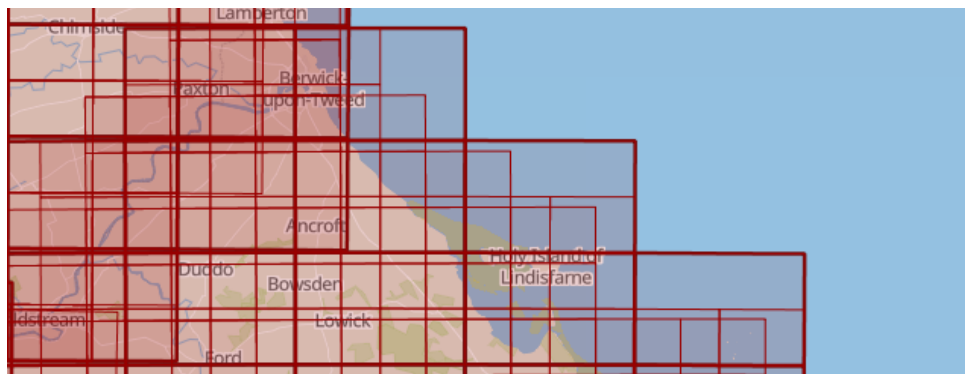


Figure 2.2 National Library of Scotland map overlay boxes

This approach was not as effective as Old-Maps from an aesthetically pleasing point of view and resulted in a cluttered map. The actual historical map display was better than earlier examined software though – selected maps are displayed in a zoomable subpage that allowed for easy viewing. Moving between historical maps was a problem, the only way to navigate through maps is to move back to the original macro scale map and start the search from the beginning. This was 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 used a version of Open Street Map.

2.3.2 Embedding Information

Having examined a few examples of actively developed GIS systems and with an idea of some industry standards, attention turned to some more specialised functions. As mentioned in Chapter 1, this is a website meant for educating children and so historical information needed to be included inside the map and in a way that is appropriate to that level of expertise. This information can appear as coloured map regions, markers or any other appropriate icon. There are several good examples of this, some of which are considered below.

Liveuamap (LM) (Live Universal Awareness Map co, 2014) is a web-based live map for tracking conflicts in areas of unrest. The display contains a series of coloured polyhedrons differentiating who controls which area of land and uses 'buttons' overlaid on the map which expand when pressed on. This design allows the user to quickly and easily examine a collection of events without making the display overly complicated. Information is dense, but the system as a whole is minimalist.

Chapter 2. Literature Survey



Figure 2.3 Liveuamap markers and information boxes.

LM is heavily geared to the 'here and now' and viewing events from previous days was 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 the creators appeared to have prioritised usability and aesthetic appeal over everything else. 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 entire map in significant detail. This minimalist approach is extremely effective at minimising screen clutter and allows only the most important information to be seen at load time. 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 drew an unusually large amount of bandwidth and processor power. The system was nigh on unusable on mobile. The user interface was generally pleasing to look at and the interactivity was extremely well done. The use of button overlays was very effective at minimising the amount of irrelevant information on screen at any one time, while maximising useful information density.

Like Old Maps Online and the NLS, LM uses a heavily modified version of Open Street Map. It appeared that Open Street Map (discussed in 3.4.3) 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 appeared to work in a similar way to the JavaScript SVG Polygon function and seemed to be comparable to how LM handles its layers. This technique appeared to be a standard in several online mapping tools. My Maps also allows a client to place markers on the map, allowing the developer to hide details until the user requests them and keeping the UI clear – again like LM. My Maps has no way of connecting a series of maps and so is limited in that capacity.

Continuing this, the Lord of the Rings Project (LoTRp) (Johansson, 2012) offers a similar approach to displaying map markers, as seen in figure 2.5.



Figure 2.4 Lord of the Rings project map markers.

Chapter 2. Literature Survey

Map markers are larger if multiple events take place at the same location. When clicked on, markers expand with an information box giving a breakdown of the details associated with it. One downside to this approach is that there was no quick way to analyse the map – each marker must be clicked on individually to understand the story. This website did offer one useful insight – it used Open layers and Leaflet (Discussed in 3.4.3). LoTRp was in many ways like the end goal of this project – navigation around arbitrary maps with attached markers and regions.

As discussed, there did appear to be a something of an industry standard through many of these systems. Coloured polygons are very popular and for good reason, they are highly effective at rapid information transfer. Markers are popular as well, but there did not seem to be any consensus as to how to present them. Side slider bars and presentation boxes were also widespread and were powerful tools for allowing users to navigate a system with speed.

2.4 Chapter Conclusion

This chapter discussed several systems that were similar to the end goal of project and discussed the pros and cons of each. Ideas were gathered and standards understood, allowing requirements to be drawn up with some certainty in how to proceed.

The following chapter is a deep dive into appropriate technologies, libraries and techniques that are required to produce a high-quality GIS tool. The chapter will also outline the end goal requirements of the project, considering the target market.

Chapter 3

System Requirements and Analysis

3.1 Chapter Introduction

With the initial analysis of current trends and standards complete, it was time to outline the project in detail. This chapter begins by defining several system requirements before discussing how the various tools, libraries and languages used throughout the project were decided upon.

See page 44 for an explicit list of requirements.

3.2 System Overview

‘A Time Traveller’s Map’ functions primarily as helpful tool for would-be map makers and for children both at home and at school. The software displays maps in an aesthetically pleasing and easy to use way and allows 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 is achieved through a slider and a side index bar.

The system is built on a five-level design. Series, Map, Event, Group and Layer. An example of this design in action would be – The American Civil War, May 1865, The battle of Palmito Ranch, Confederate troop positions, Wilson’s raid.

The user begins by outlining a series – a major event or string of events in history. They then connect several maps to that series and can navigate between them using the slider. Each map then has a number of events that represent an important occurrence in that maps time period. These events can be selected in the side bar and are navigated to automatically. Series can then be shared between users in either of two states. A Read share allows a user to view a map without making changes, Write allows for collaborative creation. A users’ series are accessible to the creator on any desktop device.

The slider displays active maps with a coloured block, allowing for immediate recognition of periods that contain interesting information. The slider automatically resizes itself to the start and end points of a selected series. This adaptive design is straight-forward for all types of users to understand.

Layers are a catchall name for a collection of possible map objects. The three main types of layers are polygons, arrows and markers. The defining trait of a layer is that they have one or more longitude and latitude values attached and can be directly interacted with on the map. Layers contain additional information within them which can be accessed by pressing on them on the map display. Generally, layers can be minimized to keep the display as clutter free as possible.

Users can connect with each other using a friends list that allows for easy collaborative map creation and sharing. Users can also block potentially malicious users. The default setting bans communication between users unless both parties have confirmed that they are trusting of one another.

3.3 Requirements

3.3.1 Requirements Overview

The following section outlines the seven key tenets of the project. These requirements are repeatedly referred to, to ensure the system stays 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 must always be easy to understand and intuitive to use. 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” (Farah Alsudani, 2009) An inescapable fact is that good applications look good and a significant amount of time was needed to ensure that the project was created in an aesthetically pleasing way.

Performance

As the literature survey showed, mapping software can be very processor intensive. The final system is geared towards minimising computational complexity and memory use where possible. This decision was further reinforced by the desire to keep the system as accessible as feasible. Important operations are fast to execute and the user is alerted when speed is impossible.

Stability

The system is expected to be usable at any time and so is generally protect against crash causing errors and invalid inputs.

Platform Independence

Ideally, the system would be able to run on any machine running any operating system or browser with no noticeable difference between the distribution. This is infeasible, but effort was spent to make the system as cross-platform compatible as possible. The system can run on Linux and Windows operating systems as well as on Chrome, Firefox, Edge, Opera and several other browsers.

System Security

Although the project is highly unlikely to be the target of an attack, it is generally good practice to be prepared. The system is resistant to many common types of attacks. In 2013, “the average prevalence rate of injectable URL of IPv4 was 5.55%” (Ying – Chiang, 2013) so SQL injections are defended against with the use of sterilised user input boxes and explicit parameterisation. Cross Site Scripting attacks are resisted where possible and manual URL manipulation is checked against the database to ensure users have appropriate access rights.

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. This is not a concern for the system in its current state but will be in the future.

3.4 System Architecture Decisions

3.4.1 Platform

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

Desktop systems have one major advantage over any other type of platform – no data storage cost. 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 the project was only aimed towards personal use, but that is not the case. Maps are meant to be shared between users for collaborative creation and for this to be achieved, the system would need to download copies of every map onto every PC and the user would need to manually conduct this update. This problem will only grow as the database expands and exists before even considering copyright issues associated with storing protected maps on personal devices.

A desktop application also suffers the problem of difficulty transferring to a mobile platform. Software does exist to convert a desktop application to mobile, but it is still a technology in its infancy and is generally limited to windows-based machines. No automated conversion to IOS or Linux compatible distributions could be found and even the Windows system has problems – “There are some important limitations... [apps] won’t work on Xbox One, HoloLens, Surface Hub, and other Windows 10 platforms” (HowToGeek, 2018) This means a full rewrite would need to be completed for the system to be transferred to a mobile platforms in the future. This would require an enormous time investment for a single developer and would be well outside of the scope of this project. A web application is relatively simple to convert to mobile, and visa-versa, allowing for superior system reach. New technology is in development to further simplify the conversion process “[The System] can effectively solve current problems in mobile applications development, such as automatic code generation for cross-platform mobile applications” (Rosales-Morales, 2019), meaning conversion may be near trivial in the future.

