KINGSTON UNIVERSITY

Faculty of Science, Engineering and Computing

29th APRIL 2019

BSc DEGREE *IN* COMPUTER SCIENCE

Team BUILDER

Volunteer Event Organizer Project

Name: **KLEVIS ZENELI**

K-Number: **K1651547**

Supervisor: **PROF. GRAEME JONES**

Project Type: **Build**

Kingston University London

Declaration

I have read and understood the University regulations on plagiarism and I understand the meaning of the word *plagiarism*. I declare that this report is entirely my own work. Any other sources are duly acknowledged and referenced according to the requirements of the School of Computer Science and Mathematics. All verbatim citations are indicated by double quotation marks ("…"). Neither in part nor in its entirety have I made use of another student's work and pretended that it is my own. I have not asked anybody to contribute to this project in the form of code, text or drawings. I did not allow and will not allow anyone to copy my work with the intention of presenting it as their own work.

Date: 29/04/2019

Signature :

Acknowledgement

I would firstly like to thank Professor Graeme Jones for his continued support, guidance and supervision throughout this project. He was a remarkable client and an outstanding supervisor, and it was a pleasure to work alongside him.

Table of Contents

Declaration **2**
Acknowledgement **3**
Table of Contents **4**
Abstract **6**
1. Introduction **7**
1.1 Background and Problem 7
1.2 Proposed Solution 8
1.3 Aims and S.M.A.R.T Objectives 9
1.4 Report Structure Overview 10
2. Literature Review **11**
3. Requirements Analysis **12**
3.1 Introduction 12
3.2 SWOT Analysis 12
3.3 Stakeholder Analysis 13
3.4 Stakeholder Power/Interest Grid 15
3.5 User Stories 16
3.6 Use Cases 16
3.7 MoSCoW 20
3.8 Project Management 22
3.9 Conclusion 23
4. Design **24**
4.1 Introduction 24
4.2 Data Modelling 24
4.3 UX and Interface design 28
4.4 Process Modelling 40
5. Implementation **42**
5.1 Introduction 42
5.2 Technologies Used 42
5.3 Architecture 44
5.4 Implementing the Database 45

5.4 Developing the frontend GUI 49
5.5 Security Measures Taken and Potential Concerns 60
6. Validation and Evaluation 62
6.1 introduction 62
6.2 Database Validation 62
6.3 Requirements Testing 64
7. Critical Review and Conclusion 65
7.1 Introduction 65
7.2 Achievements 65
7.3 Future work 65
7.4 Conclusion 66
Bibliography 67
APPENDIX A – User Stories 68
APPENDIX B – Functional Requirements 69
APPENDIX C – Schedule 71
APPENDIX D – Data Dictionary 75
APPENDIX E – Wireframes 76

Abstract

The age of digitalization is already upon us and has been for a great number of decades now, ever since the start of the digital revolution in the 1950's to 1970's. Despite this however, many event companies and organisations still operate using outdated methods, systems and technological infrastructures. This is especially true for scout event organisations that operate on a relatively smaller scale as they tend to still rely on paper-based systems to keep track of and allocate shifts to their volunteers. These organisations should have the ability to handle their events and activities, manage their volunteers, interact with their scout and parent user base and allocate teams to events accordingly without having to deal with all of the drawbacks that come with relying on an inefficient paper-based system. This project intends to address all of these issues through the creation of an efficient, user-friendly web application that will allow sea scout event organisations to manage their volunteers, create new events, create teams based on certain qualification requirements, send out invites to selected scouts and ultimately increase the interactivity from their user base. The following report will not only describe the process of creating such a solution but also outline all of the necessary decisions that had to be made along with any and all viable alternatives that could have been used.

1. Introduction

1.1 Background and Problem

Introduction

This project is about creating an intuitive system that will allow sea scout event organizations to manage and assign volunteers, based on their qualifications, to current ongoing events. The client is Graeme A. Jones, a Sea Scout volunteer and a qualified Royal Yachting Association (RYA) instructor for dinghy sailing. The organisation that he currently volunteers for is known as Ajax Sea Scouts and is situated right at the heart of Thames Ditton.

Background

As a certified RYA training centre, Ajax Sea Scouts pride themselves on being able to provide "fun, exciting and adventurous activities" to hundreds of young people through their immersive weekly events. The individuals that are responsible for creating these events are called Event Organisers (E.O.) and they consist of volunteers that have been specifically selected to take on this extra responsibility. During the creation of a new event, an E.O. will assign a volunteer to take on the role of Duty officer (D.O.) for that particular event. Afterwards, the E.O. assigns a team of volunteers to help out as part of that event and once that has been completed, scouts who match that events minimum qualification requirements are sent invites. The events themselves are created in such a way that they cater to scouts of varying ages and are typically run by multiple instructors and overseen by the appointed D.O. on-site. Attendees can expect to develop and refine their skills, under Instructor supervision, based on a set of specific learning outcomes that are assessed and converted into either a badge, an award or an RYA and British Canoe Union qualification.

The Problem

The current system in use is of a paper-based nature and has become extremely outdated due to a variety of different reasons. One of which is its susceptibility to human errors, particularly during the assessment stage, whereby misspelt words such as scout names could go unnoticed and cause confusion with little to no countermeasures in place. Similarly, its inability to automatically prevent duplicate data entries will inevitably lead to redundant data scattered across multiple documents. This becomes even more apparent when attempting to search for specific records as not only is the process itself tedious and inefficient, but the instructor might come across several similar entries and find it difficult to discern the original copy.

Resolving any of these issues requires a significant portion of the instructor's time which could be better spent on other duties as opposed to this trade-off. The system is also riddled with a plethora of security issues as it provides little to no protection against the unauthorised access of privileged information and any onlooker can read the information transcribed onto the sheet. Additionally, the documentation and paperwork generated through the use of such a system need to be stored at a physical location either in-house or offsite which directly ties up valuable real estate and results in additional monetary costs.

Furthermore, the process of backing up the data on such a system would prove to be incredibly labour intensive and will undoubtedly require a large amount of manpower in order to copy all of the contents over to a new piece of paper, potentially resulting in Ajax Sea Scouts having to suspend any water-based activity events whilst they undergo this daunting process.

1.2 Proposed Solution

This project intends to create a new system called Teambuilder that will completely replace the old one by converting it into a fully digital, modern web application that is accessible by all members from any device; provided that it has an active internet connection. It will be powered by a computerised back end database in order to help alleviate some of the real estate issues that Ajax Sea Scouts is currently facing with their paper-based system.

Validation rules, database check constraints and labels will be placed throughout various parts of the application, wherever appropriate, in order to guide the user through any decisions that they may encounter when using the application along with ensuring that the user input matches the required data type that the system is expecting to receive.

The issue of unauthorized access will be addressed by creating an account type for each of the different user groups and restricting its privilege rights and functions accordingly through user sessions. Account types will also have a bespoke dashboard that consists of helpful widgets and panels tailored to their needs and requirements.

Volunteers will be provided with the ability to interactively update their availability over the coming weeks and to respond to any event shifts that they have been invited to by either accepting or declining. On their dashboard will reside a clear and concise calendar that will display a schedule overview of all the accepted event shifts for that particular volunteer.

Volunteers with the E.O. status will be able to: create new events, edit their own events, create new D.O. from existing volunteers and create new event activities. They can then invite volunteers based on the role that they are looking for and monitor the invited volunteer's responses to see who has accepted, declined or yet to respond. This ability is also extended to inviting scouts, in bulk, based on the minimum selected qualification.

Scouts can view all the events that they have been invited to and choose to either accept or decline the invitation.

Some features will be accessible by all of the different account types and they are listed below as follows:

- Browsing all of the events
- Searching all of the events between a user defined start and end date
- Viewing the details of an event
- Viewing the day and role that a particular volunteer in a confirmed team for a confirmed event is scheduled to work.

Furthermore, a large focus will be placed on the User Experience (UX) and flexibility of the system so as to ensure that all of the various user types are able to access the information efficiently and reliably at all times.

In summary, new functionality that was not previously possible on the old system will be implemented alongside a handful of new quality-of-life features, all the while ensuring that the core functionality and previous data of the old system remain intact.

1.3 Aims and S.M.A.R.T Objectives

The main goal of this project is to create a responsive, efficient system for scout event organisations that prioritises user friendliness and flexibility as its core principles. Allowing them to host, manage and regulate their events and volunteers while also increasing the interactivity of their scout and parent user base.

Aims

- ✓ Flexible back end database design with a responsive font end user interface (UI)
- ✓ Allow Event Organisers to host, manage and regulate their events and volunteers

SMART Objectives

- ✓ Create a responsive, user friendly interface through a mixture of frameworks and custom ccs rules that can accommodate a majority of devices.
- ✓ Tooltips, labels and alert modals should be present and provide useful information to help guide or notify the user within 2 seconds of an action occurring.
- ✓ Develop bespoke dashboards for each of the user groups.
- ✓ Provide E.O. with forms that will enable them to create events, invite scouts and assign volunteers particular shifts to their created events.
- ✓ Allow E.O. to view the invite status of both scouts and volunteers.
- ✓ Split E.O. functionality from standard volunteer functionality.

1.4 Report Structure Overview

Chapter 1 – Introduction

The Introduction chapter will focus primarily on addressing who the client is, the background of the organisation and the problems that they are facing. It will also describe the proposed solution followed by a list of aims and objectives that will be met throughout the project's lifespan.

Chapter 2 – Literature Review

For the Literature Review chapter, multiple pieces of literature that relate to my chosen topic of "The different uses of NoSQL vs SQL in a database driven web application" will be reviewed and compiled into a final report along with my personal thoughts and findings.

Chapter 3 – Requirements Analysis

This chapter is designed to capture all of the necessary information needed in order to successfully achieve the required functionality of the system along with meeting all of the aims and objectives mentioned above. The wants of the different user groups and how they each interact with the system will also be explained within this chapter as they are vital to creating a dependable, fit for purpose system.

Chapter 4 – Design

The design chapter focuses on all of the decisions that have been made throughout the project along with any alternatives that were considered. It includes both front end and back end design considerations from Wireframes to Entity Relationship Diagrams and Data Dictionaries.

Chapter 5 – Implementation

This chapter will narrate all of the steps that were taken in order to create the desired system along with detailed explanations of the technologies and application programming interfaces (API) that were used. It will also further elaborate on the architectural design decisions that were initially depicted within the design chapter.

Chapter 6 – Testing and Evaluation

The Testing and Evaluation chapter presents all of the various usability and performance tests in a table like structure and concludes with a critical evaluation based on the results of analysing the undertaken tests.

Chapter 7 – Critical Review

Finally, in the Critical Review chapter, the created system functionality will be compared to the initially proposed aims and objectives. From this, we can gauge the extent to which the project has successfully satisfied the client's requirements and the wants and needs of its users. A final conclusion is then drawn depicting the strengths and limitations of the created solution along with a list of the features that were not implemented followed by any proposals for future work.

2. Literature Review

The Structure Query Language (SQL) is the most common used database language. It has played a very important role in understanding, developing and handling structured data where there are relations between different variables of the data.

D.Chamberlin and R.Boyce's first attempted to design a relational database language in (1970)[3], however it was not until 1973 after they moved to San Jose Research Laboratory when they began work on SEQUEL. This language has made without a doubt major contribution to worldwide data management. In addition, E. F. Codd of IBM Research (1970) published a paper entitled “A Relational Model of Data for Large Shared Data Banks,” which resulted to a new way for computers to manage data. “There are usually many alternative ways in which a relational model may be established for a data bank”[2].

However, using relational database is difficult, especially in cases where dealing with a large volume of data. This has become apparent in recent times where applications are not suitable to process the user’s personal information, their geographic location data as well as users of the web and mobile data. This need has resulted in the development of other systems such as the NoSQL database.

The NoSQL is a non-relational database that stores data as transaction logs rather than as tables. It is faster and more compact when dealing with very large databases and supports unstructured or irregularly structured data. The abbreviation NoSQL was first used in 1998 by Carlo Strozzi.

Both SQL and NoSQL databases are vulnerable to injection attack. Year after year we witness major data breaches that could have a devastating impact on businesses and governmental organisations, as it can lead to exposure of all of the sensitive information stored on their system.

3. Requirements Analysis

3.1 Introduction

During this phase of the project, an initial investigation of Ajax Sea Scouts is conducted and then followed up by a series of individual meetings with the client in an attempt to define a distinct set of goals for the new system. Relevant information such as the end users of the product, the desired functionality and the day to day operations of the existing system are elicited from the client and noted accordingly. This process is further aided through the use of different tools and technologies that ensure all of the necessary information has been successfully gathered. It is imperative that at this stage all ambiguities have been resolved with the client and that the needs of the different user groups and stakeholders have been sufficiently captured and differentiated from one another.

3.2 SWOT Analysis

Strengths

- ✓ Access to a multitude of different resources
- ✓ Prior experience with full stacked development and projects of a similar nature.
- ✓ Extensive knowledge of the current systems users.
- ✓ An existing user base that is willing to provide feedback and is readily available.
- ✓ All users will have the ability to view all events.
- ✓ Event Organisers will be able to track their outgoing invites easily and conveniently.
- ✓ Ongoing events can be accessed, viewed and amended in real time reducing the time spent searching through paper-based records.
- ✓ Event Organisers will be able to filter Volunteers based on their qualifications.
- ✓ The system will be up and running for 24 hours of the day, increasing accessibility.
- ✓ Familiar with the programming languages needed to create this system.

Weaknesses

- × Although appropriate security measures will be put in place in the form of input filters and validation rules, the system is still susceptible to outside attacks.
- × The system has a large number of different functionalities whose completion might not be feasible within the given time frame.
- × The system might take a longer time to load when being accessed from a slower internet connection thus preventing potential users from being able to access the information.
- × The system might not be sufficiently optimised to accommodate all of the different sized smartphone displays.
- × Certain core functionalities are dependent on third party API's.

Opportunities

- ✓ New technologies that are not currently available might be released which the system could potentially adopt.
- ✓ The system can be adapted and used for non-water-based scout organisations.
- ✓ The system can be slightly modified and extended to incorporate other volunteer-based event-hosting organisations.

Threats

- × The current system might receive extensive updates, potentially placing it in direct competition with the new system.
- × The client might outright reject the new system and not wish to use it.
- × There is also the threat of hackers, viruses and malware infiltrating the system and compromising personal user information.

It is essential that the client is involved throughout the entirety of the new system's creation to ensure that the threats and weaknesses outlined in the SWOT analysis do not get the better of the project. Additionally, by adopting this approach, the developers and project managers can be sure that the new system is in line with the client's expectations and vision at all times. This will, in turn, reduce the likelihood of the client hiring a new company to take on the project or just outright rejecting the system upon completion.

Furthermore, the system will be created using current best coding practices and the latest technology standards in an attempt to protect against the majority of external Cross-Site Scripting (XSS) or SQL injection attacks. Should time prove to be an issue, a meeting will be set up with the client to discuss which of the functionalities, if any, can be scrapped so that the build can successfully meet its intended deadline.

Upon meticulously reviewing the advantages and disadvantages of both the current and proposed system, a conclusion was drawn to go ahead with the project. The deciding factor was the sheer number of advantages that the new system provides Ajax Sea Scouts when compared to what they are currently receiving.

3.3 Stakeholder Analysis

A stakeholder is an individual or group of individuals that have a vested interest in the business, either directly or indirectly. They play a pivotal role in an organisation’s decision-making process and can heavily influence their actions. Understanding who these stakeholders are and evaluating each of their strengths is vital to the success of this project as it allows the developer to design the system to their needs and requirements. To gather this information, a meeting was set up with the client in order to clearly define the internal and external actors of the system. After the meeting was over, the developer and project manager were very familiar with the different stakeholder groups; some of which even overlapped with the system's core user types.

INTERNAL

EVENT ORGANIZERS

These are off-site individuals that are in charge of assigning duty officers, creating events and setting the up days that they will run on. Their benefits include having the ability to freely create an event and assign it a start/end date and time along with being able to manage all of their own created events. They can also filter volunteers and scouts based on specific qualifications and invite them to any of their created events. When inviting volunteers, they have the added functionality of specifying a shift start/end time that the volunteer is then invited to.

DUTY OFFICERS

These are the most senior scouts on-site and are tasked with monitoring the ins and outs of the event that they've been assigned to first-hand. Their benefits include being able to view the other members of their team, view their assigned shifts, view a history of their previous invites and respond to any new invites by either accepting or declining.

VOLUNTEERS

This stakeholder group consists of unpaid volunteers who apply with the intention of helping to provide support for ongoing events. Their benefits include the ability to register as a volunteer user account type, select their available work day for the week, upload supporting document proof of their achieved qualifications, view a list of all the events, view a history of their previous invites and respond to any new invites by either accepting or declining.

SCOUTS

These are the actual scouts who register and join with the intention to view and participate in the events that they have a particular interest with. Their benefits include the ability to register as a scout account type, view a list of all events along with their start and end dates, declare their interest with a particular event, keep track of their acquired achievements, respond to event invites, view a history of all their previous invites and also link their accounts to their respective parent's accounts.

PARENTS

These are the parents of scouts who register in order to either participate in events as a volunteer or to monitor their child's activity. Their benefits include being able to link their account with their child's account, view all events, view all the events that their child is interested in and also apply to become a volunteer. If a parent successfully applies to become a volunteer, they will reap the benefits of both "parents" and "volunteers" rather than those that just come with being a volunteer.

EXTERNAL

ROYAL YACHTING ASSOCIATION (*RYA*)

Seeing as this project will be focused on water sports, qualification accreditor associations such as the RYA and the British Canoe Union (BCU) constitute as external stakeholders. This is because the organisation that my client volunteers for is a non-commercial RYA training centre whereby official RYA and BCU qualifications can be awarded to scouts.

THE SCOUT ASSOCIATION

Not unlike Ajax Sea Scouts, the Scout Association holds events of varying activity types for scouts of all ages in order to help them advance their knowledge and expand their current skillsets. These events can be analysed and categorised based on scout attendance and participation, the results of which could be used by Ajax Sea Scouts to discern the popular events from the unpopular ones.

OTHER EXTERNAL BODIES

These consist of organisations who have a vested interest in either the safety of children in sport or getting kids to participate in outdoor sports. In both cases, this project will be indirectly benefiting their cause as it increases the accessibility of such events to a wider demographic of potential scouts or interested children.

3.4 Stakeholder Power/Interest Grid

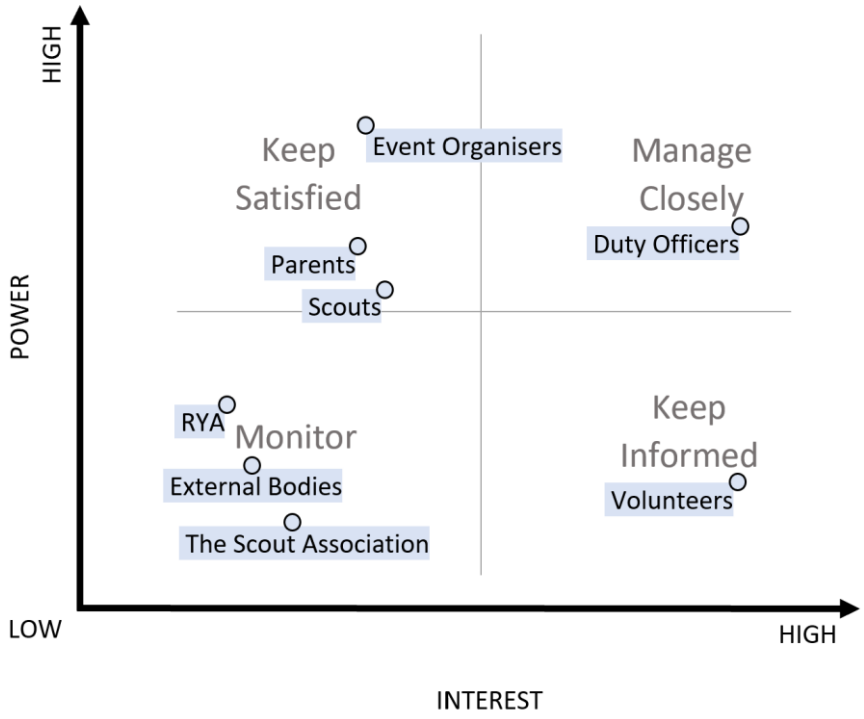


Figure 1 *– Stakeholder Power / Interest Grid*

The power/interest grid in figure 1 is compartmentalized into four bespoke categories: monitor, keep informed, manage closely and keep satisfied. These four categories represent the position of the different stakeholders along with their power and interest towards the project. This information is particularly useful when important business decisions have to be made as it provides the developer with a deeper insight into the influence that each stakeholder has over the project. However, as the project progresses through the different stages of its lifecycle, this power/interest grid will need to be modified and adapted to keep up with the stakeholder's influence so that its accuracy can be maintained.

3.5 User Stories

User stories are depictions of how the user is expected to behave when interacting with the created system and are predominantly used within agile projects. They are typically generated after the initial interviews have concluded and can be seen as end goals but expressed from a user's perspective. During this stage, it is necessary to further delve into the wants and needs of each user type and define who the core users of the system are and convert them into primary actors. A sample user story can be found below; however, for a full list of user stories please refer to *Appendix A*.

Scout

Jake is a 12-year-old sea scout who has attended multiple dinghy events hosted by Ajax Sea Scouts in the past. During his most recent event, Jake successfully met the required criteria and was awarded a level 1 qualification for dingy sailing. He now wants to attempt the level 2 qualification for dinghy sailing so he logs into Teambuilder and navigates to the events page to see if there are any "Dinghy – level 2" events scheduled in the near future. As he is looking through the events page, he finds a "kayaking – level 1" event that piques his interest. Jake decides to click on the event so that he can read obtain some additional information about the event. Whilst on the "more details" page, Jake sees that his favourite instructor is going to be teaching this course too. Jake decides that he wants to give it a try so he clicks the small star by the event in order to express his interest to that events E.O. and then continues his search for a "dinghy – level 2" event until the E.O. invites him to the "kayaking - level 1" event. Once Jake receives an invite to the "kayaking – level 1" event, he navigates to his invitations page in order to respond to the invite by either accepting or declining.

3.6 Use Cases

Use cases are generally regarded as a method of identifying and showcasing the overall system requirements. When used in tandem with use case diagrams , they become a powerful asset that allows the developer to focus on each of the user types individually rather than as a whole. This, in turn, allows for a detailed description of each user's interactions with the system and is particularly helpful in revealing any issues that might occur early on in the project.

