

# BSc DEGREE IN

*Computer Science*

## PROJECT REPORT

Name: Orsolya Toro

ID Number: K1916789

Project Title: Exploring modern web application development  
for CISSE UK

Project Supervisor: Graeme Jones

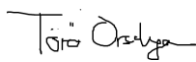
Please complete your chosen Mark Allocation Model below. The “fixed” sections may not be changed. In the other sections, you may choose only ONE of them to assign only one (1) single point. Half points may not be allocated. The total must add up to 10 and the two fixed sections may not be changed. If this is not completed or not completed correctly, the default model will be used. See Canvas and lecture notes for more information about Mark Allocation Models.

Section	Points Allocated
Introduction and Literature Review	2
Methodology and Analysis	2
Design	2
Prototype/Implementation	2
Validation and Testing	1 (fixed)
Critical Review and Conclusion	1 (fixed)
Total	10

### Declaration

I have read and understood the University regulations on academic misconduct and I understand the meaning of the word *plagiarism* and the university's definition of the word. 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 individual's work and presented it as my own. No other individuals have contributed to this work in the form of written prose, code, drawings or other figures. I did not allow and will not allow anyone to copy my work with the intention of presenting part or all of it as their own.

Date: 11/04/2022

Signature: 

## Contents

1	Introduction & Background .....	4
1.1	Background .....	4
1.2	Problem .....	4
1.3	Aims and Objectives .....	4
1.4	Solution .....	4
1.5	Methodology .....	4
1.6	Ethics, data security and protection .....	5
2	Literature review .....	6
2.1	Introduction .....	6
2.2	Development .....	6
2.2.1	Front-end Architecture .....	6
2.2.2	Back-end Architecture .....	6
2.3	Deployment .....	7
2.4	Conclusion .....	8
3	Requirements Analysis .....	9
3.1	Introduction .....	9
3.2	Stakeholder Analysis .....	9
3.2.1	Key stakeholders .....	9
3.2.2	Power Interest Matrix .....	10
3.3	Use Cases .....	10
3.3.1	Use Case Diagram .....	10
3.3.2	Brief Use Case Descriptions .....	12
3.4	User stories, Functional Requirements and Non-Functional Requirements .....	13
3.4.1	User Stories .....	13
3.4.2	Functional Requirements and MoSCoW analysis .....	13
3.4.3	Non-Functional Requirements .....	15
3.5	Risk Analysis .....	15
3.6	Summary .....	16
4	Design .....	17
4.1	Introduction .....	17
4.2	Database Design .....	17
4.2.1	ERD Entity Relationship Diagram .....	17
4.2.2	Data Dictionary .....	20
4.3	Interface Design .....	22
4.3.1	Sketches .....	22

4.3.2	Reusable components and Wireframes.....	23
4.3.3	Site Map .....	27
4.4	Sequence Diagrams.....	28
4.5	Activity Diagrams .....	29
4.6	Conclusion.....	30
5	Implementation .....	31
5.1	Introduction .....	31
5.2	Tooling.....	31
5.3	Architecture .....	32
5.4	Frontend implementation.....	33
5.4.1	Reusable Components .....	34
5.4.2	GraphQL .....	38
5.5	Backend implementation.....	40
5.5.1	Amazon Web Services Amplify .....	40
5.5.2	Databases.....	42
5.6	Conclusion.....	45
6	Validation and Testing .....	46
6.1	Introduction .....	46
6.2	Database testing .....	46
6.3	Functional Testing (Manual testing) .....	48
6.4	Automated testing .....	49
6.5	Conclusion.....	50
7	Critical Review and Conclusion .....	51
7.1	Introduction .....	51
7.2	Outcome .....	51
7.3	Future Development.....	51
7.4	Conclusion.....	52
8	References .....	53
9	Appendices.....	55
9.1.1	Appendix A – Reusable components wireframes .....	55
9.1.2	Appendix B – Wireframes .....	58
9.1.3	Appendix C – Sequence Diagrams.....	63
9.1.4	Appendix D - GraphQL Schema .....	65
9.1.5	Appendix E. – Create table statements.....	67

# 1 Introduction & Background

## 1.1 Background

This project's client is the co-founder of CISSE UK (Colloquium for Information Systems Security Education UK) Dr Charles Clarke. They focus on educating people about cyber security, raising awareness of, and collaborating on, cyber security topics. They host several events for professionals and students. Their vision is "To establish a culture of outstanding, innovative and state-of-the-art security education in the UK" (CISSE UK, 2019). Currently, they have a web application platform which is created with a third-party website builder. They would like to try to explore a more modern web application that better suits their needs.

## 1.2 Problem

The client's problem arises as they do not have a web application that would allow them to support their needs. They need a system designed to manage events for cybersecurity talks. The existing tools that they use, result in a poor user experience, and limits their ability to manage events in a more efficient way. They cannot currently reach out to many people to educate and raise awareness about cyber security. Their current digital tools limit their potential to organise and run events and encourage people to participate. It also results in a lower engagement with the community. This project will try to find a solution to these issues.

The project will focus on creating a modern web application, that is easy to use, is easily adaptable, and meets their needs. It will also use a purpose-built tool that would simplify the process of managing and deploying the whole application. By implementing all these modern technologies and tools, they will be able to concentrate more on: creating and managing events; and focusing more on encouraging people to collaborate.

## 1.3 Aims and Objectives

The main aim of this project is to review modern web technologies and tools that would help to solve the client's problems, and to implement a fully functional website that uses these modern tools and technologies.

This project's objectives are to accomplish the aims that are listed below:

1. To research modern web technologies and tools.
2. To identify requirements and prioritise them.
3. To sketch and wireframe the web application.
4. To design a useful modern web architecture that the client could benefit from.
5. To implement a web application that the client could use to manage their events and users efficiently.
6. To test the platform to verify that the application works as anticipated.
7. To evaluate and reflect on the project.

## 1.4 Solution

The proposed solution for the client will be a modern event management web application that uses an easily maintainable architecture. The web application will help manage events, help manage their members, and invite members to their events.

## 1.5 Methodology

This project will be using Agile style methods like Scrum. As this project has a client without certain requirements, who is operating an organisation in a changing environment, the Waterfall approach

would not be the best method. This is because it cannot adapt well to changing client needs (Sherman, 2014). It will likely take many meetings and checking back with the client frequently to understand what their vision is for the application. Agile methods would reduce the risk associated with eliciting requirements and building something they do not want.

The project will have twenty-four iterations throughout the year. Some more simple features will only take one week of iteration, but some more complex ones will take multiple weeks of iteration. Once every week there will be a meeting with the supervisor to discuss: what has been done; obtain feedback on the work completed; resolve any issues and discuss what is planned for the following week. A meeting with the client will be scheduled every two weeks and will cover the same topics but with a different perspective.

## 1.6 Ethics, data security and protection

This project will not store any real personal data that would be protected under the GDPR regulation. Any data that will be stored in the database will be only dummy data. This reduces the security risks and prevents any leaks of personal data. This project does not require real personal data for development and testing.

To mitigate any possible security risks, data will be encrypted through HTTPS when entering details into a form on the application. All data that will be stored in the database will also be encrypted and stored on Amazon Web Services (AWS) for additional security.

## 2 Literature review

### 2.1 Introduction

In this chapter, a set of technologies and architectures that are used in the creation of modern web applications will be discussed. Additionally, the comparison between more traditional technologies and newer modern technologies will be considered. The section will also evaluate the positives and negatives of using these modern technologies versus their traditional counterparts.

### 2.2 Development

This section focuses on the architectural styles used for web application development. It covers the front-end and the back-end architecture.

#### 2.2.1 Front-end Architecture

There are two common ways of building web application frontends. One is traditional web applications; the other is single page web applications.

##### 2.2.1.1 *Traditional Web Page Applications*

A traditional web application architecture is often called a monolith. All the application code lives in the same system and is tightly coupled together as a set of components. They also share the same server and file system. There are some advantages of this style: development can be simple, and debugging can be easier as everything is in one place. There are some drawbacks to this architecture, however. Changing and maintaining this application can be challenging because all the services are tightly intertwined (Dushenin, 2021). For example, if a change has been made to one part of the codebase it can easily break other interrelated parts. Also, if someone joins the team as a developer it can take a long time to understand the whole system and become productive (*Microservices.io*, no date).

##### 2.2.1.2 *Single Page Applications*

Prior to single-page applications, dynamic and traditional websites required the page to refresh to display relevant data. In that approach, the client sends the request to the server then the server responds to the client with an HTML page. Each time the clients want to see a different page the entire page has to be reloaded and the request-response trip has to be done each time using up bandwidth. In a single page application, only the relevant part of the page needs to be re-rendered. This is achieved by Javascript fetching the data asynchronously and then re-rendering part of the page each time something changes (Dopico, 2020). This way there is no need for multiple trips to the server to display a new look on the page. Single-page applications can offer an improved user experience for the end-user but perform poorly on low-bandwidth connections and can be more difficult for search engines to crawl and index (Fink, Flatow and Group, 2014).

#### 2.2.2 Back-end Architecture

Web applications interact with backends, to produce more meaningful systems and process data. There is a traditional way of doing things and a way that uses service-oriented architecture and APIs

##### 2.2.2.1 *Service-Oriented Architecture and Microservices*

In service-oriented architectures, applications are turned into services that are all separated and treated as their own application (Microsoft, 2021). They then work together as part of the wider application. Microservices is service-oriented architecture just with smaller services. Each service may be looked after by different teams in an organisation. Each application has its server, and they communicate with each other via APIs. (Richardson, 2018) One of the advantages of service-oriented architectures is that they are easier to scale up application infrastructure as it does not need to scale

up the whole application, just the problematic service (Richardson, 2018). That could be useful when a website's traffic is unpredictable or very high load. For instance, if it had a rush of people signing up, the authentication service can be scaled up. Testing individual parts of the system is also easier as changing one part would not affect other parts of the whole application system (Richardson, 2018 ch. 1, para. 1.5.1.) There is a significant drawback to this architecture, however, as it is more time-consuming to build as it is more complex and it requires more resources, so it is more costly. It is not the best choice for a small application or organisation for these reasons. (Richardson, 2018 ch. 1, para. 1.5.2.)

#### 2.2.2.2 REST APIs

REST APIs are often used in web applications for data exchange between the web application and the server. REST APIs stand for Representational State Transfer Application Programming Interfaces. They easily facilitate communication between a client and server using HTTP Concepts (Juviler, 2021). There are some key benefits of using REST APIs for communication. First, it is simple to use and standardised with commonly understood interfaces based on HTTP (*REST API Tutorial, 2022*). They are also scalable, so if the services grow, they can be easily modified according to that. They are also stateless and can also support caching of GET requests which can enable high performance (*REST API Tutorial, 2022*).

REST has two main components, one is the request message, and the other is the response. The client requests something from the server and the server responds with a response message for the client. The request typically has four main parts. The first is the header, the second is the HTTP operation, the third is the endpoint and the fourth is the body or parameter (IBM Cloud Education, 2021). The response from the server usually is a JSON object or XML it works similarly to CRUD in the database system. CRUD stands for Create, Read, Update and Delete. REST APIs use HTTP operations and methods. Create is equal to the POST method, Read is equal to GET request, Update is equal to the PUT method and Delete is equal to DELETE (IBM Cloud Education, 2021).

#### 2.2.2.3 GraphQL

GraphQL is another way for clients to interact with service APIs. GraphQL is a data query language for APIs and a specification. One of the main parts of a GraphQL application is the schema which describes the data that the client can ask for from the API (Stemmler, 2021). Queries are another main part, that specifies what the client wants from the API. Compared to other APIs, there is only a need to specify one endpoint location and after that, the client can specify what exact data they want to bring back from the server, which includes bringing data back from multiple locations (Stemmler, 2021). Using GraphQL can reduce the number of requests to the server and also the amount of data that comes back as it is all relevant to the client, this makes it much faster and more efficient, hence it is perfect for applications relying on low bandwidth and network quality (Linx, 2021). Similarly, to REST, GraphQL is not limited to any programming language or back-end framework, this gives developers a suitable abstraction with lots of flexibility. (Greif, 2017)

### 2.3 Deployment

This section focuses on the architectural styles used for deploying web applications.

#### 2.3.1.1 Cloud Computing

Cloud computing is having access to different computing resources (data storage, applications, servers etc.) via the internet whenever the user needs it (Vennam, 2020). There are two common ways of deploying web applications. One is to use containers and virtual machines; the other is to use serverless technologies.

#### 2.3.1.2 Containers and Virtual Machines

Both virtual machines and containers dominate the cloud computing domain upon which many web applications are built. Virtual machines have a different architecture compared to containers. The virtual machine base layer is the infrastructure (the hardware, the host operating system, etc.), on top of that there is a hypervisor that emulates hardware, the guest operating system, binaries, and libraries and finally the application (Schmitt, 2020). In containers, the architecture is a bit different. The base layer is the infrastructure, on top of that is the host operating system, the container engine and on top of that, we have the binaries, libraries and finally the application (Jones, 2018).

Containers are much more lightweight and therefore performant because each container does not create an instance of a guest operating system like in a virtual machine (Jones, 2018). They are also easily portable as they can be run on different platforms. There are some downsides to containers however because they share the same operating system/ kernel if the container engine goes down all the containers follow and would not work (Bigelow, 2015). The virtual machine, on the other hand, has its guest operating system, so if one of the guest operating systems goes down it would not affect the others. Another trade-off is containers share the operating system. Therefore, it is not possible to use different operating systems for different containers. This can be solved by combining the two architectures (Potdar *et al.*, 2020).

#### 2.3.1.3 Serverless Architecture

Serverless architecture is a rapidly evolving architecture utilised in modern web applications. Surprisingly, it does not mean there are no servers included in the system. It means a third-party provider looks after the servers. This way developers do not need to worry about the extra management of servers, and they can focus on other parts of the system (Fee, 2020). One of the most popular serverless architecture paradigms is the FaaS (Function as a Service) paradigm. This application style is where developers write their application as a set of functions that execute on third party providers (Govor, 2019). There are some significant benefits of using serverless architecture. As mentioned previously, developers do not need to worry about maintaining and looking after servers. It is easily scalable if more servers are needed for the extra load. The company only pays for the runtime of the servers which could save money in some cases. However, the cost associated with this flexibility can be quite considerable and the architecture might work out a more expensive per request for fixed high traffic applications (Jiang, Pei and Zhao, 2020).

## 2.4 Conclusion

In this chapter, some of the key aspects of modern web architecture technologies have been discussed. Many of them can be used alongside other more traditional pieces. When deciding on these technologies, it is important to look at the pros and cons and determine if they would be beneficial, and feasible and whether they would save money in the short or long run.



## 3 Requirements Analysis

### 3.1 Introduction

The below chapter discusses the stakeholders and the requirements that have been collected and created. Different techniques have been used, including stakeholder analysis, to help identify the stakeholders of this project. A power interest & matrix has been used to visualise the stakeholders' interests and power. Use case diagrams and use case descriptions have been created, to describe the system behaviours. User stories, functional requirements and non-functional requirements have also been listed with MoSCoW analysis to explain the scope of the system. Finally, a risk analysis has been included to understand the risks for the project.

### 3.2 Stakeholder Analysis

Stakeholder analysis has been done, to identify the main stakeholders for this project. They are either an individual or a group that has interest and or power in the project. Stakeholders that have more power over this project will tend to steer the results (Hoory, 2021). As the project is using an Agile methodology it is important to communicate with stakeholders regularly, mainly with the client to discuss their requirements. The requirements can also change throughout the project.

#### 3.2.1 Key stakeholders

There are a few key stakeholders.

##### **The client (CISSE UK)**

The client, CISSE UK, has high power and interest in this project. They wish to use it to make their website more efficient and find a better way to promote their site to the public and the cybersecurity professionals. They are the primary decision maker for this software and this project. Fundamentally they will also determine whether the application is used.

##### **Cybersecurity professionals**

They would have high interest in the project as they would use the site to either create and present their knowledge in different events or find events to attend and learn about. They have a medium level of power, as they can suggest different topics and ideas and will help determine the contents (the events) that the system displays. They could be a presenter for an event, an event organiser or just an attendee.

##### **University students**

In this project, they would have a high interest because they would use the website to find different events to attend and learn and network with other professionals. They would have low power with respect to the project as they would only use the site and they would not have much power to change things.