Finally, the problem of bug fixing. If a large desktop bug were to be detected after launch, it would require a huge amount of work to fix. “Defects found in testing were 15 times more costly than if found during the design phase... [and 100 times more costly if found after deployment]” (IBM, 2010) It would also require some type of software updating subsystem - this would need additional research to build safely and legally and taking that route would eventually require ethics approval. A web-based or mobile bug can be discovered, fixed and updated without the user’s knowledge.

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

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

One big advantage of a mobile based application is centred on user trend data – “in 2018, nine out of ten phones will be smart phones... sales of smart phones grew by 20.3%” (Ek-Kassas, 2017) Mobile applications are a large and growing market. This advantage *could* be taken advantage of in an online application by specifically designing the website to operate on a mobile phone. If this route were to be explored, it would likely be outside the immediate scope of the project and would need to be delayed until a later stage of development when the core functions are in a more mature state.

Finally, consider common successful mapping systems like Google Maps, web-based mapping software is very much the industry standard for casual map viewing. That is not to say mapping software does not exist on desktop or mobile. There are many desktop-based mapping software systems - 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 platform of choice could only really be web-based. In the future, attention may be turned towards creating a limited, processor friendly and specifically designed mobile version.

3.4.2 Programming Languages and Tools

Considering the system was to be a web-based application, the choice of which programming language to use was 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 were used as the 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 was sought. 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. These differences will be considered shortly. First, as mentioned in 3.2.1, performance was 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 needed to be maintained and server queries had to be handled quickly. There are two commonly used ways of handling server requests -

Concurrent Threading is 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. Concurrent threads are moderately reliant on manually developing switching rules.

Asynchronous Processing is very similar to concurrency but allows 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 (Node documentation, 2017)

Hypothetically, if “Fast I/O and a limited number of connections” (Mansnun, 2017) could be relied on, manual multithreading would have resulted in slightly increased query execution efficiency. Unfortunately, this was not a guarantee and so it was deemed preferable to plan for the worst-case scenario – low server responsiveness and numerous active connections. Asynchronous is heavily favoured in this type of scenario. (Mansnun, 2017)

Chapter 3. System Requirements and Analysis

Because of the advantages discussed above, asynchronous architecture would have to exist in the choice of server-side tool. This decision discounted PHP from the running - the language does not fully support asynchronous programming, instead “relying on external dependencies like Spatie” (Brent, 2017) to emulate an asynchronous nature. Ruby could 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 to function correctly.

This decision left 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 many use cases (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 could be developed in JavaScript, significantly simplifying the project structure and allowing both ends to be developed in the same IDE, vastly accelerating development time.

There was one downside to Node - it is widely regarded as having a steep learning curve. This downside resulted in a small number of false starts as familiarity with the language first needed to be nurtured. “Callback hell” (Alex Soft, 2018), was also a problem that reared its head repeatedly throughout the development process – nesting callbacks and forcing a wait for Promises to be returned. As experience with the Promise process developed, this complication eased significantly and was a near non-issue towards the end of development. Because of the arguments above, Node was the tool of choice.

The standard and by far most documented and used Node.js server framework is Express. The framework is “fast, minimal and flexible... the defacto API for Node.” (TecMint, 2019) Because of that, the project used Express.

For the complete full stack experience, a templating engine had to be used to minimise the amount of repeated code. The differences between the various options is limited and most of the candidates would work perfectly well with Node and Express. The two most popular templating engines are Embedded Java Script (EJS) and Handlebars.

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) This is not an enormous advantage, but it would help in keeping the project structure in a clear and useable state and accelerated development speed when familiarity with the library was achieved. As well as this, “EJS has no overlap with Angular, React or any other common frameworks (Raj, 2015). This is not of any great importance at this present stage of development but may prove beneficial in the future. For these reasons, EJS was the templating engine of choice.

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

3.4.3 Mapping Software

As anticipated, finding reliable comparisons between various GIS tools was a difficult task - the community who develop with them is relatively small and very little documentation exists. Finding accurate comparisons of performance was even more difficult as there does not seem to have been a systematic study produced at any point.

When deciding on a GIS map package provider, there were only two major options to pick between – OpenLayers and Leaflet. Examples of both packages were shown in Chapter 2. As discussed repeatedly throughout this report, performance is king, and a small dependency is typically preferable to a monolithic one. One of the few comparisons that could be made is package size. OpenLayers – 151.7kb, Leaflet – 39.6kb (npm Trends, 2019) As is clear to see, leaflet is a far smaller package - just one fifth the size of OpenLayers. This size difference also means that leaflet provides a smaller number of useable development functions - but this is not necessarily a bad thing. The general design of this project is fairly unique and most of the functions used in it were always going to be bespoke. A lightweight package that provided the bare minimum of functions but with vast room to customise would always be preferable and that is essentially what leaflet was made for.

One additional advantage for leaflet over Open Layers was MapBox. “Mapbox extends the popular Leaflet Library... providing robust geospatial data with our Streets, Terrain, Traffic, and Satellite tile sets” (MapBox Documentation, 2018) The extension essentially provides an array of map styles that can be displayed to the user. MapBox could be easily incorporated into the system and greatly improved user experience. Using MapBox would also eliminate the need to use any additional dependencies – Open Street Map for instance – as MapBox provides similar functionality but pre-packaged with Leaflet.

Leaflet is small, customisable and has several useful extensions. All great attributes for a system that is expected to be largely bespoke. Because of this, Leaflet was chosen to handle map rendering.

3.4.4 Database

Considering this application was going to be web-based, some thought had to be put into what type of data storage to use. The main decision to make was 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 anytime soon. Performance is also something to consider. According to (Kumar and Prabhu, 2017), “NoSQL style databases perform queries approximately 40% faster than their SQL counterparts. In the same paper, Kumar outlines that this effect is only noticeable when considering many thousands of queries at once and the speed difference will not be seriously felt on this application. This may change at some point in the future if the software expands and demands a greater number of query requests. A further discussion may have to take place at that later stage of development.

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, 2014) and (Lewitt, 2015) agrees saying “there is a lack of support tools to help.” Due to the limited time frame available to develop this system, attention needed to be paid to development speed and a well-documented style typically results in far faster development.

Experience is a lesser concern but was still something to consider. The author had a deal of experience developing applications using SQL style databases and minimal with NoSQL. That experience would transfer into rapid development. It would have been 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 would have very limited effects on the system, the only real differentiating factors were ease of development and experience. It was likely that development would have been far faster using an SQL style database. For that reasons, it was decided that the system would be built with an SQL backend.

3.4.5 DBMS and Mapping Data

One thing to consider when choosing a database flavour is whether the chosen service can store and use data in an efficient way. The project, as mapping software, would rely heavily on longitude and latitude values and a database that could store and manipulate data in the form $[(X, Y)]$ would be heavily preferred.

This requirement immediately discounted the use of MySQL as the service offered no support for geospatial data. MySQL is also not appropriate because it only offers library support for calculations based in Euclidian geometry. (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 or use longitude and latitude with accuracy.

Because of this, one of the few remaining popular SQL style databases to consider was 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. This position is supported by Agarwal, saying “PostgreSQL/PostGIS inherits more than one thousand geo- functions. MongoDB only supports three geo-functions,” and “Currently, [NoSQL databases] support only a limited type of geometry and have very few functions to manipulate them.” (Agarwal, 2017)

Before settling on a final database type, there were two final things that need to be considered – format of date data types and of BC dates. These two concepts needed to be easily translatable into a JavaScript usable format and visa-versa. According to the (PostgreSQL documentation), “dates are stored in the standard format “yyyy-mm-dd”. This is preferable, but there was another problem – PostgreSQL stores months 1-indexed where JavaScript is 0-indexed. The system would require conversion code to be created for communication with the database.

Further, BC date formats are stored “...with the suffix ‘ BC’” (PostgreSQL documentation) attached to the end of a string being inserted into the database. This would 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 was a potential problem that was considered before implementation. 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 (JavaScript Documentation) - this will cause a crash in PostgreSQL. Generally, the new Date function does not handle negative dates very well in any situation and this problem is compounded with 0 inputs. Error handling code was necessary to ensure these inputs were never entered, and general caution was paid in around 1BC/1AD to ensure the phantom 'year' 0 had no effect on the normal functions of the website and that the additional year did not create an offsetting factor into calculations around this date range.

Because of the considerations above, the system would rely on PostgreSQL to handle data storage, though extra effort went into using negative dates and around the 'year' 0.

3.5 Ethical Considerations and Pre-Implementation Changes

From the start, this project was intended as an educational aid for children - this was always going to be a key focus during development, but some changes took place before implementation began. First, the amount of data that was to be stored on the system raised some important ethical concerns. Storing large collections of data - especially from children - brings additional unnecessary risk and is something to be extremely wary of in general. "Children require specific protection with regard to their personal data as they may be less aware of the risk (GDPR Recital 38, 2016) Because of this, the required information was slimmed to a bare minimum - no delicate personal information is stored at all. In the event this system was distributed to the public, the only details to be requested on sign up would be an email address, username and suitable password. Usernames would be the only public information available to users. Note that ethics approval would have to be sought because of the need for email addresses to deter bots and malicious users. (Sheffield University Ethics Procedure, 2017)

Cyberbullying and abuse were also considered before development began. "The number of young people reporting instances of cyberbullying has doubled since 2011" (Comparitech, 2018) It is conceivable that users could upload offensive images in place of maps. To combat this, users can only share maps with people whose usernames they know and have whitelisted. Only admins can distribute maps to all users. This should prevent unsolicited images being shared to a wide audience. Users can also un-whitelist their contacts and can add users to a block list.

3.6 Chapter Conclusion

This chapter decided on a list of requirements that would be the guide for the development process and discussed the tools, packages, libraries and platform that would be used throughout. A full list of explicit requirements can be found in the appendix.