Use Case Diagrams

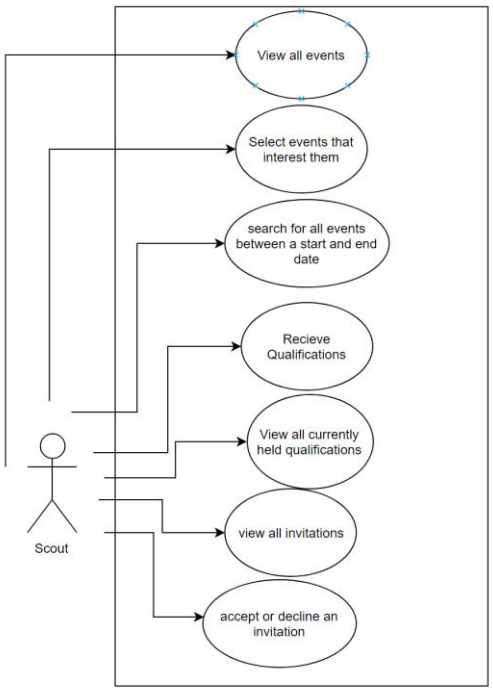


Figure 2 – *Scout Interaction with the system*

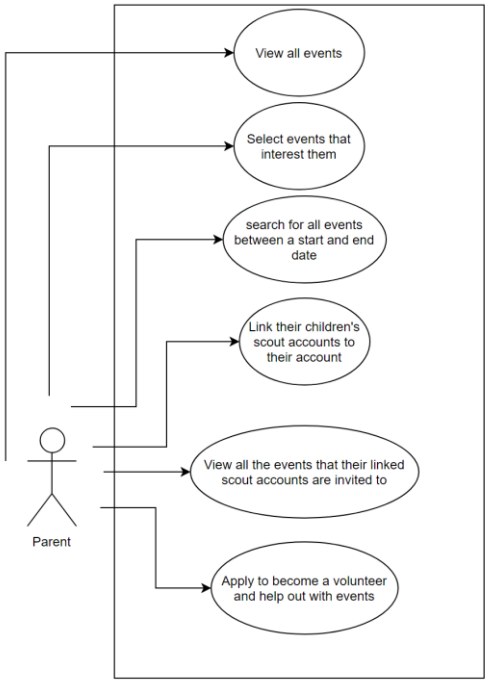


Figure 2.1 – *Parent Interaction with the system*

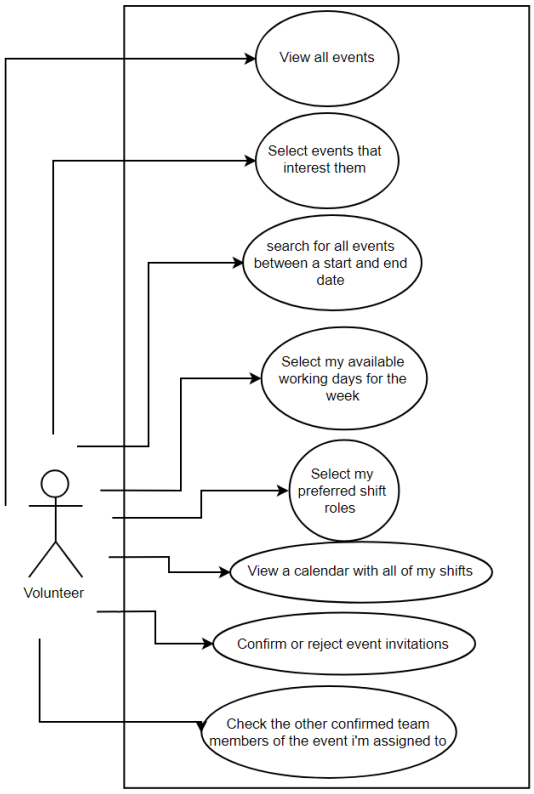


Figure 2.2 – *Volunteer Interaction with the system*

Use Case Descriptions

Use Case	1. Creating an event
Overview	The Event Organizer (EO) wants to create an event that has an activity assigned to it. He types in a name of the event, its start date/time and end date/time and selects an activity for it.
Primary Actor	Event Organizer (EO)
Trigger	Event Organizer decided a new event should be created
Precondition 1	The EO is logged in.

Basic flow	
1	EO navigates the menu and opens the “create new event form”
2	EO enters a name and description for the event
3	EO sets the start and end date and times for the event.
4	EO selects an activity to run during the event
5	EO sets the date and time for the duration the activity will run
Alternative Flow 4A	The activity that the EO wants to add is not available or doesn’t exist
4A1	EO selects the “add a new activity” option
4A2	EO enters a name and description for the activity
4A3	EO re-fills in the “create new event form” with the information.
4A4	EO selects an activity to run during the event
Termination outcome	Activity that wasn’t available is now added and attached to the event

Figure 3 *– Create an event Use Case Description*

3.7 MoSCoW

MoSCoW is a prioritisation technique that can be found within the DSDM agile framework and is primarily used to aid in organising the system requirements from most to least important. Since this project has a strict deadline that it must adhere to, it is inevitable that some of the functionality will be left unimplemented; therefore, this technique was used to help shift the focus onto the most important requirements so that the vital, core features of the system work appropriately before any of the additional quality-of-life features are touched upon. MoSCoW prioritisation breaks down the requirements into the following four sections:

- **MUST** have – The requirements that are essential to the core operations of the system and must be implemented in order for the project to be deemed a success.
- **SHOULD** have – An important requirement but one that is not necessarily vital to the project.
- **COULD** have – Additional desired functionality that is generally regarded as less important than the previous two sections.
- **WON'T** have – Functionality that will not be delivered as part of this project's time frame.

Figure 4, below, shows a select portion of the Functional Requirements of the system. For an extensive list, please refer to Appendix B.

Functional Requirements

KEY: ALL = All user groups, **E.O** = Event Organisers, **D.O** = Duty Officers,

ID	REQUIREMENT	LEVEL	USER GROUPS
1	Log into the system	MUST	ALL
2	Register for an account	MUST	ALL
3	Create event	MUST	E.O
4	Edit event	MUST	E.O
5	Create activity	MUST	E.O
6	Create duty officer	MUST	E.O
7	Assign duty officer	MUST	E.O
8	Invite volunteers	MUST	E.O
9	Filter volunteers	MUST	E.O
10	Filter scouts	MUST	E.O
11	Invite scouts	MUST	E.O

Figure 4 *– Functional Requirements*

Non-Functional Requirements

SECURITY

- ✓ Passwords are hashed before stored into the database using the bcrypt algorithm.
- ✓ User privileges are restricted based on the account type to prevent unauthorised access.
- ✓ The system must authenticate the user upon logging in and redirect them to their appropriate dashboard.
- ✓ The system must prevent anyone who is not logged in from accessing the information.

PERFORMANCE

- ✓ The database must be able to hold 7000 records, at minimum, without any issues.
- ✓ The system must be able to run seamlessly with 100 scouts and 100 volunteers using it simultaneously.
- ✓ There should be no more than 2 seconds of latency for the end user when they perform an operation on the system or add new entries to the database.

COMPATIBILITY

- ✓ The system must be able to run in the following browsers: Google Chrome, Firefox, Safari, Internet Explorer and Microsoft Edge.
- ✓ There should be a responsive UI design that adapts to screens of all sizes
- ✓ The validation rules in place must work and be applied at all times on any device

DESIGN

- ✓ There must be a clear UI design that persists across all pages of the system
- ✓ The colour scheme used must be readable under all lighting conditions

AVAILABILITY

- ✓ The system must be up and accessible 99% of the time during the year

USABILITY

- ✓ The system should have appropriate labels and messages to provide guidance to the user when performing specific tasks.

3.8 Project Management

Multiple different methodologies were considered for adoption during the planning stages of this project with the main two being the Waterfall and Agile approaches. The waterfall approach is a very linear one in the sense that it's broken up into specific, distinct stages; each of which must first be completed before the next can start. This becomes a major drawback against the waterfall approach as it essentially impedes the project from progressing until the stage before it has been completed. However, there are advantages to this model with one being that it simple and easy to understand.

On the other hand, the Agile approach is an iterative type of Rapid Application Development (RAD) whereby deliverables are "time-boxed" and grouped into "sprints". This allows for much more flexibility and communication with the client throughout the project as deliverables can be reviewed by both the team and the client upon completion along with reprioritised based on any changes the client might come up with later on in the project. Alongside this, agile also has many available frameworks, such as scrum, which could also be adopted providing further benefits to the project. However, agile does have its negatives too with one in particular being that the progress of the project is difficult to track since sections are completed in increments and therefore you are required to look across sprints and piece them together so that a full picture of "project progress" can be generated. [6] (Segue Technologies, 2018).

After a careful analysis of the waterfall model and the agile approach, a decision to go with an agile approach, which adopts some of the scrum principles, was made. The reasoning for this came down to it being the most appropriate fit for the project due to its flexible and iterative nature; allowing for the segmentation of work into deliverables that are completed separately, tested and re-joined once again at a later stage. Having this flexibility will help ensure that the project is completed successfully.

Major Risks and Contingencies

Risk	Likelihood	Impact	Mitigation
Project might not be completed within the timeframe set	Medium	High	Schedule regular meet ups with my client and identify which features can be dropped or amended so that project deadline can be met
The dummy data that will be used might not be close enough to the real data that the system is intended for and thus could affect the client's ability to validate the finished system.	Low	High	Obtain samples of previous data captured by the old system from the client and crosscheck it against the dummy data to ensure that it is in line with what the client usually holds.

Unable to sufficiently use the programming languages required	Low	High	Ask supervisors that specialise in those specific languages for assistance alongside participating in online courses to accrue the missing skills

Figure 5 *– Major Risks and Contingencies*

Project Schedule

A complete project schedule with an accompanying Gantt chart has also been created so as to ensure that the deliverables are handed in on time. This can be found in Appendix C.

3.9 Conclusion

To summarise this phase, all of the different client's needs were collected through various interviews and have been successfully converted into a set of distinct functional requirements.

Additionally, the potential risks of the project have been appropriately identified and are each accompanied by a solution that has been specifically crafted to mitigate the dangers that issue presents to the project as a whole. As a result, the project is able to progress at a steady pace with minimal downtime or delays, throughout its development life cycle.

The chosen methodology for the project was also defined in this phase and a decision was ultimately made to go with Agile Software Development because of its flexibility and iterative nature. Despite this however, it is important to understand that as a direct result of incorporating such a methodology, the client's requirements are subject to change as the project goes on.

4. Design

4.1 Introduction

Before the implementation of the system can begin, certain design decisions need to be made so that the developer can visualise what the final product's appearance and functionality will look like. This visualisation is created through utilising a combination of different tools and techniques and applying them to the requirements gathered during the Requirements Analysis stage.

The design chapter makes use of these techniques and considerations whilst also expanding upon the principles and ideas outlined in the previous chapters. This process of converting the requirements into actual system functionality will be broken down and showcased within the following sections: Data Modelling, Process Modelling and UX and Interface design.

Furthermore, particular importance will be placed on the front-end UX aspect of the system so as to ensure that the information displayed within the developed system is easily accessible by its primary user base.

4.2 Data Modelling

Initial Entity Relationship Diagram (ERD)

An initial ERD was created based on the requirements derived from the client and demonstrates the overall structure and architecture of the database in the form of tables, attributes and relationships. This gives the user a clear overview of the overall database along with a providing a general idea of the information that will be captured and how that information links to/interacts with one another.

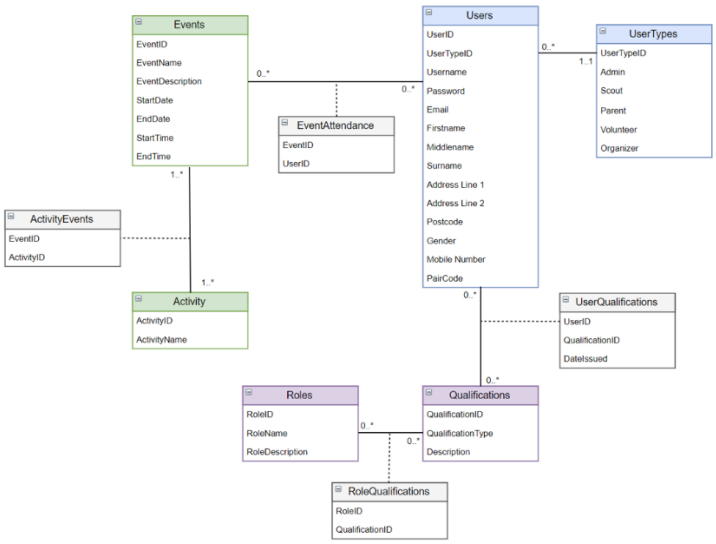


Figure 6 *– Initial Entity Relationship Diagram*

Figure 6 illustrates the initial ERD that was created and shown to the client in order to explain how the requirements will be captured and assess the extent to which it lines up with their expectations. Any and all feedback received from the client was then funnelled back into the ERD improving its complexity.

The ERD itself is comprised of 10 tables that have been normalised in 3rd normal form and represent the various pieces of information that are required in order to achieve the desired system functionality.

The initial idea was to have a user's table, aptly named "Users" , that would contain all of the log in information and personal details. This would be then followed by an additional table called "UserTypes" that would link to the Users table through a many to one relationship allowing each user in the Users table to have a particular account type associated with their account. The "Qualifications" and "Events" tables would then be linked to the "Users" table through many to many relationships so that a user can be part of multiple events and have multiple qualifications. However, modern database design principles prohibit us from leaving these relationships as many to many. As a result, these relationships had to be further broken down into constituent parts and are represented in the form of two additional tables named "EventAttendance" and "UserQualifications" respectively. Records in these two tables are uniquely identified by a combination of two keys, formally referred to as composite primary keys, and allow for multiple instances of the same user id to be mapped to different event and qualification ids.

During the presentation of the initial ERD to the client, newly desired functionality and workflow restrictions were established which had to be added to the system in the form of additional requirements. This request was relatively simple to accommodate because of the project's adopted agile methodology whereby revisiting and altering previous sections, after receiving feedback from the client, comes as a standard procedure.

Please find the revised Entity Relationship diagram, that has been amended based on the client's feedback, as *figure 7* below.

Revised Entity Relationship Diagram

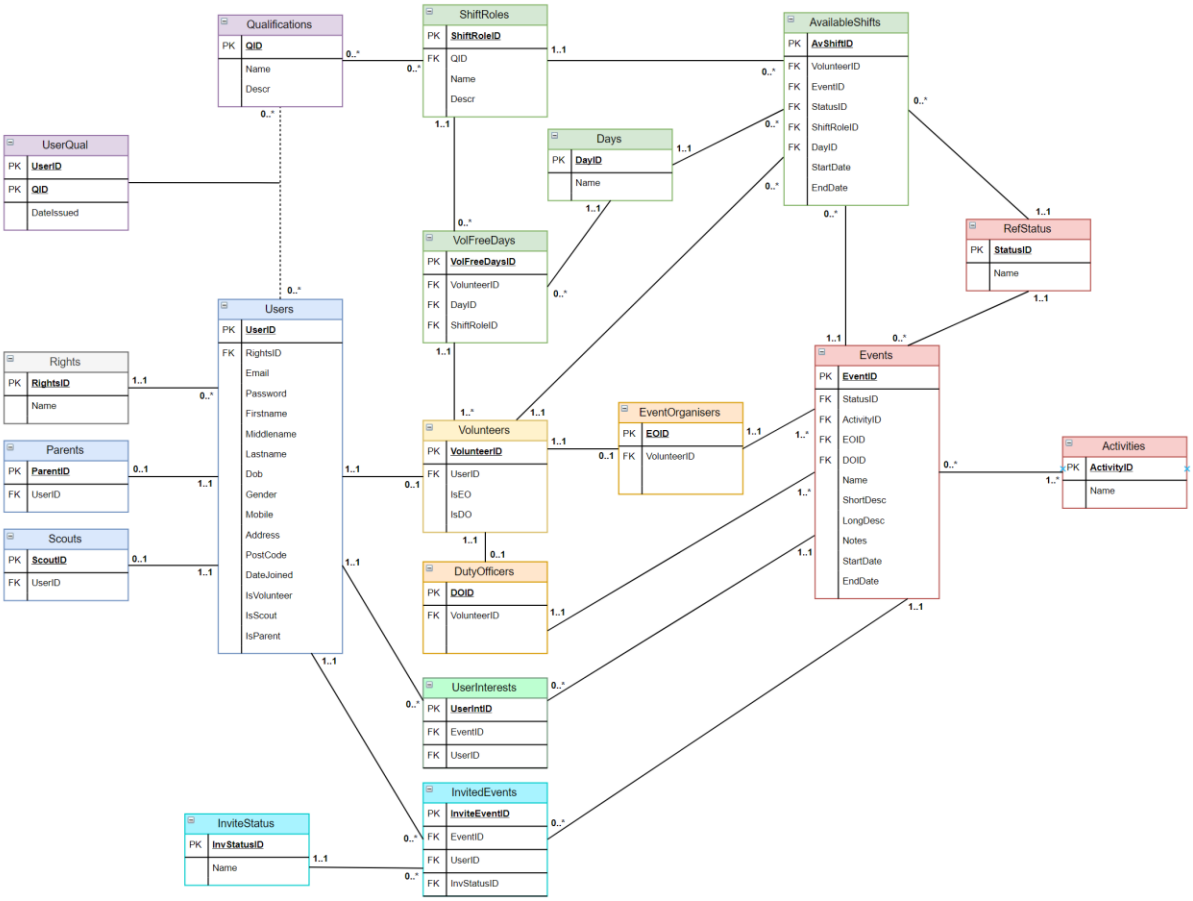


Figure 7 *– Revised Entity Relationship Diagram*

The revised ERD illustrated in *figure 7* has undergone multiple changes to accommodate for the additional workflow's and requirements requested by the client. It is a magnitude greater in terms of flexibly, functionality provided and overall complexity when compared to its predecessor in *figure 6*. An immediately noticeable difference is that the number of tables has almost doubled from the previous value of 10 and is now at 19. The core idea of how to separate the users into their respective types has also changed and is now done through what's known as subclasses.

USERS, ACCOUNT TYPES AND PRIVILEGE LEVELS

Every user of the system will have their information stored within the "Users" table and will have a unique primary key that can identify them along with their details. Afterwards, the attributes "isVolunteer", "isScout" and "isParent" will be filled in with a value of either 1 or 0 whereby 1 symbolises "yes" and 0 symbolises "no", respectively. Following this, a new record is then created within the subclass tables of either "Scouts", "Parents" or "Volunteers" using the unique UserID that was generated from the Users table. These subclass tables are linked to the "Users" table through a one-to-one relationship. Privilege levels on the other hand, are

managed slightly differently and consist of a Rights table that is linked to the Users table through a one-to-many relationship whereby a RightsID foreign key is declared within the RightsID attribute of the Users table. Each record in the Users table is required to have a value of either 1 or 2, which translates to 'Standard' or 'Admin' respectively, in their RightsID attribute.

USERS AND QUALIFICATIONS

Much like the initial ERD, the relationship between the Qualifications table and Users table remains unchanged as a many to many and similarly has to be broken into an additional table named "UserQual". Records in this newly created "UserQual" table are uniquely identified by a combination of two keys, formally referred to as composite primary keys, and allow for multiple instances of the same UserID to be mapped to different qualification ids called QID.

VOLUNTEERS, EVENT ORGANISERS AND DUTY OFFICERS

Once a UserID is added as a new record into the "Volunteers" table they can officially be considered a volunteer and the system will recognise them as such, provided that their "isVolunteer" attribute is also set to 1. At this stage a volunteer's "isEO" and "isDO" attributes are set to 0 and they can select the days in which they're free to work through the "VolFreeDays" table along with the additional ability of being invited to events through the "AvailableShifts" table. Should an existing E.O decide that they want to assign a volunteer as the D.O for one of their events then the volunteers "isDO" attribute will be updated from 0 to 1 and a new record will be placed for that "VolunteerID" inside of the "DutyOfficers" table. The relationship between the volunteers and both the duty officers or event organisers is a one-to-one relationship whereby "EventOrganisers" and "DutyOfficers" are subclasses of Volunteer.

Database Assumptions

The following assumptions were discovered and outlined during this phase:

- ✓ Scout accounts cannot become parent or volunteer accounts and vice-versa.
- ✓ Events cannot last for more than a week at a time.
- ✓ E.O must verify and approve a volunteer request to add a qualification
- ✓ Only an E.O or D.O may award scouts with qualifications
- ✓ D.O must be supplied at the creation stage of an event
- ✓ The volunteer cannot be assigned multiple shifts on the same day for the same event

Data Dictionary

This is an incredibly useful piece of documentation that outlines the overall structure of the database and illustrates the properties of each of the different attributes within a table such as its name, data type, length, and constraints.

A shortened version of the data dictionary table can be found below; however, to view the complete table please refer to *Appendix D*.

(Table Name) Relation Name	PK/FK	*(Field Name)* Attribute Name	Data Type & length	NOT NULL	Further Constraints
Users	PK	UserID	INT(5)	•	
	FK	RightsID	INT(1)	•	
		Email	VARCHAR(255)	•	UNQIUE
		Password	VARCHAR(64)	•	
		Firstname	VARCHAR(30)	•	
		MIddlename	VARCHAR(30)		
		Lastname	VARCHAR(30)	•	
		Dob	DATE		
		Gender	VARCHAR(6)	•	CHECK (Gender= 'Male' OR Gender= 'Female')
		Mobile	INT(11)		
		Address	VARCHAR(200)		
		Postcode	VARCHAR(8)	•	
		DateJoined	DATETIME		
		IsVolunteer	CHAR(1)	•	
		IsScout	CHAR(1)	•	
		IsParent	CHAR(1)	•	

Table 1 – *Data Dictionary*

As seen in *table 1*, Password is of type of VARCHAR and has a length of 64 because it will be hashed before stored into the database and email has a type of VARCHAR and a length of 255 to accommodate for all potential email types. Furthermore isVolunteer, isScout and isParent are of type CHAR(1) because their values will only ever be either a 0 or a 1 and therefore essentially acts as a Boolean. Attributes such as gender have additional check constraints placed on them to ensure that only the values of "Male" or "Female" are accepted however the majority of the systems constraints will be placed on the input fields of the teambuilder application through validation rules. This allows the user to be visually notified of any incorrect inputs whilst simultaneously reducing the workload put onto the server as the request will not be sent unless the validation rules put in place have been met.

4.3 UX and Interface design

Frameworks

There were a variety of different frameworks that were considered for adoption during the font-end design consideration stage and they are listed below as follows:

- Materialize
- Bootstrap 4
- Material Design Bootstrap (MDB)

The documentations and API's for each of these three responsive font-end frameworks were read extensively in order to determine which one would be the best fit for a system such as Teambuilder.

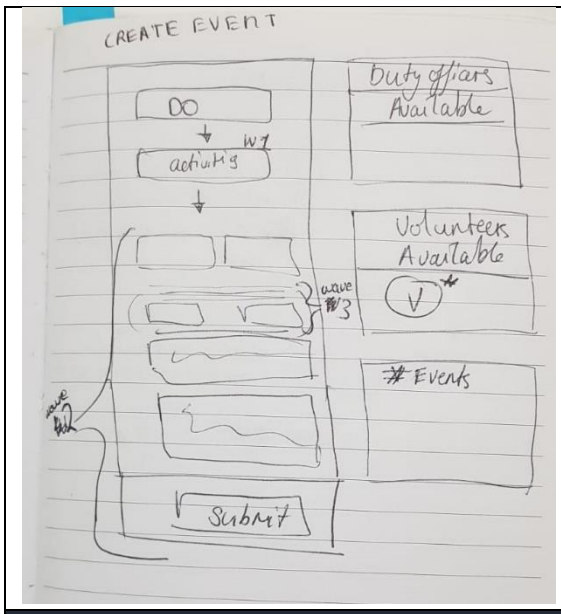
Initially, materialize seemed as though it would be the best fit as it is built based on googles material design principles; however, it is still a relatively new framework with a small team behind it so the features and functionality that it offered were very limited.

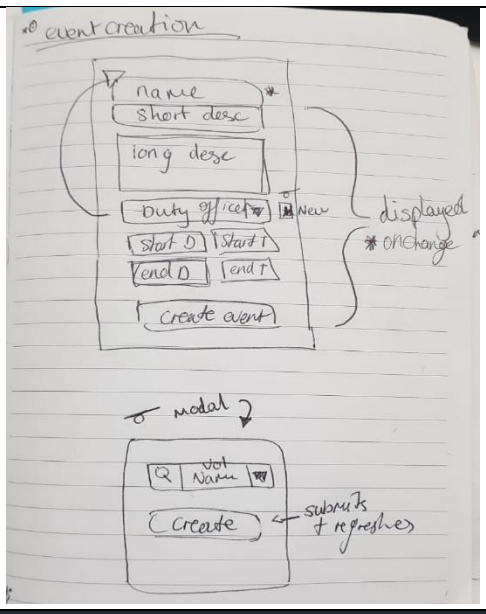
Bootstrap was then considered as it is a renowned framework that has been in circulation for 4 iterations and is used by thousands of developers worldwide and bests materialize in terms of the sheer amount of functionality it brings; however, this was still not the exact fit for the project.