### 3.2.2 Power Interest Matrix

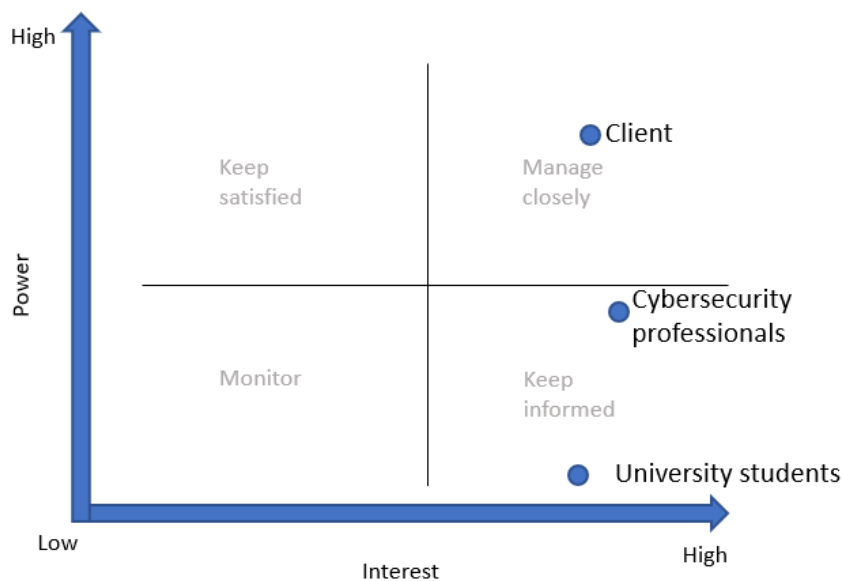


Figure 3.1. Power Interest Matrix

The above diagram shows the Power Interest Matrix that is used to visually show the key stakeholders' positions. There are four main sections, stakeholders in the low interest and low power quadrant must be monitored. They need little communication, however. Those in the keep satisfied quadrant have high power but low interest. They do not need too much communication, but they must be satisfied. The Keep informed section has stakeholders that have high interest but low power. These will be communicated with to keep them in the loop. The final section is the Manage closely, they have high interest and power. They must be communicated with regularly to keep them in the loop and to ensure they are satisfied (MindTools, no date).

## 3.3 Use Cases

### 3.3.1 Use Case Diagram

A use case diagram has been created to help to identify systems requirements and to show how the system would behave. It models the actors and the system and their interactions (*IBM Docs, 2021*). There are two use case diagrams that have been created, Figure 3.2. presents how the admin interacts with the system, Figure 3.3. presents how a user would interact with the system. These two actors were selected as they represent the two sides of the market that this application facilitates.

Figure 3.2 Use case diagram admin



Figure 3.3. Use case diagram user



### 3.3.2 Brief Use Case Descriptions

Use case descriptions have been created, to show the steps in greater detail, describing the actor interactions with the system. It describes the preconditions, success end conditions, failure conditions, the main actors, and the steps that the actor needs to take to achieve the use case. (Friedenthal, Moore, and Steiner, 2014, ch. 12, p. 12.4.2.)

Below are four use case descriptions.

Use Case 1		Add event
Preconditions	Admin is logged in	
Success end condition	Event added to the database, the event is visible	
Failed end condition	The event is not added to the database, the event is not visible	
Primary, secondary actors	Admin	
Description	Step	Action
	1	Click on create event tab
	2	Fill in all details about the event
	3	Click on submit button
	4	The system confirms successful completion

Use Case 2		Submit Presentation
Preconditions	The user is logged in	
Success end condition	Presentation added to the database, the contribution is visible	
Failed end condition	Presentation is not added to the database, contribution is not visible	
Primary, secondary actors	User	
Description	Step	Action
	1	Click on the event
	2	Click on add an event presentation
	3	Fill in presentation metadata
	4	Click on submit button
	5	The system confirms successful completion

Use Case 3		Register for an event
Preconditions	The user is logged in and, on the events page	
Success end condition	User attendance for an event is added to the database, a confirmation page is visible, email received	
Failed end condition	User registration for an event is not added to the database, the confirmation page is not visible, email is not received	
Primary, secondary actors	User	
Description	Step	Action
	1	Click on the particular event
	2	Click on the register button
	3	Fill in required details
	4	Click on submit button
	5	System redirects to the confirmation page

Use Case 4		View registered users for an event
Preconditions	Admin is logged in	
Success end condition	Registered users are visible	
Failed end condition	Users are not visible	
Primary, secondary actors	Admin	
Description	Step	Action
	1	Click on View event participants
	2	Select the particular event
	3	System shows the number and the list of participants

### 3.4 User stories, Functional Requirements and Non-Functional Requirements

All the requirements that have been collected from the client and the project have been listed in three different parts, user stories, functional requirements, and non-functional requirements.

#### 3.4.1 User Stories

A list of user stories has been written to help to see what the system needs to be able to do and it has been categorised by different users.

As an...	I would like to...	So that...
event organiser	add new events,	people can attend.
event organiser	edit events,	the correct information is displayed.
event organiser	alert future participants about event changes,	participants know the correct details.
event organiser	send out newsletters,	people would know about upcoming events.
event organiser	be able to delete events,	they would not show up.
event organiser	Upload a presentation	I can share it with the participants
presenter	earn contribution badges,	it shows I am an active participant.
presenter	present on different events,	I can educate participants.
user	edit my details (email, name, organisation, status),	my details are up to date.
user	be able to delete my profile,	my account no longer exists.
user	register to addend to events,	I can participate in events.
attendee	earn contribution badges,	it shows I am an active participant.
student	register to events,	I can addend.
student	earn contribution badges,	it shows I am an active participant.
professional	register to events,	I can addend.
professional	earn contribution badges,	it shows I am an active participant.

#### 3.4.2 Functional Requirements and MoSCoW analysis

After the user stories have been written, a list of functional requirements has been created. It has been written in verb – noun form. This is being used to help identify all the requirements that the system needs to facilitate. Every functional requirement has the description to elaborate on each function.

MoSCoW analysis has been used after listing all the functional requirements. This analysis helps rank each of the functional requirements to see which one is a must have, should have, or could have requirement. Must have requirements have to be implemented to ensure the success of the project; should have should be implemented but the success of the project is not heavily reliant on the implementation; could have, is an additional extra that can be done to make it more interesting but is not strictly necessary, Won't represent those requirements that have been identified that will not be implemented for this project (Agile Business Consortium, no date).

ID	Functional Requirements	Description	MoSCoW
1	Create an account	People must be able to create an account.	Must have
2	Log in/ Log out	All registered users must be able to log in or log out.	Must have
3	Reset password	All registered users should be able to reset their passwords.	Should have
4	Change personal details	All registered users must be able to change their name, email, organisation, and status.	Must have
5	Delete account	All registered users must be able to delete their accounts.	Must have
6	Register for an event	All registered users must be able to register for an event.	Must have
7	Add events	The administrator must be able to add events.	Must have
8	Edit events	The administrator must be able to edit events.	Must have
9	Delete events	The administrator must be able to delete events.	Must have
10	Alert about event change	The administrator could be able to alert future participants about an event change.	Could have
11	Send newsletter	The administrator could be able to send out newsletters to registered users.	Could have
12	View the list of future events	All users must be able to see the list of events.	Must have
13	View the list of past events	All users must be able to see the list of all past events.	Must have
14	View specific event details	All users must be able to see all details about an event.	Must have
15	View registered users	The administrator should be able to see all registered users.	Should have
16	View registered users for an event.	The administrator should be able to see all registered users for an event.	Should have
17	Submit Presentations	All presenters must be able to submit their Presentations	Must have
18	View Presentations	All attendees must be able to view the presentations from the event.	Must have

19	View confirmation for registered event	All users should be able to see a confirmation page after registering for an event.	Should have
20	View confirmation for creating an account	All users should be able to see a confirmation page after creating an account.	Should have
21	Cancel participation	The registered user for an event could be able to cancel their participation.	Could have
22	Contribution Badges	All users should be able to see badges of users based on their level of contribution and activity in the community.	Could have

### 3.4.3 Non-Functional Requirements

Non-functional requirements are listed to show the quality of the project. This describes how the systems should act.

ID	Non-Functional Requirements
1	The website should have 99.99% uptime for availability.
2	All sensitive data should be encrypted in the database.
3	SQL injections should be unsuccessful.
4	The page should render in under 6 seconds.
5	The application should be easy to use.
6	The application should be easily maintainable.

### 3.5 Risk Analysis

A risk analysis has been written to help minimise any possible risks that may arise with the project. A risk description describes the possible risk. Likelihood, impact, and severity are scored either low, medium, or high. Mitigating action and contingent action describe how the risk can be avoided and what to do if it materialises.

ID	Risk description	Likelihood	Impact	Severity	Mitigating action	Contingent action
1	The data stored in the database is breached	Low	Medium	Medium	All data should be stored safely. Passwords should be one-way encrypted.	The administrator should notify users if the data has been compromised.
2	The documentation of the project will not be finished on time	Medium	High	High	Ensure to document everything throughout the project and have a timeline when each chapter should be completed.	Prioritise the documentation over the implementation.

3	Implementation will not be finished on time	Medium	Medium	Medium	Have a written timeline for each implementation within the project. Prioritise features and implement more important ones first.	Focus on main features of the project.
4	Requirements will change throughout the project	High	Medium	Medium	Ensure if requirements change other scope / requirements are traded off for it.	Drop some features to make room for the new requirement.
5	Documentation gets lost	Medium	High	High	The documentation of the project should be stored on the cloud on top of storing it on the machine.	Cloud storage of the project should be updated regularly.
6	Implementation gets lost	Medium	High	High	Store the project implementation in source control on GitHub.	Push to GitHub regularly.

### 3.6 Summary

The above section described and visualised the requirements for this project. It was important to identify a large proportion of the requirements before building the project. Because this is an Agile-based project, these requirements could change, and the client can change their mind about all the set-out requirements. Stakeholder analysis helped identify individuals or groups with power or interest in this project. The use case diagram assisted with showing how the system behaves with different actors. Writing down and prioritising all the functional requirements helped identify the different and important functions of the system. Finally, the non-functional requirements detail the quality of the system, and a risk analysis has been done to minimise any risks associated with this project.



## 4 Design

### 4.1 Introduction

This chapter will show all the details of the system design. The Entity Relational Diagram (ERD) that has been created maps out all the entities and their relationship to each other. That is followed by the data dictionary, listing the details of the tables that need to be implemented into the database. The initial design includes the sketches that have been created to present different screens. After that, a few wireframes are presented with extra details. The site map follows to visually show the structure of different pages. Lastly, a sequence diagram and activity diagram have been included to show key interactions between system components.

### 4.2 Database Design

#### 4.2.1 ERD Entity Relationship Diagram

The entity-relationship diagram has been created to show the different entities, their attributes and how those entities are connected to each other. With the entity-relationship diagram design, it is easy to see how the data will be stored in the database system. Each entity has a primary key marked with “(PK)”, also if a foreign key is present “(FK)” marks it. The relationship between entities is marked with a line and each end cardinalities are shown. All entities are in 3<sup>rd</sup> normal form because there are no duplicates and each of the attributes is reliant on the primary key directly.

Figure 4.1. shows the first initial database design. It has six entities including, user type, user, attendee, event booking, event, and location.

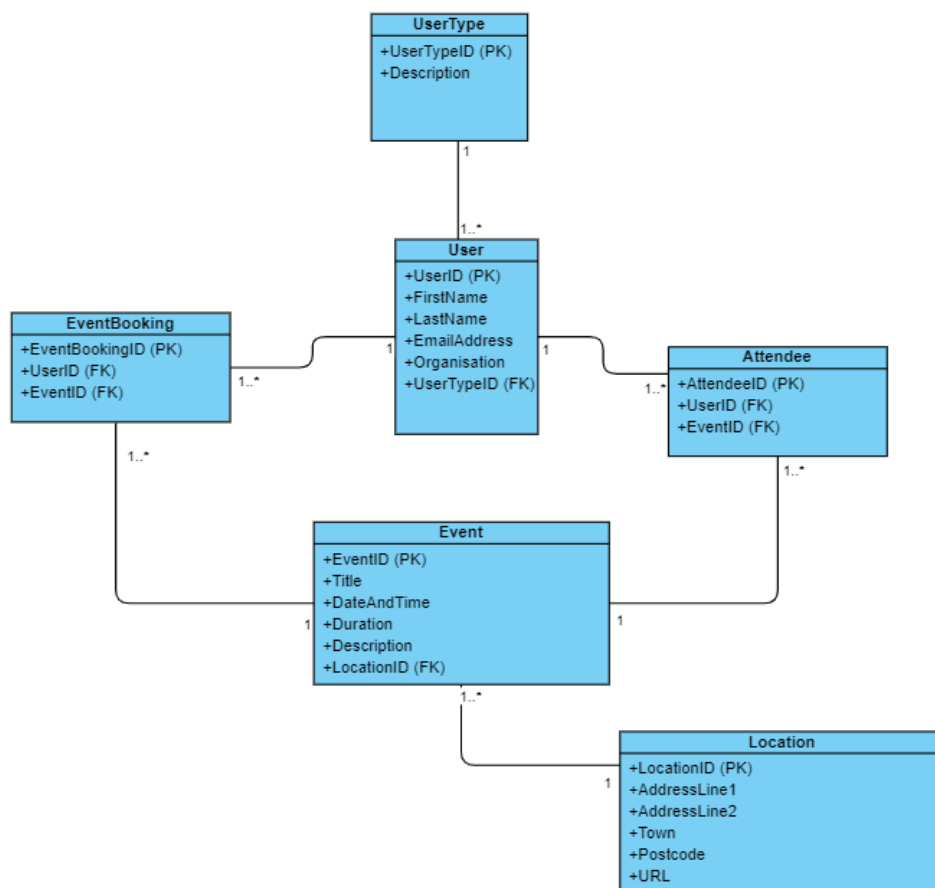


Figure 4.1. First ERD

After going back to the first ERD, it made sense to change some of the entities to better fit the requirements. Adding the Presentation entity would give a place to store presentations and it could be reused in different events. The Presentation Material table would allow the presenters to store links to their materials stored in the cloud. That makes it possible to access those links after an event. The Badge entity has been created to show levels of engagement with the community with users being awarded status based on participation. The event booking table has been removed as it is just a duplicate of the attendee table. The final ERD can be seen below.

This ERD supports the functional requirements. A user can be stored in the system with the relevant attributes. The Role entity makes it possible to control the access of different pages and give control over who can modify certain things, for example, events. With the Attendee entity, the admin would be able to see users that attend an event. Figure 4.2 shows the final ERD.