Chapter 4 contains the initial draft designs for the final product and considers various database design possibilities

Chapter 4

System Design

4.1 Chapter Introduction

From the start, the plan for this project was to minimize the amount of developed pages and to make the display as minimalistic as possible, while maximising usable information density. To achieve this, a dynamic approach to design was needed as discoveries of what worked and what did not would become clear through extended use and testing. To this end, an expectation of dramatic change throughout the course of development was prepared for.

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.

4.2 User Interface Design

4.2.1 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 features accessible through menus. The initial design for the general front page of the website was created with this expectation in mind. The map takes up most of the screen, like both Lord of the Rings Project and LiveuaMap. (Chapter 2)

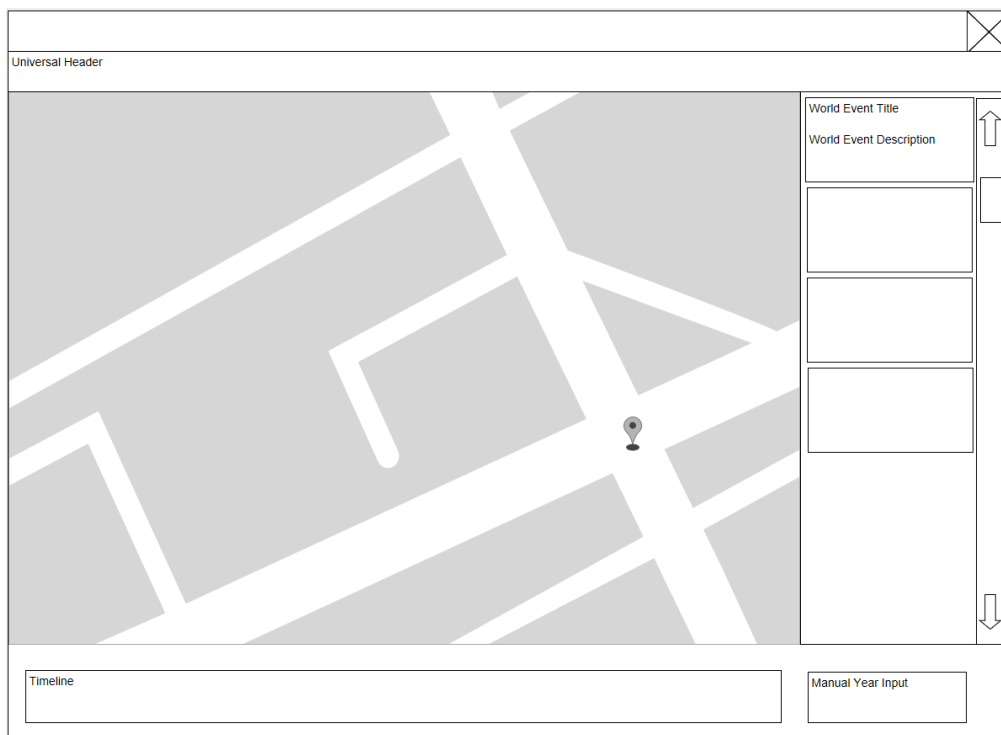


Figure 4.1 Initial user interface design mock-up.

This design allows for easy use and immediate understanding of how the system works. Users can instantaneously see the main function without needing to navigate through menus. This index screen is identical for both the logged in and logged out users.

4.2.2 Display Boxes

The display box is a generic container box for Series, Maps and Events on the map. The idea behind them is to allow users to get a quick overview of what that box contains so they can rapidly choose whether to investigate deeper.

Display boxes have several on-click functions attached to them, dependant on what level of the Series-Map-Event hierarchy they are being used to display. A series display box begins a new series transition and updates the timeline with associated maps. A map display box will move the timeline to the selected map, displaying associated events. An Event display box will move the user to that events location on the map.

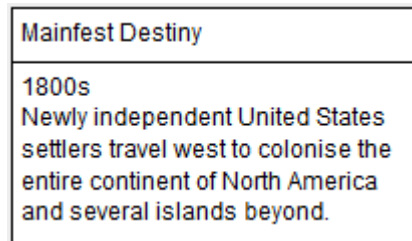


Figure 4.2 Initial series display box design mock-up.

This unified design allows for max usability as users do not need to become comfortable with multiple different designs of display boxes.

4.2.3 Timeline

The timeline design focuses on the principles stated above – minimalist but with maximum information transmission. Figure 4.3 shows this in action.

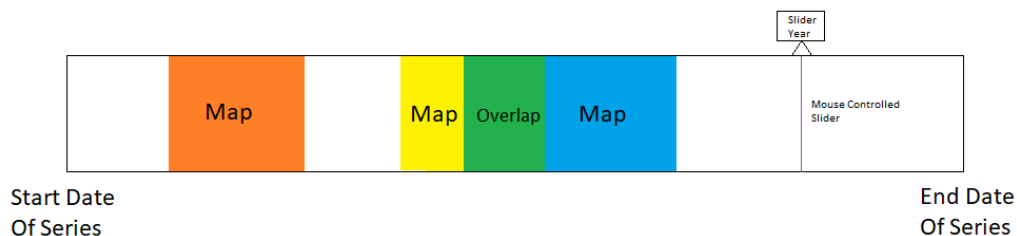


Figure 4.3 Timeline design mock-up.

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 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 a user wishes to go to a specific date, a popup above the slider makes this very easy. When multiple maps overlap, a special display on the timeline appears, and when selected, the display will enter a unique state, allowing users to flick between them.

This design was primarily created with children in mind – the colours immediately make clear that *something* interesting is happening at that location on the timeline, which should draw users there naturally.

The design also has those who are colour blind in mind – the end product for this feature will have a colour-blind mode, which only displays colours that have the correct contrast.

4.2.4 Map Annotation

Map annotation is a key feature of this system and a significant amount of thought needed to go into how to display the map tools correctly so that all potential users can make use of them without issue. To that end, the design had to be simple, but expansive. Ideally, all functions needed to be in the same tool so that users do not need to delve through endless menus to find the one feature they need.

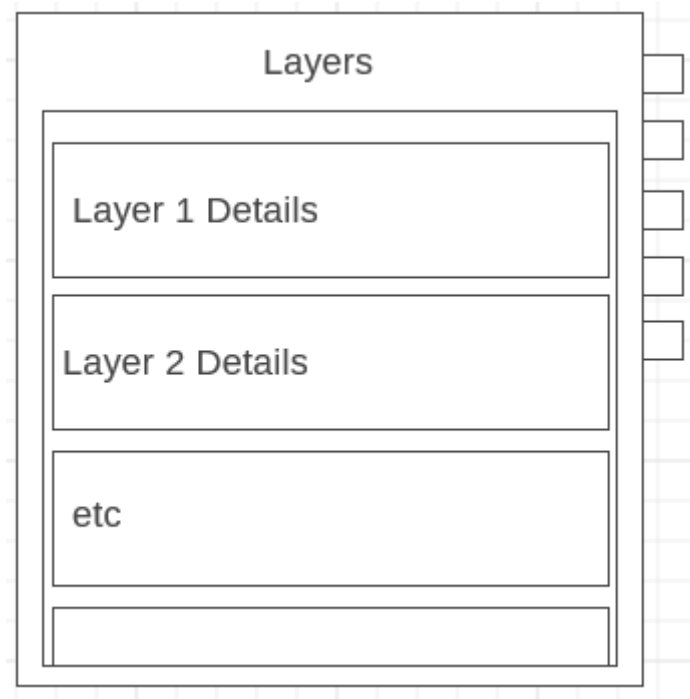


Figure 4.4 Initial map tools design

Figure 4.4 shows the basic design for the map creation tools. Each tab contains one stage of the Series-Map-Event hierarchy and contains all needed tools in that one tab.

This design would prove to be insufficient in a space effective and aesthetically pleasing way and would require significant redesign to make work correctly. The tabs were moved to the bottom and top and significantly more were added, giving additional functionality to the tool. This is discussed in detail in 5.6.1.

To complement this, the map needed markers to indicate to the user that some microscale event was happening at that location – the location of a cavalry charge, for instance. This design is the most basic of possible designs and the expectation was that colour and images would be used in the production version. This turned out to be correct, and a wide variety were added. Discussed in 5.6.4.



4.5 Map marker design

These designs made up the skeleton that would end up as the final implementation of this system, but they were a good foundation, and most were implemented as expected.

4.3 Database Design

The original design for the database can be seen in Figure 4.4: (Arrows indicate One → Many, text fields substitute a [longitude, latitude] data type) The final design can be seen in Figure 4.5.