Material Design Bootstrap on the other hand, combines the best parts of googles material design principles with fork of Bootstrap to create an incredible front-end framework that had the exact features Teambuilder was looking for. Therefore, a decision was made to go ahead with MDB as the chosen front-end framework of choice.

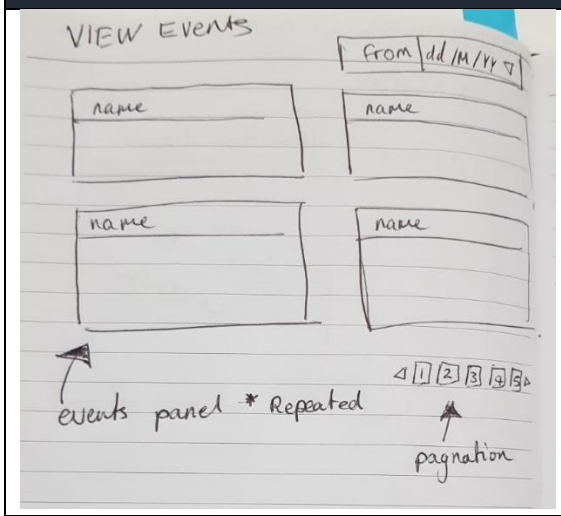
Low Fidelity - Sketches

Sketches were used to create a rough visual overview of how the layout of the menu navigation and different dashboards will look like. They were generated together with the client during interview sessions and meetings as they were quick to create and easy to amend making it a perfect way to bounce ideas back and forth. set up the foundation for what will eventually become the applications layout. Below are a few examples of low fidelity sketches, the rest can be found in Appendix E.

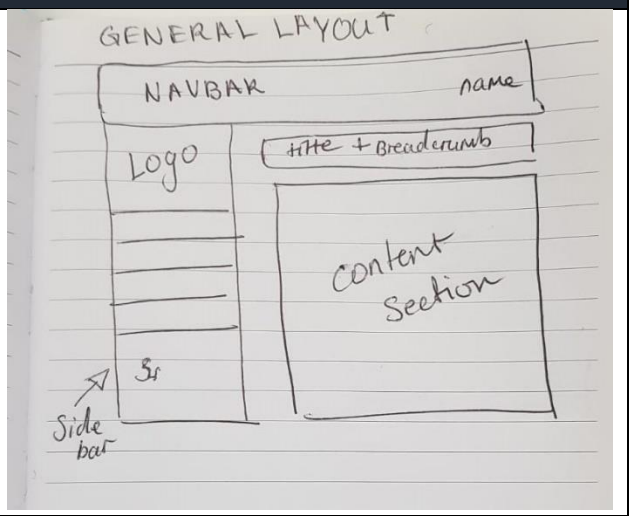




Create an Event



VIEW ALL EVENTS



GENERAL DASHBOARD LAYOUT

Table 2 – *Low Fidelity Sketches*

Medium Fidelity - Wireframes

After carefully reviewing the sketches , a select few were converted into more refined, medium fidelity wireframes using a free online software called InVision. This was primarily done so that additional details could be added, and the content could be dynamically changed in real time to allow for further experimentation of potential layout structures. These can be found below.

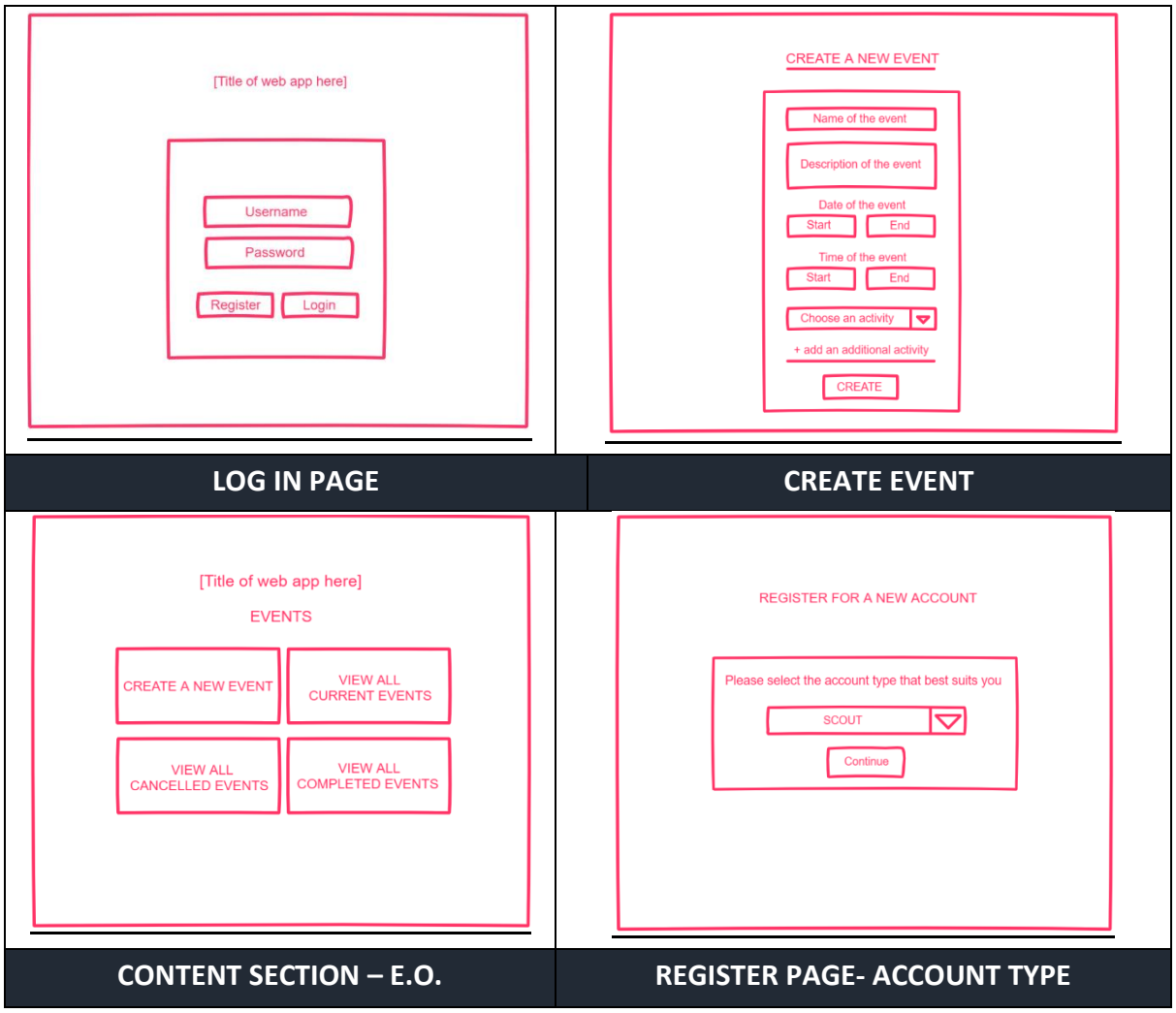


Table 3 – *Medium Fidelity Wireframes*

As can be seen from the medium fidelity wireframes in *table 3*, the layout of some of the pages previously seen in the sketches section have been altered. This was intentionally done so that multiple renditions of slightly different layouts could be created and then later assessed, together with the client, in order to find out which structures would best fit the requirements of the system and would be the most user friendly. Sections prefaced with "CONTENT SECTION" such as Content Section – E.O. refer to the position that it will be placed inside of and is actually referencing the content section panel inside of the "GENERAL DASHBOARD LAYOUT" sketch from *table 2.*

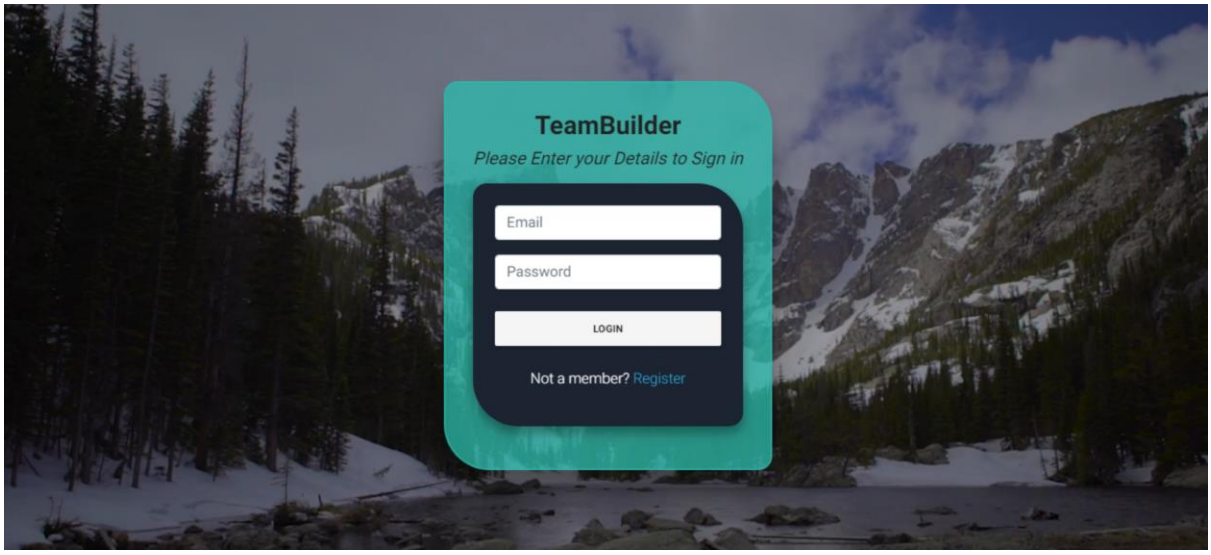
High Fidelity – Mock-ups

Once all of the low fidelity sketches and medium fidelity wireframes have been assessed and the desired layout has been defined, a series of HTML mock-ups were created to serve as the backbone of the application whereby dynamically generated content, from PHP database queries, is displayed upon.

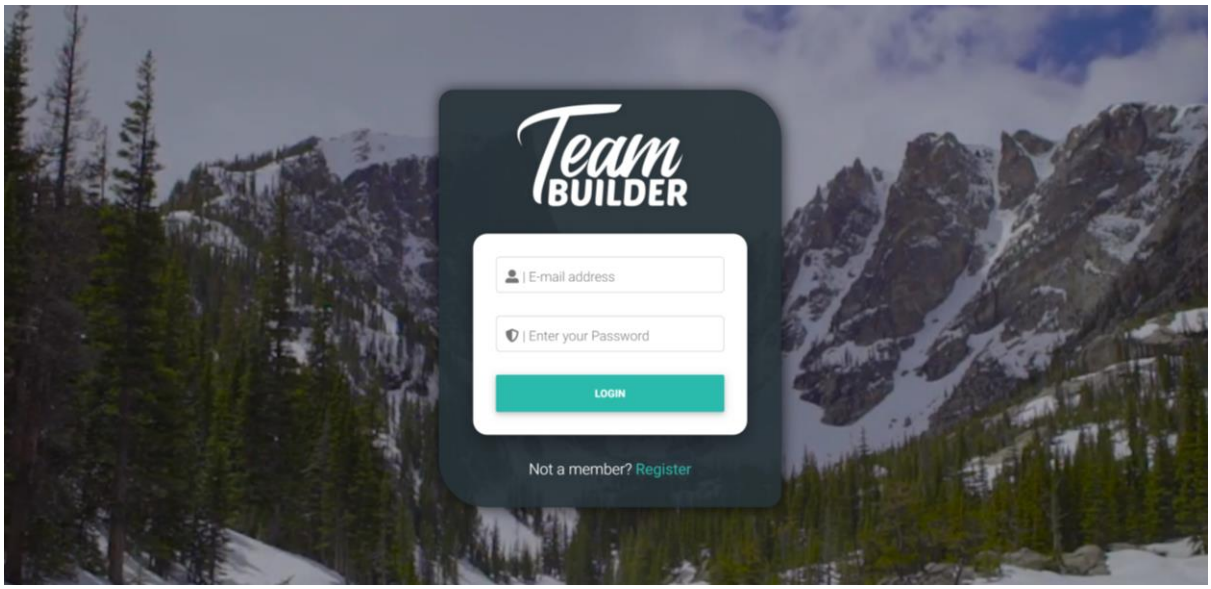
LOGIN PAGE

The mock-ups themselves are not always perfect first time and they were also slightly changed and improved upon after further consultations with the client as is depicted below:

Initial mock-up:



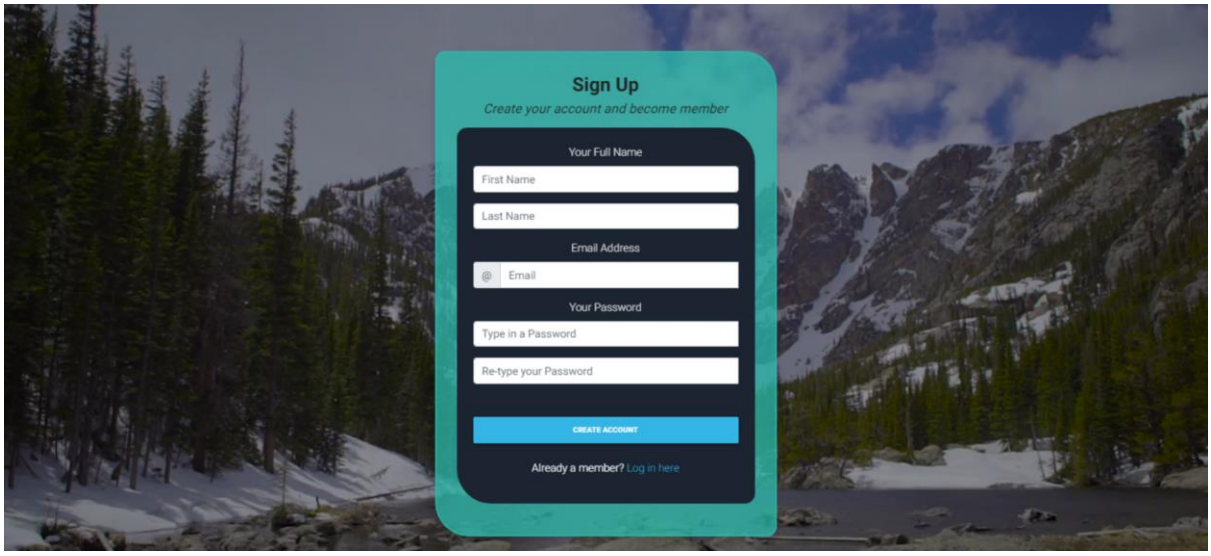
Updated mock-up:



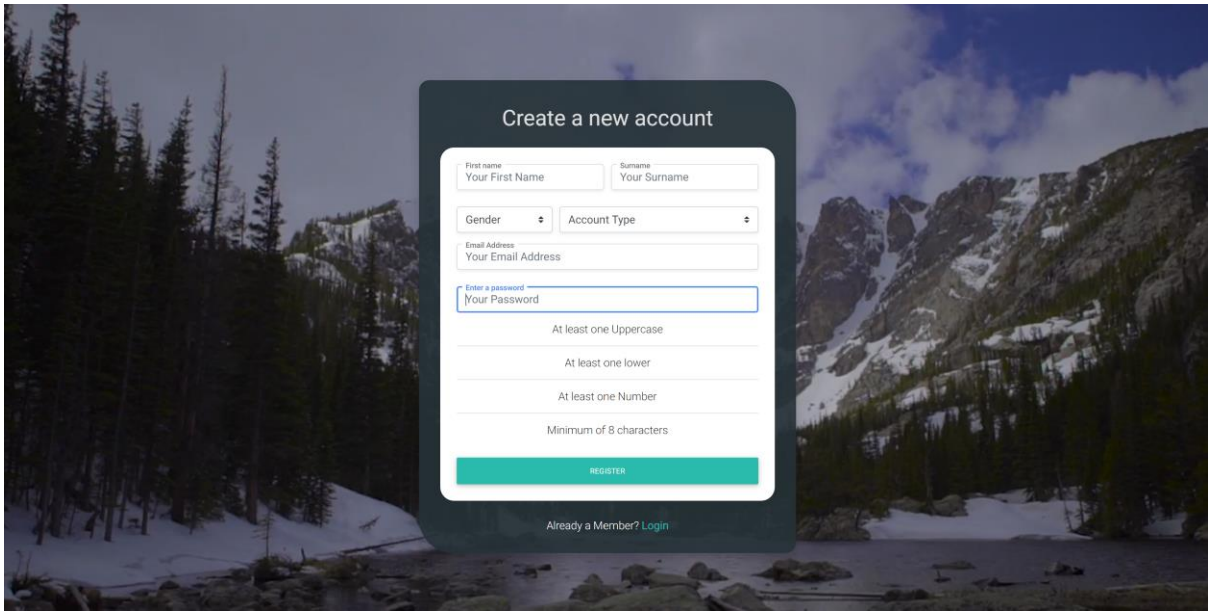
Some of the changes include: the colour scheme, the padding and roundness of the borders, the positioning of the input fields and the replacement of the logo.

REGISTER PAGE

Initial mock-up:

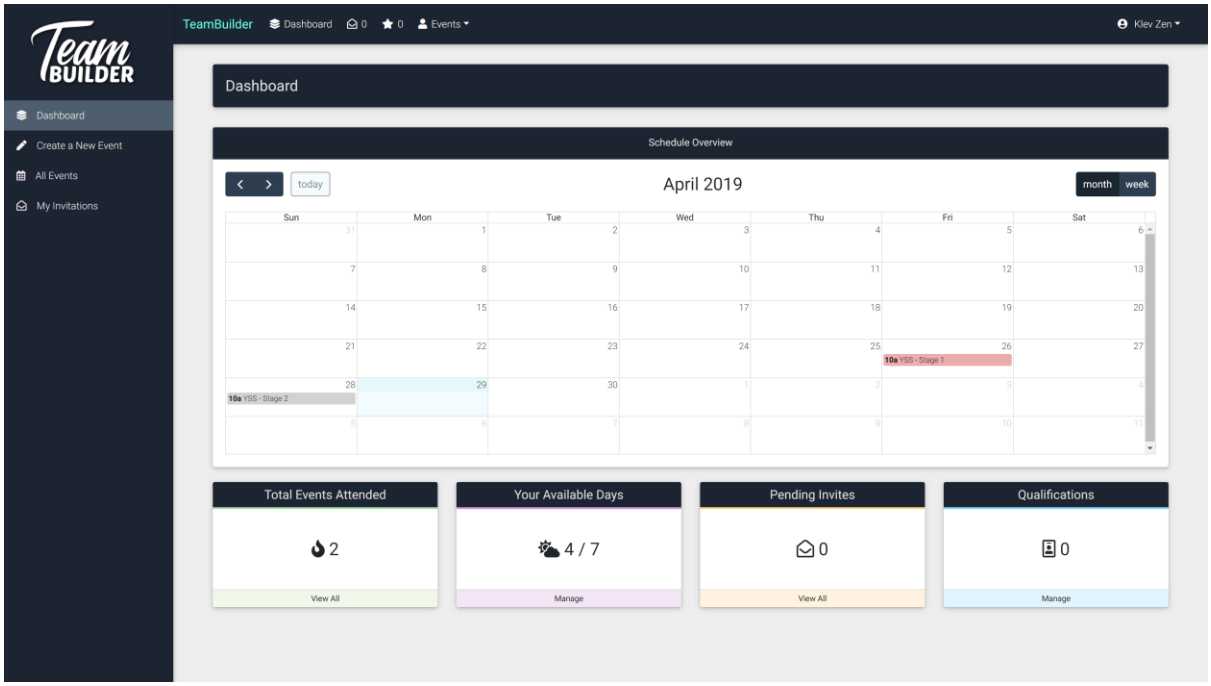


Updated mock-up:



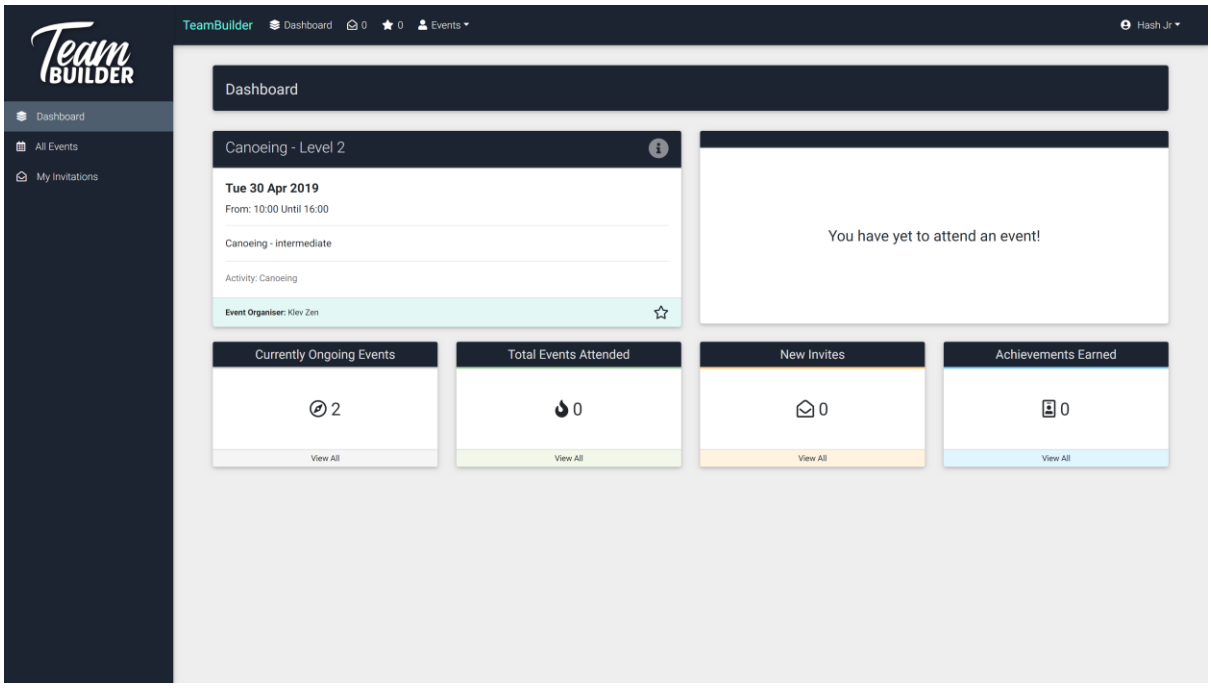
Some of the changes include: the colour scheme, the padding and roundness of the borders, the positioning of the input fields and the password advice panel that expands when the password input field in is focus.