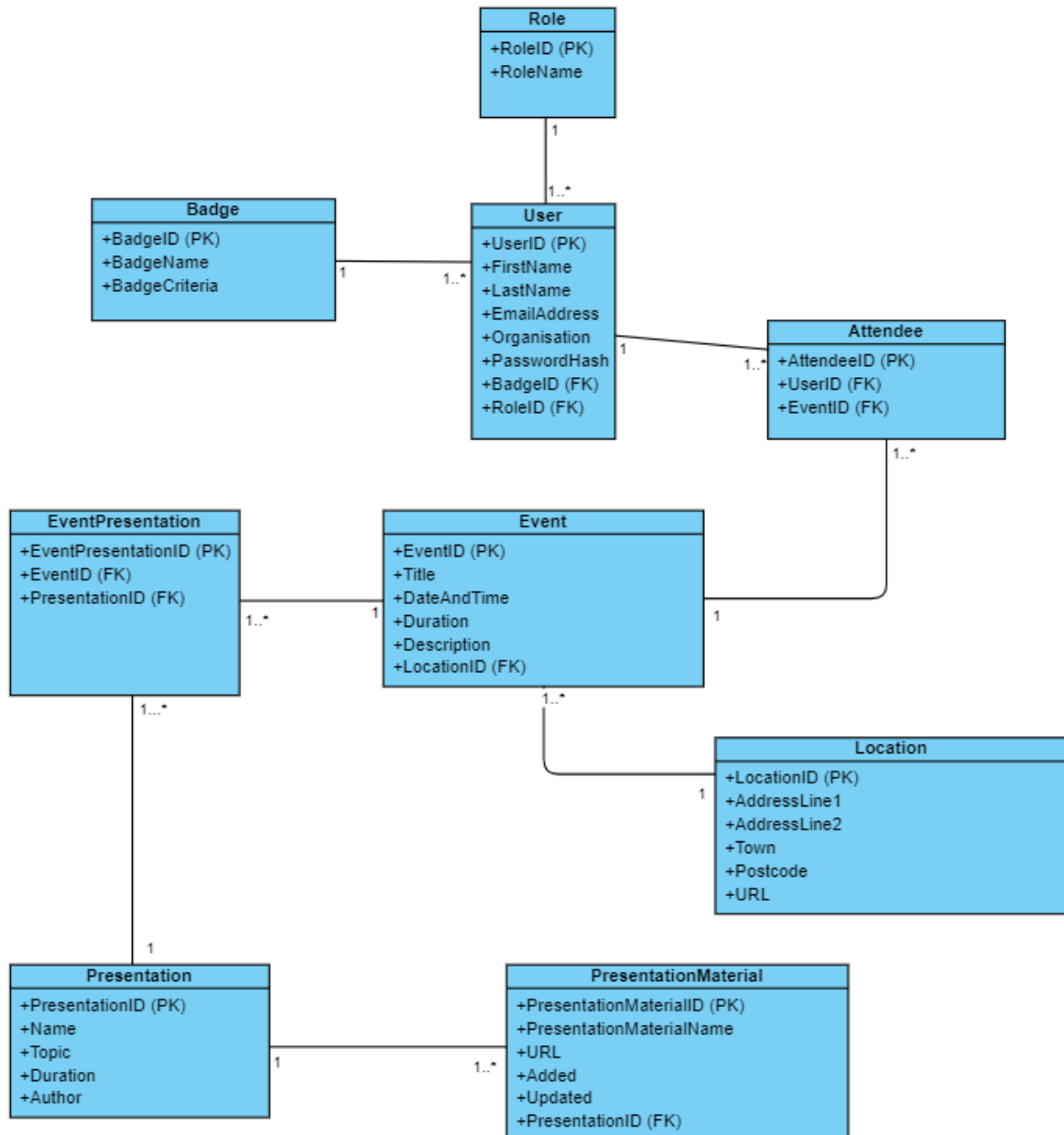


Figure 4.2. Final ERD

The first main entity is the **User** table. It stores all the information needed about a user. It also contains the RoleID as a foreign key from the **Role** table. The **Role** table contains the RoleID and the RoleName, which would be admin, student, and presenter. With the **Role** table in place, that would enable the use of role-based access control to the site. Different roles, for example, the admin would have the permission to change an event and see that page, but a regular user like a student would not see that option. The **User** table also stores the BadgeID as a foreign key. The **Badge** table is created to store different types of badges. It is a sort of reward system for users, who contribute or attend events multiples times, this form of social proof is designed to create an encouragement to participate more often.

The second main entity is the **Event** table. This entity contains the main information about an event that a user would see on the site. The location information is stored in a LocationID, and a foreign key connects the record to the **Location** table. The **Location** entity contains either the physical

address or a URL for an event. The **Attendee** table was created as an intermediary table between **User** and **Event** because they have a many to many relationship. The **Attendee** table has the information about which user booked a place for an event.

The **Presentation** table has been created to store information about the presentation, like the name, topic, duration etc. As an event can have many presentations and a presentation can belong to many events, The **EventPresentation** table has been made as an intermediary table between **Event** and **Presentation** to resolve this relationship. The **PresentationMaterial** entity contains URLs to the many materials used for a presentation.

A simplifying assumption has been made, that a user can have only one role and one badge at a given time.

#### 4.2.2 Data Dictionary

After creating the ERD, a data dictionary has been written to show the structure of the database system, each of the attributes, their data types, and to show the constraints between each of the related entities. The data dictionary contains all the information that is needed to insert tables into the database. It specifies the name of the table in the first column. The second column contains all the attribute names for the table. The data type states the type of the attribute; for example Int would store numbers, and String would store characters. The length is either the number limit for that attribute or the character limit that can be stored. The Not null column constraint specifies if the input data cannot be empty. Each table must have a PK (Primary key) and it must be unique. The FK (Foreign Key) field shows the connection to another table.

Table 4.1. shows the data dictionary for this project.

Table	Column	Data type	Length	Not null	PK/FK	Other constraints
Role	RoleID	Int	8		PK	
	RoleName	varchar	50	not null		
User	UserID	Int	8		PK	
	FirstName	varchar	50	not null		
	LastName	varchar	50	not null		
	EmailAddress	varchar	250	not null		
	Organisation	varchar	100	not null		
	RoleID	Int	8	not null	FK	References Role
	BadgeID	Int	8		FK	References Badge
Event	EventID	Int	8		PK	
	Title	varchar	250	not null		
	DateAndTime	Int	15	not null		
	Duration	Int	3	not null		
	Description	varchar	500	not null		
	LocationID	Int	8	not null	FK	References Location
Attendee	AttendeeID	Int	8		PK	
	UserID	Int	8	not null	FK	References User
	EventID	Int	8	not null	FK	References Event
EventPresentation	EventPresentationID	Int	8		PK	
	EventID	Int	8	not null	FK	References Event
	PresentationID	Int	8	not null	FK	References Presentation
Location	LocationID	Int	8		PK	
	Address1	varchar	250			
	Address2	varchar	250			
	Town	varchar	40			
	Postcode	varchar	8			
	URL	varchar	250			
Presentation	PresentationID	Int	8		PK	
	Name	varchar	50	not null		
	Topic	varchar	50			
	Duration	Int	3			
	Author	varchar	100			
PresentationMaterial	PresentationMaterialID	Int	8		PK	
	PresentationMaterialName	varchar	50	not null		
	URL	varchar	250	not null		
	Added	Int	10	not null		
	Updated	Int	10	not null		
	PresentationID	Int	8	not null	FK	References Presentation
Badge	BadgeID	Int	8			
	BadgeName	varchar	50	not null		
	BadgeCriteria	varchar	50	not null		

Table 4.1. Data dictionary

## 4.3 Interface Design

### 4.3.1 Sketches

The first step when creating a UI is to do some sketches with pen and paper. It is a quick process, it does not cost much, and it provides a basic look. This is simple and cheap to iterate upon providing a basis from which the wireframes can be built. It does not have many details, but it gives a better understanding of how the UI might look and drives conversation.

Figure 4.3. provides six different screens. The login screen with an email and a password field. The sign-up page with all the input fields that needed to be stored. The event details page, where all the details about a particular event would be listed. The edit event page, where the admin could change event details. The registered users' profile page, where users can view and change their information.

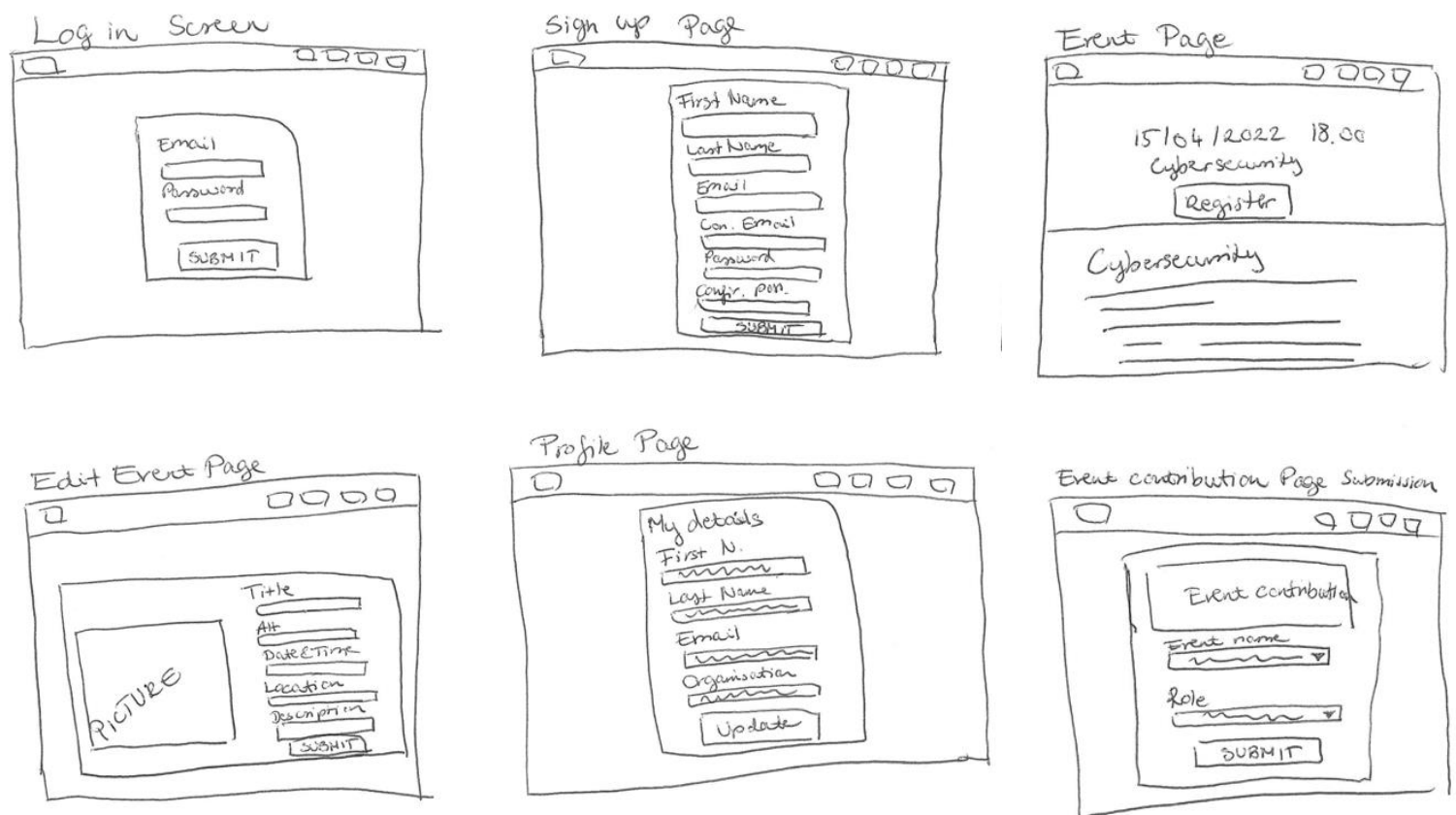


Figure 4.4. Sketches

Figure 4.5. shows the look of the contribution summary page; on this page, users can see their contribution to the events. The register for an event page is also shown. This is where the users would fill out their details to register for an event. The confirmation page is also drawn; this page is displayed after successfully registering for an event. Finally, the admin event management page is shown. This is where the admin would see for each event, who is attending.

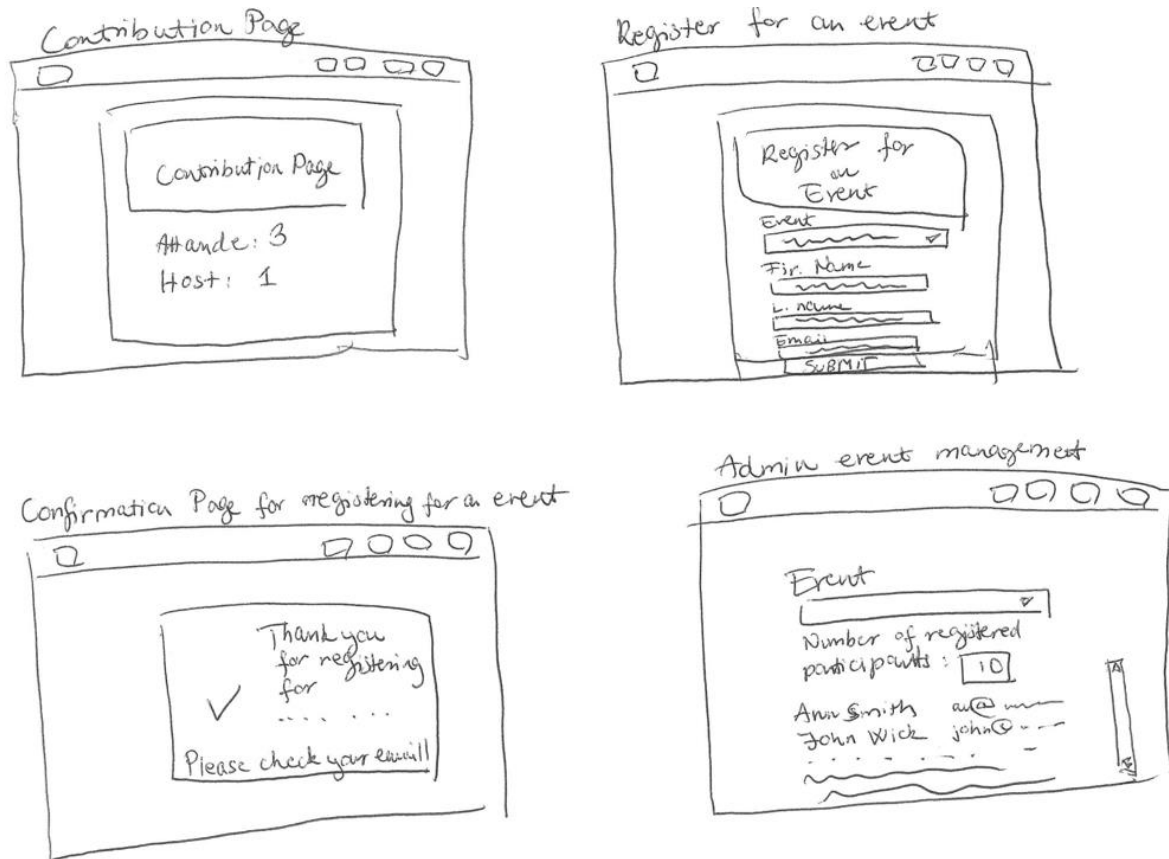


Figure 4.5 -Sketches

#### 4.3.2 Reusable components and Wireframes

After doing the sketches, reusable components have been created for the main components in the UI. Reusable components help to make the design uniform and facilitate visual consistency throughout all the pages. In the below pictures, the reusable components can be seen for this project. The main components are: the form field (green rectangle); the button item (blue rectangle); a picture item (purple rectangle); the description container (pink rectangle); and the card container (orange rectangle). More reusable containers can be found in Appendix A.

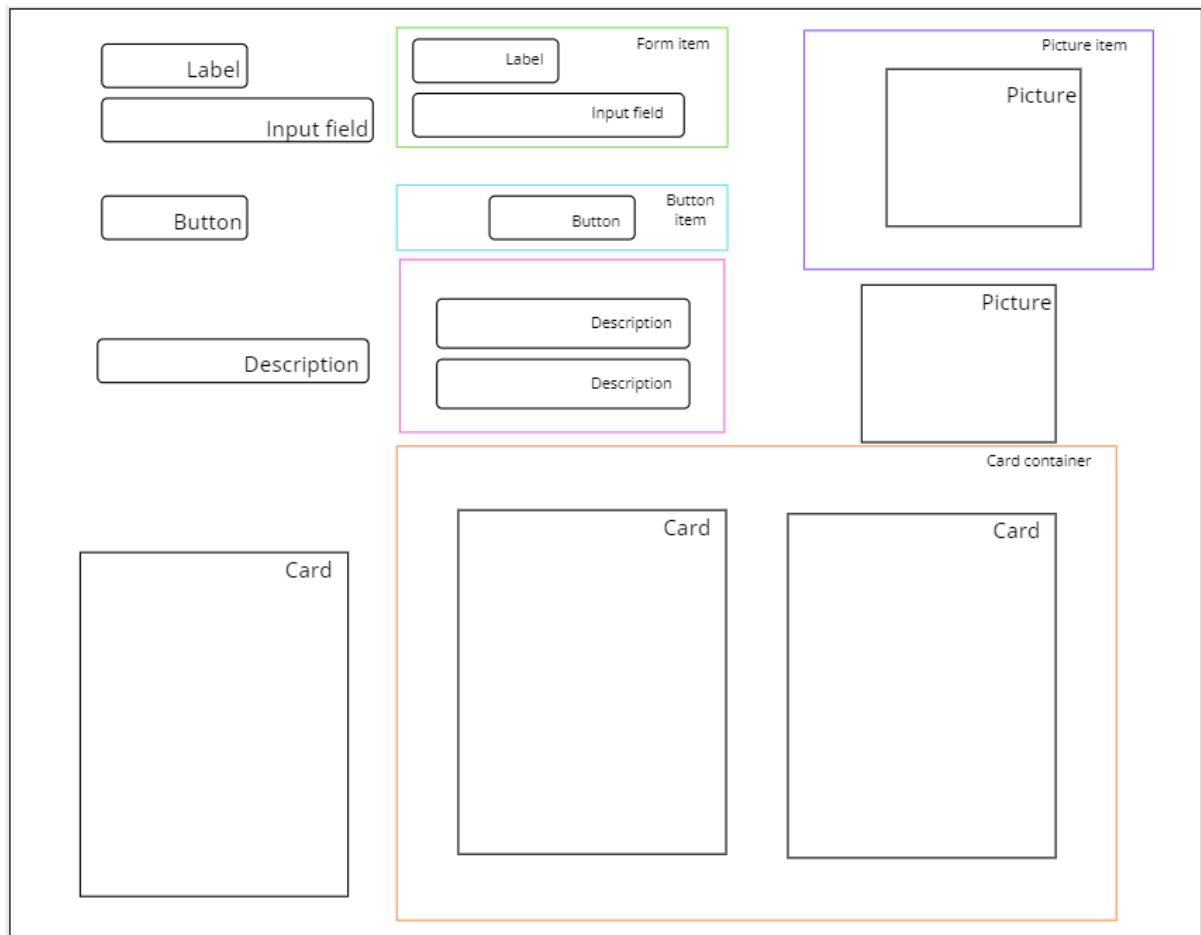


Figure 4.6. Reusable component wireframe

The next step was to create the wireframes. The design for the wireframes gives a good starting point for the implementation. On the below wireframes, the reusable components are highlighted with the same colours, as in the reusable component wireframe above. The rest of the wireframes can be found in Appendix A and B.