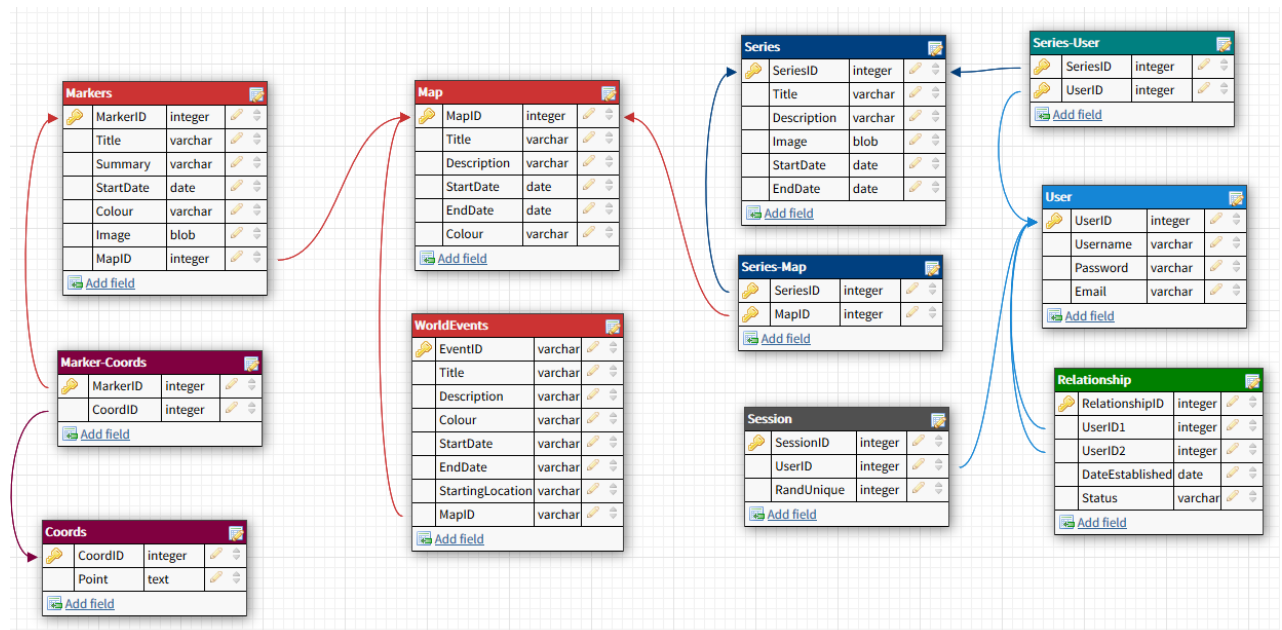


Figure 4.6 Initial database design

This initial design was born from an undeveloped understanding of the information required to store and recreate layer objects in the database. The biggest misunderstanding was in how layers and layer groups worked – ‘layers’ is a general term for a number of different objects which have similar properties, they can largely be recreated from the same few shared data values, allowing the Events and Markers tables to be rolled into one. This streamlining allows the data structure to follow roughly the same structure as the leaflet package – Map, Group Layer, Layer with the addition of Series before Map and Event as a collection of layer groups on a single map.

The session table was also removed and replaced with a single column in the user table – ‘loggedIn’, the ramifications of this are discussed in 5.4.

The final design had one further advantage over the previous – it vastly simplified the table structure. The general database flow is now linear, and the number of tables has been reduced by one. User-Series now also contains additional unique information, meaning it is no longer ‘just’ a linker table and has a useful purpose – the ‘ReadWrite’ column gives users the ability to edit another users maps if granted permission from the owner.

‘Readwrite’, as well as ‘StartLocation’ and ‘StartZoom’ in the ‘MapEvents’ table are examples of scope creep – discussed in detail in 7.2.

One last thing to note in relation to the database is that the ‘WorldEvents’ table – discussed throughout Chapter 3 – is not present in the final design. This is because the path to implementing them was unclear and their inclusion was delayed until a later stage, as outlined in 5.5.3.

Chapter 4. System Design

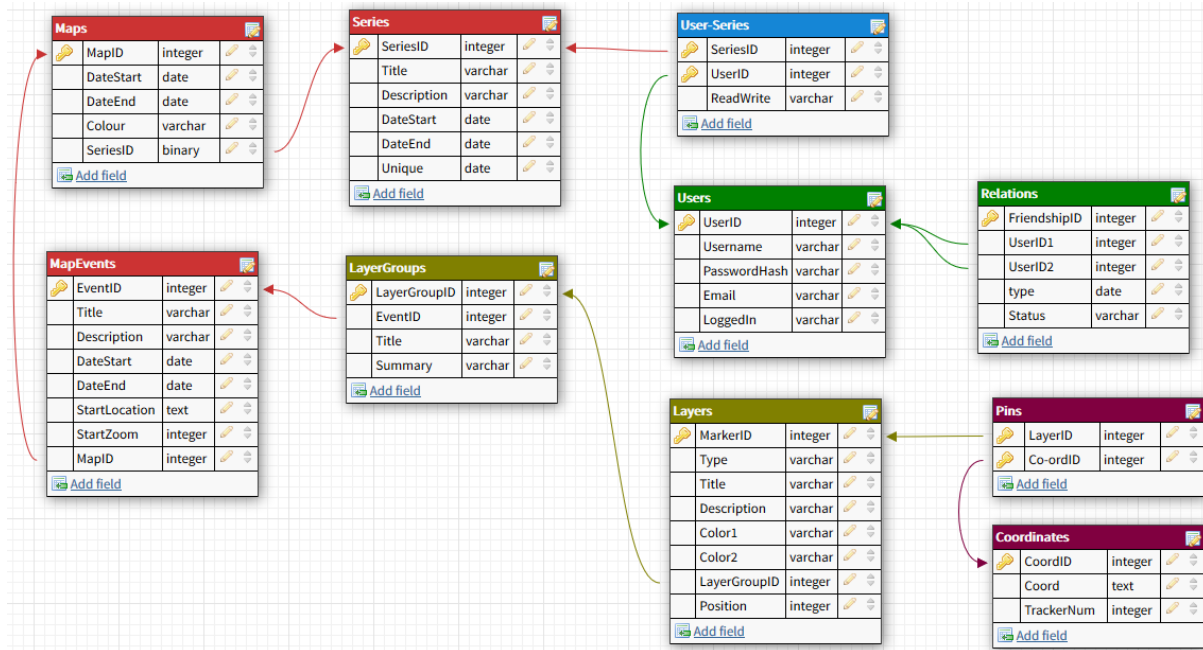


Figure 4.7 Final, normalised database design

4.4 Chapter Conclusion

This Chapter was a brief chance to plan out some of the designs that would act as a guide for the development period.

The following Chapter digs into the implementation details and discusses how the project progressed, outlines some of the challenges faced and describes a few solutions to those challenges.

Chapter 5

Implementation

5.1 Chapter Introduction

Having decided on the specific tools and libraries that would be used throughout this project, and following the completion of initial user interface designs, it was time to begin development. This Chapter begins with discussion of general system features, before focusing on analysis of individual components and some of the problems encountered.

5.2 Software Lifecycle and Development Paradigm

Throughout the course of this stage of development, the author attempted to follow a highly agile design methodology built around the Feature Driven Development (FDD) (Palmer, 2001) and Just Enough Design Initially (JEDI) models. The ideas behind these models are to work feature to feature with new development plans written up at each step. Both models expect significant levels of change throughout the project and encourage developers to build that expectation into their planning. Testing is completed throughout the project, followed by a final catch-all test at the end. These principles are critical in a development situation like this, where time constraints are by far the most significant limiting factor and where 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.

5.3 Project Structure

One of the biggest hurdles for projects of this size is losing track of which files do what. This makes continuous, uninterrupted development difficult and maintenance near impossible.

The general file structure of this project followed a Model-View-Controller (MVC) methodology where the project is divided into three parts – models, where data and database communication is handled. Views, what is displayed to the end user to allow for interaction. Controllers, handle inputs and commands the models and views. This layout vastly simplified the standard monolithic project layout and made for very quick development as related files were close to each other and in a highly ordered and organised way. As well as this, each part of the MVC structure was divided into further sub divisions to allow for fast pinpointing of required functions.

Regular reorganisation of the file structure took place throughout development to ensure that the project did not slip into disorder. A full display of the final file structure can be found in the appendix.

5.4 Server Security

The author chose to use the ORM library Sequelize to handle SQL query creation and sterilization. The library is well documented, powerful and reduces database inputs to Strings, making it much harder to inject malicious code. This should protect users from any potential hostile actors.

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’. 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’ is compromised, it would only be useful for a short period of time – until the next login, logout or tab close. The function used to generate the low collision string can be seen in the appendix.

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 they lack appropriate access to the required page, they are redirected to the logged in/out index appropriate to them. ‘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 does not need to remember the 128-character unique identifier and the ID still does not need to be exposed to the client-side.

As well as this, passwords are hashed upon arrival to the server and are never stored in plain text form. Finally, if this project does have a public launch, it will use the HTTPS protocol as authentication.

5.5 Timeline Development

5.5.1 Timeline Granularity

The original timeline design was discussed in section 4.1.3.



Figure 5.1 Extended timeline design

As is clear to see in Figure 5.1, 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 could not 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 in figure 5.2.



Figure 5.2 Simplified timeline design

The same data as Figure 5.1, 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.



Figure 5.3 Simplified timeline 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 constraints and a moderate priority, it was decided that the extended timeline would have to be delayed until after launch so other, more critical requirements could be focused on. This is a regrettable decision, but one that allows many timelines use of 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.

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 was a problem that required a significant amount of time to solve completely.

The PostgreSQL documentation lists the appropriate way of handling BC as appending – “BC, bc, AD or ad – Era indicators” to dates. This, in and of itself, was not a problem if converting directly into a date format on the server side. Unfortunately, this was 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 (JavaScript Documentation). This method causes unexpected results from some inputs.

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, which will result in 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 were particularly troublesome when parsing.

To solve this problem, there is a relatively unknown input to the Date constructor that allows easy use of BC dates. (“...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. Amazingly, this technique worked almost immediately with the code already written to handle AD dates, requiring only minimal alterations to fully generalise. Care was still be taken adding 1 to months of dates retrieved from the database and subtracting 1 from those being inserted.

Despite this success, there was 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 was not counted when calculating map block locations and offsets, but otherwise, there were no significant problems.



Figure 5.4 Timeline transition BC - AD

The simplicity of the final timeline design makes for an easy to use navigation tool that is both minimalist and information dense. The final timeline is also appealing to the eye and – with the additional of a small animation when changing timelines – made for an attractive display.

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 created by admins.

The intention was for this bar to display all three of these types, but complications with the timeline – like those in 5.5.1 – 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 completion, so the idea of a global timeline was put on hold for the near future. To facilitate the user browsing these maps without the timeline, any of the three types can be hidden with a tick box in the user tools menu.