DASHBOARD – WITH VOLUNTEER VIEW (E.O Status)

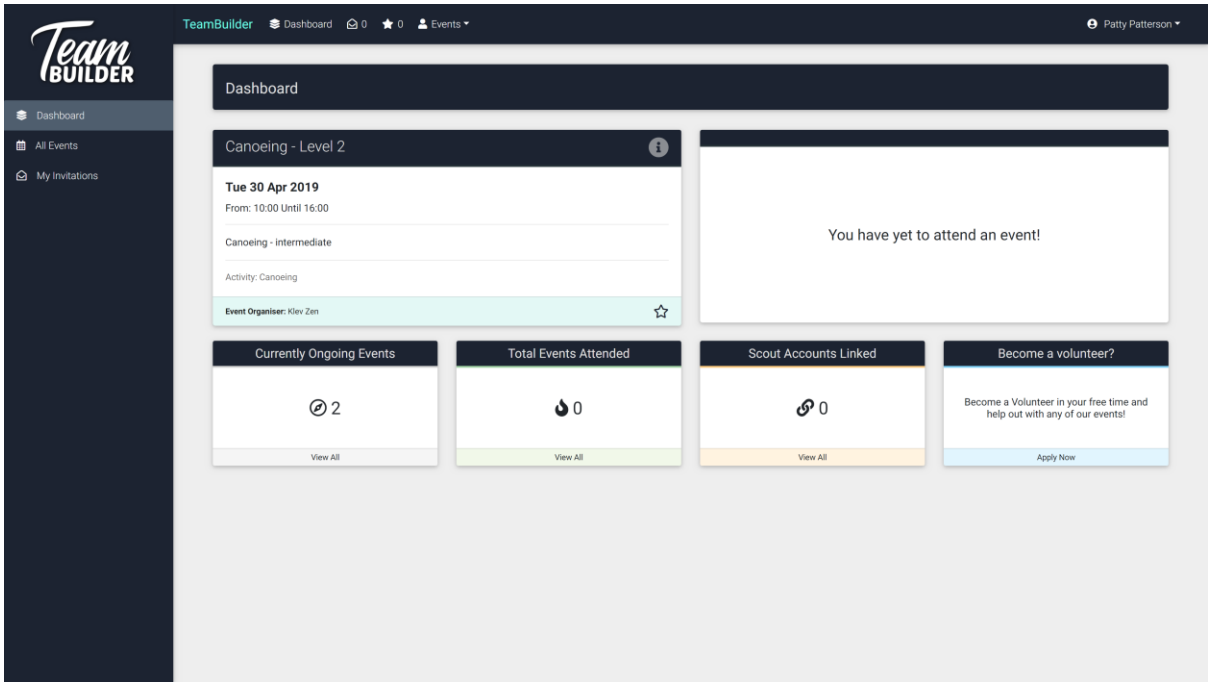


It follows the "general dashboard layout" structure depicted in the preliminary sketches coupled with a variety of helpful panels and a calendar created using the fullcalendar.io API. This particular screenshot is taken from a volunteer account that is also an event organiser; however, a standard volunteer account will see the exact same layout barring the "create a new event" menu navigation links.

DASHBOARD – WITH SCOUT VIEW

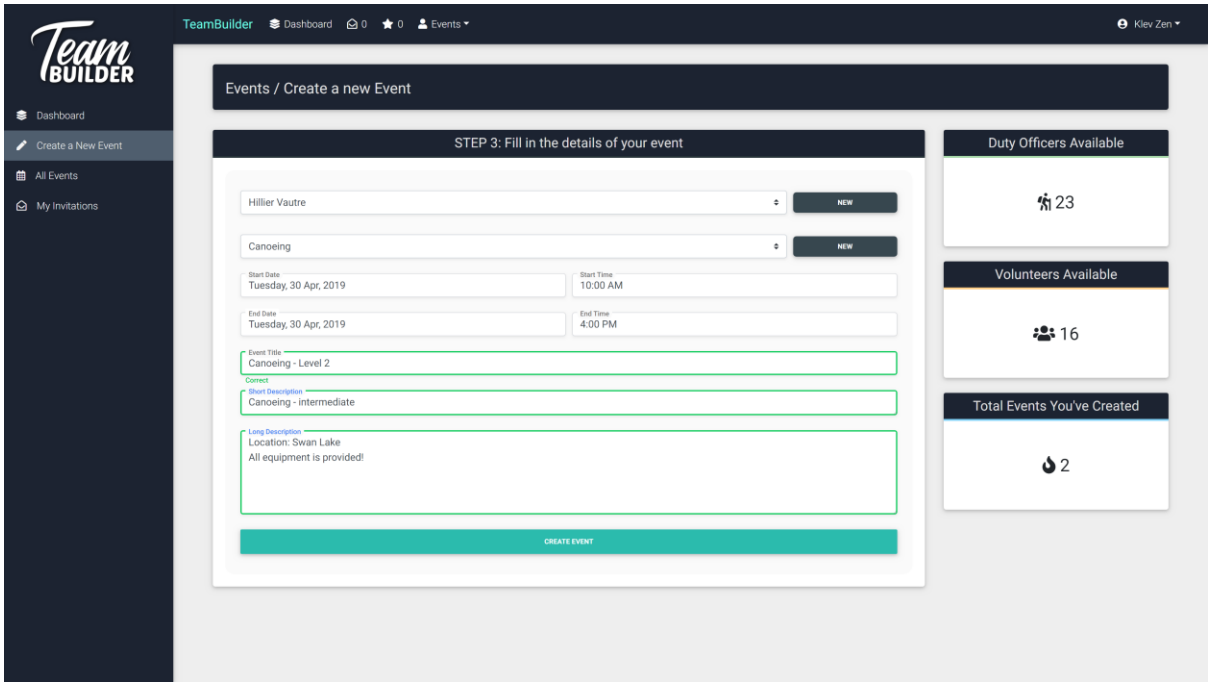


DASHBOARD – WITH PARENT VIEW



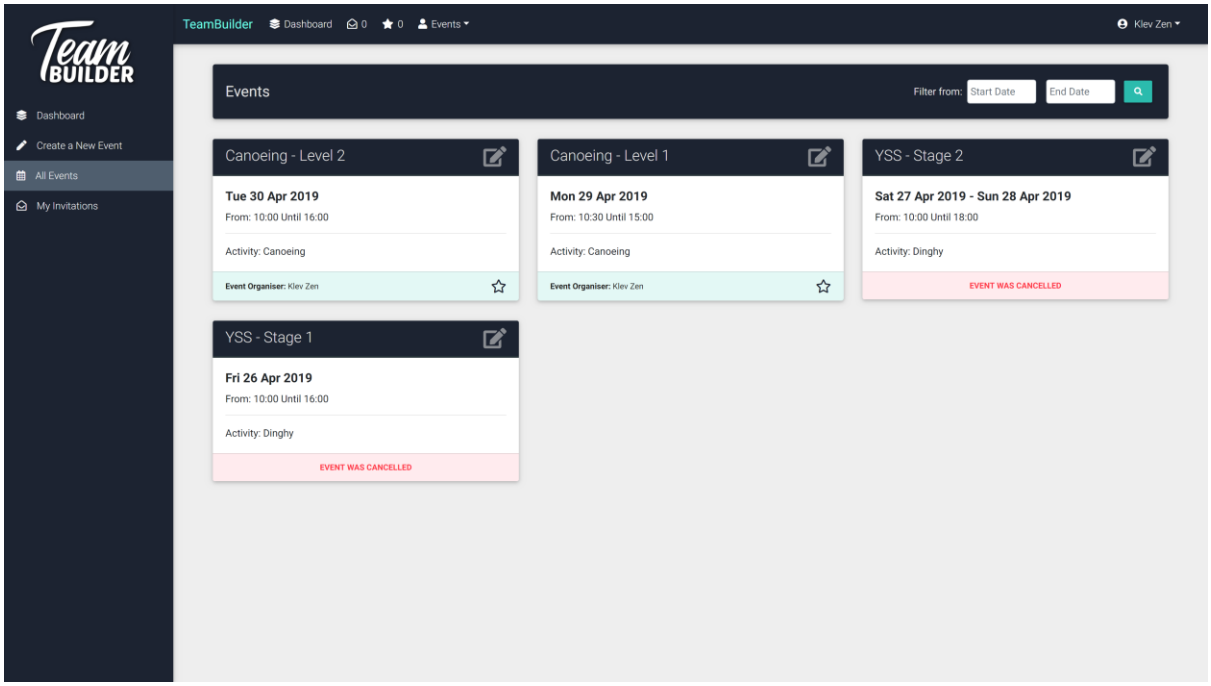
This is similar to the scout view apart from the panels where a parent account can apply to become a volunteer and they can also link their child's scout account to their account and by doing so unlock additional functionality such as view all of the events that their child has been invited to.

CREATE A NEW EVENT



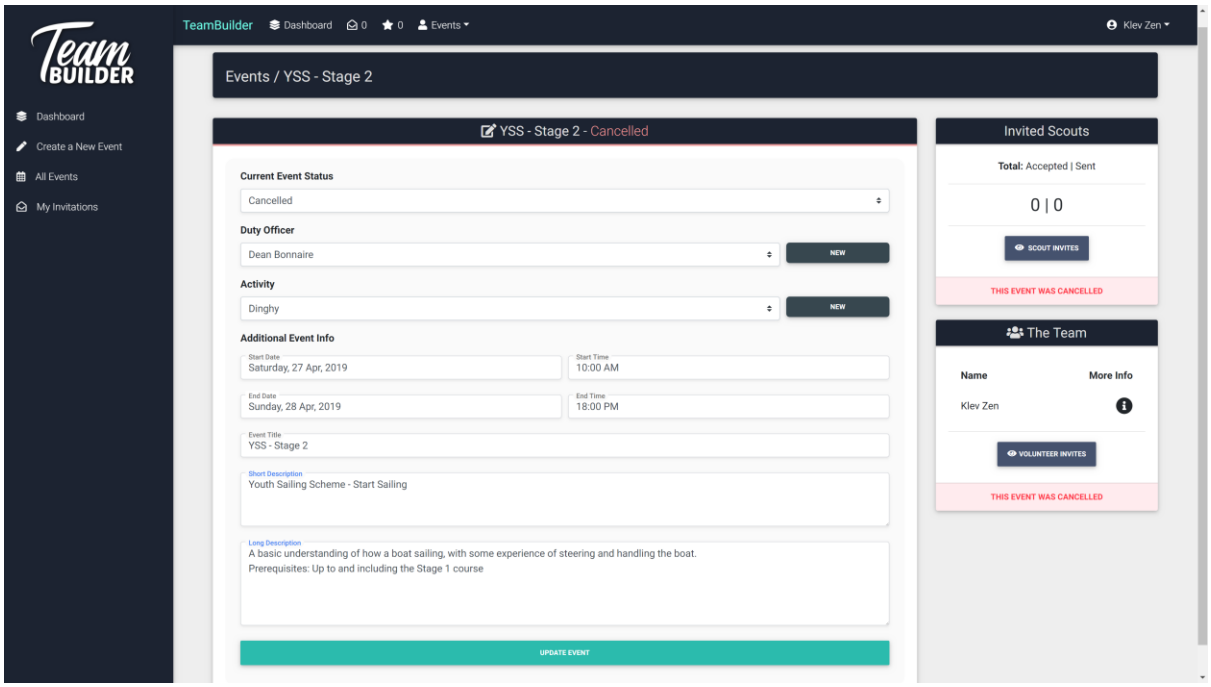
All of the input forms have the appropriate validation rules applied to them along with a date picker and time picker for the date and time input fields respectively. The API pickadate.io was used to enable the date and time picking functionality.

VIEW ALL EVENTS

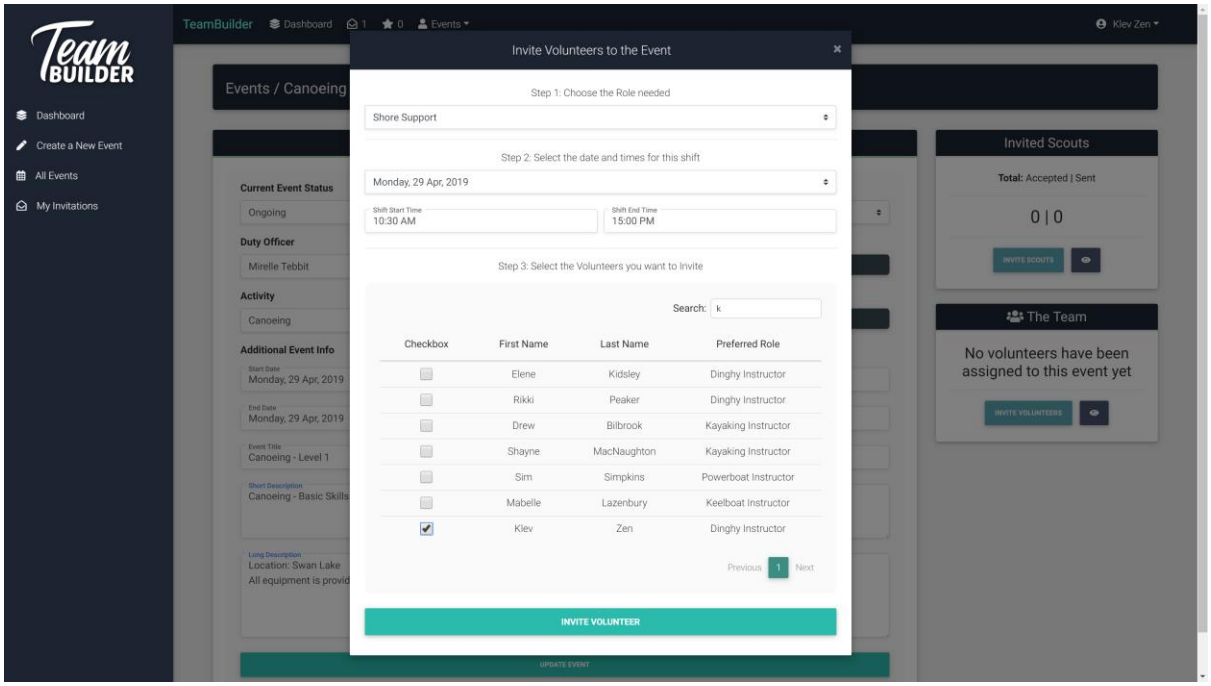


This page is the same for all user types with the only difference being that the edit icon is displayed over the events instead of the information icon and that is because the currently logged in account is of E.O status and those events have been created by him and are therefore linked to his account. Should an additional event be created by another E.O rather than “Klev Zen” then the information icon will be displayed for that particular event instead of the edit icon as Klev Zen would not be the E.O for that particular event.

EDIT AN EVENT (As the E.O of that event)

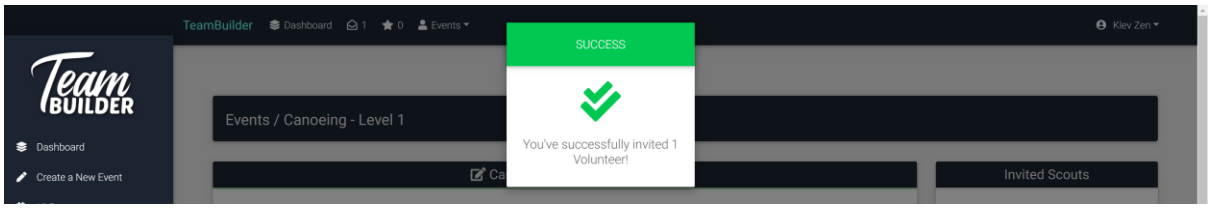


INVITE A VOLUNTEER (As the E.O for that Event)



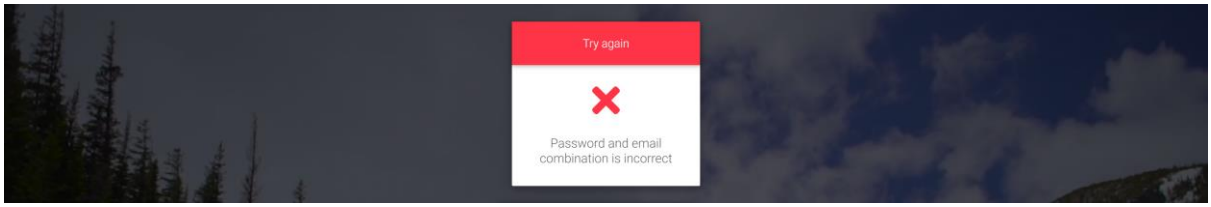
This is a modal that is triggered by clicking the “invite volunteers” button at the bottom of “the team” panel. Once you progress through all the appropriate steps, a table with a list of volunteers is displayed. This table is from created using the Datatable JavaScript API.

SUCCESS MODAL



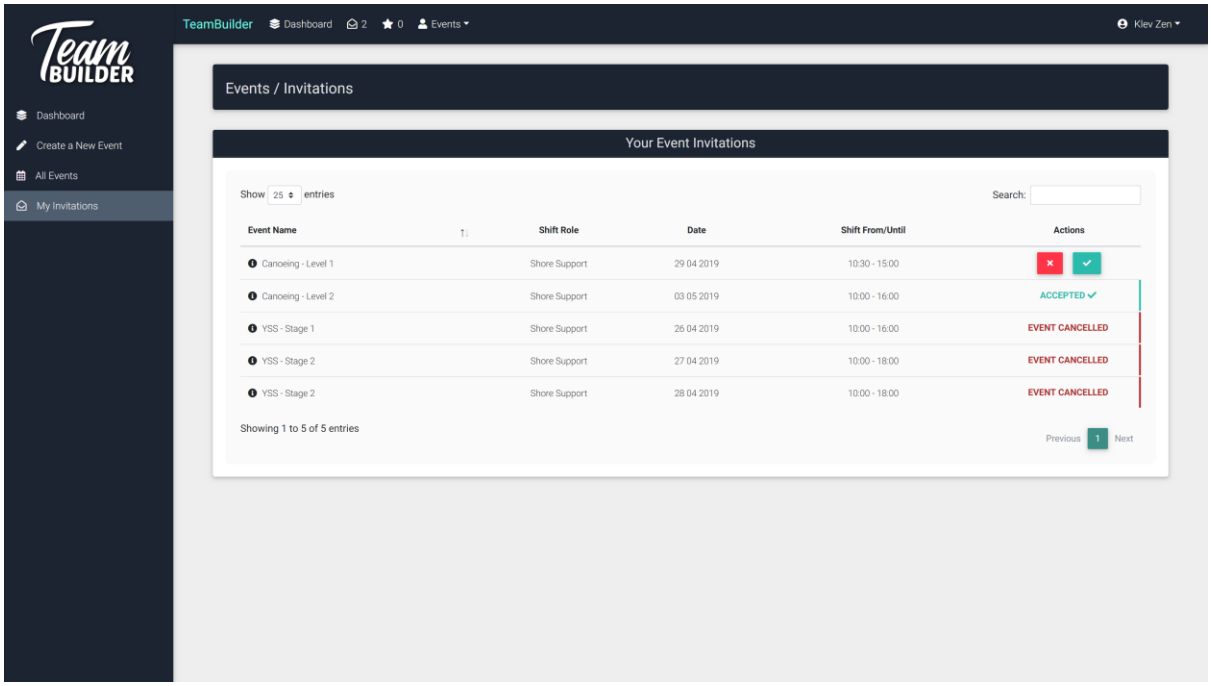
This is an example of how a success modal will appear to the user. The inner text is changeable.

ERROR MODAL



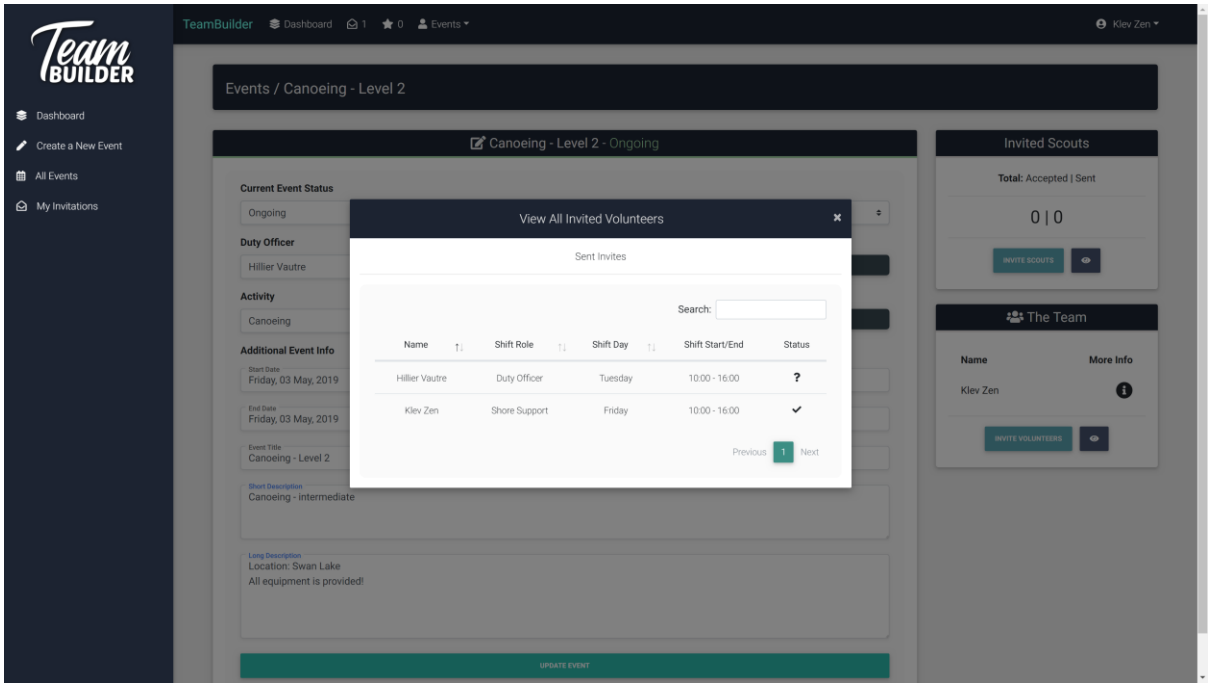
This is an example of how an error modal will appear to the user. The inner text is changeable.

INVITATIONS (E.O status)



Showcases all the events that the user has been invited to and provides them with the ability to either accept or decline any invites that are still pending a response provided that the event is still ongoing and not cancelled. This particular screenshot is taken from a volunteer account that is also an event organiser; however, every account will see the exact same layout barring the “create a new event” menu navigation links.

VIEW INVITED VOLUNTEERS (As the E.O for that Event)

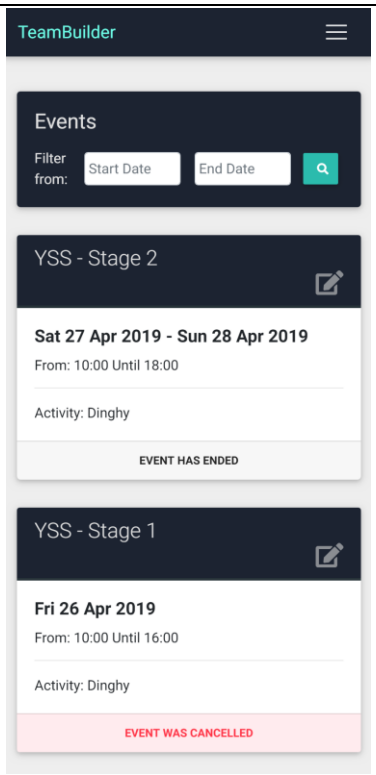


Uses the Datatable JavaScript API to create a table that holds the current response statuses of the invited volunteers. Viewing invited scouts is very similar to this but with scout accounts.