Figure 4.7. Events list page wireframe



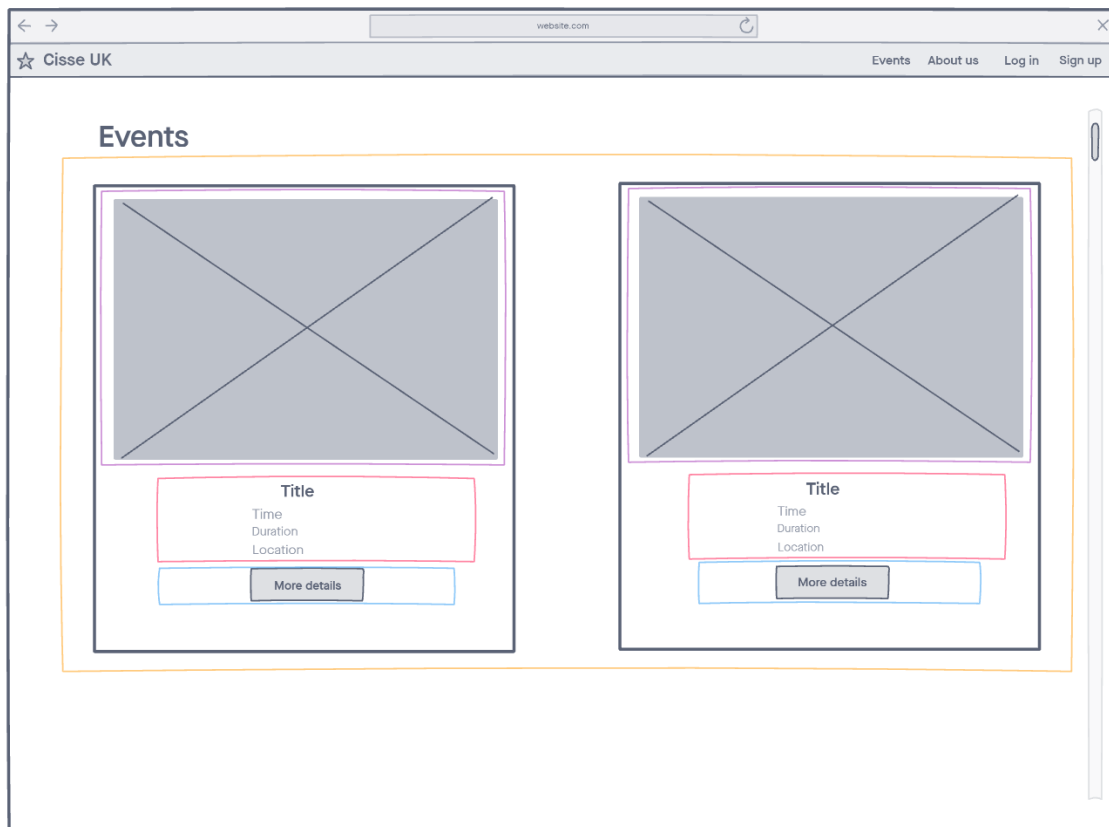
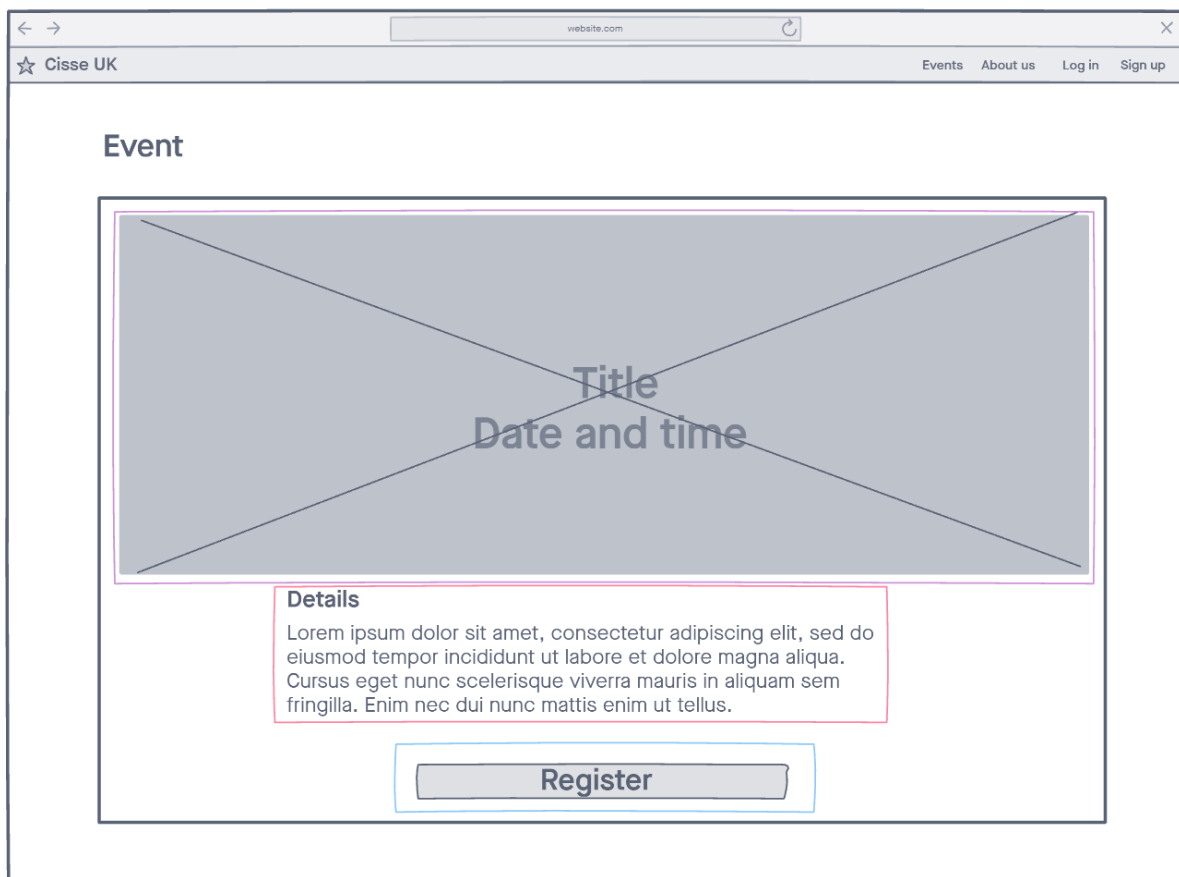


Figure 4.7. shows wireframes of the events page, where all the events would be listed. Each event has a card view with: a picture, a title, a description, and the button that would take the user to the event details page. The card view has been chosen to display the individual events as it is a very popular design pattern in User interface design. It gives the user a simplistic, visually satisfying look. The card view focuses on displaying only the most important information. This also helps them to remember information more easily (Justinmind, 2019).

Figure 4.6. Event detail page wireframe



The above diagram is the screen that would show all the details about a particular event. The title, the date and the time would be displayed on an image. Below would be the details about the event and a register button. The register button would take the user to the register for an event page. Using an image on the top of the page is designed to help users remember the content; it will also draw more attention to the page than the text alone (Wilson, 2015).

Figure 4.7. Register for an event page wireframe

The wireframe shows a web browser window. The address bar contains 'website.com'. The page title is 'Cisse UK'. The navigation menu includes 'Events', 'About us', 'Log in', and 'Sign up'. The main content area is divided into two sections. The top section is a large grey rectangle with a diagonal cross and the text 'Register for an event'. The bottom section is a white rectangle containing a form. The form has four input fields: 'Event' (a dropdown menu with 'Select an event' and a downward arrow), 'First name' (a text input with 'Type your first name'), 'Last name' (a text input with 'Type your last name'), and 'Email' (a text input with 'Type your email'). A 'Submit' button is located below the form fields.

The last wireframe in this list is the register for an event page. This is where the user would fill out their details to be able to register for an event. It would have an image on top, a form below and the submit button. Labels for the input fields have been placed above. This helps users complete the form faster as they use just a single eye movement to understand the form (Bakusevych, 2020).

#### 4.3.3 Site Map

The site structure can be seen below. It shows how different pages would connect to each other.

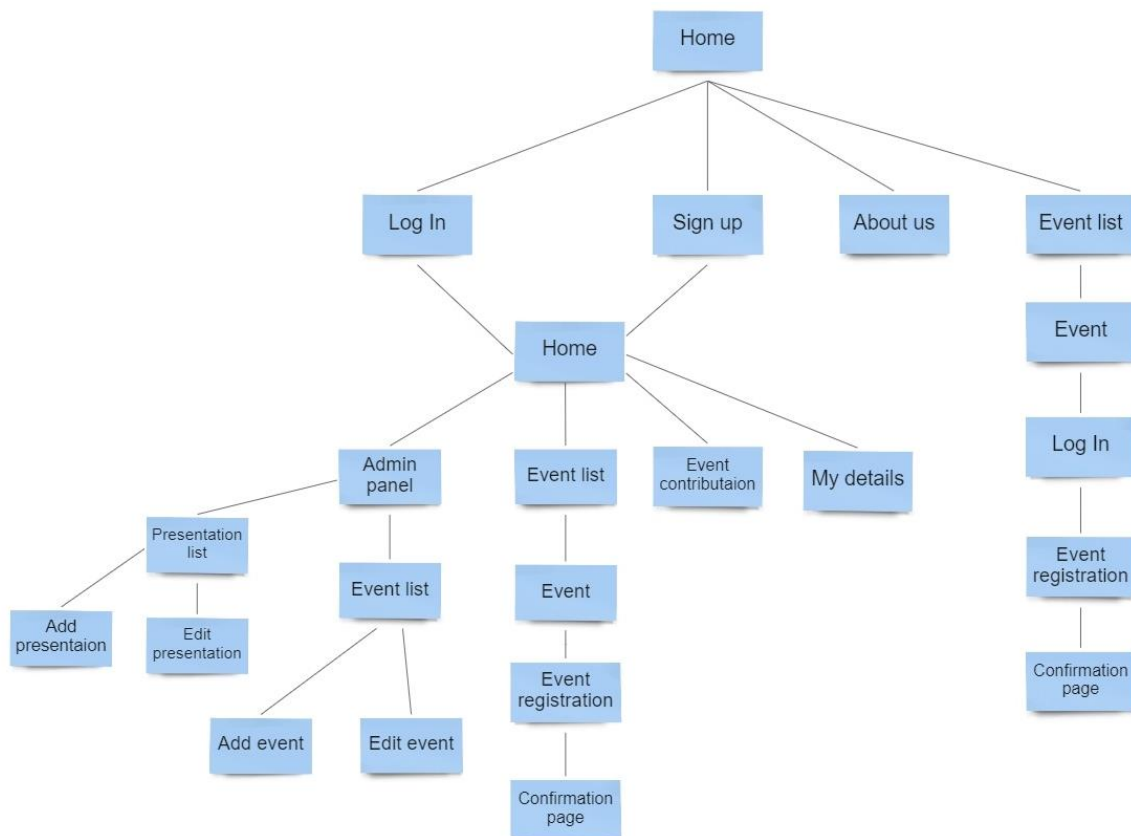


Figure 4.8. Site map

#### 4.4 Sequence Diagrams

Sequence diagrams show the interactions between the different systems and external actors. In this case, it shows the chain of system interactions between the admin, the user interface, the API, and the database. The Figure 4.9. sequence diagram represents how the admin would delete an event from the system. The Starting point is the home page. This is where the admin would click events on the navbar. Next, the user interface would tell the API to fetch all the events. Then the API would query all the events from the database. The database would respond by returning all the details to the API, then the API would return all the events to the user interface. Then the user interface would display all the events to the admin.

To delete an event, the admin would click on a delete button on an event, where the user interface tells the API to delete a certain event, then the API passes the message to the database to delete that event from the database, identifying it by its id. After the event has been deleted from the database, an updated event list would be sent back to the API. Then the API would pass that message down to the user interface. Finally, the user interface would display the updated event list.

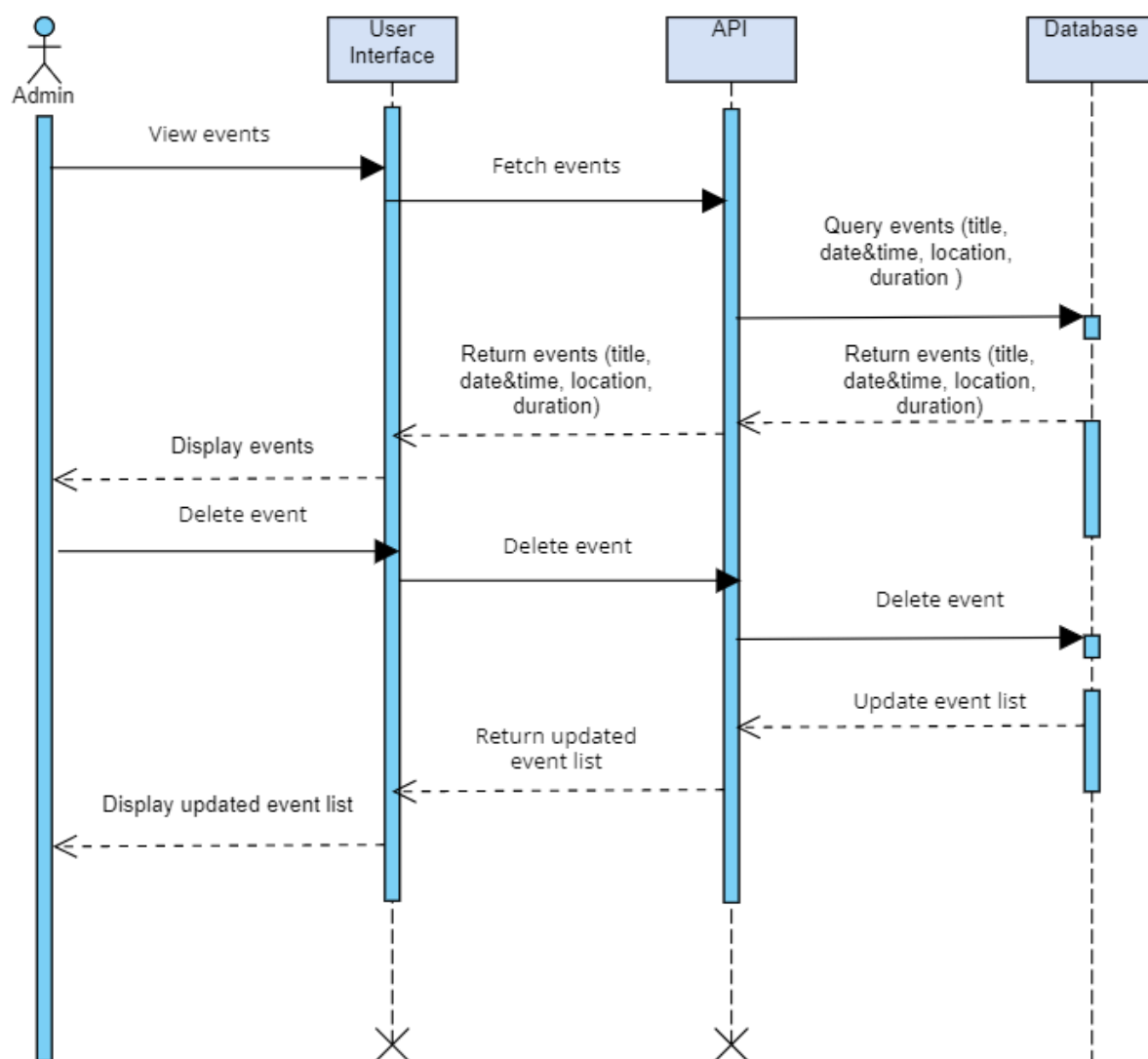


Figure 4.9. Admin delete an event sequence diagram

More sequence diagrams can be seen in Appendix C.