The significant downside to this decision was that the ‘W’ requirements, as shown in the appendix, will also have to be delayed until after publication as they were reliant on a global timeline working as intended. The option of developing a temporary work around was considered and then dismissed, to save time for more pressing requirements.

5.6 Map Tools

As discussed in 3.3.3, Leaflet applications contain three tiers of container object. The top most of these is the map itself, the second is layer groups and the third is individual layers. Layer groups allow for mass manipulation of layers and permit the website access to individual layers by use of their id number. This section discusses each of these container objects and how they can be combined to make a useable map series.

5.6.1 Tool Summary

The leaflet package is generally considered light but highly flexible. There are very few major functions beyond the main map display, the above-mentioned tier system and a rendering system for shapes. This meant that most functions had to be created bespoke. As discussed in 3.3.3, the lightweight nature of Leaflet was one of the major reasons for choosing it as a map provider as it allowed for maximum customisability.

Considering this product is supposed to be used by children, the design was kept as simple and easy to understand as possible. The map creation tool design had to balance the original goal of minimalism with ease of use and some compromises had to be made in pursuit of that goal. Those compromises resulted in the design shown in Figure 5.5 (with placeholder images)

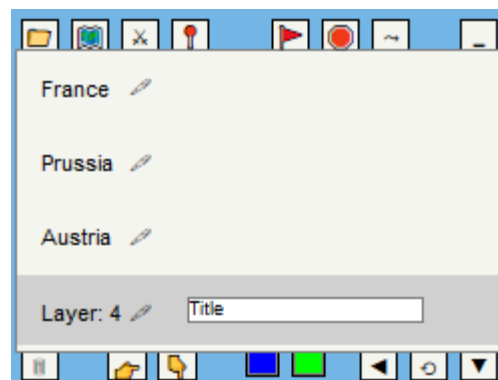


Figure 5.5 Map Tools overview

This layout allowed for a high level of information density without compromising too much on appearance or minimalism. The general idea was that all the functions should be included in a single tool so users were not required to delve through menus to find a specific method. After an initial period of learning, the tool became intuitive to use.

Layers, shown in Figure 5.5, are each composed of a custom object that contains the leaflet group layer itself, a collection of ordered polygon longitude and latitude values, a list of ordered pins, a dictionary of unordered information markers, colours, a title and related historical information. Each layer can then be saved to a temporary local dictionary for the duration of the edit, before being uploaded to the database. This system, while appearing a little complicated at first, made for easy manipulation of layer objects, fast switching between different layer groups and kept database upload and retrieval simple.

5.6.2 Polygon Functions

The process of creating new maps and events is simple to use and powerful. The inbuilt Leaflet polygon drawing function is highly limited – It can only take an array of longitude and latitude values and returns a coloured polygon. There is no inbuilt way of changing the polygon after creation or moving points without manually rewriting the list. To fix this, a wide array of custom functionality had to be added.

To start, a function was created to translate user mouse presses into longitude and latitude. These values could then be pushed to the layer array and the polygon redrawn. This allowed custom polygons to be created freely without any need for keyboard input from the user. After that, pins were made to appear at each vertex in the polygon to show the user where the selected polygon vertices were. Pins only appear on layers that have been selected in the above layer tool. This function vastly improved usability and made it easy to add additional functions to the system at later stages.



Figure 5.6 User made Polygons. Left region selected.

The design of the first iteration of the polygon drawing algorithm had one significant downside - It was difficult to return to a map later to add additional regions or detail. New pins could only be placed at the very end of the longitude and latitude array which caused unwanted lines to appear when returning to an earlier region of the map.

Trying to expand a map using the original naive polygon drawing algorithm can be seen in Figure 5.7.

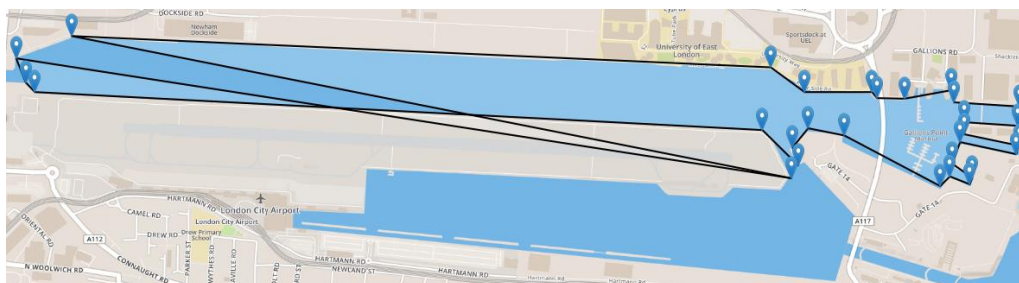


Figure 5.7 Naive map drawing algorithm

To solve this, two new functions had to be created. The first allowed users to select a pin as an 'anchor' point. The second inserts new vertices into the polygon array after the anchors' index.

Chapter 5. Implementation

The red pin has been clicked on and is now the selected anchor point. Afterwards, both regions of the reservoir are inside the desired area, as seen in Figure 5.8.

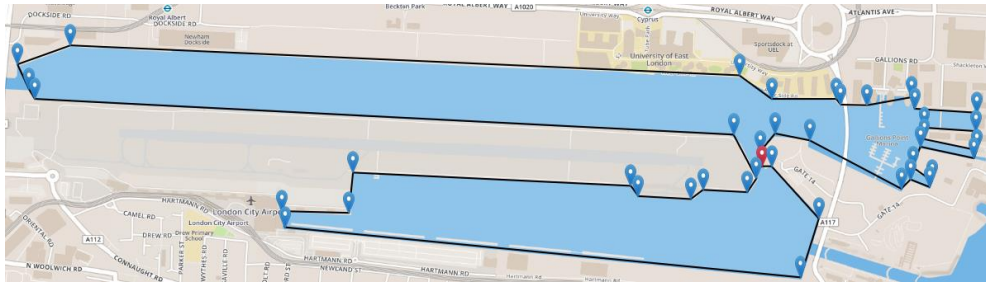


Figure 5.8 Advanced map drawing algorithm

This was a good start, but there were still several limitations to that tool. Sometimes it may be useful to create a quick first draft of a map before returning to add detail later. This process can be done by adding a small number of roughly place pins around the required area.

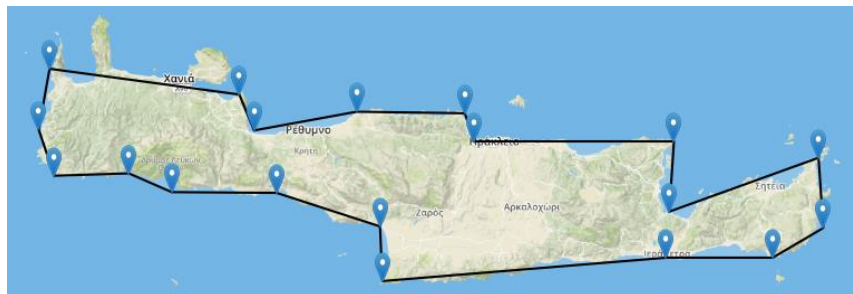


Figure 5.9 Limited polygon function

This can result in badly placed pins that are far from their correct locations. To fix this, a function was created to allow pins to be moved and deleted with ease. Pins can be moved by right clicking after an anchor has been selected and deleted by clicking the back button on the map tools popup. Pins also track their order of placement, allowing the user to undo as many pin placements as they wish.

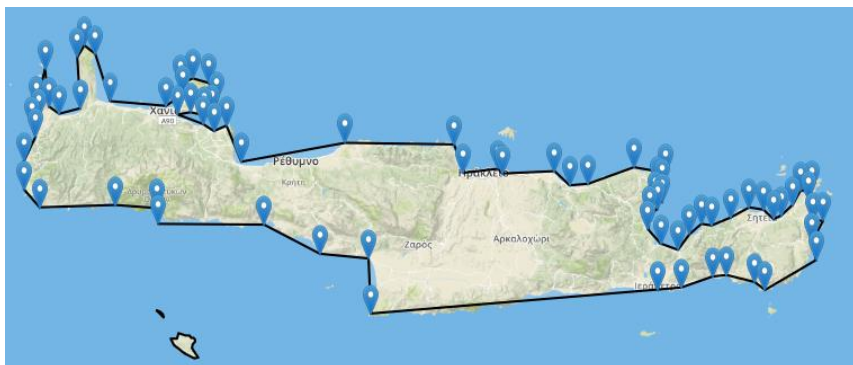


Figure 5.10 Expanded Polygon function

5.6.3 Creating New Series, Maps and Events

As discussed in 3.3.3 - A series is a collection of maps, a map is a collection of events and an event is a collection of layers groups.

Layer group creation is discussed in the previous section and event creation requires only a small extension to formalise. The only addition required to create an event and upload it to the database is a name, summary, starting location and start / end dates. Input fields to add this information are in the map tool.

Series are created in a similar way by navigating to the series menu in the map tool and inputting the required information. Note that Figure 5.11 is not the final design for this tool and acts as a proof of concept to ensure model functions work correctly. The final design will use drop down boxes instead of text input to ensure valid date inputs. Series can then be selected in a second tab and can subsequently have new maps, events and layers attached or have their information edited in the third tab. The information is displayed in simple English and in a very easy to understand way, even for children (3.3).