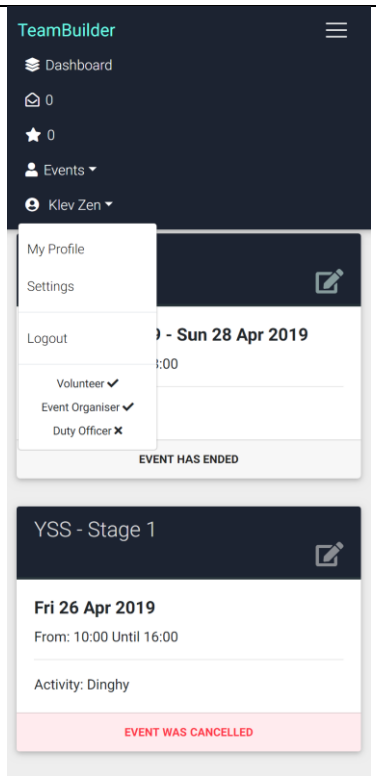
Navigation / Menu Structure

As scout events are typically held outdoors, a responsive navigation menu that could collapse and expand was necessary. Through added a feature such as this the navigation remains clean and uncluttered even when the system is being accessed from a handheld device such as a smartphone or tablet. This allows the user to traverse through the teambuilder application seamlessly just as they would if they were on a desktop pc, making for a great user experience.

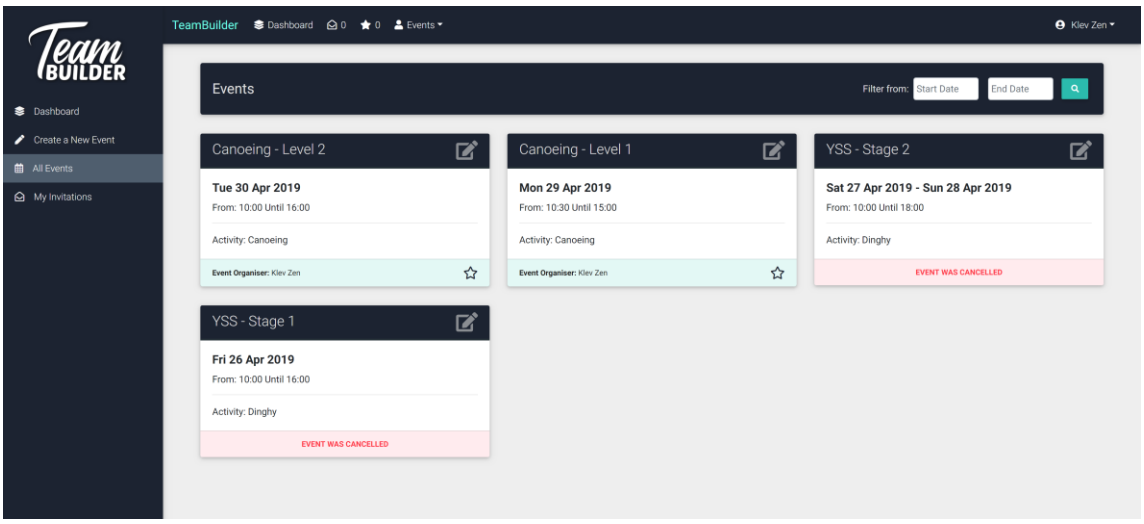
Below is an example of this menu in action



HAMBURGER MENU - COLLAPSED



HAMBURGER MENU - EXPANDED



FULL DESKTOP NAVIGATION MENU WITH SIDEBAR

Pages

Teambuilder application will be made up of numerous HTML and PHP pages that collectively deliver the overall system functionality. It is imperative that these pages are successfully identified and clearly separated based on the different user types. In an attempt to reduce redundant pages, the system will adopt an MVC style format whereby multiple user types will request access to the same page but be routed a different view depending on their account type. An example of where this is in use would be the following pages: dashboard, invitations and events. PHP sessions will be used to prevent users from accessing pages that they are not allowed to whilst jQuery will be used to enhance the user experience of the system. The "x" represents the inability to access the page and an blank field represents the ability to access.

Page Name	User Types			
	Scout	Volunteer	Event Organiser	Parent
Dashboard				
Events				
Invitations				
NewEvent	×	×		×
Index (the login page)				
Register				

4.4 Process Modelling

Activity Diagrams

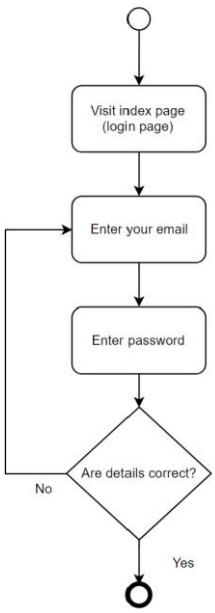


Figure 8 – *Log in activity diagram*

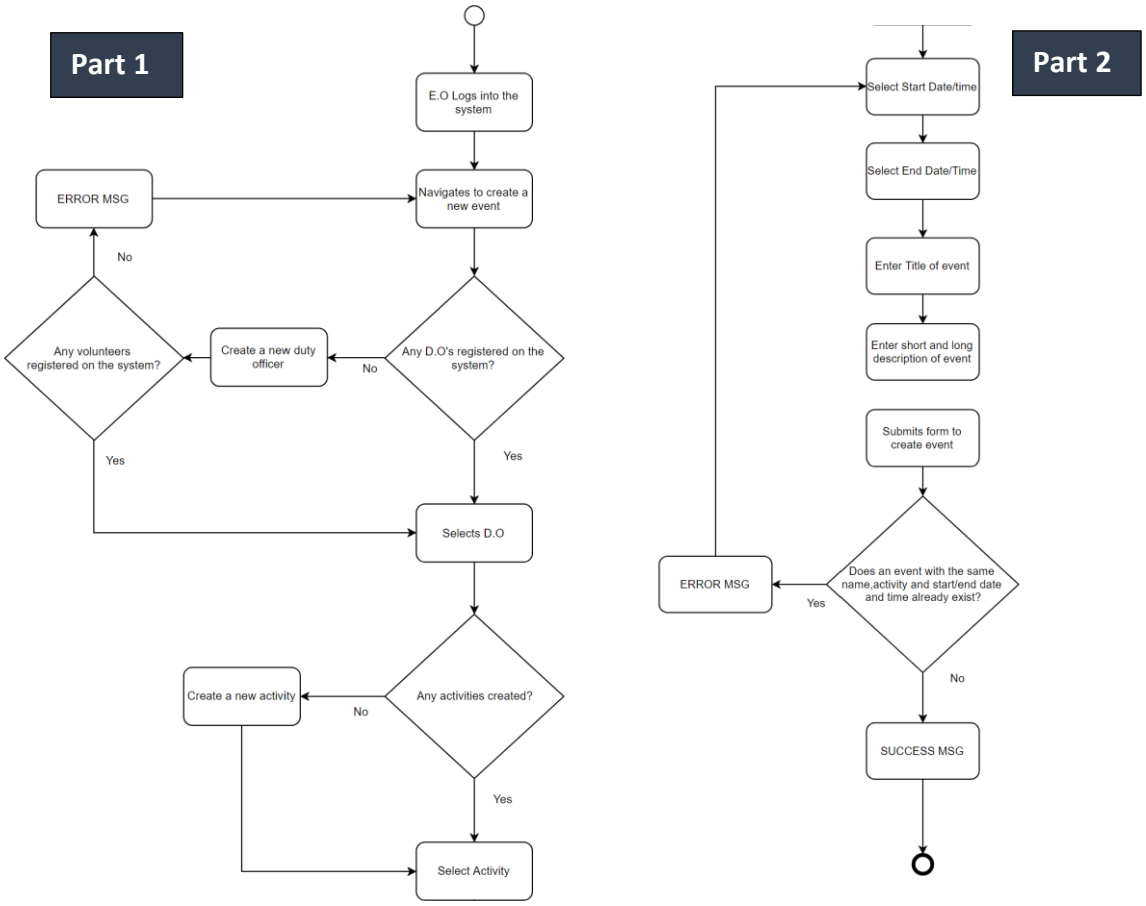


Figure 8.1 – *Create a new event activity diagram*

5. Implementation

5.1 Introduction

This chapter will detail the steps taken in order to actually develop the teambuilder application along with describing all of the different technologies and API's that were used to deliver the systems desired functionality. These have been split into two major sections and they are: the front-end Graphical User Interface (GUI) and the back-end Database Implementation.

The limitations of the system and any problems that were encountered during this phase will also be clearly outlined. Furthermore, due to the nature of our chosen agile methodology the requirements of the project can change during client feedback iterations even at this stage of the project; therefore, the final HTML layout iterations have been depicted within the design chapter and have been aptly labelled. The fundamental layouts of the pages won't change in this section but will instead become dynamic through the use of PHP, jQuery, JavaScript API's, CSS and validation rules.

5.2 Technologies Used

ATOM

This is an open-source text editor upon which the entire teambuilder application will be coded using. It offers a clean and simple UI along with a vast amount of community developed plugins that will greatly enhance the productivity of the developer and in turn greatly reduce the time it takes to add features or make changes to the project. It is compatible with most operating systems making it extremely portable and provides support for all of the languages that the project requires such as SQL, HTML, JavaScript, CSS and PHP.

HTML5

This is formally referred to as Hypertext Mark-up Language (HTML) and is a staple in any web development project. This will be used to create the structure and provide the system with a layout through a variety of different tags and attributes on top of which dynamic content will be generated. This language will be used in tandem with other programming languages such as JavaScript and PHP.

CSS3

Cascading Style Sheets (CSS) will be used to manipulate and style the presentation of the application by defining its paddings, margins, colours, sizes, fonts and overall positioning of its components. This will be achieved through custom written CSS code coupled together with front-end frameworks such as MDB.

JavaScript

JavaScript is client-side programming language that is essential to the success of this project. It will predominantly be used in the form of JavaScript libraries and API's such as jQuery and fullcalendar.io rather than on its own; however, there are most defiantly instances where

custom written pure JavaScript will need to be written alongside a third-party API code in order to make it work.

PHP

This is known as Hypertext Pre-processor (PHP) and is a server-side scripting language. This is the core language that the Teambuilder application will be run on and will allow for the majority of the features to be implemented through the use of PHP sessions, database queries and native PHP functions. The interface that will be used to access the database is known as php data objects (PDO) and provides an abstraction layer that allows functions to still execute correctly regardless of the type of database being used. When this is used in tandem with an object orientated (OO) style of programming it results in an extremely efficient, secure and flexible system that can protect against numerous SQL attacks.

SQL and phpMyAdmin

Teambuilder will be hosted on the Kingston University Web server and WinSCP will be used to connect to these servers and upload all of the necessary files needed to run the system. The database will be managed using phpMyAdmin running MariaDB server version 10.0.21 and developed using PDO with an OO, Model View Controller (MVC) approach to programming. This will allow us to write clean, maintainable code that is easily updated and can protect the system from SQL injections through the use of sanitizing input filters and binding parameters to statements rather than passing raw, un-sanitized variables straight into SQL queries. This ultimately provides the system with the flexibly, maintainability and security it needs to successfully meet the client's requirements.

JavaScript libraries and API's

jQuery

This will be used to handle multiple front-end user-experience related tasks throughout multiple pages of the system. The use of event listeners, custom validation rules using its selectors, AJAX calls to the database through PHP files and inbuilt functions are paramount to the overall look and feel of the application making this a vital dependency of the project necessary to successfully meet the client's requirements.

Fullcalendar.io

This is a free, customizable and open-source JavaScript event calendar that will be used within the volunteer dashboard to display their accepted events.

Pickadate.js

Is a lightweight, responsive date and time picker that is built using jQuery as its foundation, a JavaScript library that is already a dependency for our application. Pickadate.js is utilised throughout the teambuilder application wherever a date or time input is necessary.

MDBootstrap

This is a loaded front-end framework that incorporates both google material design principles and bootstrap functionality. Teambuilder will incorporate this framework in its entirety and as a result take on all of its dependencies including the likes of jQuery, bootstrap, fontawesome and Datatables. This framework will be used in tandem with

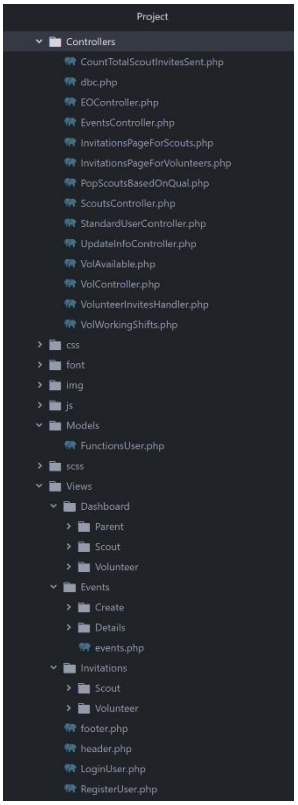
custom written CSS and media queries to make the application responsive for all types of devices and will save a lot of time as opposed to the alternative of writing and creating the interface components from scratch. It is also backed by a large, dedicated team so it has a plethora of available features that are all kept up to date and are fully customisable. Examples of some of the components provided by this framework include the likes of navigation bars, buttons, cards, panels, video backgrounds, and styled lists.

Alternatives

The following list consists of alternative software, servers or frameworks and have their counterparts labelled respectively.

- Bootstrap can be used as an alternative to MDBootstrap
- WAMP server can be used as an alternative to XAMPP
- Node.js can be used as an alternative to PHP
- FileZilla FTP can be used as an alternative to WinSCP
- Notepad++ can be used as an alternative to Atom
- SiteGround can be an alternative to the Kingston web Server
- MyWebSQL can be an alternative to phpMyAdmin
- jQuery UI datepicker can be an alternative to Pickadate.js
- mobiscroll can be an alternative to fullcalendar.io

5.3 Architecture



Teambuilder is created in such a way that all of its interactions are done through the browser alone. This is ideal as it means that the system can be accessed from any device, provided that it has an active internet connection and a suitable browser installed, regardless of any hardware or operating system constraints.

At its core, teambuilder follows a slightly adapted rendition of the popular Model View Controller (MVC) architectural pattern and as a result can reap the benefits it provides such as reusability, reliability and maintainability. It differs slightly from the traditional form of MVC due to the fact that the web server that will host the application does not allow modifications to a file that's known as ".htaccess" and access to this file is necessary in order to redefine the root directory and resolve issues such as routing. Nevertheless, the system still incorporates the core principles of an MVC application and object orientated approach to programming like the separation of concerns whereby business logic is grouped together and handled away from the views.

Figure 10 – *MVC file structure*

5.4 Implementing the Database

Creating Tables

One of the biggest advantages of using phpMyAdmin to manage the applications database is that you can execute queries straight from an inbuilt console within phpMyAdmin. This feature is particularly useful when coupled with the data dictionary depicted in the design phase as it contains all of the necessary information needed in order to develop a MySQL create table statement. A MySQL create table statement is a short-hand way to create tables within phpMyAdmin without having to go through its tedious wizard; an example of this is illustrated below.

```
CREATE TABLE users (
    UserID int(5) NOT NULL AUTO_INCREMENT,
    rightsID int(1) NOT NULL,

    Email varchar(255) NOT NULL UNIQUE,
    Password varchar(64) NOT NULL,

    Firstname varchar(30) NOT NULL,
    Middlename varchar(30),
    Lastname varchar(30) NOT NULL,
    Dob DATE,

    Gender varchar(6) NOT NULL,

    Mobile varchar(11),
    Address varchar(200),

    Postcode varchar(8),

    DateJoined DATETIME,

    IsVolunteer char(1) NOT NULL,
    IsScout char(1) NOT NULL,
    IsParent char(1) NOT NULL,

    PRIMARY KEY (UserID),
    KEY fk_rights (rightsID),

    CONSTRAINT users_ibfk_1
    FOREIGN KEY (rightsID)
    REFERENCES rights (rightsID)
    ON DELETE NO ACTION
    ON UPDATE CASCADE
);
```

Figure 11 – *MySQL Create Table Statement*

Once executed, this statement will create a table called “users” with all of the attribute names as columns and will then set primary identifying key of this table to the “UserID” attribute and also declare the “rightsID” attribute as a foreign key (FK), linking it to the “rightsID” column of the “rights” table. As you can imagine, once these statements have been created, they can be reused multiple times without having to be rewritten. These statements can also be layered on top of one another and executed in bulk thus saving the developer even more time. However, there is a caveat to this method of creating tables and that is the fact that the order in which the create table statements are executed matters. The tables without any FK’s must be created first and if this is neglected then when a create table statement with a FK attempts to apply the its FK constraint an error will occur as it will not be able to find the referenced table.

Eliciting Data Requirements

In order to sufficiently test the database to see if it can facilitate large amounts of data without issues, which is one of the non-functional requirements of the system, the client had to be consulted once more. From this additional meeting, further information such as the types of activities, the types of qualifications and the shift roles were extracted from the client and used to populate the database with relevant information.

Populating the Tables with Data

Dummy Data

In order to populate the database with bulk dummy data, the relevant information that was supplied to me by my client had to be converted into MySQL insert into statements. Some examples of which are presented below:

```
/*
*       Activities
*/

INSERT INTO activities (`ActivityID`, `Name`) VALUES (NULL, 'Dinghy'),
(NULL, 'Kayaking'),
(NULL, 'Canoeing'),
(NULL, 'Windsurfing'),
(NULL, 'Powerboat'),
(NULL, 'Cruising'),
(NULL, 'Keelboat');
```

```
/*
*      Shiftroles
*/

INSERT INTO shiftroles (`ShiftRoleID`, `QID`, `Name`, `Descr`) VALUES (NULL, '1', 'Shore Support', 'General shore support to assist the scouts'),
(NULL, '2', 'Duty Officer', 'The most senior scout at an event'),
(NULL, '3', 'Dinghy Instructor', 'Dinghy instructor for the event to the scouts'),
(NULL, '4', 'Kayaking Instructor', 'Kayaking instructor for the event to the scouts'),
(NULL, '5', 'Canoeing Instructor', 'Canoeing instructor for the event to the scouts'),
(NULL, '6', 'Powerboat Instructor', 'Powerboat instructor for the event to the scouts'),
(NULL, '7', 'Cruising Instructor', 'Cruising instructor for the event to the scouts'),
(NULL, '8', 'Keelboat Instructor', 'Keelboat instructor for the event to the scouts'),
(NULL, '9', 'Medic', 'On-site Medic'),
(NULL, '10', 'Safety Boat Crew', 'Crew member for the satefy boat');
```

Figure 12 *– MySQL Insert into Statements*

These statements work in a similar fashion to the MySQL create table statements mentioned earlier and the ones in this particular example will add 7 records into the ‘activities’ table and 10 records into the ‘shiftroles’ table respectively. This process is relatively simple to complete

when the number of records is small; however, since the system is required to handle 200 users simultaneously, we will need much more than what is illustrated from *figure 12* alone.

To solve this, a free online dummy data generator called mockaroo.com was used to generate large amounts of dummy data such as first names, last names and emails that would then be manually copied into the relevant 'VALUE' parameters in order to create the remainder of the insert into statements.

Verifying the data population

Once all of the create table and insert into statements have been executed, it is essential to confirm whether or not the database has been successfully populated with the desired data. This can be done through the use of phpMyAdmin statistics and can be seen in *Figure 13*.

Table	Action	Rows
activities	Browse Structure Search Insert Empty Drop	7
availableshifts	Browse Structure Search Insert Empty Drop	14
days	Browse Structure Search Insert Empty Drop	7
dutyofficers	Browse Structure Search Insert Empty Drop	23
eventorganisers	Browse Structure Search Insert Empty Drop	1
events	Browse Structure Search Insert Empty Drop	4
invitedevents	Browse Structure Search Insert Empty Drop	5
invitestatus	Browse Structure Search Insert Empty Drop	4
parents	Browse Structure Search Insert Empty Drop	51
qualifications	Browse Structure Search Insert Empty Drop	36
refstatus	Browse Structure Search Insert Empty Drop	6
rights	Browse Structure Search Insert Empty Drop	2
scouts	Browse Structure Search Insert Empty Drop	102
shiftroles	Browse Structure Search Insert Empty Drop	10
userinterests	Browse Structure Search Insert Empty Drop	0
userqual	Browse Structure Search Insert Empty Drop	553
users	Browse Structure Search Insert Empty Drop	255
volfreedays	Browse Structure Search Insert Empty Drop	108
volunteers	Browse Structure Search Insert Empty Drop	127
19 tables	Sum	1,315

Figure 13 *– phpMyAdmin Row Statistics*

ActivityID	Name
3	Canoeing
6	Cruising
1	Dinghy
2	Kayaking
7	Keelboat
5	Powerboat
4	Windsurfing

ShiftRoleID	QID	Name	Descr
1	1	Shore Support	General shore support to assist the scouts
2	2	Duty Officer	The most senior scout at an event
3	3	Dinghy Instructor	Dinghy instructor for the event to the scouts
4	4	Kayaking Instructor	Kayaking instructor for the event to the scouts
5	5	Canoeing Instructor	Canoeing instructor for the event to the scouts
6	6	Powerboat Instructor	Powerboat instructor for the event to the scouts
7	7	Cruising Instructor	Cruising instructor for the event to the scouts
8	8	Keelboat Instructor	Keelboat instructor for the event to the scouts
9	9	Medic	On-site Medic
10	10	Safety Boat Crew	Crew member for the satefy boat

Figure 14 *– Data imported into the Database Tables*

Establishing a Database Connection

The connection to the database was established through a file called 'dbc.php' the contents of which can be seen in *figure 15*. Since the system uses PHP Data Objects with an Object Orientated Programming (OOP) approach, this file consists of a class named 'DatabaseCon' that extends the class of 'PDO' and has a constructor method which, when an instance of the 'DatabaseCon' class is created, returns a new PDO object (line 14) based on the variables from lines 3 to 9.

```
1   <?php
2   class DatabaseCon extends PDO{
3           public $db;
4           private $eng = 'mysql';
5           private $serv = 'kunet.kingston.ac.uk';
6
7           private $u = 'k1651547';
8           private $p = '[redacted]';
9           private $n = 'dbck1651547';
10
11          function __construct()
12          {
13              try{
14                  $this->db = new PDO("".$this->eng.":host=$this->serv; dbname=$this->n", $this->u,$this->p);
15              }
16              catch (PDOException $e){
17                  $e->getMessage();
18              }
19          }
20
21  }
```

Figure 15 *– dbc.php*

This file is referenced using the 'require_once' function and is done so within the 'header.php' file. The 'header.php' file is then referenced in all of the views so that the functions can have access to the database. Again, since this is a PDO with OOP approach the PDO object that is returned from creating a new instance of the 'DatabaseCon' object must be passed through as a parameter to a new instance of a controller class object in order for that controller to have access to the database. An illustration of this can be found in *Figure 16*.

```
<?php
    session_start();
    require_once('Models/FunctionsUser.php');
    require_once('Controllers/dbc.php');


    $dbpdo  = new DatabaseCon();
    $fuser  = new FunctionsUser($dbpdo);

?>
<html lang="en">

<head>
  <meta charset="utf-8">
```

Figure 16 – *Top half of the header.php file*

5.4 Developing the frontend GUI

Once the database has been successfully implemented, the UI will now need to be constructed based on the High-Fidelity mock-ups generated in the design phase. A range of different tools, technologies and programming languages were used to create the most appropriate interface and they have been detailed above within the 'Technologies Used' section of this chapter.