## 4.5 Activity Diagrams

Activity diagrams show a flow of an activity and its alternate routes. Figure 5. Shows the steps that the admin can take when viewing the event list.

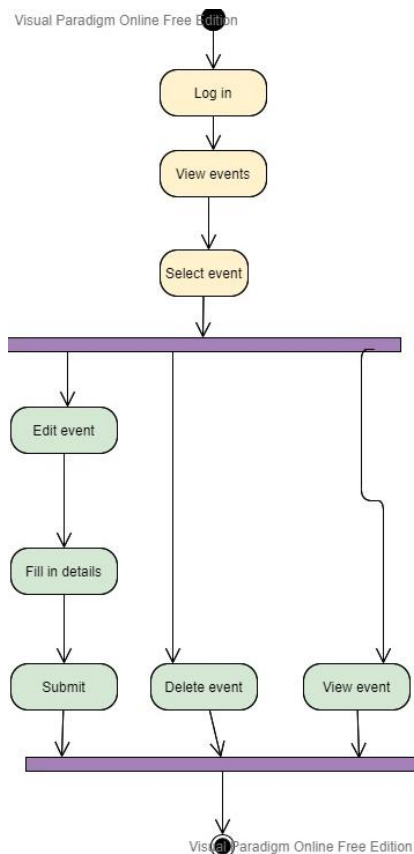


Figure 5. Activity diagram

#### 4.6 Conclusion

The design chapter was broken into two main sections. The first is the database design and the second is the interface design. The database design gave an insight into the structure of the database. The ERD help visualise the database, and the data dictionary helps with the implementation. The interface design included the sketches for different screens; followed by wireframes with more details. The site map was presented to show the different pages. Finally, the sequence diagrams and activity diagrams have shown interactions between the systems.

## 5 Implementation

### 5.1 Introduction

In this chapter, the implementation of this project will be discussed. This will detail all the tools and libraries that have been used and the architecture that has been designed for this project. There are two main parts to the application, the front-end and the backend. In the front-end, the react single page application will be detailed, then the reusable components. In the backend part, the databases will be discussed. Finally, the GraphQL API and queries that link the two together will be explained.

### 5.2 Tooling

**Node.js** is a runtime environment for JavaScript. It runs the JavaScript outside of the browser, for example on the server or on a local machine. This is the basis on which AWS amplify works. For this project, the node is just used to run the front-end and run the AWS amplify backend, no dedicated node.js code is written.

**NPM** – npm is a package manager for JavaScript. It supports package installation, dependency management and scripting. It manages dependencies as packages can be dependent on other packages, and npm will automatically install the right sub-dependencies. It also manages all packages versions; npm in this project is also used to run pre-packaged scripts for running the application.

**AWS Amplify** – This is a framework designed to simplify the creation and operation of application backends. It has a library that supports Node.js apps that will be used in this project. It has been selected, to help to build a full-stack application using its built-in tools (Amazon Web Services, 2022a). The backend of the application can be set up with Amplify Studio through the AWS user interface. The application's database can also be seen and handled through this interface. In this project two different databases have been used: firstly DynamoDB (NoSQL); and the second version Serverless Amazon Aurora (MySQL). To make the front-end communicate with the database a GraphQL API has been configured inside of Amplify. Authentication of the project is handled by Amazon Cognito another AWS Amplify tool.

**Aws AppSync (GraphQL API)** – In this project, a GraphQL API is created by AWS Amplify and it handles all the data insertion, modification, deletion and fetching. It is an alternative to REST API.

**GraphiQL** – GraphiQL is a client for GraphQL with a simple user interface. It makes it possible to create and test different queries inside the browser. It is used in this project to test different requests for data, before implementing the queries inside the application.

**React.js** – This is currently a popular, open-source Javascript Library that is being used to build a front-end application. Other popular libraries are used for front-end development, one is called Angular, and the other is Vue. React was chosen for this project because it has a less steep learning curve compared to Angular. Vue is easier to learn compared to React, but it is less visible in the job market (Daityari, 2021; Google Trends, no date). The combination of easy to learn and, simple to use made it attractive for this project.

**React date picker** – This is an external library that is being used to display a calendar view when selecting a date for an event. This has been chosen for this project because it simplifies the code and provides a better user experience.

**React-router-dom** – This is an important React library that handles routing between the different pages in the application and creates the single-page experience for the user across the components.

**GitHub** – GitHub is an online source code repository. This has been used throughout the project to store all code on the cloud. In conjunction with Git, it was used to help manage the source code and changes. It was also connected to AWS Amplify to trigger application builds and deployment.

**DynamoDB** – This is an Amazon-owned and designed non-relational (NoSQL) database. This works as a key-value store and it is simple to work with (Amazon Web Services, 2022b). This has been used as the first database for the project as it was the default database recommended when setting up AWS Amplify.

**MySQL (Serverless Amazon Aurora)** – This was the second version of the database. It has been chosen, because this is a relational database, and it works with AWS Amplify. This gave a more traditional approach to storing data that better suited the client's requirements and design. Later in this section, the rationale behind switching databases is described in more detail.

**Amazon Cognito** – This is an AWS tool used to handle user authentication and access management for different user roles. This can also be set up to use multi-factor authentication for extra security (Amazon Web Services, 2022c). In this project, this has been chosen to simplify the set-up of the authentication process as the time is limited. After setting it up to the front-end, it gave a prebuilt user interface for sign-in and sign-up, that can be customised. It also takes away the complexity and security risks associated with creating a dedicated authentication system (Amazon Web Services, 2022c).

**Cascading Style Sheets (CSS)** – This has been used in the project to style the frontend components. This has been selected for this project as it is the default tool used for styling pages. In this project, the CSS sits with the React component code making components reusable.

**HTML & JSX** – HTML stands for Hypertext Markup Language, which is used for documenting the web page content & structure. JSX stands for JavaScript Syntax Extension. With React, it makes it possible to write JavaScript with HTML elements. Some rules need to be followed when using JSX. Some examples of that: are that the attributes of HTML have to be written in camelcase. Also, JSX must return a single parent element (Chris, 2021). This has been chosen because of its ease of use with React.

### 5.3 Architecture

How the different tooling fits together is as follows. The whole system runs in an AWS Virtual Private Cloud. The React frontend does not have direct access to the database. The front end utilises the AWS Amplify Library which uses GraphQL to interact with the authentication system and the database. To get the relevant data that is needed for the front end, the React app calls the AWS Amplify library which sends the request to the right GraphQL API. Then GraphQL API fetches all the relevant data from the Amazon Aurora (MySQL) database, and it returns that to the React frontend.

The Authentication process is handled by Amazon Cognito. The frontend does not have direct communication with Amazon Cognito. Similarly, the front-end application sends a request to AppSync (GraphQL API), and then that request is passed to Amazon Cognito to handle. When the authentication process is done, Amazon Cognito sends the response back to AppSync and then AppSync sends that to React.



The Figure 5.3.1. is shown in the diagram below:

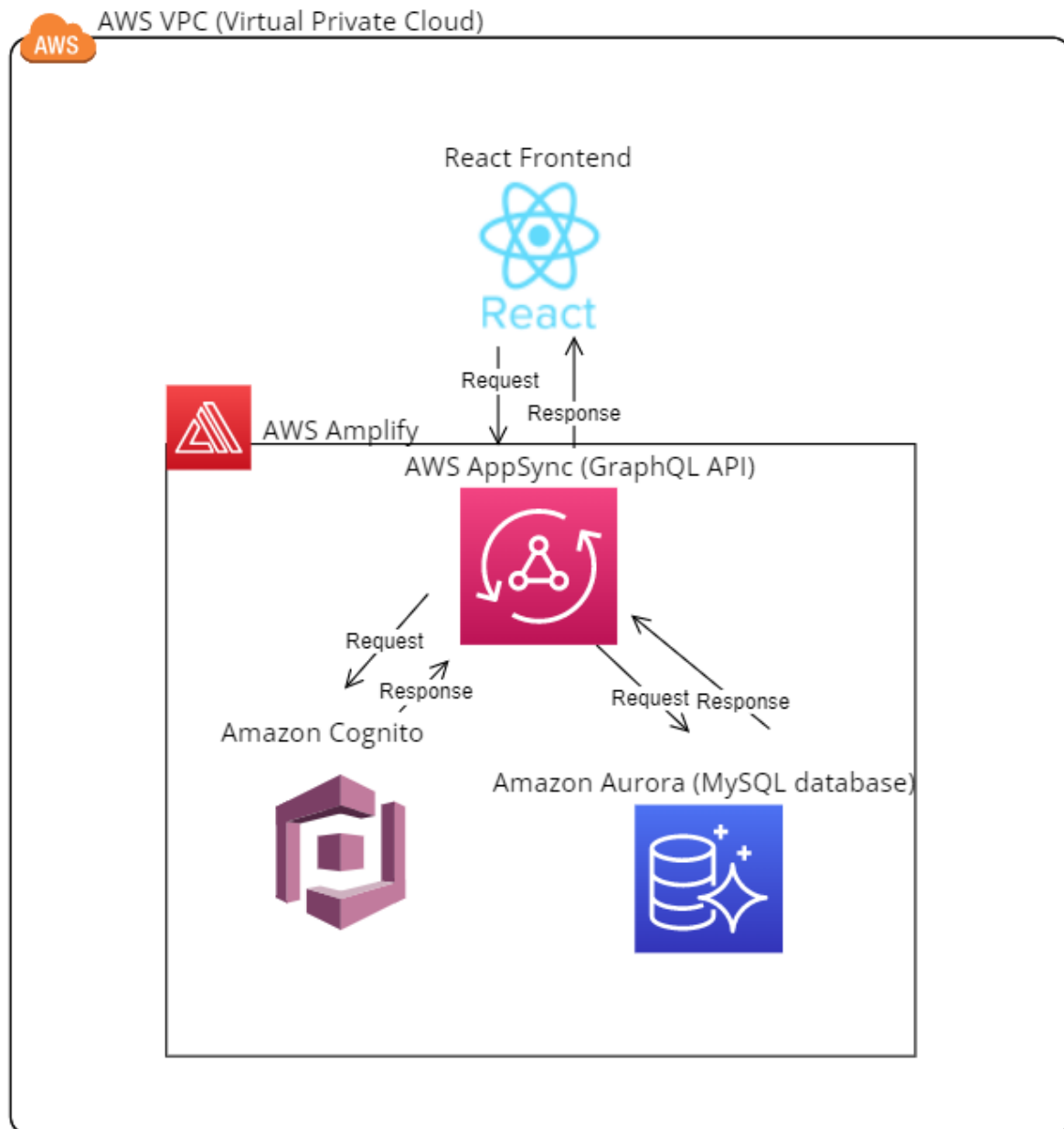


Figure 5.3.1. – Architecture diagram

## 5.4 Frontend implementation

The frontend application is a react application implemented as a single page application making use of reusable components. In this section, the structure of the front end will be considered. Some example pages and components will also be described.

The front-end component of the project was created with React, a Javascript library. It does not use traditional separate HTML and Javascript, instead, it uses JSX, a special syntax. With the traditional approach, the markup (HTML) is separated from the logic (Javascript). JSX combines both. React applications are composed of lots of components interacting with each other. These reusable components are then organised and imported into larger page components. This makes it visually consistent and more maintainable. The front-end application is broken into many folders and files.

The Figure 5.4.1. shows the folder structure. The public folder holds the index.html file which contains the link to the React Javascript bundle. This is the root of the project. The src folder holds three main folders: the component folder, the graphql folder and the pages folder. The component folder holds all the smaller elements of the pages, including reusable components. The graphql folder has the auto-generated mutation and queries files. These files hold the queries that are needed for retrieving, modifying, and deleting data from the database. The App.js hold all the pages, that the application has and the index.js file is used as the entry point.

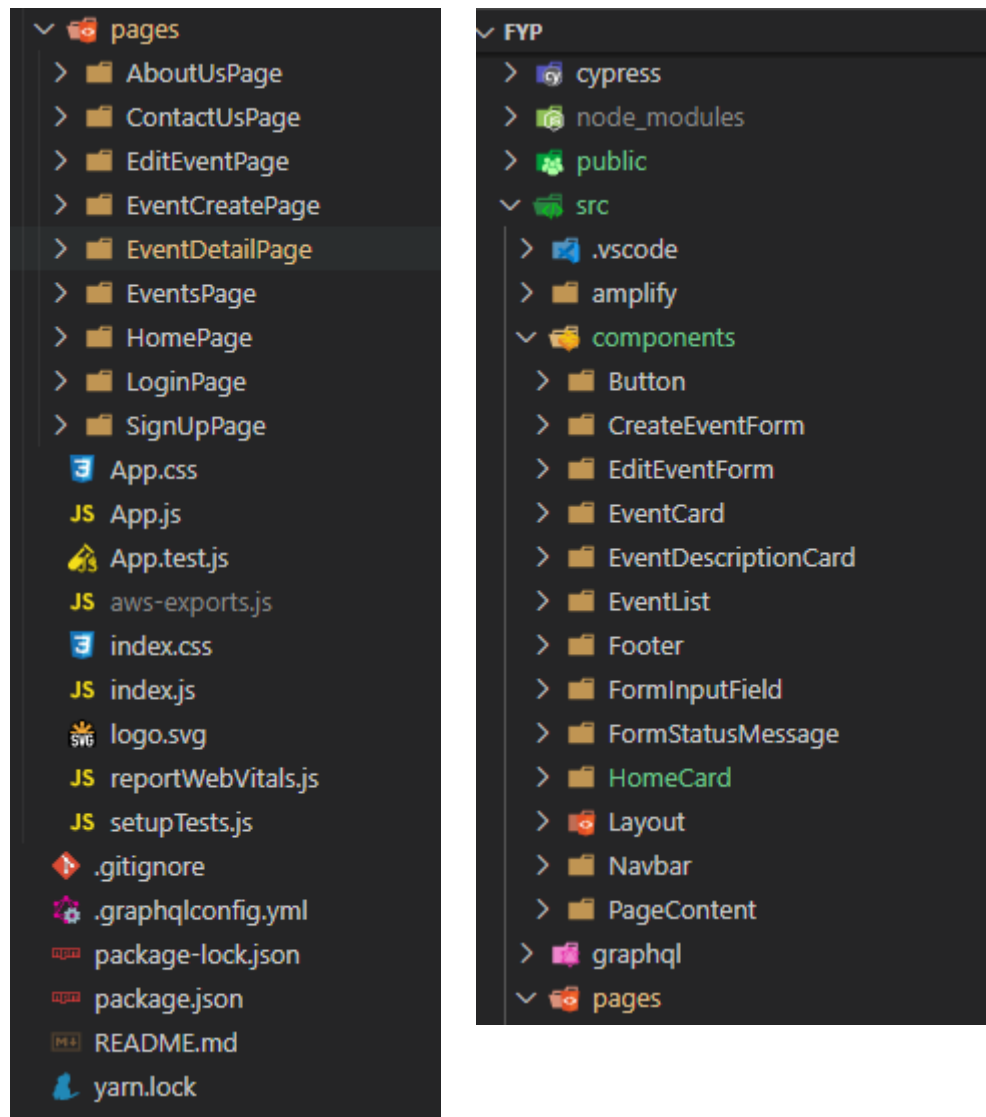


Figure 5.4.1. – Folder structure

### 5.4.1 Reusable Components

There are multiple reusable components have been created for this project. Figure 5.4.2. and 5.4.3. show an example of the reusable components that have been created. The formInputField(props) is a small function. This function is essentially a configurable template that takes an object with properties as its parameters. It is used in multiple places in the codebase, for example: on the event form page, or on the update event form page but it could be used by any other pages. The label text, input fields size, the onChange handler and the value can be set with different values to make it flexible for re-use. These properties are configured In the JSX element. The point of creating and using components is to facilitate re-use across the platform.

```
import React from "react";
import "../FormInputField.css";

export function FormInputField(props) {
  return (
    <div className="fields">
      <label>{props.labelText} </label>
      <input
        type="text"
        size={props.size}
        onChange={props.changeHandler}
        value={props.inputValue}
      />
    </div>
  );
}
```

Figure 5.4.2. – Reusable component Form input field

The below figure shows the component being reused in the create event page. In the example below a custom, change handler is being created to update the state of the form. The state of the form is then fed into the input value so when the user types something, the form state is updated and the form input field re-renders with the state value in the text.

```
<FormInputField
  labelText="Image:"
  size="50"
  inputValue={eventFormData.img}
  changeHandler={(e) =>
    | setEventFormData({ ...eventFormData, img: e.target.value })
  }
/>
<FormInputField
  labelText="Alt:"
  size="50"
  inputValue={eventFormData.alt}
  changeHandler={(e) =>
    | setEventFormData({ ...eventFormData, alt: e.target.value })
  }
/>
```

Figure 5.4.3. – Reusable component Form input field used in Create event page

The card component on the event list page is another example. In this component, the properties are wrapped in HTML to display in the browser. Some manipulation is done with time to get it to display in a user-friendly way. In addition to this, the button reusable component is imported and wrapped inside a Navigation Link from ReactRouterDOM. This special component is used to navigate to the event details page, depending on the id of the event.