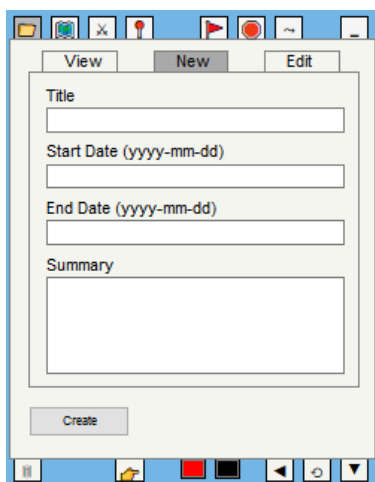


Figure 5.11 Create new Series tab

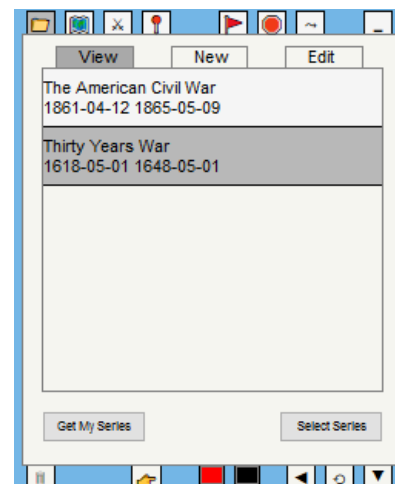


Figure 5.12 Select Series tab

Maps are created in two ways. The first is through the map making tool above and the second is by pressing on a link in the side scroll box. The general process of adding a map to a series is the same as that for creating a series.

5.6.4 Embedded Historical Information

One of the most important endeavours of this project was attempting to achieve usability through minimalism and information density. To that end, markers were designed in such a way as to display implicit information openly while hiding detailed information until requested. The simplest way this can be done is through colour. The current system has ten colours of pins, allowing them to be associated with various coloured polygon region. The colour can be changed in the map tools' layer menu. As can be seen in the image, pressing on a marker reveals the hidden information which is later minimised when the cursor is moved away.

This was very much the first stage of marker development and several ideas are in the process of being developed. The first of these ideas was discussed in 2.3 - using a library of general symbols with a variety of colours for each to give a clear hint of what took place at that location at first glance (as seen in LiveuaMap in 2.3.2). A few examples could be a speech bubble representing negotiations between parties, crossed swords showing the site of a battle or a bulb showing where an important discovery took place and coloured to show the relevant party.

Arrows also exist in the system, but they are not currently in a deployment ready state after numerous issues with them appeared.

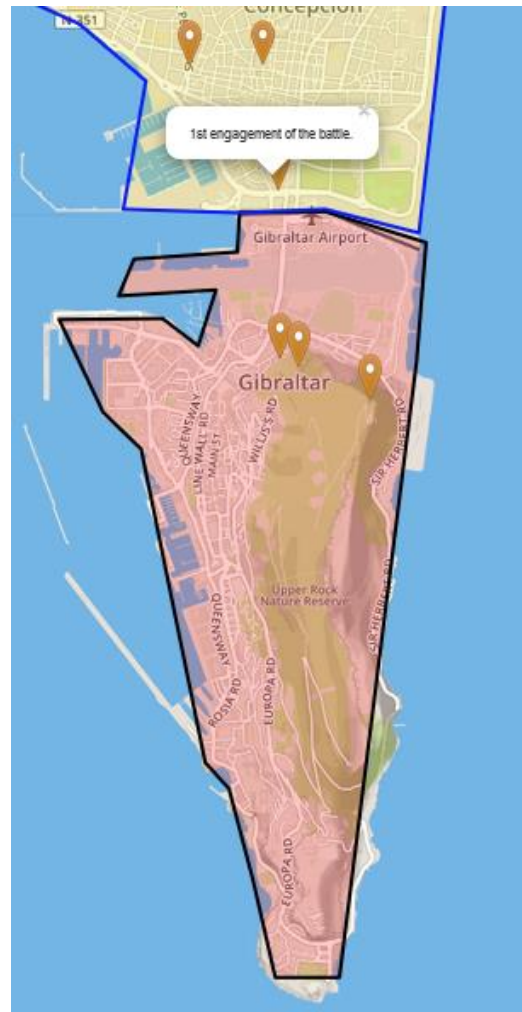


Figure 5.13 Embedded information markers in maps

5.7 User Tools

Early on in development, the decision to use a single page to handle most functions was made. The real meat of this project was in developing functions in the back end. To facilitate that, all user functions can be accessed in a single popup partial that can be created by accessing the drop-down menu in the index – similar to how to map tool works. This decision vastly simplifies the development process and allowed focus to be concentrated 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.

Chapter 5. Implementation

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. This allows for easy and quick transition between projects.

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. This action then updates the “Relations” table (4.3) 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. As mentioned in 3.5, protecting users is the most important concern for this project and users cannot communicate before they have both acknowledged one another.

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. This status is saved as ‘hidden_block’ in the database.

5.8 Administrator Tools

The system has a few administrator tools, though time constraints and a relatively low priority slowed their development considerably. The most visible of these tools is seen when creating a new series. Administrators have the power to create maps that are visible to all users.

5.9 Chapter Conclusion

This chapter has been a discussion on the implementation of the project – what went right, what went wrong and what required some change to work effectively. Some requirements were scaled back, and some new requirements appeared during development.

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

Chapter 6

Testing

6.1 Chapter Introduction

This Chapter discusses the testing that took place after the development process. The chapter is divided into two parts – Generic Testing and manual End-to-End Testing.

6.2 Generic Testing

Generic Testing is the process of ensuring the website is fast, secure and widely available. This was largely conducted manually.

6.2.1 Compatibility Testing

As mentioned in 3.3.2, compatibility is crucial to this project. To test compatibility, the full end-to-end requirements list was tested on all available major browsers.

Browser	Compatibility	Problems	Fixed?
Firefox	Full		
Chrome	Full		
Opera	Partial	Login issues	Yes
Edge	Full		
i.e 8.1 +	Partial	Some graphical problems	Partially
Vivaldi	Full		
Brave	Full		
Safari	Untested	No Mac / emulator available.	

The only major problem was found in Opera. Initially, the login option would not appear in the navigation bar (Shown in Figure 6.1). This turned out to be a cookie error and was simple to solve. Safari will be tested when necessary equipment is acquired.

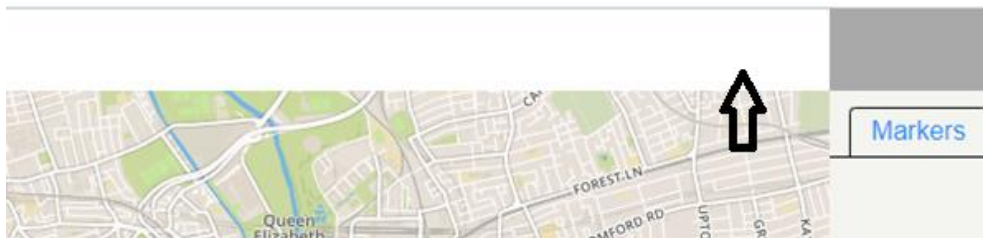


Figure 6.1 Opera testing error

6.2.2 Performance Testing

As discussed repeatedly throughout this project, good websites are fast and mapping websites tend to struggle with speed. To that end, the performance was tested to ensure the system is responsive to the user. Unfortunately, there are a limited number of useful performance tests that can be conducted on an offline website, as communication with the database is near instantaneous. The only tests that could be conducted are tests at either end of the system – database speed and client facing actions.

Most time consuming actions on the client side are handled through an $O(1)$ complexity hash map lookup and so there was minimal room for performance upgrades. The most time-consuming action was redrawing markers after a marker has been selected at $O(n)$ complexity. Tracking the full reset time for polygons has complexity $O(2n) = O(n)$ on marker placing and there was no visible delay between pressing on the map and a marker appearing on the map until over 600 markers were attached to the same polygon. This seemed an acceptable number. There was one problem on the client side – initial unique authorisation key generation (5.4). There was a key generation inefficiency discovered that was leading to delayed appearance of the 'Login' button by almost 1 second. This was caused by a Promise return logic ordering error that was delaying the key generation function, an example of 'Callback Hell' as discussed in 3.4.2. After fixing this bug, login time was reduced from 1.3 seconds to 8.2ms, vastly increasing login usability.

As for the database, there was little that could be done to significantly increase interaction speed as most of the execution time was due to database upload speed. The only significant avenue for improvement would be hosting a dedicated server – a consideration for the future.

6.2.3 Security Testing

For security testing, two main concerns were considered – SQL injections and URL manipulation.

To conduct the SQL injection tests, every user input box was tested with help from the SQL injection Cheat Sheet (Netsparker, n.d.). An example of this can be seen in Figure 6.2.

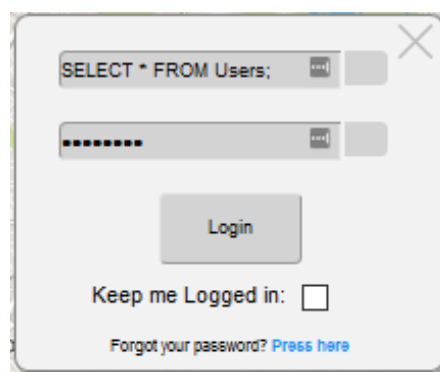


Figure 6.2 SQL injection example

There was never an expectation of any SQL vulnerability and this proved to be an accurate assessment. Every input box is sterilised (5.4) and reduced to a string with Sequelize. There were no discovered opportunities to SQL inject into the website.

Chapter 6. Testing

A URL manipulation attack attempts to manually input a server route into the URL to indirectly access a function and, in turn, the database. To conduct this test, every server route and model was tested in both a logged in and logged out user state to ensure verification was being carried out.