Reusable Components

HEADER

```
1   <?php
2       session_start();
3       require_once('Models/FunctionsUser.php');
4       require_once('Controllers/dbc.php');
5
6
7       $dbpdo  = new DatabaseCon();
8       $fuser  = new FunctionsUser($dbpdo);
9
10  ?>
11  <html lang="en">
12
13  <head>
14    <meta charset="utf-8">
15    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">
16    <meta http-equiv="x-ua-compatible" content="ie=edge">
17    <title>TeamBuilder</title>
18    <!-- Font Awesome -->
19    <link rel="stylesheet" href="https://use.fontawesome.com/releases/v5.6.3/css/all.css">
20    <!-- Bootstrap core CSS -->
21    <link href="css/bootstrap.min.css" rel="stylesheet">
22    <!-- Material Design Bootstrap -->
23    <link href="css/mdb.min.css" rel="stylesheet">
24    <!-- Your custom styles (optional) -->
25    <link href="css/style.css" rel="stylesheet">
26
```

Figure 17 – *header.php*

To begin with, a 'header.php' file Is created within the views folder and contains the dependency declarations, lines 19 to 25 in *figure 17*, for the various CSS files that are vital to styling the teambuilder application. The code within the <meta> html tags are also of great importance as they define the character encoding for this document followed by a set of instructions that are given to the browser detailing how to scale the page through the viewport. This is also extended to set the width and zoom of the page. Line 15 is of the upmost importance when it comes to the responsivity of the website because it enables the browser's page to automatically scale to the correct width and height. This 'header.php' file is 'required_once' at the top of every single view page so that access to the database can be established for the controllers of that particular view, along with maintaining the user session across multiple pages thanks to the session_start() function on line 2.

A similar thing has been done with the 'footer.php' page also but rather than being 'required_once' towards the top of the page, it is 'required_once' towards the bottom of the view's page.

Redirecting users that are logged in

If a user decides to manually type in the page extension to go to the register or index page (login page) then they are redirected back to their dashboard by the following code which is placed at the top of the register page and login page (index.php).

```
1   <?php
2   require_once('Views/header.php');
3   require_once('Views/RegisterUser.php');
4   if(isset($_SESSION['u_sesh']))
5   {
6       header("Location: dashboard.php");
7   //  echo "youre logged in!";
8   }else if(isset($_SESSION['success'])){
9     header("Location: index.php");
10  }
11  ?>
```

Figure 18 *– register.php*

Lines 4 to 6 handle this logic and check if the 'u_sesh' variable has been set; if it has then the user is logged in (because they have an ongoing session assigned to them) they are redirected to the dashboard.php file. Similarly, is the session 'success' is set, then they have just successfully registered an account so the system will redirect them to the index page where the 'success' session gets unset.

Register for an account

One of the requirements of the system is that a user should be able to sign up for their desired account type and their detailed are stored in the databased with their password hashed, this has been implemented through the following set of pages. On the register page*, figure 18 line 3,* the RegisterUser.php file is required by this page and within that , *figure 19 line 8*, there is an if statement which checks if the register button from the register.php page has been submitted. For this if statement to be true, the user must have successfully met all of the input validation rules put in place for the register form and successfully submitted the form. Should this condition be met, the input variables that were submitted are sanitized using the filter_var() method and have the second parameters set to the expected data type for that variable. The $email variable is additionally converted to a lowercase string so that when it is

stored into the database and compared against later a match can be successfully found, regardless of case.

```php
<?php
require_once('Models/FunctionsUser.php');
require_once('Controllers/dbc.php');

$dbpdo  = new DatabaseCon() ;
$fuser  = new FunctionsUser($dbpdo);

if(isset($_POST['registerbutton']))
{
    $firstname = filter_var($_POST['firstname'],FILTER_SANITIZE_STRING);
    $lastname = filter_var($_POST['lastname'],FILTER_SANITIZE_STRING);
    $email = filter_var($_POST['email'],FILTER_SANITIZE_EMAIL);
    $email = strtolower($email);
    $password = $_POST['password'];
    $gender = "";
    $acctype = "";

    if(isset($_POST['gender'])){
      $gender = filter_var($_POST['gender'],FILTER_SANITIZE_STRING);
    }

    if(isset($_POST['acctype'])){
      $acctype = filter_var($_POST['acctype'],FILTER_SANITIZE_STRING);
    }
```

Figure 19 – top half of RegisterUser.php

After the variables have been sufficiently filtered, the first if in a series of nested if's will be run and it consists of a function that takes the $email variable submitted with the form as a parameter and queries the database to see if it already exists (*Figure 20 – line 26*).

```
26      if(!$fuser->duplicatemail($email)){
27
28        if($fuser->newUser($firstname,$lastname,$email,$password,$gender,$acctype)){
29
30          if($fuser->addDefaultQualifications($email,$firstname,$lastname))
31          {
32            $_SESSION['success'] = "Your ".$acctype." account has been successfully created";
33
34            if($acctype == "Volunteer")
35            {
36              $_SESSION['chosenfreedays'] = "no";
37            }
38          }
```

Figure 20 – middle half of RegisterUser.php

should this if statement return false, then we progress on to the next if statement which consists of a function call that takes the user submitted details as parameters, hashes their password and inserts the user into the database. The function newUser Exists in the functionsUser.php file and is illustrated in *Figure 21.*

```
216         public function newUser($firstname,$lastname,$email,$password,$gender,$acctype)
217         {
218           //echo "-- started function newUser --";
219             if(!empty($firstname) && !empty($lastname) && !empty($email) && !empty($password) && !empty($gender) && !empty($acctype))
220             {
221             //  echo "-- passed empty vars check --";
222                 $boolVol = 0;
223                 $boolScout = 0;
224                 $boolParent = 0;
225
226                 if($acctype === "Scout")
227                 {
228                   $boolScout = 1;
229                 }
230                 else if($acctype === "Parent")
231                 {
232                   $boolParent = 1;
233                 }
234                 else if($acctype === "Volunteer"){
235                   $boolVol = 1;
236                 }
237
238                 $pass = password_hash($password,PASSWORD_DEFAULT);
239
240                 $st = $this->db->prepare("INSERT INTO `users`
241                   (`RightsID`,`Email`,`Password`,`Firstname`,`Lastname`,`Gender`,`DateJoined`,`IsVolunteer`,`IsScout`,`IsParent`)
242                   VALUES (1,:u_email,:u_pass,:u_fn,:u_ln,:u_gender,:u_date,$boolVol,$boolScout,$boolParent)");
243
244                 $st->bindParam(':u_email',$email);
245                 $st->bindParam(':u_pass',$pass);
246                 $st->bindParam(':u_fn',$firstname);
247                 $st->bindParam(':u_ln',$lastname);
248                 $st->bindParam(':u_gender',$gender);
```

Figure 21 – half of the newUser Function

When this function is run, the parameters are first checked to ensure that they're not null or empty. Afterwards the acctype variable passed in as a parameter is checked to see which account type the user had selected and updates the corresponding $boolScout,$boolVol or $boolParent variable accordingly. Once that's done line 238 hashes the password using the bcrypt hashing algorithm and then an SQL statement is prepared. As you can see, the only variables directly passed into the SQL statement are the $bool variables as we know what these values will be at all times, that is either a 1 or a 0 so they are generally regarded as safe to directly reference. The user submitted variables however have to each be bound to a corresponding label using the bindParam function so that no SQL injection or malicious code gets executed along with our SQL statement.

```
254                 if($st->execute())
255                 {
256                   if($boolScout == 1){
257                     return $this->addScout($email);
258                   }else if($boolParent == 1){
259                     return $this->addParent($email);
260                   }else if($boolVol == 1){
261                     return $this->addVolunteer($email);
262                   }else{
263                     return true;
264                   }
265                 }
```

Figure 22 – *lower half of the newUser Function*

Line 254 executes the SQL statement mentioned above inside of an 'if' statement condition, if the execution was successful then the criteria of the 'if' statement is met and then the respective function is called depending on the $bool value of your chosen account type. For this example lets say that our chosen account type is volunteer, then our $boolVol variable

will equal to 1 so the function addVolunteer will be run and the variable $email will be passed through to it as a parameter. The contents of the addVolunteer function are shown in *figure 23*.

```
278         public function addVolunteer($email)
279         {
280           if(!empty($email)){
281             $r_userid = $this->getIdfromEmail($email);
282
283             if($r_userid === false){
284               return false;
285             }else{
286               $st = $this->db->prepare("INSERT INTO `volunteers`(`UserID`, `IsEO`, `IsDO`) VALUES (:u_id,0,0)");
287               $st->bindParam(':u_id',$r_userid);
288
289               if($st->execute()){
290                   return true;
291               }else{
292                   return false;
293               }
294             }
295           }else{
296             return false;
297           }
298         }
```

***Figure 23** – addVolunteer function*

This function will take the $email parameter and run a check on it to ensure that it is not null or empty. It will then run an additional helper function called getIdfromEmail and will pass in that $email variable as a parameter. This function will query the database for that particular email and if a match is found, it will return the UserID that is associated to that email address otherwise it will return false. This value is then stored inside of the $r_userid variable and, if it is not false, will be passed through to a prepared SQL insert into statement using the bindParam function. This statement is then executed and creates a new record in the volunteers table for that UserID. Now that this has passed, the value “true” is returned back to the RegisterUser.php , *in Figure 20,* and the function addDefaultQualificaitons, *Figure 20 line 30, is run and adds the qualification of “No qualification” to the ‘userqual’ table for that users UserID.*

Updating the calendar to show the volunteer's shifts

The calendar on the volunteer's dashboard was created using the fullcalendar.io JavaScript API and collects its data based on ajax calls, through php files, which query the availableshifts (avs) table for that volunteers VolunteerID. Before any of this can occur though, the fullcalendar.io dependencies along with a native JavaScript file called "voldashboard.js" must be imported, in this particular order, within the file " Views/Dashboard/Volunteer/ Index.php ". This is presented in Figure 24 below :

```
<!-- Calendar CSS Dependancies -->
<link href='https://unpkg.com/@fullcalendar/core@4.0.2/main.min.css' rel='stylesheet' />
<link href='https://unpkg.com/@fullcalendar/daygrid@4.0.1/main.min.css' rel='stylesheet' />
<link href='https://unpkg.com/@fullcalendar/timegrid@4.0.1/main.min.css' rel='stylesheet' />
<link href='https://unpkg.com/@fullcalendar/list@4.0.1/main.min.css' rel='stylesheet' />
```

```
<!-- Calendar JavaScript Dependancies -->
<script src='https://unpkg.com/@fullcalendar/core@4.0.2/main.min.js'></script>
<script src='https://unpkg.com/@fullcalendar/daygrid@4.0.1/main.min.js'></script>
<script src='https://unpkg.com/@fullcalendar/timegrid@4.0.1/main.min.js'></script>
<script src='https://unpkg.com/@fullcalendar/list@4.0.1/main.min.js'></script>
<script src='https://unpkg.com/@fullcalendar/bootstrap@4.0.1/main.min.js'></script>

<!-- Custom JS-->
<script type="text/javascript" src="js/voldashboard.js"></script>
```

Figure 24 – *Calendar.io Dependencies*

Since we imported our custom JavaScript file after the fullcalendar.io dependencies we can reference their functions within voldashboard.js; however, before that the volunteer id that is currently stored in the user's session will need to be placed as the value for a hidden input field within the file " Views/Dashboard/Volunteer/ Index.php ".

```
<input type="hidden" name="hiddenvolid" id="hiddenvolid" value="<?= $_SESSION['u_sesh']['VolId'] ?>">
```

This value will then be taken and stored into a JavaScript variable and passed along with our ajax call in order to tailor the returned results to the currently logged in volunteer. This is further illustrated within *figure 25*.

```
1  document.addEventListener('DOMContentLoaded', function() {
2    var calendarEl = document.getElementById('calendar');
3    var volunteerId = $('#hiddenvolid').val();
```

Figure 25 – *Top part of voldashboard.js*

Line 1 to 3 of *Figure 25* first wait for the Document Object Modal content to finish loading and one it has concluded, the calendar with the html id attribute of 'calendar' is stored inside of the calendarEl variable. Similarly, the value of the html input with the id attribute of #hiddenvolid' is stored inside of the VolunteerId variable. Afterwards, the calendar is

initialised by creating a new FullCalendar object and setting its 'Calendar' property to two parameters, one of which being our previously declared calendarEl variable , *Figure 25 line 2,* followed by an object with different properties that we will now fill in with the appropriate options so that the event calendar can be successfully initialised. Please refer to *figure 26:*

```
6       var calendar = new FullCalendar.Calendar(calendarEl, {
7         header: {
8           left: 'prev,next today',
9           center: 'title',
10          right: 'dayGridMonth,timeGridWeek'
11        },
12        eventSources: [
13
14          // your event source
15          {
16            url: 'Controllers/VolWorkingShifts.php',
17            method: 'POST',
18            extraParams: {
19              volid: volunteerId,
20              calendarAjaxOngoingEvents: 'search'
21            },
22            failure: function() {
23              alert('there was an error while fetching events!');
24            },
25            color: 'lightblue',   // a non-ajax option
26            textColor: 'black' // a non-ajax option
27          },
28
```

Figure 26 – *Middle part of voldashboard.js*

This piece of code follows the recommendations and guidelines detailed on the fullcalender.io API website for populating event sources using ajax requests. Lines 16 to 27 is our first ajax call and will submit the variables 'volid', which holds the volunteer id of the current logged in user and 'calendarAjaxOngoingEvents' to the VolWorkingShifts.php file using the POST method.

```
<?php
require_once('dbc.php');

if(isset( $_POST['calendarAjaxOngoingEvents'] ))
{
     $volid = $_POST['volid'];

     $pdo = new DatabaseCon();
     $st = $pdo->db->prepare("SELECT e.Name AS 'title', avs.StartDate AS 'start', avs.EndDate AS 'end'
FROM `availableshifts` avs
INNER JOIN `events` e ON e.EventID = avs.EventID
WHERE `VolunteerID` = :vol_id
AND avs.`StatusID` = 6
AND avs.EndDate > :curdate
AND e.StatusID != 2
AND e.StatusID != 3");
date_default_timezone_set("Europe/London");
$date = new DateTime();
$date = $date->format('Y-m-d H:i:s');

$st->bindParam(':vol_id',$volid);
$st->bindParam(':curdate',$date);
     $st->execute();
     $results = $st->fetchAll(PDO::FETCH_OBJ);
     echo json_encode($results);

}
```

Figure 27 *– Top part of VolWorkingShifts.php*

For Figure 27 above. Since the variable 'calendarAjaxOngoingEvents' was passed through as a parameter, the if statement condition on line 4 will resolve to true and a new SQL Select statement will be prepared. This statement will query the availableshifts (avs) table based on 5 conditions:

1. Volid variable passed to it from the ajax request matches a VolunteerID inside of that table
2. avs table's EndDate attribute is greater than today's date,
3. avs table's StatusID is equal to 6 (which is the status id for "Accepted")
4. Event's StatusID is not 2 ("cancelled")
5. Event's StatusID is not 3 ("ended")

The query has also been formatted through the addition of the "AS" command, ensuring that the results returned back from the query follow the pattern of "title", "start" and "end" for the Event Name, avs StartDate and avs EndDate attributes respectively, as these are the names that the fullcalendar.io ajax call is looking for when It receives a response back.

It then binds the appropriate parameters of $volid (the volunteer id passed in from the ajax call) and $date (todays current datetime created using the php datetime object and format function) to the statement through the bindParam function and executes the statement. The results are received through the fetchAll function and will be stored inside of the $results variable. This $results variable is then json encoded so that it can be correctly interpreted when it is sent back to the ajax call and parsed. If that VolunteerID had any records in the avs table for an event that was still ongoing and that they had accepted the invite to they would see that event show on their calendar much like *figure 28* below.

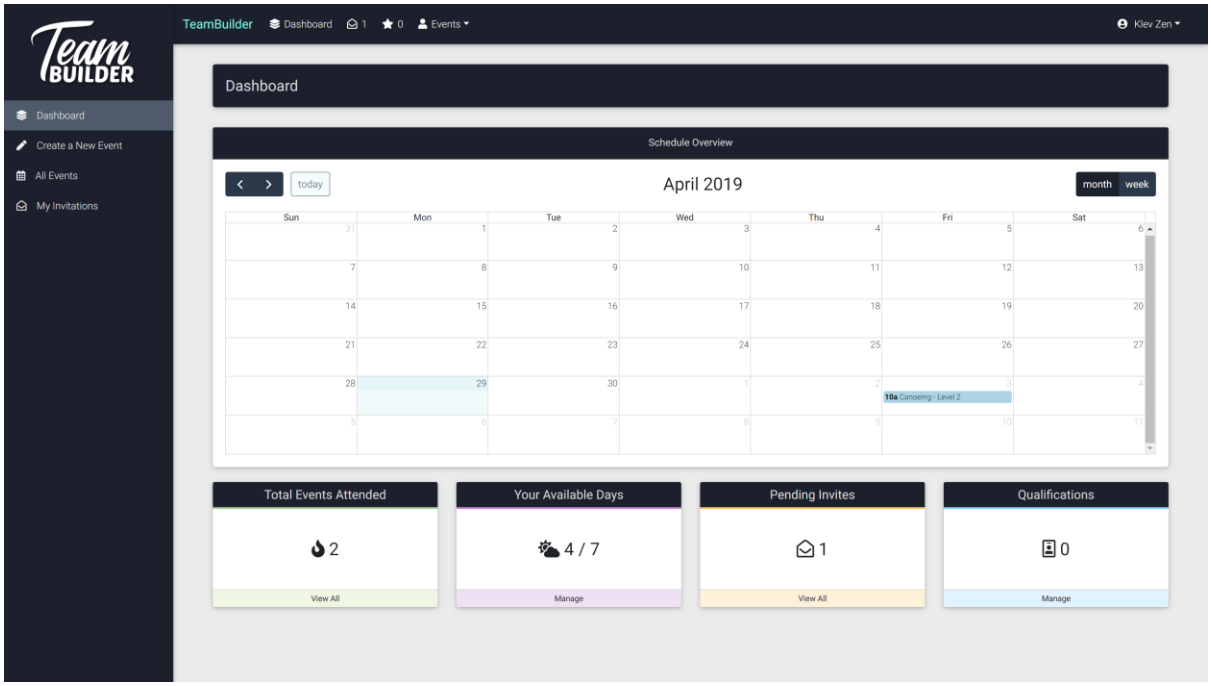


Figure 28 *– Volunteer Dashboard Calendar after successful ajax call*

This is then done two additional times but with different ‘extraParam’ variables each time so that the events that have ended and the events that were cancelled can also show up on the volunteer’s calendar, provided that they chose to accept the shift invites in both instances.

```
29          {
30            url: 'Controllers/VolWorkingShifts.php',
31            method: 'POST',
32            extraParams: {
33              volid: volunteerId,
34              calendarAjaxEndedEvents: 'search'
35            },
36            failure: function() {
37              alert('there was an error while fetching events!');
38            },
39            color: '#d2d2d2',   // a non-ajax option
40            textColor: 'black' // a non-ajax option
41          },
42          {
43            url: 'Controllers/VolWorkingShifts.php',
44            method: 'POST',
45            extraParams: {
46              volid: volunteerId,
47              calendarAjaxCancelledEvents: 'search'
48            },
49            failure: function() {
50              alert('there was an error while fetching events!');
51            },
52            color: '#e9adad',   // a non-ajax option
53            textColor: 'black' // a non-ajax option
54          }
```

Figure 29 *– Bottom part of voldashboard.js*

For *figure 29* above. Lines 30 to 41 is our second ajax call and will submit the variables ‘volid’, which holds the volunteer id of the current logged in user and ‘calendarAjaxEndedEvents’ to the VolWorkingShifts.php file using the POST method.

```
28  else if(isset( $_POST['calendarAjaxEndedEvents'] ))
29  {
30    $volid = $_POST['volid'];
31
32    $pdo = new DatabaseCon();
33    $st = $pdo->db->prepare("SELECT e.Name AS 'title', avs.StartDate AS 'start', avs.EndDate AS 'end'
34  FROM `availableshifts` avs
35  INNER JOIN `events` e ON e.EventID = avs.EventID
36  WHERE `VolunteerID` = :vol_id
37  AND avs.`StatusID` = 6
38  AND avs.EndDate < :curdate
39  AND e.StatusID != 2");
40
41  date_default_timezone_set("Europe/London");
42  $date = new DateTime();
43  $date = $date->format('Y-m-d H:i:s');
44
45  $st->bindParam(':vol_id',$volid);
46  $st->bindParam(':curdate',$date);
47    $st->execute();
48    $results = $st->fetchAll(PDO::FETCH_OBJ);
49    echo json_encode($results);
50
51  }
```

Figure 30 *– Middle part of VolWorkingShifts.php*

For Figure 30 above. Since the variable 'calendarAjaxEndedEvents' was passed through as a parameter, the if statement condition on line 28 will resolve to true and a new SQL Select statement will be prepared. This statement will query the availableshifts (avs) table based on 4 conditions:

1. Volid variable passed to it from the ajax request matches a VolunteerID inside of that table
2. avs table's EndDate attribute is less than today's date,
3. avs table's StatusID is equal to 6 (which is the status id for "Accepted")
4. Event's StatusID is not 2 ("cancelled")

The steps from here are the same as for the first ajax query mentioned ***above figure 28.*** *Therefore, the outcome will be displayed instead*

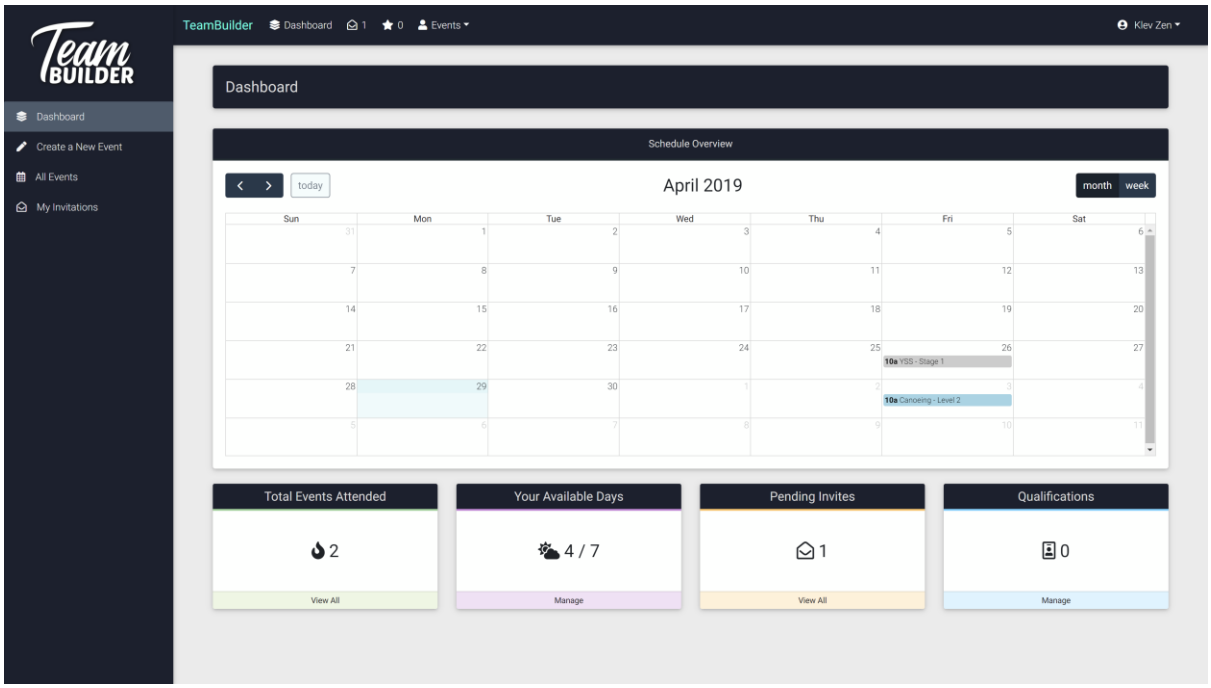


Figure 31 *– Volunteer Dashboard Calendar after 2nd successful ajax call*

For *figure 29* above. Lines 43 to 53 is our second ajax call and will submit the variables 'volid', which holds the volunteer id of the current logged in user and 'calendarAjaxCancelledEvents' to the VolWorkingShifts.php file using the POST method.

```
52  else if(isset( $_POST['calendarAjaxCancelledEvents'] ))
53  {
54    $volid = $_POST['volid'];
55
56    $pdo = new DatabaseCon();
57    $st = $pdo->db->prepare("SELECT e.Name AS 'title', avs.StartDate AS 'start', avs.EndDate AS 'end'
58  FROM `availableshifts` avs
59  INNER JOIN `events` e ON e.EventID = avs.EventID
60  WHERE `VolunteerID` = :vol_id
61  AND avs.`StatusID` = 6
62  AND e.StatusID = 2");
63
64    $st->bindParam(':vol_id',$volid);
65    $st->execute();
66    $results = $st->fetchAll(PDO::FETCH_OBJ);
67    echo json_encode($results);
68  }
```