Each card's details dynamically change based on the data that is fetched from the backend API via GraphQL. The events data is iterated by using a map function which creates event card components with different properties based on the details of the event. Additionally, a custom deletion function is passed into each of these with the event of the id. Figure 5.4.4. shows the reusable component and Figure 5.4.5. shows when it is used to display different events dynamically.

```
import React from "react"; // you, 5 months ago * eventpage, event cards, event list class componen...
import "../EventCard.css";
import { Button } from "../Button/Button";
import { NavLink } from "react-router-dom";
export const EventCard = (props) => (
  <div className="event-card">
    <img src={props.event.img} alt={props.event.alt} width="700" height="500" />
    <h2>{props.event.title}</h2>
    <p>
      {`${props.event.eventDate.getDay()}/${
        props.event.eventDate.getMonth() + 1
      }/${props.event.eventDate.getFullYear()}`}
      {`${props.event.eventDate.getHours()}:${(props.event.eventDate.getMinutes() < 10 ? '0': '')+props.event.eventDate.getMinutes()}`}
    <br />
      {`${props.event.duration} hour(s)`}
    <br />
      {props.event.location}
    </p>
    <NavLink to={`/eventDetails/${props.event.id}`}>
      <Button text="View More Info" />
    </NavLink>
    <NavLink to={`/editEvent/${props.event.id}`}>
      <Button text="Edit" />
    </NavLink>
    <button onClick={props.onClickHandler}>Delete</button>
  </div>
);
```

Figure 5.4.4. – Event Card reusable component

```
return (
  <div className="event-list">
    {events?.map((event) => (
      <EventCard key={event.id} event={event} onClickHandler={() => deleteEventData(event.id)} />
    ))}
  </div>
);
```

Figure 5.4.5. – Reusable component being used

Styling of the pages and components is done with CSS specific for the components. In the Figure 5.4.6. the event list page is presented. Each event is inside a card component. At the top of each card there is an image for the event and below the important details can be seen with different button options. Flexbox is being used to display the components in the right order on the card.

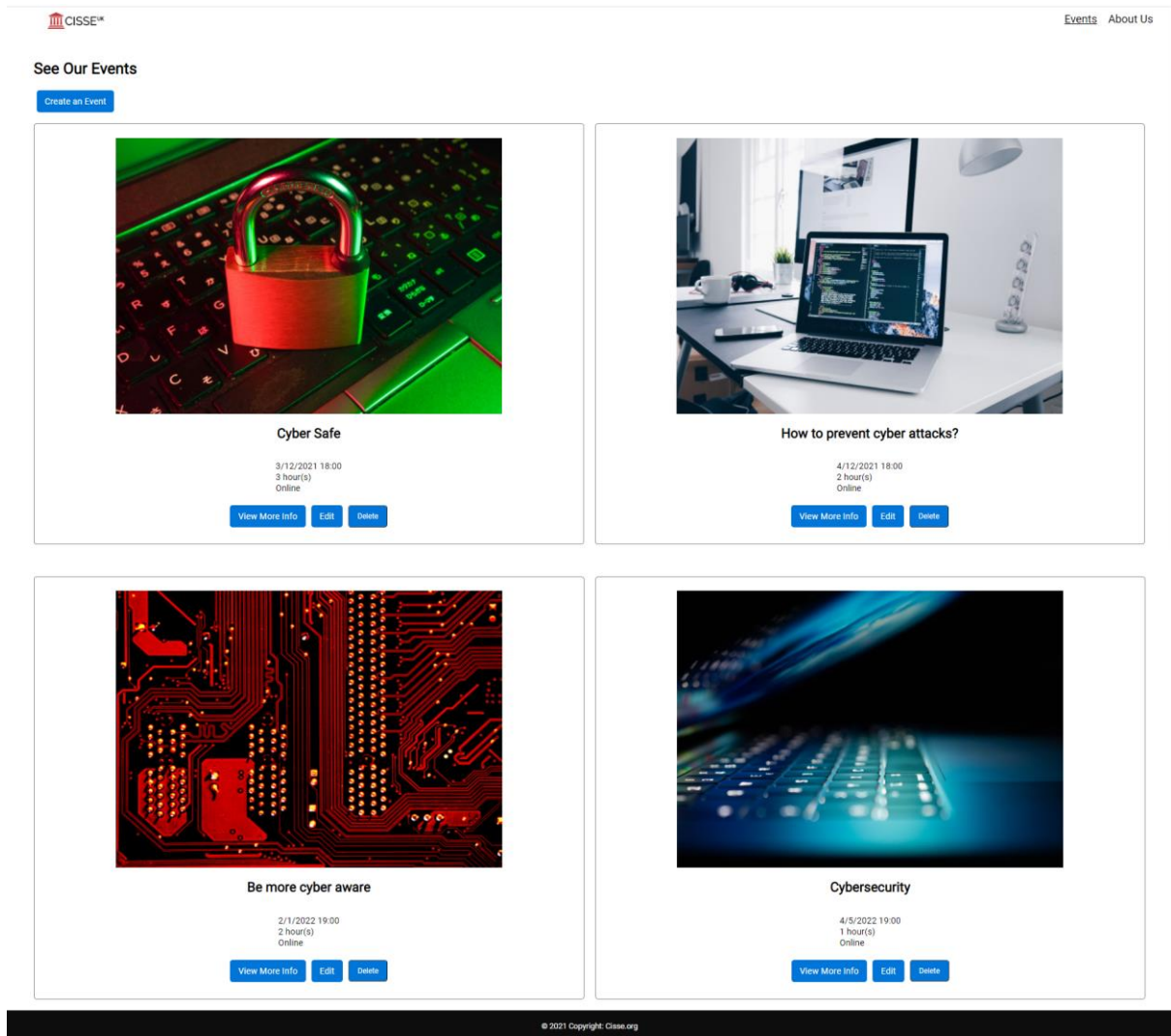


Figure 5.4.6. – The create an event page

The border and the border radius of the card element have also been set to have a grey border with

```
.event-card {
  border: 0.5px solid grey;
  border-radius: 5px;
  padding: 25px;
}

@media only screen and (max-width: 600px) {
  .event-card {
    margin-left: 30px;
    margin-right: 30px;
  }
}

.event-card-top {
  display: flex;
  justify-content: center;
  align-items: center;
  flex-direction: column;
}

.event-card-bottom {
  display: flex;
  justify-content: center;
  align-items: center;
  flex-direction: row;
}
```

rounded corners. Additionally, media queries are used to make the components responsive for smaller mobile displays. The corresponding CSS can be seen below for the individual cards:

```

button {
  display:inline-block;
  border-radius: 5px;
  padding-top: 10px;
  padding-bottom: 10px;
  padding-left: 15px;
  padding-right: 15px;
  background-color: #0275d8;
  text-decoration: none;
  color: white;
  margin: 5px;
}

.eventPic{
  width:700px;
  height:500px;
}

@media only screen and (max-width: 600px) {
  .eventPic {
    width:300px;
    height:200px;
  }
}

```

Figure 5.4.7. – EventCard.css

The page components are very simple because they are composed of smaller reusable components used elsewhere in the application.

Figure 5.4.8. shows the CreateEventForm is wrapped in the PageContent component which is a layout component.

```

import React from 'react';
import { PageContent } from '../../components/PageContent/PageContent';
import { CreateEventForm } from '../../components/CreateEventForm/CreateEventForm';
import { withAuthenticator } from '@aws-amplify/ui-react';
import { AmplifySignOut } from '@aws-amplify/ui-react';
import './EventCreatePage.css';

const EventCreatePage = () => (
  <PageContent>
    <CreateEventForm/>
    <AmplifySignOut/>
  </PageContent>
)

export default withAuthenticator(EventCreatePage);

```

Figure 5.4.8. – Event Create Page

## 5.4.2 GraphQL

This project is using the GraphQL API from the Amplify library to communicate with the backend database. This is a built-in tool that has been configured when setting up AWS Amplify with the application. The GraphQL API is controlled by the GraphQL Schema in the application code. After modifying it, the changes just need to be pushed to Amplify and it updates everything in the background. When setting up the GraphQL API, mutations.js and queries.js files have been created automatically by Amplify. These have the necessary code needed to talk to AWS and the database. The corresponding database tables do have to be created separately in the database too, however. The queries.js file has the code for getting and listing data from the database. The mutations.js file has the code for updating and deleting data via the API. The schema.json graphql file defines the types and relationships between data entities. This is covered in the database section of this report.

Figure 5.4.9. is an example of one of the ways the GraphQL API and AWS Amplify have been used in this project. A function was created to retrieve all the events on the page (loadEventData). This makes use of the GraphQLOperation function imported from AWS Amplify. The listSecurityEvents variable is being used to define one of the graphql queries to execute. Inside the load event data function, data is being pulled off the response from the GraphQL API operation to get the events. Next, all the events in the results have their date converted from a timestamp into a Javascript date. After the event dates have been fixed, the state of the component is set to include the event data and it re-renders the page. In the useEffect hook, this loadEventData () function is called.

```
import React, { useState, useEffect } from "react";
import { API, graphqlOperation } from "aws-amplify";
import "../EventList.css";
import { EventCard } from "../EventCard/EventCard";
import { deleteSecurityEvents } from "../../graphql/mutations";

const listSecurityEvents = `
query ListSecurityEventss {
  listSecurityEventss {
    id
    title
    alt
    eventDate
    img
    location
    duration
  }
}
`;

export function EventList() {
  const [events, setEventData] = useState([]);

  const loadEventData = async() => {
    let {data} = await API.graphql([graphqlOperation(listSecurityEvents)]);
    let events = []
    if(data.listSecurityEventss){
      events = data.listSecurityEventss.map((event)=>{
        event.eventDate = new Date(event.eventDate * 1000)
        console.log(event.eventDate)
        return event
      })
    }
    setEventData(events)
  }

  useEffect(() => {
    loadEventData();
  }, [])
}
```

Figure 5.4.9. – Event List, GraphQL API being used

Figure 5.4.10. has the function to delete an event from the database. It has one parameter, the id of an event to be deleted. The AWS Amplify library runs the deleteSecurityEvents mutation (this is imported from the mutation file) and passes in the id of the item to delete. This is a better way of running the queries rather than defining them directly, as was done for the list events operation. Next, a new array is created by filtering out the event that's been deleted and then setting the state of the component to the new event list. This causes it to re-render / re-display. The events could have been fetched again, but this achieves the same goal.

The last section of code is used to display and render the event cards. It maps through the events and displays a reusable event card component for each event.

```
async function deleteEventData(id){
  console.log(id)
  await API.graphql({ query: deleteSecurityEvents, variables: {id: id}})
  const newSecurityEvenstArray = events.filter(event => event.id !== id);
  setEventData(newSecurityEvenstArray);
  console.log("delete function fired")
}

return (
  <div className="event-list">
    {events?.map((event) => (
      <EventCard key={event.id} event={event} onClickHandler={() => deleteEventData(event.id)} />
    ))}
  </div>
);
}
```

Figure 5.4.10. – Delete an event function

## 5.5 Backend implementation

The backend application is entirely driven by AWS and AWS Amplify which will be described in this section. Also in this section, the databases storing the data for the system will be explored.

### 5.5.1 Amazon Web Services Amplify

#### 5.5.1.1 *Setting up Amazon Amplify*

The heart of this project is driven by AWS Amplify. This simplifies, the building and the deployment of the application through its user interface. To use AWS Amplify, @aws-amplify/cli needed to be installed globally. The next step was to run amplify configure, this brings up a window in the browser then it needs to sign into AWS Console. After signing in, an IAM user needed to be set up for the project. After the setup process in the command line, the react project had to be selected and then the amplify project could be initiated. This created the Amplify project and created a folder called amplify that had the back-end configuration of the application. A src/awes eports.js file had also been created. This has the configuration details for the services.

Next, the @aws-amplify/ui-react library had to be installed. Amplify had to be imported in the UI in the index.js file and its configure method is called This is shown in Figure 5.5.1.



```

import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
import App from './App';
import reportWebVitals from './reportWebVitals';
import Amplify from 'aws-amplify';
import config from './aws-exports';

Amplify.configure(config);

ReactDOM.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>,
  document.getElementById('root')
);

// If you want to start measuring performance in your app, pass a function
// to log results (for example: reportWebVitals(console.log))
// or send to an analytics endpoint. Learn more: https://bit.ly/CRA-vitals
reportWebVitals();

```

Figure 5.5.1. – Amplify imported

With the amplify project configured and the UI package imported, the GraphQL API needed to be set up next. Through the command line, an API was added. After defining the GraphQL schema needed for the objects, it automatically created the tables. First, it was using DynamoDB and generated a default schema file in the code. This also created the GraphQL API under AWS AppSync. When that was all set up, the schema had to be modified to suit the requirements and it was updated and pushed to deploy the changes to the GraphQL API and the databases. Note, when switching to the Serverless MySQL database, the automated database table generation no longer worked as it is not supported by Amplify. At this point, the corresponding tables had to be created using the Create statements described later in the chapter. Amplify was then configured to use a resolver to fetch data from the database for the GraphQL API.

#### 5.5.1.2 *Setting up Amazon Cognito*

User authentication is used on the create an event page. This is done with Amazon Cognito, a tool that handles user authentication. It comes with prebuilt sign-up and login forms. Figure 5.5.2. below shows how it is been used in this project. To use it in the application first, the `withAuthenticator` and `AmplifySignOut` functions must be imported from `@aws-amplify/ui-react` library. Then the page is wrapped with a `withAuthenticator` component. When this page is visited, the system only lets the user in if they sign-up or log in to the page. `AmplifySignOut` is a sign-out button on the page, that lets users sign out.

```
import React from 'react';
import { PageContent } from '../../components/PageContent/PageContent';
import { CreateEventForm } from '../../components/CreateEventForm/CreateEventForm';
import { withAuthenticator } from '@aws-amplify/ui-react';
import { AmplifySignOut } from '@aws-amplify/ui-react';
import './EventCreatePage.css';

const EventCreatePage = () => (
  <PageContent>
    <CreateEventForm/>
    <AmplifySignOut/>
  </PageContent>
)

export default withAuthenticator(EventCreatePage);
```

Figure 5.5.2. – Authentication being used

### 5.5.2 Databases

In this project, two different databases have been used. Initially, it was started with a (NoSQL) non-relational database called Amazon DynamoDB. This was the default database used by Amazon Amplify. The key name in the schema type is equal to the table name in the database, and the @model describes the attributes. The name of the attributes is the key for this type and the value of the attributes is the data type. The exclamation marks mean it is a non-nullable field.