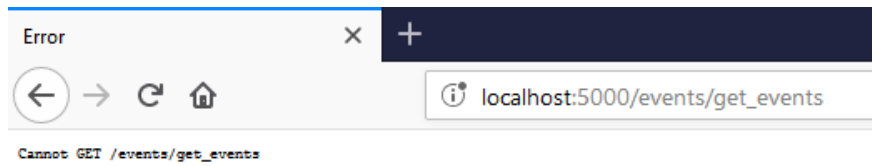


Figure 6.3 A basic URL manipulation attack example

This testing returned no anomalous results, each route was correctly checked and nothing unexpected was returned to the client.

6.3 End-to-End / Acceptance Testing

The last thing to consider is if the website fulfils the original requirements, as set out in 3.1/2 and listed in detail in the appendix, have been fulfilled.

Requirement	Priority	Completed?	Comments
M1	Maximum	Full	
M2	Maximum	Full	
M3	Maximum	Full	
M4	Maximum	Full	
M5	High	No	
M6	High	Partial	Some features, not all
M7	Medium	Full	
M8	Low	Partial	Admin and owned show at start
M9	Low	Full	
M10	Low	No	
M11	Low	No	

E1	Maximum	Full	
E2	Maximum	Full	
E3	Maximum	Full	
E4	Maximum	Full	
E5	High	Full	
E6	High	Full	
E7	Medium	Partial	Viewable in side slider
E8	Medium	Partial	
E9	Low	Partial	Some events can be toggled, not all
E10	Low	No	

T1	Maximum	Full	
T2	Maximum	Full	
T3	Maximum	Full	
T4	High	Full	
T5	High	Partial	Future is visible, past is not
T6	High	No	
T7	High	No	
T8	Low	No	

U1	Optional	Full	
U2	Optional	Full	
U3	Optional	Partial	Needs Ethics approval first
U4	Optional	Full	
U5	Optional	Full	
U6	Optional	Full	
U7	Optional	Full	
U8	Optional	Full	
U9	Optional	Full	

Chapter 6. Testing

To summarise, testing produced a fair number of successful results and showed that requirements were generally met. In total, 24 / 44 requirements were met in their entirety and 7 more were partially fulfilled. The 7 World Event requirements were discounted for reasons discussed in 5.5.3.

6.4 Chapter Conclusion

This Chapter discussed some of the testing that went into ensuring the project would function as intended.

The following Chapter will reflect on the project, analyse how the system developed and consider plans for the future.

Chapter 7

Evaluation and Conclusion

7.1 Chapter Introduction

With the first phase of implementation complete, discussion turned to evaluation in preparation for the future. This chapter discusses several decisions that took place during development, evaluating the effectiveness of those decisions and reflecting on how well they performed in fulfilling the initial requirements. Near-term development plans will also be discussed in this chapter.

7.2 Implementation Evaluation

The system generally performs as was intended and offers a wide variety of tools to map makers. The physical mapping features are well fleshed out and robust and the user-to-user interaction system is developing quickly.

One of the biggest recurring problems faced during the development process was the need for several database redesigns. (The original and final designs can be found in 4.2) This was a problem that stemmed from two sources – unanticipated library complexities and scope creep. The biggest factor in database redesign came from the leaflet package and an insufficient understanding of its core mechanics and composition. Layer objects required significantly more stored information to recreate than anticipated and this was a problem that could only be partially offset by a deeper reading of the documentation. It required an understanding that could only be gained through hands-on experience over an extended period. As familiarity with the software increased, this problem diminished, and the result was a finely tuned database that stored all the necessary information in an easy to use way.

Scope creep was another concern throughout the project and focus constantly had to be returned to the original requirements. This problem did not directly slow development, but it did require a few alterations to the database. Adding a read/write system to all shared maps is a good example of this. Scope creep *did* have a benefit though, it helped with development morale and led to some features being produced faster than they would otherwise have been. If this process can be done in a more orderly fashion, it could well produce a better overall product in less time. In the future, it may be a good idea to designate a certain number of days a month to focus on innovative ideas and reconsideration of niche requirements.

As for systems architecture, Node.JS was certainly the right choice for this project as development was remarkably smooth after the initial learning curve. ‘Callback hell’ was a problem at the start, but soon diminished. EJS presented some difficulty and this project is far from using its full potential, some of the more advanced features of the library were ignored in the name of dwindling development time. As anticipated, a Model-View-Controller (MVC) model vastly simplified implementation and made finding required features easy – even after the total number of functions crossed the 150 mark.

The decision to pursue a rapid 'Just Enough Design Initially' (JEDI) development paradigm was something of a double-edged sword – it allowed for rapid development, but with significantly less structure. This lack of structure caused unnecessary delays as the systems complexity increased and counteracted the benefits of the MVC model. JEDI also made for a 'here, there, everywhere' style of development where features were being created in an almost random order, rather than a tight, methodical sequence. As the project advanced, JEDI became more and more of a relic until it was completely abandoned in the last month of development. A more organised paradigm was adopted, and development accelerated because of it. That will likely be the style used in any future development as well.

7.3 Project Limitations

Most of the original designs were successfully implemented, but some proved too time consuming or too difficult to implement under strict time pressures. There are two significant missing features from this phase of development – A global timeline (5.5.3) and custom maps. Custom maps are a big loss for the project and will be one of the first features produced in the next implementation stage. Currently, the user can only create maps using the default Mapbox street view map – this is a little bit emersion breaking as modern streets are visible in a location they would not be in the past. There should not be any major roadblocks in implementing custom maps in the future, but significant research will need to go into it first. All the features created for the standard Mapbox system should be transferable to custom user maps.

The global and expanded timelines are the other feature missing from this interim product. Both resulted from a technical problem that will require a significant redesign to timeline granularity display, as discussed in detail in 5.6.2/3. This is a problem that is likely to take a large amount of time to find a solution to, but which will also be prioritised in the next phase of development. Because of this technical hitch, series can only be displayed if they contain less than 35 years. This is unfortunate but should still allow a significant percentage of series to be displayed.

Both features simply proved too time consuming to implement in a way that was worth the investment, considering the strict temporal pressures. This was probably the correct decision as the complete system functions largely in its entirety and that may not have been the case if multiple days were dedicated to fixing unnecessary minor issues.

The website would ideally be live and in a viewable state to the public, and that was an option in the closing week of this project. Unfortunately, it was deemed that the site was not in a complete enough condition to be released. Before that can happen, two things will need to take place – the first is ethics approval. Users must submit an email addresses to prevent bots and malicious users from abusing the service. This is an unavoidable step, but one that will benefit the system in the long run. The second is live user testing. Live user testing was a goal from the start that simply proved infeasible in the time available. The service was only in a good enough state for users in the last two weeks of development and time pressure made it difficult to arrange for people to participate. Live testing will be one of the first actions carried out when the system is in a more complete state and the responses will be used to make necessary improvements.

7.4 Future Development

The priority for the next stage of development is likely to be consolidation of existing functions, cleaning up the code base and generalisation of more query functions. The process of generalisation has already begun, and progress can be seen in the 'general models' file. Generalisation is going to be an ongoing auxiliary project throughout all future development and should eventually lead to smoother advancements when the library is in a more mature state.

Continuing, some parts of the code base are not in a perfect condition - there are a small number of functions in a disorderly state and a few known inefficiencies exist. This was largely the result of strict time pressures and a hurried development cycle. Improving these functions will be a priority in the near future, but some areas of the code base will need to be fully redesigned to allow additional features to be added.

In general, the user interface should be improved in the next development cycle. The current UI is not in a bad state, but some quality of life improvements could be implemented with only a small amount of additional work. The user tools in particular are not as pleasant in appearance as they could be.

In terms of future development, custom maps and a global timeline will be the top priority in the next stage - the reasons for this are discussed above. World Events are a whole group of requirements that had to be delayed until a later cycle and rely on a global timeline to be implemented first. As well as this, additional admin tools should be designed to allow indirect database access. Those tools should help in combating any potentially malevolent parties.

Beyond this, additional user to user interaction should be a priority. The system began to organically evolve from a map sharing platform to more of a social network type arrangement and it may be worth considering developing more heavily into this area in the future. Currently, no collaborative map creation social networking platforms exist despite a significant number of potential users who would likely be interested in such a system. If this were a route to go down, vast amounts of time and energy will need to go into protecting users from malicious agents - sharing protocols would need to be tightened significantly, anti-bullying processes created, and ethics approval sought.

7.5 Final Conclusion

Considering the original definition outlined in Chapter 2, a GIS application is "A system designed to capture, store, manipulate, analyse, manage, and present geographical data" (University of Wisconsin, 2018), it is fair to say that the final system falls into that definition nicely.

Many of the original design features were fully implemented and the system is well on the way to fulfilling many the rest. Despite this, there is still a long way to go before the system is in a state for general release and a sizable amount of work still needs to be done to reach that position. With a little more work and time this project should grow into a great tool for students looking to learn and casual map creators with an interest in history.

Perhaps the most important result of this process was the significant amount of experience gained in working with GIS systems and full stack development in general. Experience that is certain to come in use in the future.