Figure 32 *– Bottom part of VolWorkingShifts.php*

For Figure 32 above. Since the variable 'calendarAjaxCancelledEvents' was passed through as a parameter, the if statement condition on line 28 will resolve to true and a new SQL Select statement will be prepared. This statement will query the availableshifts (avs) table based on 4 conditions:

1. Volid variable passed to it from the ajax request matches a VolunteerID inside of that table
2. avs table's StatusID is equal to 6 (which is the status id for "Accepted")
3. Event's StatusID is equal to 2 ("cancelled")

The steps from here are the same as for the first ajax query mentioned ***above figure 28.*** *Therefore, the outcome will be displayed instead*

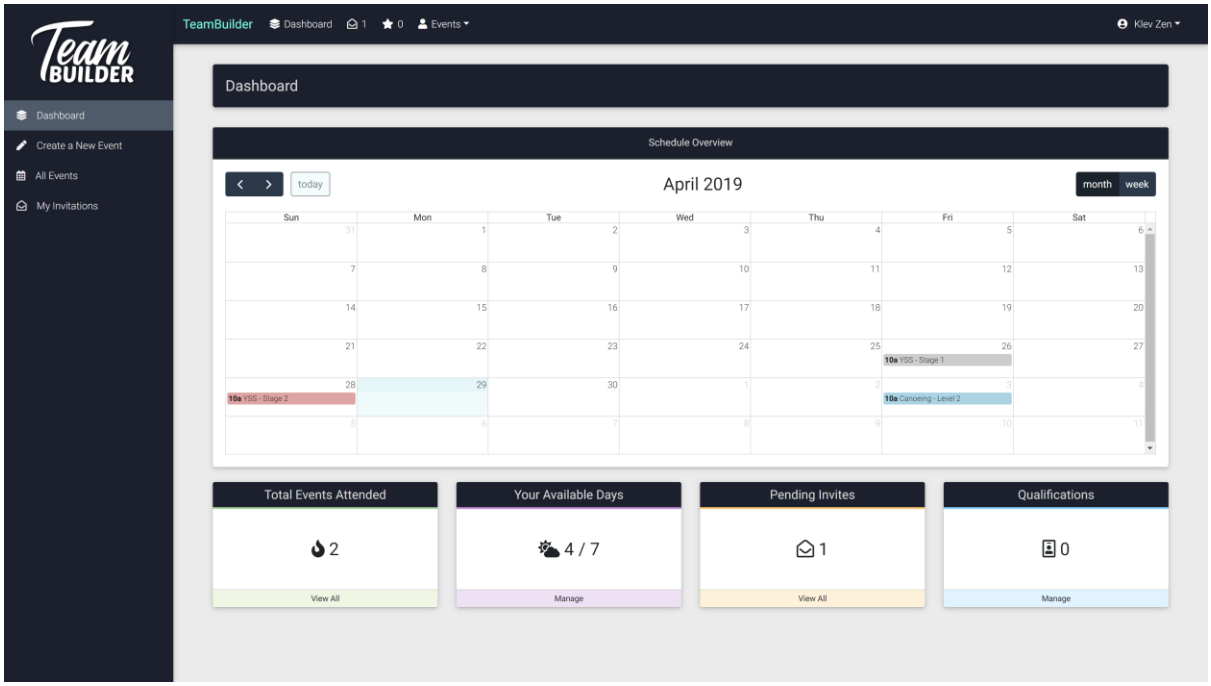


Figure 33 *– Volunteer Dashboard Calendar after 3rd successful ajax call*

5.5 Security Measures Taken and Potential Concerns

Client Side

On the client side, there are multiple custom validation rules and Regular Expression (RegExr) patterns in place, created using JavaScript, that have been applied to each and every one of the system's inputs accordingly thus providing an additional layer of protection to the system.

Server Side

On the server side all new registered users will have their passwords hashed and salted using the bcrypt one-way hashing algorithm so that in the event of a database breach the user's passwords are safe from the attacker's eyes. Furthermore, any and all data that is received from a user input has been put through additional sanitization filters, empty value checks and have had their special characters escaped through the use of inbuilt PHP functions in order to reduce the systems susceptibility to Cross Site Scripting (XSS) attacks .

Protecting against SQL injections

An SQL injection is when an attacker will attempt to exploit an un-sanitized input field by entering malicious SQL code that will be executed and could cause records to be deleted or even whole tables to be dropped which, if the developer hasn't backed the system up, could cause irreparable damage. There are several prevention methods in place that can prevent against such a threat and is showcased in the code below:

```
18      public function getInviteStatusOfAllVolunteersForEvent($evtid){
19        if(!empty($evtid)){
20          $evtid = filter_var($evtid,FILTER_SANITIZE_NUMBER_INT);
21          $st = $this->db->prepare("SELECT avs.AvShiftID, avs.StartDate, avs.EndD
22      FROM `availableshifts` avs
23      INNER JOIN volunteers v ON v.VolunteerID = avs.VolunteerID
24      INNER JOIN users u ON u.UserID = v.UserID
25      INNER JOIN shiftroles sr ON sr.ShiftRoleID = avs.ShiftRoleID
26      INNER JOIN days day ON day.DayID = avs.DayID
27      INNER JOIN refstatus rs ON rs.StatusID = avs.StatusID
28      WHERE avs.EventID = :evtid");
29          $st->bindParam(':evtid',$evtid);
30          $st->execute();
31          $rc = $st->rowCount();
32
33          if($rc>0){
34            $res = $st->fetchAll(PDO::FETCH_OBJ);
35            return $res;
36          }else{
37            return false;
38          }
39        }
40      }
```

Figure 34 *– SQL injection code*

Figure 34 Line 21 sets up a prepared SQL query statement but doesn't directly pass in any variables, instead a label, *line 28*, called ':evtid' is put in as a placeholder instead. After the SQL statement has been prepared, the $evtid variable can now be bound to the ':evtid' through a bindParam() function on *line 29*. Now when the SQL statement is executed, *line 30*, the parameter values that we bound to the statement can't interfere with it as they are not embedded into the query string but are instead sent over using a different protocol.

6. Validation and Evaluation

6.1 introduction

Within this chapter, teambuilder application will undergo a series of different types of tests the results of which will be detailed accordingly in order to confirm whether or not the functionality operates as expected and produces the desired results.

6.2 Database Validation

As the back-end database is a vital component of the teambuilder application it is imperative to ensure that all of the desired database create, read, update and delete (CRUD) functionalities are fully operational at all times. Furthermore, tests must also be run on the database to demonstrate that it is in a consistent state, this will involve testing for all entity, referential, domain and general check violations in order to see how the database will respond.

Create Read Update Delete - Tests

ID	REQUIREMENT	SQL QUERY	RESULT
1	Create Activity	INSERT INTO activities (`ActivityID`, `Name`) VALUES (NULL, 'Paddleboating');	PASS

✔ 1 row inserted.
Inserted row id: 9 (Query took 0.0360 seconds.)

```
INSERT INTO activities (`ActivityID`, `Name`) VALUES (NULL, 'Paddleboating')
```

2	Delete Activity	DELETE FROM `activities` WHERE `ActivityID` = 8	PASS

✔ 1 row affected. (Query took 0.0487 seconds.)

```
DELETE FROM `activities` WHERE `activities`.`ActivityID` = 8
```

3	View Events	SELECT * FROM `events`	PASS

✔ Showing rows 0 - 3 (4 total, Query took 0.0013 seconds.)

```
SELECT * FROM `events`
```

4	Update Event	UPDATE `events` SET `ActivityID` = '2' WHERE `EventID` = 1;	PASS

✔ 1 row affected.

```
UPDATE `events` SET `ActivityID` = '2' WHERE `events`.`EventID` = 1;
```

Integrity Violation - tests

ID	DESCRIPTION	SQL QUERY	INTEGRITY	RESULT
1	Trying to insert a new record and setting its id to a Primary key that already exists	INSERT INTO activities (`ActivityID`, `Name`) VALUES (1, 'Swimming');	Entity	PASS

Error

SQL query:

```
INSERT INTO activities (`ActivityID`, `Name`) VALUES (1, 'Swimming')
```

MySQL said:

```
#1062 - Duplicate entry '1' for key 'PRIMARY'
```

2	Inserting a non-existent FK value into the UserID field	INSERT INTO `parents`(`ParentID`, `UserID`) VALUES (NULL,1000)	Referential	PASS

Error

SQL query:

```
INSERT INTO `parents`(`ParentID`, `UserID`) VALUES (NULL,1000)
```

MySQL said:

```
#1452 - Cannot add or update a child row: a foreign key constraint fails
```

3	Trying to update a value so that it is the same as another records value; therefore, violating the unique constraint placed	UPDATE `users` SET `Email` = 'vv@vv.vv' WHERE `users`.`UserID` = 254	(UNIQUE constraint)	PASS

Error

SQL query:

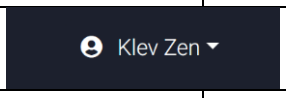
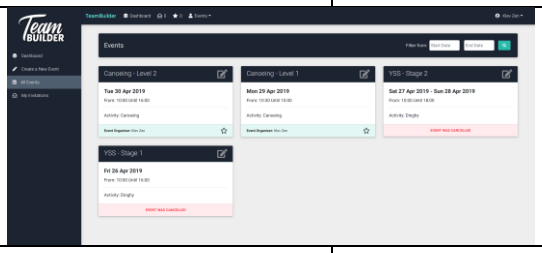
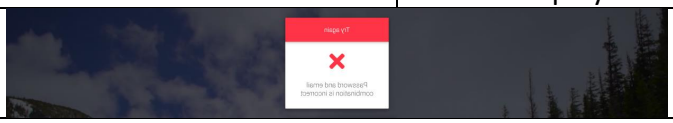
```
UPDATE `users` SET `Email` = 'vv@vv.vv' WHERE `users`.`UserID` = 254
```

MySQL said:

```
#1062 - Duplicate entry 'vv@vv.vv' for key 'Email'
```

6.3 Requirements Testing

The terms functional testing and black box testing are one in the same and refer to a particular software testing technique whereby a primary focus is placed on testing the system against its functional requirements. Tests are crafted in such a way that they do not look at how the output was produced by the system but instead solely focus on whether or not It matches the what was expected. The following test cases will consist of a test id, functional requirement ID, a scene, an expected result and an actual result. These will also be accompanied by a screenshot to illustrate the test case.

ID	FK ID	SCENE	EXPECTED	ACTUAL
1	1	User logs on	Their name is displayed in the nav	PASS
				
2	1	User views all events	All events are displayed	PASS
				
3	1	User types in wrong credentials	Error message is displayed	PASS
				

7. Critical Review and Conclusion

7.1 Introduction

As the project has now been concluded, this chapter will be primarily focusing on describing the hardships that were faced, feats that were accomplished and any potential improvements that could be adopted by the system.

7.2 Achievements

The created project is a responsive, flexible user-friendly application that has strived to meet all of the client's requirements and has successfully met the majority of the desired functionality. The project greatly benefitted from all of the interviews and meetings that were had with the client each of which drew the project ever closer towards achieving the initially defined goals and objectives outlined in the introduction chapter. Since the primary focus of this system was its flexibility and responsiveness for its various different user types, the fact that the information is easily accessible and that it can run seamlessly on a multitude of different devices without an issue speaks to the degree of which it has successfully met the its intended aims and objectives.

The design chapter has put the complex ERD created to good use and has laid out a very strong foundation from which an event better system could be built upon provided that the timeframe and technological expertise of the developer allow for it.

The agile methodology adopted by the system proved to be an invaluable asset to the project and has helped keep the project on track through timeboxing techniques and iterative development; making sure that the client was kept in the loop at all stages of the project so as to not diverse off track and end up wasting time on features that were not wanted or required by the client in the first place. This was further corroborated by the strong sense of communication that was built with the client throughout the entirety of the project's life cycle.

7.3 Future work

Various unforeseen issues had occurred on multiple occasions during this project the majority of which having to do with understanding and integrating all of the different API's and libraries together in an attempt to get them working in unison. One particular example was when the developer was attempting to incorporate a JavaScript Promise Library called bluebirdjs. Promises are essentially glorified call-back functions and work in much the same way; however, it was not very accommodating for a new developer who did not have much prior experience with using and manipulating asynchronous JavaScript functions so there is definitely still a lot of room for improvement and growth for both the project and the team tasked with developing and overseeing it.

Following on from this there are a few changes that would provide a substantial benefit to the existing project but could not be implemented or utilised during this project life cycle and they are listed below as follows:

1. Introduction of an admin user to the system – With an admin user, the assignment of E.O along with the overall management of the systems day to day operations become simpler as there is a single definitive authoritative user group that all of the other user groups and look to should an issue with the system arise. Furthermore, the created ERD allows for an admin user so it is a shame that it could not be accommodated for during this project.

2. Integrate the google maps API so that scouts, volunteers and general attendee's can view the area that the even is taking place in real time rather than tabbing out of their browser to go and search the post code or address manually themselves. This would make for a more pleasant user experience.

3. Moving from the Kingston university servers in favour of a paid web host – this would be ideal as the current Kingston university servers, upon which the application is hosting on, doesn't allow the modification of a file known as "htaccess" therefore the application is stymied in terms of its security potential right off the bat. Some of the various benefits of being able to amend the "htaccess" file is the addition of the HTTPS protocol and how it can be added to the teambuilder application to make for an even more secure and robust system.

4. Restricting sign ups to only desired individuals – this would be a better approach to the system as opposed to what's currently in place as it allows the client to better manage who is using the application at any given time. This functionality shouldn't be too difficult to implement and could be done through incorporating some form of code word that the E.O or Admins know and can change and could be set as a requirement to enter the code word correctly in order to sign up for a new account.

5. Prevent direct access to php files – this is another security feature that the system as it is right now would greatly benefit from as it can greatly improve the security and, in turn, reliably of the system.

7.4 Conclusion

This project has been an invaluable and extremely insightful journey. Every moment of which felt as though it was a new learning experience and after having achieved all of the desired system goals the project is now ready to be used as a basis for different applications of a similar nature.

Bibliography

[1] A Tajpour, S. Ibrahim and M. Masrom, "SQL Injection Detection and Prevention Techniques, "International Journal of Advancements in Computing Technology, vol. 3, pp. 82-91, 2011

[2] Codd, Edgar F (June 1970). "A Relational Model of Data for Large Shared Data Banks". Communications of the ACM. 13 (6): 377–87.

[3]https://www.ibm.com/support/knowledgecenter/SSGU8G_11.50.0/com.ibm.virtapp.doc/TD_item3.htm. [Accessed 20 Mar. 2019].

[4] J.Clarke, SQL Injection Attacks and Defense, Elsevier, 2009.

[5] H.William G. J and O. Alessandro, "A Classification of SQL Injection Attacks," in International Symposium on Secure Software Engineering 2006

[6] Segue Technologies. (2018). *Waterfall vs. Agile: Which Methodology is Right for Your Project?* [online] Available at: https://www.seguetech.com/waterfall-vs-agile-methodology/ [Accessed 14 Jan. 2019].

APPENDIX A – User Stories

Scout

Jake is a 12-year-old sea scout who has attended multiple dinghy events hosted by Ajax Sea Scouts in the past. During his most recent event, Jake successfully met the required criteria and was awarded a level 1 qualification for dingy sailing. He now wants to attempt the level 2 qualification for dinghy sailing so he logs into Teambuilder and navigates to the events page to see if there are any “Dinghy – level 2” events scheduled in the near future. As he is looking through the events page, he finds a “kayaking – level 1” event that piques his interest. Jake decides to click on the event so that he can read obtain some additional information about the event. Whilst on the “more details” page, Jake sees that his favourite instructor is going to be teaching this course too. Jake decides that he wants to give it a try so he clicks the small star by the event in order to express his interest to that events E.O. and then continues his search for a “dinghy – level 2” event until the E.O. invites him to the “kayaking - level 1” event. Once Jake receives an invite to the “kayaking – level 1” event, he navigates to his invitations page in order to respond to the invite by either accepting or declining.

APPENDIX B – Functional Requirements

KEY:
ALL = All user groups, **E.O** = Event Organisers, **D.O** = Duty Officers,
P = Parents, **S** = Scout, **VOL** = Volunteer, **THEMSELVES** = The current user logged in

ID	REQUIREMENT	LEVEL	USER GROUPS
1	Log into the system	MUST	ALL
2	Register for an account	MUST	ALL
3	Create a new event	MUST	E.O
4	Edit existing event	MUST	E.O
5	Create a new activity	MUST	E.O
6	Create a new duty officer	MUST	E.O
7	Assign a duty officer to an event	MUST	E.O
8	Invite volunteers to an event	MUST	E.O
9	Filter volunteers based on their qualifications	MUST	E.O
10	Filter scouts based on their qualifications	MUST	E.O
11	Invite scouts to an event	MUST	E.O
12	Confirm event attendance	MUST	ALL
13	Decline event attendance	MUST	ALL
14	Filter events based on a to / from date	SHOULD	ALL
15	Cancel an already accepted invitation	COULD	ALL
16	Remove an invited scout from an event	SHOULD	E.O
17	Remove an invited volunteer from an event	SHOULD	E.O
18	Update a scout's qualification	SHOULD	D.O,E.O
19	Import Scouts	WON'T	E.O
20	Export Scouts	WON'T	E.O
21	Import Volunteers	WON'T	E.O
22	Export Volunteers	WON'T	E.O
23	Search scouts during invite stage	MUST	E.O
24	Search volunteers during invite stage	MUST	E.O
25	Request to add a qualification to their account	SHOULD	VOL
26	Accept a volunteer's qualification request	COULD	E.O
27	View all the qualifications of a particular volunteer	COULD	E.O, THEMSELVES
28	View all the qualifications of a particular scout	COULD	D.O,E.O, THEMSELVES
29	View the official, confirmed team for an event	MUST	ALL
30	View all events	MUST	ALL
31	View more details about an event	MUST	ALL

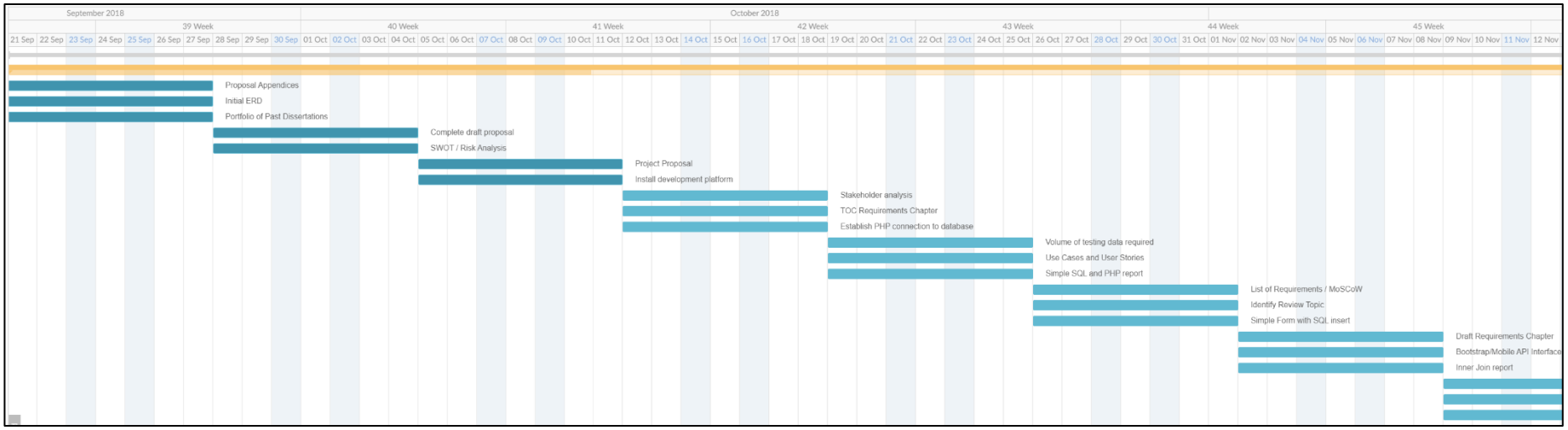
32	Select days available along with preferred roles	MUST	VOL
33	Replace volunteers assigned to a team with another vol	COULD	E.O
34	Show interest in a particular event	SHOULD	ALL
35	View the events that their scout has accepted	COULD	P
36	Delete an event	WON’T	E.O
37	Drop out of an accepted shift	COULD	VOL
38	Ask for a shift role or shift time change	COULD	VOL
39	View the events that their scout is interested in	COULD	P
40	View all personal event invitations	MUST	ALL
41	Change your account details	COULD	ALL
42	Email on event invite	SHOULD	E.O
43	Email on event cancelled	SHOULD	E.O
44	Email on invite declined	COULD	ALL
45	Log out of the system	MUST	ALL
46	View my days available for the week	MUST	VOL
47	View the day and role that a confirmed volunteer is working on for a particular event	SHOULD	ALL
48	View a calendar schedule of your confirmed shifts	MUST	VOL
49	Apply to become a volunteer	COULD	P
50	View all of the users that have shown interest in an event	COULD	E.O
51	View volunteer invite responses to a particular event	MUST	E.O
52	View scout invite responses to a particular event	MUST	E.O
53	View last attended event and most recent event	MUST	S,P