Figure 5.5.3. below shows the DynamoDB GraphQL schema.

```
type Event @model {
  id: ID!
  img: String!
  alt: String!
  title: String!
  date: String!
  duration: String!
  location: String!
}
```

Figure 5.5.3. – DynamoDB GraphQL Schema

After implementing the non-relational database, the client decided, they would prefer to use a relational database. AWS Amplify currently uses the Amazon Aurora MySQL database as the default relational database which suited the use case. Figure 5.5.4. shows the part of the schema for the relational database, it slightly differs from DynamoDB's schema. The full schema can be seen in Appendix D.

```
type SecurityEvents {  
  id: Int!  
  img: String!  
  alt: String!  
  title: String!  
  eventDate: Int!  
  duration: Int!  
  location: String!  
  description: String  
}
```

Figure 5.5.4. – Relational database GraphQL Schema

When setting up the database for a relational database, the GraphQL API schema had to be updated. For each table in the database a type, an input type and an update type had to be declared. The mutation and query for each type had to have a corresponding list, get delete, create, and update operation declared. This was done in the `schema.graphql` file and that had to be pushed to Amplify to update it. This then pre-generated some code (queries, mutations). Then in the AWS Console, under AWS RDS Query Editor the tables had to be created using create table statements. That way the schema was synchronised with the database and AWS could use the preconfigured resolver function to map the GraphQL API requests to the Serverless MYSQL database. All the create table statements can be seen in Appendix E as can the `schema.graphql` file. All the tables in the database can be seen below:



Rows returned (9)	
<input type="text" value="Search rows"/>	
Tables_in_Events	
Attendee	
Badge	
EventPresentation	
Location	
Presentation	
PresentationMaterial	
Role	
SecurityEvents	
User	

Figure 5.5.5. – Tables in the database

On the AWS website, the current region was set to Europe (London) eu-west-2, but Amazon Aurora only worked in US East (N. Virginia) us-east-1 region at the time. This meant the database was stored in a different region. Figure 5.5.6. shows the query editor, where different queries can be written, and it will display the results. When “Select \* From SecurityEvents” is typed in and then run the query, there are three results found. This is matching with the events list page, it also has those three events visible.

Amazon RDS

Dashboard

Databases

Query Editor

Performance insights

Snapshots

Automated backups

Reserved instances

Proxies

Subnet groups

Parameter groups

Option groups

Custom engine versions

Events

Event subscriptions

Recommendations

Certificate update

RDS > Editor: rds-1

Editor

Recent

Saved queries

```
1 select * from SecurityEvents
```

Run

Save

Clear

Change database

Output

Result set 1 (3)

Rows returned (3)

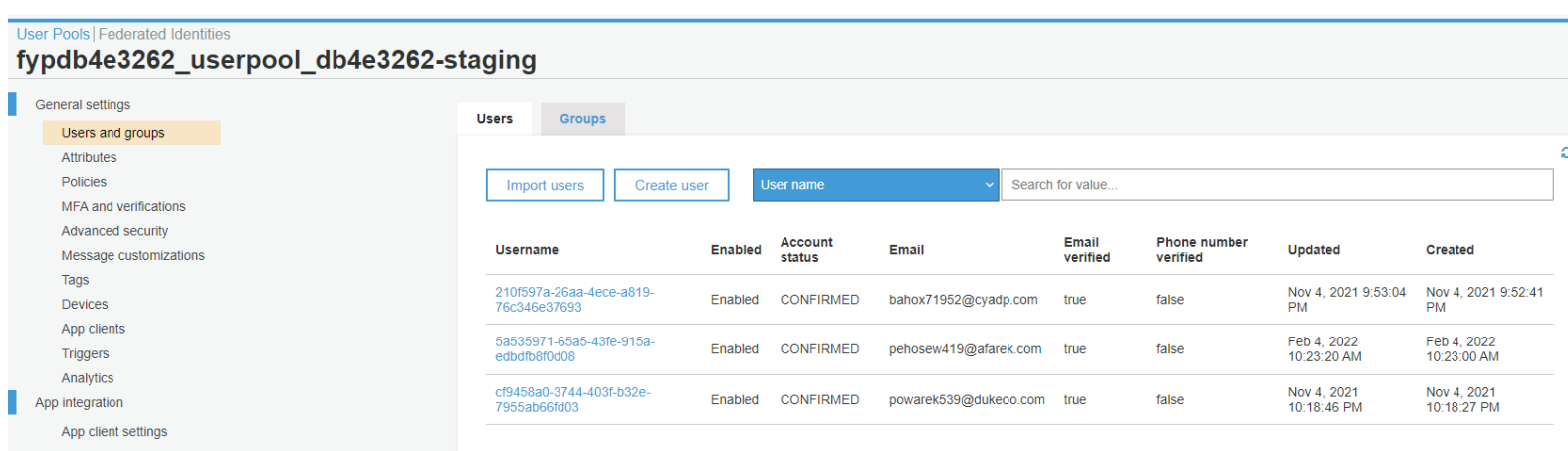
Export to csv

< 1 >

id	img	alt	title	location	eventDate	duration	description
1	https://images.unsplash.com/photo-1614064641938-3bbee52942c7?ixid=MnwxMjA3fDB8MhwaG90by1wYVdlfHx8fGVufDB8fHx8&ixlib=rb-1.2.1&auto=format&fit=crop&w=1170&q=80	padlock on a laptop keyboard	Cyber Safe	Online	1638986400	3	The cyber safe event will focus on teaching people about best practices online. If you would like to be cyber safe, then don't miss out attended to this wonderful event.
2	https://images.unsplash.com/photo-1498050108023-c5249f4df085?ixid=MnwxMjA3fDB8MhwaG90by1wYVdlfHx8fGVufDB8fHx8&ixlib=rb-1.2.1&auto=format&fit=crop&w=2072&q=80	laptop with code on a desk	How to prevent cyber attacks?	Online	1638468000	2	How to prevent cyber attacks? the event will focus on teaching people about different cyber attacks and tricks on how to prevent them. If you would like to learn more, then don't miss out attended to this wonderful event.
3	https://images.unsplash.com/photo-1580584126903-c17d41830450?ixid=MnwxMjA3fDB8MhwaG90by1wYVdlfHx8fGVufDB8fHx8&ixlib=rb-1.2.1&auto=format&fit=crop&w=1939&q=80	red and black chip card	Be more cyber aware	Online	1642532400	2	The Be more cyber aware event will focus on teaching people how to stay safe online. If you would like to learn more, then don't miss out, attended to this wonderful event.

Figure 5.5.6. – Amazon RDS Query Editor

Users of the system can be seen in the AWS dashboard using Cognito and the user pool associated with the application. It is possible to view the attributes of the users that have authenticated with the system (see the screenshot below). Figure 5.5.7. has all the data about the signed-up users.



User Pools | Federated Identities

**fypdb4e3262\_userpool\_db4e3262-staging**

General settings

Users and groups

Attributes

Policies

MFA and verifications

Advanced security

Message customizations

Tags

Devices

App clients

Triggers

Analytics

App integration

App client settings

Groups

Import users

Create user

User name

Search for value...

Username	Enabled	Account status	Email	Email verified	Phone number verified	Updated	Created
210f597a-26aa-4ece-a819-76c346e37693	Enabled	CONFIRMED	bahox71952@cyadp.com	true	false	Nov 4, 2021 9:53:04 PM	Nov 4, 2021 9:52:41 PM
5a535971-65a5-43fe-915a-edbdfb8f0d08	Enabled	CONFIRMED	pehosew419@afarek.com	true	false	Feb 4, 2022 10:23:20 AM	Feb 4, 2022 10:23:00 AM
cf9458a0-3744-403f-b32e-7955ab66fd03	Enabled	CONFIRMED	powarek539@dukeoo.com	true	false	Nov 4, 2021 10:18:46 PM	Nov 4, 2021 10:18:27 PM

Figure 5.5.7. – Registered users

## 5.6 Conclusion

The main parts of this chapter were focused on tooling and describing the implementation for the front end and back end of the project. The front-end of the application utilises a React single page application with reusable components. In the back-end part, the databases are explained along with GraphQL API that joins everything together.

## 6 Validation and Testing

### 6.1 Introduction

In this chapter, different testing methods will be discussed. First, the database will be tested by running different SQL statements and observing their results. This will be followed by manual functional testing; this type of testing checks if the application functions work as expected against the functional requirements outlined in the prior chapter. Finally, some end-to-end test automation will be done using Cypress and Cucumber; an end to end (E2E) JavaScript-based testing framework.

### 6.2 Database testing

The database has been tested using the Query editor inside the Amazon console. This has been done by running SQL statements and checking the results. It is important to test the database to see if it gives the right outcome as dynamic applications are heavily dependent upon the database. All CRUD (create, read, update, and delete) statements have been tested against the main security events table.

ID	Requirement	SQL statement	Result
1	Display all events	SELECT id, img, alt, title, location, eventDate, duration, description FROM SecurityEvents;	Pass

```
1 SELECT id, img, alt, title, location, eventDate, duration, description FROM SecurityEvents;
```

Run

Save

Clear

Output

Result set 1 (3)

Rows returned (3)

Export to csv

Search rows

id	img	alt	title	location	eventDate	duration	description
1	<a href="https://images.unsplash.com/photo-1614064641938-3bbee52942c7?ixid=MnwzMjA3fDB8MHxwaG90by1wYWdlHx8fGVuLDB8fHx8&amp;ixlib=rb-1.2.1&amp;auto=format&amp;fit=crop&amp;w=1170&amp;q=80">https://images.unsplash.com/photo-1614064641938-3bbee52942c7?ixid=MnwzMjA3fDB8MHxwaG90by1wYWdlHx8fGVuLDB8fHx8&amp;ixlib=rb-1.2.1&amp;auto=format&amp;fit=crop&amp;w=1170&amp;q=80</a>	padlock on a laptop keyboard	Cyber Safe	Online	1638986400	3	The cyber safe event will focus on teaching people about best practices online. If you would like to be cyber safe, then don't miss out attended to this wonderful event.
2	<a href="https://images.unsplash.com/photo-1498050108023-c5249f4df085?ixid=MnwzMjA3fDB8MHxwaG90by1wYWdlHx8fGVuLDB8fHx8&amp;ixlib=rb-1.2.1&amp;auto=format&amp;fit=crop&amp;w=2072&amp;q=80">https://images.unsplash.com/photo-1498050108023-c5249f4df085?ixid=MnwzMjA3fDB8MHxwaG90by1wYWdlHx8fGVuLDB8fHx8&amp;ixlib=rb-1.2.1&amp;auto=format&amp;fit=crop&amp;w=2072&amp;q=80</a>	laptop with code on a desk	How to prevent cyber attacks?	Online	1638468000	2	How to prevent cyber attacks? the event will focus on teaching people about different cyber attacks and tricks on how to prevent them. If you would like to learn more, then don't miss out attended to this wonderful event.
3	<a href="https://images.unsplash.com/photo-1580584126903-c17d41830450?ixid=MnwzMjA3fDB8MHxwaG90by1wYWdlHx8fGVuLDB8fHx8&amp;ixlib=rb-1.2.1&amp;auto=format&amp;fit=crop&amp;w=1939&amp;q=80">https://images.unsplash.com/photo-1580584126903-c17d41830450?ixid=MnwzMjA3fDB8MHxwaG90by1wYWdlHx8fGVuLDB8fHx8&amp;ixlib=rb-1.2.1&amp;auto=format&amp;fit=crop&amp;w=1939&amp;q=80</a>	red and black chip card	Be more cyber aware	Online	1642532400	2	The Be more cyber aware event will focus on teaching people how to stay safe online. If you would like to learn more, then don't miss out, attended to this wonderful event.

2	Insert a new event	INSERT INTO SecurityEvents(img,alt,title,location,eventDate,duration, description) VALUES('https://images.unsplash.com/photo-1548092372-0d1bd40894a3?ixlib=rb-1.2.1&ixid=MnwxMjA3fDB8MHxwaG90by1wYWdlfHx8fGVufDB8fHx8&auto=format&fit=crop&w=1470&q=80','half closed laptop','Cybersecurity','Online',1651773600,1, 'If you would like to learn more about cybersecurity, do not hesitate to sign up for this event.')	Pass												
<div><pre>1 INSERT INTO SecurityEvents(img,alt,title,location,eventDate,duration, description) 2 VALUES('https://images.unsplash.com/photo-1548092372-0d1bd40894a3?ixlib=rb-1.2.1&amp;ixid=MnwxMjA3fDB8MH 3 xwaG90by1w 4 YWdlfHx8fGVufDB8fHx8&amp;auto= 5 format&amp;fit=crop&amp;w=1470&amp;q=80','half closed laptop','Cybersecurity','Online',1651773600,1,'If you would like to learn more about cybersecurity, do not hesitate to sign up for this event. 6</pre></div> <div><div>RunSaveClear</div><div>Output</div><div>Statements (1)<div>Search rows</div><div>Export to csv</div><div>&lt; 1 &gt;</div><table><tr><th></th><th>Start</th><th>Statement</th><th>Status</th><th>Response</th><th>Duration</th></tr><tr><td>1</td><td>14:22:46</td><td>INSERT INTO SecurityEvents(img,alt,title,location,eventDate,duration, description) VALUES('https://images.unsplash.com/photo-1548092372-0d1bd40894a3?ixlib=rb-1.2.1&amp;ixid=MnwxMjA3fDB8MHxwaG90by1wYWdlfHx8fGVufDB8fHx8&amp;auto=format&amp;fit=crop&amp;w=1470&amp;q=80','half closed laptop','Cybersecurity','Online',1651773600,1,'If you would like to learn more about cybersecurity, do not hesitate to sign up for this event.')</td><td>Success</td><td>1 row updated</td><td>275 ms</td></tr></table></div></div>					Start	Statement	Status	Response	Duration	1	14:22:46	INSERT INTO SecurityEvents(img,alt,title,location,eventDate,duration, description) VALUES('https://images.unsplash.com/photo-1548092372-0d1bd40894a3?ixlib=rb-1.2.1&ixid=MnwxMjA3fDB8MHxwaG90by1wYWdlfHx8fGVufDB8fHx8&auto=format&fit=crop&w=1470&q=80','half closed laptop','Cybersecurity','Online',1651773600,1,'If you would like to learn more about cybersecurity, do not hesitate to sign up for this event.')	Success	1 row updated	275 ms
	Start	Statement	Status	Response	Duration										
1	14:22:46	INSERT INTO SecurityEvents(img,alt,title,location,eventDate,duration, description) VALUES('https://images.unsplash.com/photo-1548092372-0d1bd40894a3?ixlib=rb-1.2.1&ixid=MnwxMjA3fDB8MHxwaG90by1wYWdlfHx8fGVufDB8fHx8&auto=format&fit=crop&w=1470&q=80','half closed laptop','Cybersecurity','Online',1651773600,1,'If you would like to learn more about cybersecurity, do not hesitate to sign up for this event.')	Success	1 row updated	275 ms										
3	Update an event	UPDATE SecurityEvents SET title = "Annual cybersecurity meeting" WHERE id = 4	Pass												
<div><pre>1 UPDATE SecurityEvents 2 SET title = "Annual cybersecurity meeting" 3 WHERE id = 4</pre></div> <div><table><tr><th></th><th>Start</th><th>Statement</th><th>Status</th><th>Response</th><th>Duration</th></tr><tr><td>1</td><td>17:23:42</td><td>UPDATE SecurityEvents SET title = "Annual cybersecurity meeting" WHERE id = 4</td><td>Success</td><td>1 row updated</td><td>361 ms</td></tr></table></div>					Start	Statement	Status	Response	Duration	1	17:23:42	UPDATE SecurityEvents SET title = "Annual cybersecurity meeting" WHERE id = 4	Success	1 row updated	361 ms
	Start	Statement	Status	Response	Duration										
1	17:23:42	UPDATE SecurityEvents SET title = "Annual cybersecurity meeting" WHERE id = 4	Success	1 row updated	361 ms										
4	Delete an event	DELETE FROM SecurityEvents WHERE id=4	Pass												
<div><pre>1 DELETE FROM SecurityEvents 2 WHERE id=4</pre></div> <div><table><tr><th></th><th>Start</th><th>Statement</th><th>Status</th><th>Response</th><th>Duration</th></tr><tr><td>1</td><td>17:29:5</td><td>DELETE FROM SecurityEvents WHERE id=4</td><td>Success</td><td>1 row updated</td><td>397 ms</td></tr></table></div>					Start	Statement	Status	Response	Duration	1	17:29:5	DELETE FROM SecurityEvents WHERE id=4	Success	1 row updated	397 ms
	Start	Statement	Status	Response	Duration										
1	17:29:5	DELETE FROM SecurityEvents WHERE id=4	Success	1 row updated	397 ms										

### 6.3 Functional Testing (Manual testing)

Table 6.3.1. lists all the tests that have been performed to see if the application functions work as expected. The tests have been performed by the creator of the application and the author of this dissertation. The table details the list of functions; the steps that are needed to be carried out to achieve the expected behaviour; the expected behaviour from the system and the result.