References

Agarwal .S (2017), 'Analysing the performance of NoSQL vs. SQL databases for Spatial and Aggregate Queries' in 'Free and Open conference for Geospatial Conference'

URL: <https://scholarworks.umass.edu/cgi/viewcontent.cgi?article=1028&context=foss4g> pp.8 - 9

Alexsoft (2018), 'The Good and the Bad of Node.js Web App Development',

URL: <https://www.altexsoft.com/blog/engineering/the-good-and-the-bad-of-node-js-web-app-development/>
[Accessed 14 03 2019]

Alsudani. F. and Casey. M. (2009), 'The effect of Aesthetics on Web Credibility', pp. 1,

URL: https://www.bcs.org/upload/pdf/ewic_hci09_paper66.pdf [Accessed 25 02 2019]

Brent (2017), 'Asynchronous PHP',

URL: <https://stitcher.io/blog/asynchronous-php> [Accessed 13 02 2019]

Code Combat (2013), 'Code Combat',

URL: <https://codecombat.com/> [Accessed 20 04 2019]

Comparitech (2018), 'Cyber bullying facts and statistics for 2016 – 2018',

URL: <https://www.comparitech.com/internet-providers/cyberbullying-statistics/> [Accessed 25 03 2019]

Creative Bloq (2013), 'The top 5 JavaScript Templating Languages',

URL: <https://www.creativebloq.com/web-design/templating-engines-9134396> [Accessed 13 02 2019]

Debian Benchmarks (2018), 'Node.js vs Python 3 fastest programs',

URL: <https://benchmarksgame-team.pages.debian.net/benchmarksgame/faster/node-python3.html>
[Accessed 20 03 2019]

El-Kassas, W and Abdullah B, (2017), 'Taxonomy of Cross-Platform Mobile Applications Development Approaches',

URL: <https://www.sciencedirect.com/science/article/pii/S2090447915001276> [Accessed 17 03 2019]

Esri Institute (2012). 'What is GIS',

URL: <https://www.esri.com/~media/Files/Pdfs/library/bestpractices/what-is-gis.pdf> [Accessed 15 02 2019]

General Data Protection Regulation (2016), 'Recital 38 EU GDPR',

URL: <http://www.privacy-regulation.eu/en/recital-38-GDPR.htm> [Accessed 17 03 2019]

Google (2005), 'Google Maps',

URL: <https://www.google.com/maps>

How to Geek (2018), 'How to Convert a Windows Desktop App to a Universal Windows App',

URL: <https://www.howtogeek.com/250041/how-to-convert-a-windows-desktop-app-to-a-universal-windows-app/> [Accessed 14 04 2019]

IBM (2010), 'Integrating Software Assurance into the Software Development Life Cycle (SDLC)' in 'JISTP - Volume 3, Issue 6', pp. 51.

URL:

https://www.researchgate.net/publication/255965523_Integrating_Software_Assurance_into_the_Software_Development_Life_Cycle_SDL_C [Accessed 12 04 2019]

JavaScript Documentation (n.d.), 'The Date Constructor',

URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Date

Johansson E. (2012), 'Lord of the Rings Project',

URL: <http://lotrproject.com/> [Accessed 15 02 2019]

Kumar S. M. and Prabhu J. (2017), 'Comparison of NoSQL Databases and Traditional Database – An emphatic analysis'

URL: <http://joiv.org/index.php/joiv/article/view/58> [Accessed 20 03 2019]

Li, Yishan (2013), 'A Performance Comparison of SQL and NoSQL databases' in 'Communications, Computers and Signal Processing (PACRIM), 2013 IEEE Pacific Rim Conference'. pp.17

Live Universal Awareness Map (2014), 'Liveuamap',

URL: <https://liveuamap.com/> [Accessed 05 02 2019]

Masnun (2017), 'Async Python: The Different Forms of Concurrency',

URL: <http://masnun.rocks/2016/10/06/async-python-the-different-forms-of-concurrency/> [Accessed 20 04 2019]

Mapbox Documentation (2018), 'How Mapbox works',

URL: <https://docs.mapbox.com/help/how-mapbox-works/> [Accessed 20 03 2019]

McClure (2013), 'Parsing BC dates with JavaScript',

URL: <https://scholarslab.lib.virginia.edu/blog/parsing-bc-dates-with-javascript/> [Accessed 29 03 2019]

NASA (2019), 'GISS Surface Temperature Analysis (GISTEMP v3)',

URL: <https://data.giss.nasa.gov/gistemp/> [Accessed 25 04 2019]

National Library of Scotland (n.d.), 'NLS',

URL: <https://maps.nls.uk/> [Accessed 13 02 2019]

NetSparker (n.d.), 'SQL Injection Cheat Sheet',

URL: <https://www.netsparker.com/blog/web-security/sql-injection-cheat-sheet/> [Accessed 27 04 2019]

Node Documentation (2017), 'Don't block the Event Loop (or the Worker Pool)',

URL: <https://nodejs.org/es/docs/guides/dont-block-the-event-loop/> [Accessed 10 03 2019]

npm Trends (2019), 'leaflet vs mapbox.js vs openlayers',

URL: <https://www.npmtrends.com/leaflet-vs-mapbox.js-vs-openlayers> [Accessed 08 03 2019]

- Open Layers (2006), 'Open Layers',
URL: <https://openlayers.org/> [Accessed 16 03 2019]
- Open Street Map Community (2004), 'Open Street Map',
URL: <https://www.openstreetmap.org/#map=6/54.910/-3.432> [Accessed 03 03 2019]
- Oxford University Press (2011), 'My Maths',
URL: <https://www.mymaths.co.uk/> [Accessed 21 02 2019]
- Palmer. S. R (2001), 'A Practical Guide to Feature-Driven Development',
URL: <https://dl.acm.org/citation.cfm?id=600044> [Accessed 27 03 2019]
- PostgreSQL Documentation 1 (n.d.), 'Data Type Formatting Functions',
URL: <https://www.postgresql.org/docs/8.2/functions-formatting.html> [Accessed 20 03 2019]
- PostgreSQL Documentation 2 (n.d.), 'Date / Time Types',
URL: <https://www.postgresql.org/docs/9.1/functions-formatting.html> [Accessed 20 03 2019]
- Rascia. T (2017), 'Understanding Date and Time in JavaScript',
URL: <https://www.digitalocean.com/community/tutorials/understanding-date-and-time-in-javascript>
 [Accessed 20 03 2019]
- Rosales-Morales (2019), 'A New Approach for Developing Automatic Cross-Platform Mobile Applications Using Image Processing Techniques', Abstract,
URL: <https://academic.oup.com/comjnl/advance-article-abstract/doi/10.1093/comjnl/bxz029/5476716?redirectedFrom=fulltext> [Accessed 21 04 2019]
- Sheffield University (n.d.), 'The University Procedure in practice',
URL: <https://www.sheffield.ac.uk/rs/ethicsandintegrity/ethicspolicy/approvalprocedure/proceduralelements>
 [Accessed 20 04 2019]
- TecSoft (2019), '14 Best NodeJS Frameworks for Developers in 2019',
URL: <https://www.tecmint.com/best-nodejs-frameworks-for-developers/> [Accessed 12 03 2019]
- USC Dornsife, (2018), 'Predicting and Managing Chaos: How GIS Has Transformed Natural Disaster Relief',
URL: <https://gis.usc.edu/blog/predicting-and-managing-chaos-how-gis-transformed-natural-disaster-relief/>
 [Accessed 20 02 2019]
- Xhafa, S (2015), 'Geographic Information Systems (GIS) in Urban Planning',
URL: http://journals.euser.org/files/articles/ejis_jan_apr_15/Sonila_Xhafa.pdf [Accessed 05 02 2019]
- Ying - Chiang, C (2013), 'Vulnerability Assessment of IPv6 Websites to SQL Injection and Other Application Level Attacks',
URL: https://www.researchgate.net/publication/260396729_Vulnerability_Assessment_of_IPv6_Websites_to_SQL_Injection_and_Other_Application_Level_Attacks [Accessed 25 02 2019]

Appendix

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

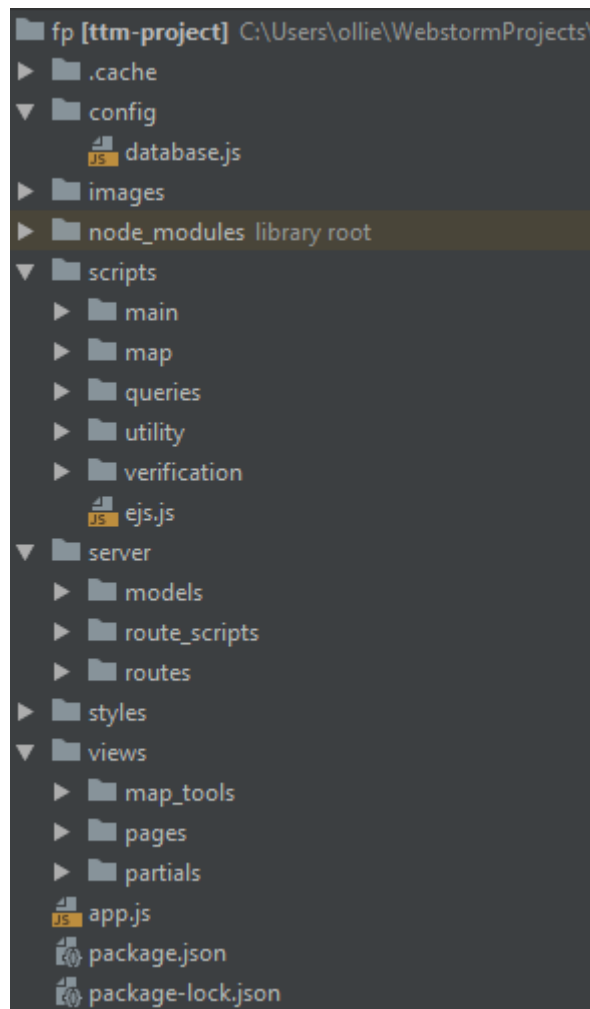
Identifier	Requirement	Priority	Exp. Difficulty
------------	-------------	----------	-----------------

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

Project Structure



Unique String Generation

```
gen_unique: function() {  
  let unique = "";  
  let possible = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz";  
  for (let i = 0; i < 70; i++)  
    unique += possible.charAt(Math.floor(Math.random() * possible.length));  
  return unique;  
},
```