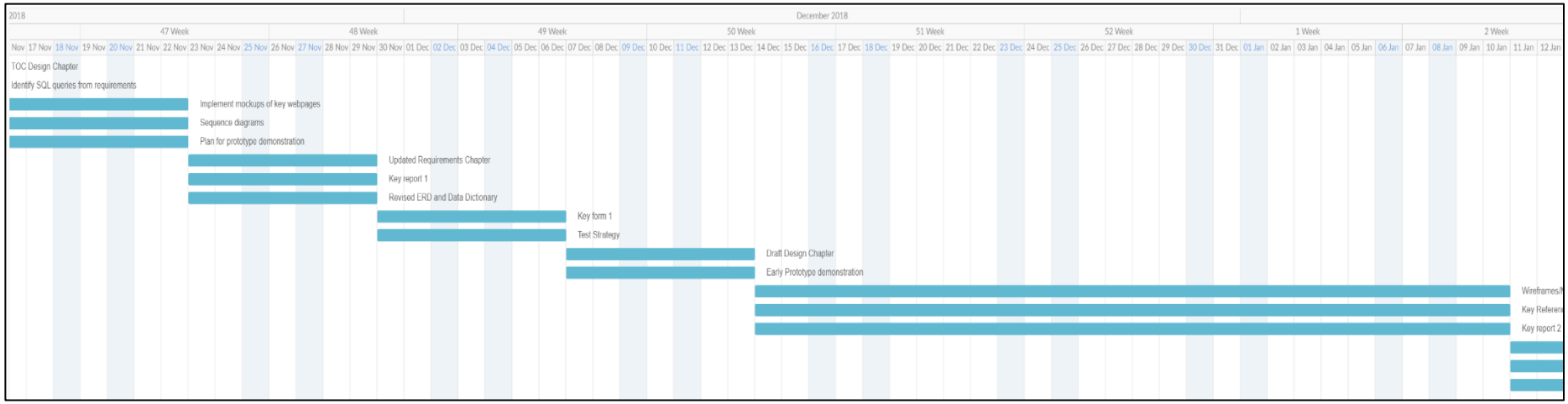
APPENDIX C – Schedule

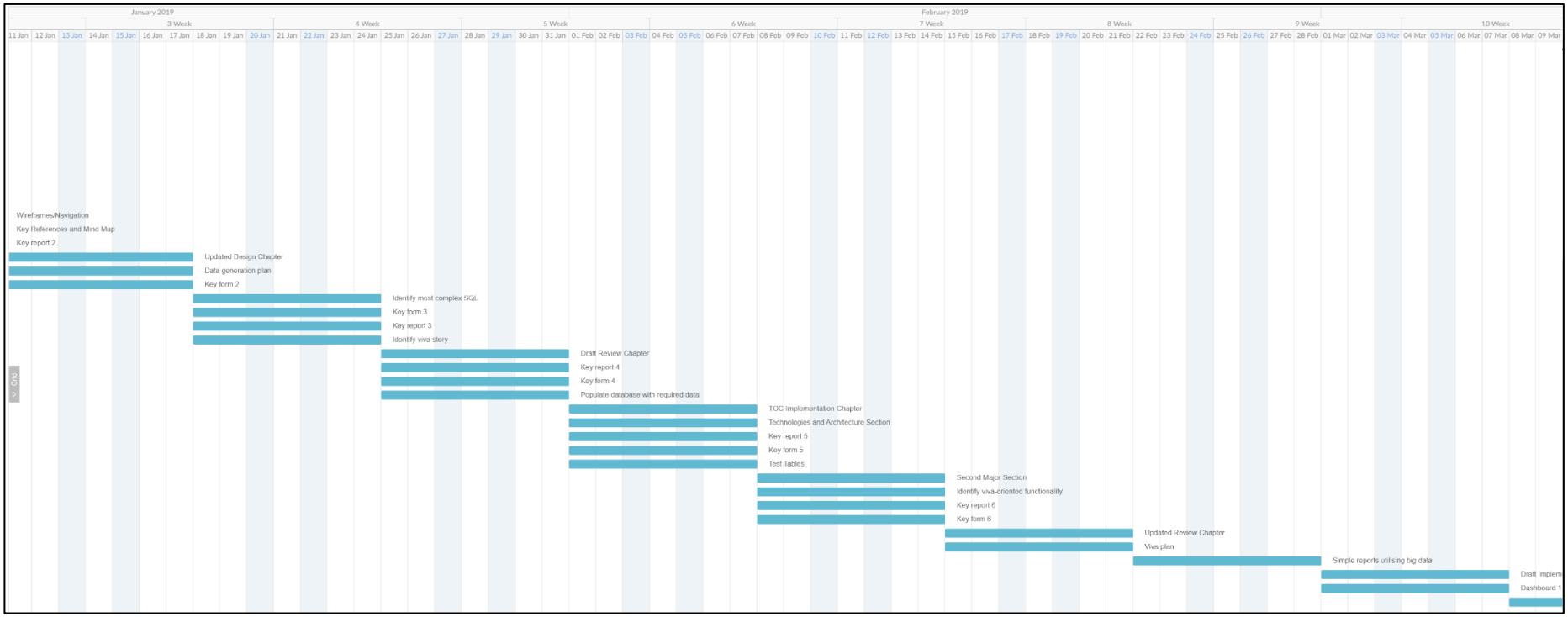
The project had to follow a strict timetable (*Table Below*) of weekly deliverables at all times in order to monitor and assess how the creation of the system was progressing. These weekly deliverables were established together with the client through multiple meetings and encounters and were then converted into a Gantt chart (*Figure 1*) that highlighted the major development phases.

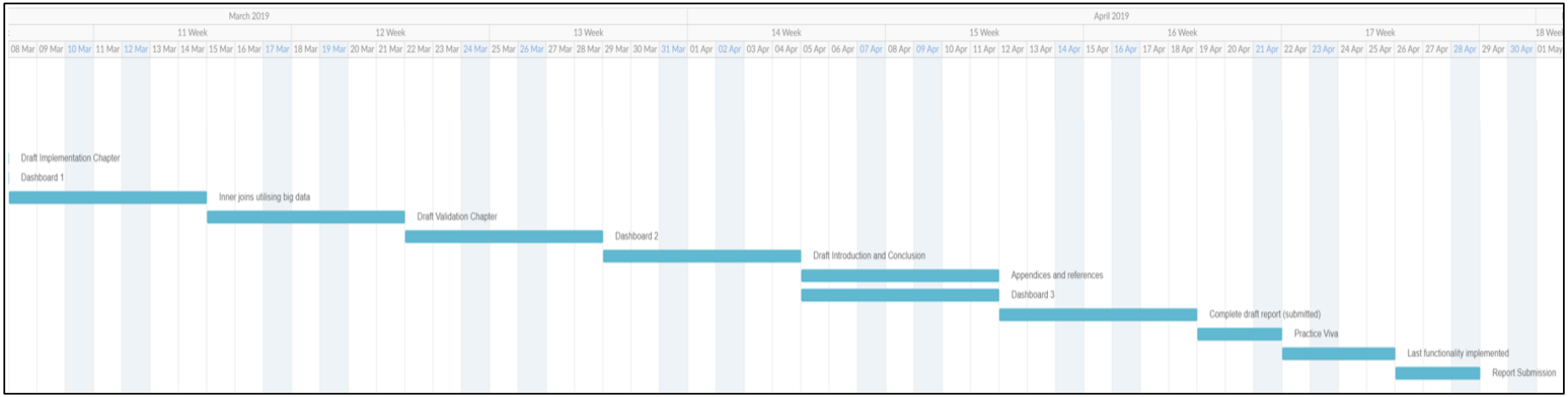
Date	***Deliverable***	
28/09/2018	Documentation	Proposal Appendices
28/09/2018	Methodology	Initial ERD
28/09/2018	Research	Portfolio of Past Dissertations
05/10/2018	Documentation	Complete draft proposal
05/10/2018	Methodology	SWOT / Risk Analysis
12/10/2018	**Documentation**	**Project Proposal**
12/10/2018	Implementation	Install development platform
19/10/2018	Methodology	Stakeholder analysis
19/10/2018	Documentation	TOC Requirements Chapter
19/10/2018	Implementation	Establish PHP connection to database
26/10/2018	Research	Volume of testing data required
26/10/2018	Methodology	Use Cases and User Stories
26/10/2018	Implementation	Simple SQL and PHP report
02/11/2018	Methodology	List of Requirements / MoSCoW
02/11/2018	Research	Identify Review Topic
02/11/2018	Implementation	Simple Form with SQL insert
09/11/2018	Documentation	Draft Requirements Chapter
09/11/2018	Research	Bootstrap/Mobile API Interface
09/11/2018	Implementation	Inner Join report
16/11/2018	Methodology	Sketches
16/11/2018	Documentation	TOC Design Chapter
16/11/2018	Research	Identify SQL queries from requirements
23/11/2018	Implementation	Implement mockups of key webpages
23/11/2018	Methodology	Sequence diagrams
23/11/2018	Research	Plan for prototype demonstration
30/11/2018	Documentation	Updated Requirements Chapter
30/11/2018	Implementation	Key report 1
30/11/2018	Methodology	Revised ERD and Data Dictionary
07/12/2018	Implementation	Key form 1
07/12/2018	Research	Test Strategy
14/12/2018	Documentation	Draft Design Chapter

14/12/2018	**Presentation**	**Early Prototype demonstration**
11/01/2019	Methodology	Wireframes/Navigation
11/01/2019	Research	Key References and Mind Map
11/01/2019	Implementation	Key report 2
18/01/2019	Documentation	Updated Design Chapter
18/01/2019	Research	Data generation plan
18/01/2019	Implementation	Key form 2
25/01/2019	Research	Identify most complex SQL
25/01/2019	Implementation	Key form 3
25/01/2019	Implementation	Key report 3
25/01/2019	Research	Identify viva story
01/02/2019	Documentation	Draft Review Chapter
01/02/2019	Implementation	Key report 4
01/02/2019	Implementation	Key form 4
01/02/2019	Evidence	Populate database with required data
08/02/2019	Documentation	TOC Implementation Chapter
08/02/2019	Documentation	Technologies and Architecture Section
08/02/2019	Implementation	Key report 5
08/02/2019	Implementation	Key form 5
08/02/2019	Evidence	Test Tables
15/02/2019	Documentation	Second Major Section
15/02/2019	Research	Identify viva-oriented functionality
15/02/2019	Implementation	Key report 6
15/02/2019	Implementation	Key form 6
22/02/2019	Documentation	Updated Review Chapter
22/02/2019	Research	Viva plan
01/03/2019	Evidence	Simple reports utilising big data
08/03/2019	Documentation	Draft Implementation Chapter
08/03/2019	Implementation	Dashboard 1
15/03/2019	Evidence	Inner joins utilising big data
22/03/2019	Documentation	Draft Validation Chapter
29/03/2019	Implementation	Dashboard 2
05/04/2019	Documentation	Draft Introduction and Conclusion
12/04/2019	Documentation	Appendices and references
12/04/2019	Implementation	Dashboard 3
19/04/2019	Documentation	Complete draft report (submitted)
22/04/2019	Presentation	Practice Viva
26/04/2019	Implementation	Last functionality implemented
29/04/2019	**Documentation**	**Report Submission**


September 2018
October 2018
39 Week
40 Week
41 Week
42 Week
43 Week
44 Week
45 Week
Proposal Appendices
Initial ERD
Portfolio of Past Dissertations
Complete draft proposal
SWOT / Risk Analysis
Project Proposal
Install development platform
Stakeholder analysis
TOC Requirements Chapter
Establish PHP connection to database
Volume of testing data required
Use Cases and User Stories
Simple SQL and PHP report
List of Requirements / MoSCoW
Identify Review Topic
Simple Form with SQL insert
Draft Requirements Chapter
Bootstrap/Mobile API Interface
Inner Join report


2018
December 2018
47 Week
48 Week
49 Week
50 Week
51 Week
52 Week
1 Week
2 Week
TOC Design Chapter
Identify SQL queries from requirements
Implement mockups of key webpages
Sequence diagrams
Plan for prototype demonstration
Updated Requirements Chapter
Key report 1
Revised ERD and Data Dictionary
Key form 1
Test Strategy
Draft Design Chapter
Early Prototype demonstration

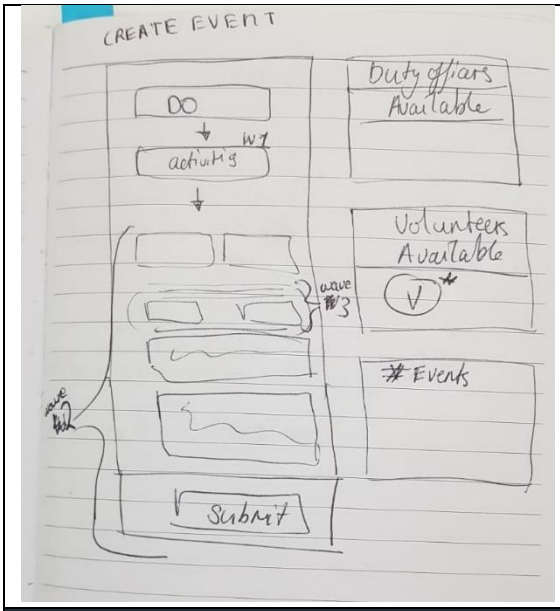

January 2019
February 2019
3 Week
4 Week
5 Week
6 Week
7 Week
8 Week
9 Week
10 Week
Wireframes/Navigation
Key References and Mind Map
Key report 2
Updated Design Chapter
Data generation plan
Key form 2
Identify most complex SQL
Key form 3
Key report 3
Identify viva story
Draft Review Chapter
Key report 4
Key form 4
Populate database with required data
TOC Implementation Chapter
Technologies and Architecture Section
Key report 5
Key form 5
Test Tables
Second Major Section
Identify viva-oriented functionality
Key report 6
Key form 6
Updated Review Chapter
Viva plan
Simple reports utilising big data

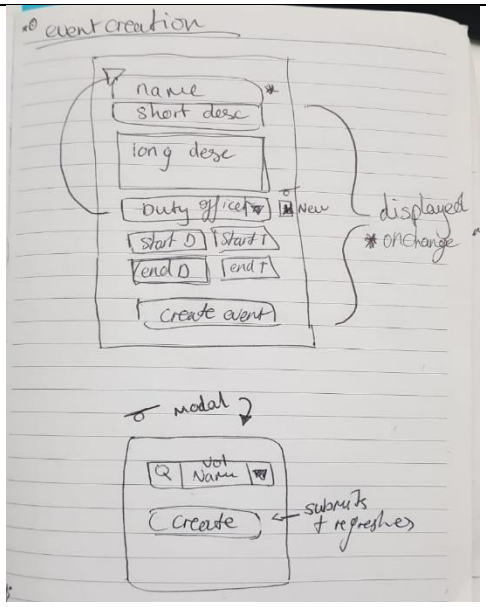

March 2019
April 2019
11 Week
12 Week
13 Week
14 Week
15 Week
16 Week
17 Week
Draft Implementation Chapter
Dashboard 1
Inner joins utilising big data
Draft Validation Chapter
Dashboard 2
Draft Introduction and Conclusion
Appendices and references
Dashboard 3
Complete draft report (submitted)
Practice Viva
Last functionality implemented
Report Submission

APPENDIX D – Data Dictionary

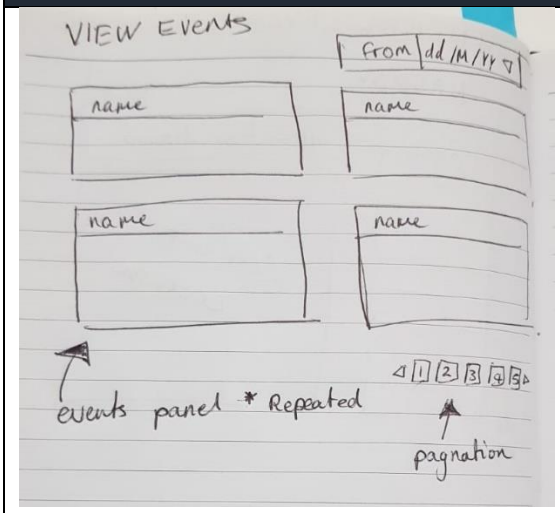
(Table Name) Relation Name	PK/FK	*(Field Name)* Attribute Name	Data Type & length	NOT NULL	Further Constraints
Users	PK	UserID	INT(5)	•	
	FK	RightsID	INT(1)	•	
		Email	VARCHAR(255)	•	UNQIUE
		Password	VARCHAR(64)	•	
		Firstname	VARCHAR(30)	•	
		MIddlename	VARCHAR(30)		
		Lastname	VARCHAR(30)	•	
		Dob	DATE		
		Gender	VARCHAR(6)	•	CHECK (Gender= 'Male' OR Gender= 'Female')
		Mobile	INT(11)		
		Address	VARCHAR(200)		
		Postcode	VARCHAR(8)	•	
		DateJoined	DATETIME		
		IsVolunteer	CHAR(1)	•	
		IsScout	CHAR(1)	•	
		IsParent	CHAR(1)	•	

APPENDIX E – Wireframes

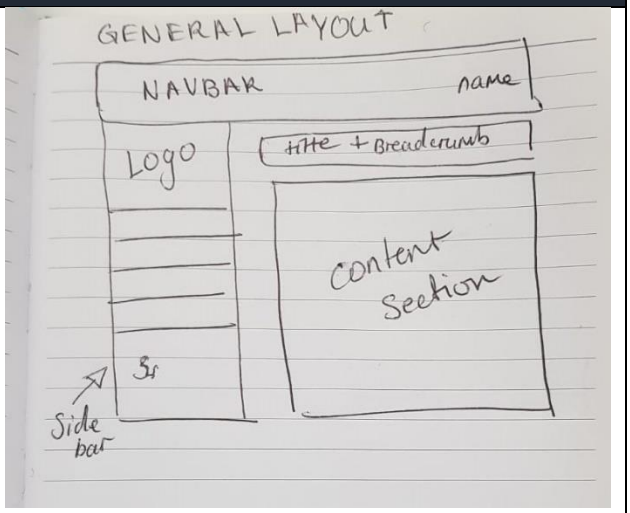




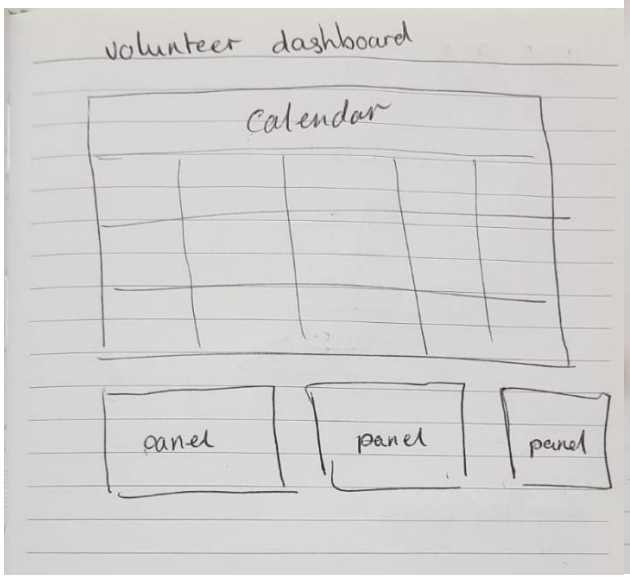
Create an Event

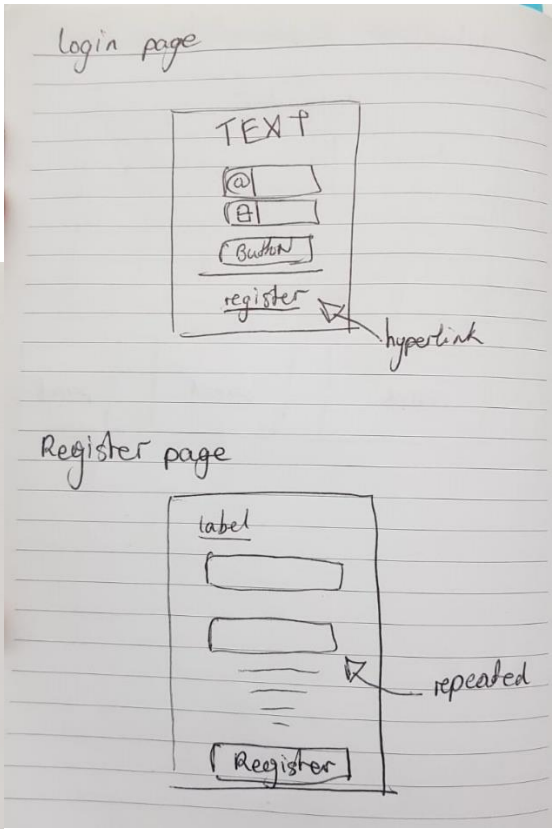


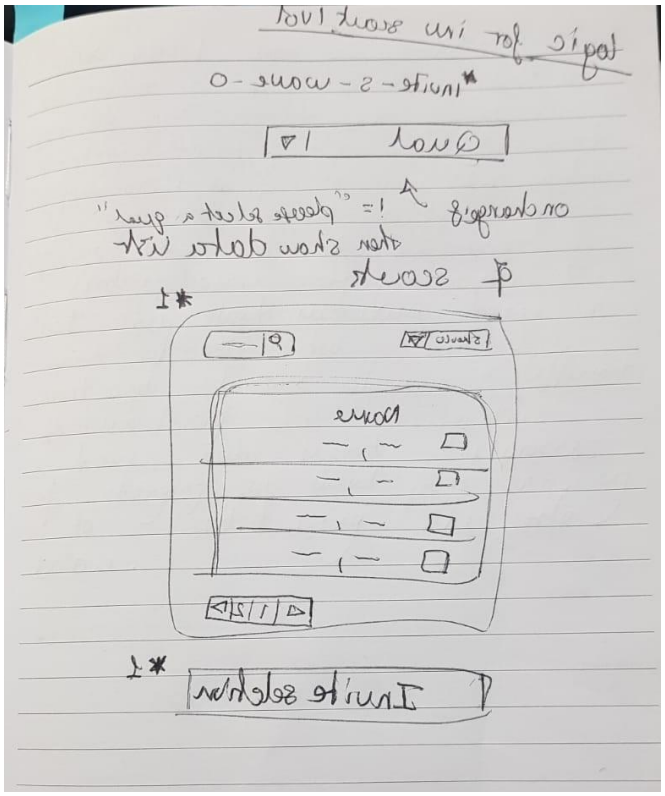
VIEW ALL EVENTS

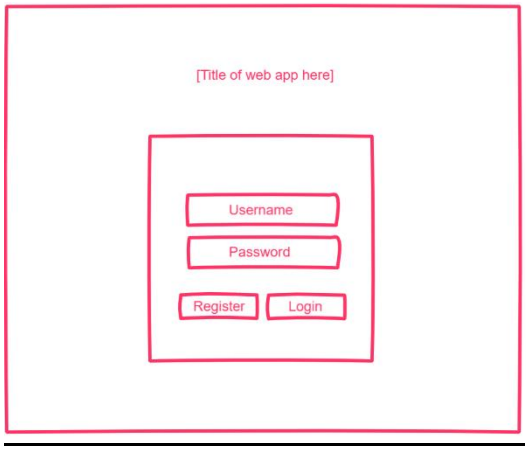


GENERAL DASHBOARD LAYOUT

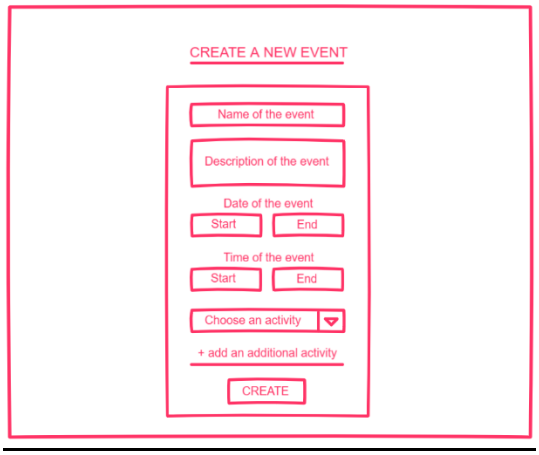

volunteer dashboard
Calendar
panel
panel
panel


login page
TEXT
@
Button
register
hyperlink
Register page
label
repeated
Register

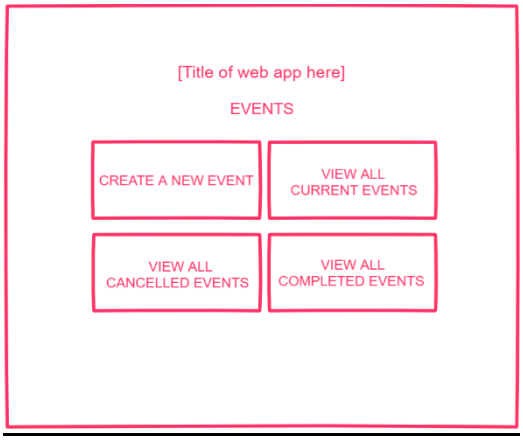

Invite selection
Home
Search
Search
Invite - 3 - more - 0
Quit



LOG IN PAGE



CREATE EVENT



CONTENT SECTION – E.O.

REGISTER PAGE- ACCOUNT TYPE