ID	Function	Steps	Expected behaviour	Result
1	Edit an event	<ol style="list-style-type: none"> <li>1. Click on Events on the navbar</li> <li>2. Click on the Edit button on an event</li> <li>3. Fill out the login form</li> <li>4. Click submit</li> <li>5. Fill in details in the form</li> <li>6. Click update event</li> </ol>	Users should see a message saying, "Events successfully created!"	Pass
2	Create an event	<ol style="list-style-type: none"> <li>1. Click on Events on the navbar</li> <li>2. Click on Create an Event button</li> <li>3. Fill out the login form</li> <li>4. Click submit</li> <li>5. Fill in details in the form</li> <li>6. Click Add event</li> </ol>	Users should see a message saying, "Events successfully created!"	Pass
3	Delete an event	<ol style="list-style-type: none"> <li>1. Click on Events on the navbar</li> <li>2. Click on the Delete button</li> </ol>	The event should disappear from the event list page	Pass
4	Sign out	<ol style="list-style-type: none"> <li>1. After signing in on the create an event page</li> <li>2. Click on the Sign out button</li> </ol>	Users should see the login page after signing out	Pass
5	Sign up	<ol style="list-style-type: none"> <li>1. Click on Events on the navbar</li> <li>2. Click on Create an Event button</li> <li>3. Click on create account</li> <li>4. Fill out the form</li> <li>5. Click create account</li> <li>6. Fill out the form, get the confirmation code from the email</li> <li>7. Click confirm</li> </ol>	Users should see the create an event page	Pass
6	Log in	<ol style="list-style-type: none"> <li>1. Click on Events on the navbar</li> <li>2. Click on Create an Event button</li> <li>3. Fill out login details</li> <li>4. Click submit</li> </ol>	Users should see the create an event page	Pass



7.	Log in with the wrong email or password	<ol style="list-style-type: none"> <li>1. Click on Events on the navbar</li> <li>2. Click on Create an Event button</li> <li>3. Fill out login details with the wrong email or password</li> <li>4. Click submit</li> </ol>	Users should see a banner saying, "User does not exist." And form should be cleared	Pass
8.	Reset password	<ol style="list-style-type: none"> <li>1. Click on Events on the navbar</li> <li>2. Click on Create an Event button</li> <li>3. Click on reset password</li> <li>4. Fill in the username field</li> <li>5. Click send code</li> <li>6. Enter the code from the email</li> <li>7. Enter new password</li> <li>8. Click submit button</li> </ol>	Users should get a verification code for their email. After going through the whole process, they should see the login page	Pass
9.	Signing up with an existing account	<ol style="list-style-type: none"> <li>1. Click on Events on the navbar</li> <li>2. Click on Create an Event button</li> <li>3. Click on create account</li> <li>4. Fill out the form</li> <li>5. Click create account</li> </ol>	Users should see a banner saying, "An account with the given email already exists." They should also stay on the sign-up page.	Pass
10.	View event details	<ol style="list-style-type: none"> <li>1. Click on Events on the navbar</li> <li>2. Click on the View more info button</li> </ol>	Users should see all the information about the event	Pass

Table 6.3.1. – Functional manual testing

Note: In this application logging in / signing up is facilitated with a wrapper for the create event form. To access the login / sign up you will need to be logged out and visit the create event page. There is no dedicated login or sign-up page.

## 6.4 Automated testing

An automated end to end test framework has been used for part of the project. This was done to create a fast way to automate testing allowing for safe changes to the code. The tests have been automated using Cypress, a JavaScript-based end to end (E2E) automation framework and a Cucumber testing plugin that assists with BDD (Behaviour Driven Development). These tests are meant to be simple so they can be read by anyone. This is useful when the application is deployed as it helps establish requirements and testable acceptance criteria for designers and product people (Mustafić, 2020). When writing Cypress tests a scenario must be written. This would detail the steps that an end-user would perform to archive a task. This checks if all the steps get the expected outcome. Cypress controls the browser and automates the activity as if it was a real end-user performing the tests.

An example test is shown and described below:

```
Scenario: View Event Details
  Given I am on the event List
  When I click on an event
  Then I should be taken to the event details page
```

Figure 6.4.1. – Feature file describing the scenario

```
import { Given, When, Then } from 'cypress-cucumber-preprocessor/steps'
```

```
Given('I am on the event List',()=>{
  cy.visit('http://localhost:3000/events')
})

When('I click on an event',()=>{
  cy.contains('View More Info').click()
})

Then('I should be taken to the event details page',()=>{
  cy.url().should('include','/eventDetails/1')
  cy.contains('Cyber Safe').should('be.visible')
})
```

Figure 6.4.2. – The cypress test file

Automated tests setup for this project are detailed in the below table:

ID	Function	Steps	Expected behaviour	Result
1	View event list	- Go to the event list	See the list of events	Pass
2	View event details	-Go on the event list page - Click on the view more info button on an event	Should be navigated to the event details page	Pass
3	Create an event	-Go on the event list page -Click on create an event button -Enter all the detail for the event -Click submit button -Go back to the event list page	A confirmation message should be visible, and when navigating back to the event list page the new event should be visible	Pass

## 6.5 Conclusion

In this chapter, the different techniques used to test the application have been described. The database has been tested by using SQL statements. The application has then been tested using functional manual testing to see if the system works as expected. The final part of the testing has been using test automation to create end to end tests for key scenarios.

## 7 Critical Review and Conclusion

### 7.1 Introduction

This final chapter of the project will reflect on the outcome of the whole project. It will detail what has been achieved; also, what has not been achieved; and how that situation could have been better managed. The future work paragraph will explain what else could be implemented to make it better. Finally, the conclusion will talk about the whole project's learnings.

### 7.2 Outcome

The project has a mixed outcome. There have been some successes. One of the goals of this project was to implement a web application that uses a modern approach, techniques, and tools. The whole project was focused on this goal. This has been achieved by utilising the AWS Amplify framework that simplifies the creation and management of application backends. This helps the client because they can focus more on the content that would go on their web application and increase the number of events and users' contributions rather than spending time on maintaining the platform. The project also has been implemented with ReactJS, a modern web technology. This gives the user a modern single page application experience. In addition to this, it is easy to build out new pages and functions through the reusable components that have been created.

In addition to this, software engineering tooling was used well throughout the project. There is an automated build pipeline that deploys the code when changes are pushed to the version control meaning that the application is deployed automatically.

Another part that has gone well, was that the project was successfully adapted to the client's changing needs. This project started by implementing a NoSQL database, AWS DynamoDB, that is widely used by different companies. After implementing it, the client realised it would better fit their business needs to have a relational database. They already have a relational database with many tables, and it would be too complicated and time-consuming to translate it into a non-relational database. Because the project implemented the non-relational database on a smaller scale, it did not take too much time to realise it would not meet their needs.

On the other hand, some parts of the project did not go as planned. Many of the set-out requirements have not been implemented. This was due to a lack of time management and competing demands from other university modules. This meant that not all the Must requirements set out in section 3 were implemented. However, the database design and GraphQL APIs were fully implemented meaning that part of the implementation is there.

Another challenge was that it was also time consuming to learn complex new tools and technologies and implement them afterwards in such a small period of time. AWS amplify was particularly difficult to learn and implement due to an assumed knowledge of AWS. The project was also built inside a Linux virtual machine that kept having trouble with the git source control system which impacted the author's productivity.

Some sprint deliverables were a bit too ambitious as the learning time had not been factored in properly and some deadlines were missed. The documentation of this project also ended up taking longer than anticipated. It could have been written consistently, breaking up into smaller pieces rather than rushed in the last few weeks of the year.

### 7.3 Future Development

This project could be taken to next level by implementing more of the requirements that have been set at the beginning of the project and written up in the design section. The web application would

also benefit from a more appealing design; this is not complicated but was time-consuming, so the effort was spent elsewhere. Having said that, the site does implement some basic responsive web design using media queries, flexbox, and the CSS grid. The time spent on the learning and implementing new technologies and tools took away the time from implementing a better UI for the application. Fundamentally, the application would be more useful if more features would be available for the different types of users. That includes the more advanced admin panel to manage the users. Additionally, permissions and separation of the different types of users would have better fit the requirements outlined. The application also could store some information about the user sand usage that could be more beneficial to the client and help them understand their audience.

#### 7.4 Conclusion

To conclude, the whole project gave a good insight into the real working world and the software development lifecycle. The author learned the importance of breaking up the project into smaller pieces and the value of time management and working with a client. Having the client change their minds throughout the project added complications to the project that had not been factored into the timescale, but the Agile approach makes it possible to change requirements regarding the clients' needs, the scope was therefore exchanged, and fewer less important features were implemented as a result. Overall, the author developed lots of new skills, both soft skills and technical skills that they hope will help them in their first role.

## 8 References

Agile Business Consortium, (no date). *Chapter 10: MoSCoW Prioritisation | The DSDM Handbook*. Available at: [https://www.agilebusiness.org/page/ProjectFramework\\_10\\_MoSCoWPrioritisation](https://www.agilebusiness.org/page/ProjectFramework_10_MoSCoWPrioritisation) (Accessed: 22 March 2022).

Amazon Web Services. (2022a) *AWS Amplify Build extensible, full-stack web and mobile apps faster. Easy to start, easy to scale*. Available at: <https://aws.amazon.com/amplify/?nc=sn&loc=0> (Accessed: 28 March 2022)

Amazon Web Services. (2022b). *Amazon DynamoDB*. Available at: <https://aws.amazon.com/dynamodb/> (Accessed: 10 April 2022).

Amazon Web Services. (2022c). *Amazon Cognito*. Available at: <https://aws.amazon.com/cognito/> (Accessed: 10 April 2022).

Bakusevych, T. (2020) *Text fields & Forms design — UI components series, Medium*. Available at: <https://uxdesign.cc/text-fields-forms-design-ui-components-series-2b32b2beebd0> (Accessed: 26 March 2022).

Bigelow, S.J. (2015). How important is a container's host OS? Available at: <https://searchservervirtualization.techtarget.com/answer/How-important-is-a-containers-host-OS> (Accessed: 10 April 2022).

Chris, K. (2021). HTML vs. JSX - What's the Difference? Available at: <https://www.freecodecamp.org/news/html-vs-jsx-whats-the-difference/> (Accessed: 10 April 2022).

CISSE UK. (2019). *Our vision Collaborate to innovate!* Available at: <https://cisseuk.org/our-vision> (Accessed: 10 April 2022)

Daityari, S. (2021) *Angular vs React vs Vue: Which Framework to Choose in 2021, CodeinWP*. Available at: <https://www.codeinwp.com/blog/angular-vs-vue-vs-react/> (Accessed: 28 March 2022).

Dopico, A. (2020). *What is AJAX in single page application?* Available at: <https://janetpanic.com/what-is-ajax-in-single-page-application/> (Accessed: 10 April 2022).

Dushenin, O. (2021) *Monolithic Architecture. Advantages and Disadvantages*. Available at: <https://datamify.medium.com/monolithic-architecture-advantages-and-disadvantages-e71a603eec89> (Accessed: 23 March 2022)

Fee, N. (2020). *What Is Serverless Architecture? Key Benefits and Limitations*. Available at: <https://newrelic.com/blog/best-practices/what-is-serverless-architecture> (Accessed: 10 April 2022).

Fink, G., Flatow, I. and Group, S. (2014) *Pro Single Page Application Development: Using Backbone.js and ASP.NET*. Apress.

Friedenthal, S., Moore, A. and Steiner, R. (2014) *A Practical Guide to SysML*. 3rd edn. Elsevier Science. Available at: <https://www.perlego.com/book/1809648/a-practical-guide-to-sysml-pdf> (Accessed: 22 March 2022).

Google Trends (no date) *Google Trends*. Available at: <https://trends.google.com/trends/explore?cat=31&q=Vue%20jobs,React%20jobs,Angular%20jobs> (Accessed: 28 March 2022).

Govor, J. (2019). *Survival guide to Serverless and FaaS (Function-as-a-Service)*. Available at: <https://www.linkedin.com/pulse/survival-guide-serverless-faas-function-as-a-service-julia-govor> (Accessed: 10 April 2022).

Greif, S. (2017) *So what's this GraphQL thing I keep hearing about?* Available at: <https://www.freecodecamp.org/news/so-whats-this-graphql-thing-i-keep-hearing-about-baf4d36c20cf/> (Accessed: 4 February 2022).

Hoory, L. (2021) *What Is A Stakeholder Analysis? Everything You Need To Know, Forbes Advisor*. Available at: <https://www.forbes.com/advisor/business/what-is-stakeholder-analysis/> (Accessed: 22 March 2022).

IBM Cloud Education. (2021) *REST APIs*. Available at: <https://www.ibm.com/uk-en/cloud/learn/rest-apis> (Accessed: 21 March 2022).

IBM Docs (2021). Available at: <https://prod.ibmdocs-production-dal-6099123ce774e592a519d7c33db8265e-0000.us-south.containers.appdomain.cloud/docs/en/rational-soft-arch/9.6.1?topic=diagrams-use-case> (Accessed: 22 March 2022).

Jiang, L., Pei, Y. and Zhao, J. (2020) 'Overview Of Serverless Architecture Research', *Journal of Physics: Conference Series*. Available at: <https://iopscience.iop.org/article/10.1088/1742-6596/1453/1/012119/pdf> (Accessed: 23 March 2022).

Jones, D. (2018) Containers vs. Virtual Machines (VMs): What's the Difference? Available at: <https://www.netapp.com/blog/containers-vs-vms/> (Accessed: 10 April 2022).

Justinmind, (2019), *Card UI design: fundamentals and examples* (no date). Available at: <https://www.justinmind.com/blog/cards-ui-design/> (Accessed: 26 March 2022).

Juviler, J. (2021). *REST APIs: How They Work and What You Need to Know*. Available at: <https://blog.hubspot.com/website/what-is-rest-api> (Accessed: 10 April 2022).

Linx. (2021). *GraphQL vs REST - a low-code API showdown*. Available at: <https://linx.software/graphql-vs-rest-a-low-code-showdown/> (Accessed: 10 April 2022).

Microservices.io. (no date) Microservices Pattern: Monolithic Architecture pattern. Available at: <http://microservices.io/patterns/monolithic.html> (Accessed: 4 February 2022).

Microsoft. (2021). *Service-oriented architecture*. Available at: <https://docs.microsoft.com/en-us/dotnet/architecture/microservices/architect-microservice-container-applications/service-oriented-architecture> (Accessed: 10 April 2022).

MindTools, (no date). *Stakeholder Analysis: Winning support for your projects*. Available at: [https://www.mindtools.com/pages/article/newPPM\\_07.htm](https://www.mindtools.com/pages/article/newPPM_07.htm). (Accessed 23 March 2022).

Mustafić, A. (2020) Cypress Tests in BDD Style. Available at: <https://dev.to/bornfightcompany/cypress-tests-in-bdd-style-52n5> (Accessed: 10 April 2022).

Potdar, A.M. et al. (2020) 'Performance Evaluation of Docker Container and Virtual Machine', *Procedia Computer Science*. Available at: <https://reader.elsevier.com/reader/sd/pii/S1877050920311315?token=7A3D8F243D6305E79B7ADA E96D4F1D6E8531403BE43D04A8E6096AC6F8824AEA4C152C7DBA422CCD80AE070350041657&originRegion=eu-west-1&originCreation=20220327001021> (Accessed: 23 March 2022)

REST API Tutorial. (2022). *What is REST*. Available at: <https://restfulapi.net/> (Accessed: 21 March 2022).

Richardson, C. (2018) *Microservices Patterns*. Manning Publications. Available at: <https://www.perlego.com/book/1469067/microservices-patterns-pdf> (Accessed: 01 February 2022).

Schmitt, M. (2020). *Virtual Machine (VM): Why and When Do You Need One?*. Available at: <https://cloudzy.com/virtual-machine-vm-what-why-when/> (Accessed: 10 April 2022).

Sherman, R. (2014) *Business Intelligence Guidebook*. [edition missing]. Elsevier Science. Available at: <https://www.perlego.com/book/1810028/business-intelligence-guidebook-pdf> (Accessed: 10 April 2022).

Stemmler, K. (2021). *Building a GraphQL API- GraphQL API example*. Available at: <https://www.apollographql.com/blog/graphql/examples/building-a-graphql-api/> (Accessed: 10 April 2022).

Vennam, S. (2020). *Cloud Computing*. Available at: <https://www.ibm.com/cloud/learn/cloud-computing> (Accessed: 10 April 2022).

Wilson, P. (2015). *How to Fully Engage Your Readers' Brains with Images [SlideShare]*. Available at: <https://copyblogger.com/images-engage-readers/> (Accessed: 26 March 2022).

## 9 Appendices

### 9.1.1 Appendix A – Reusable components wireframes

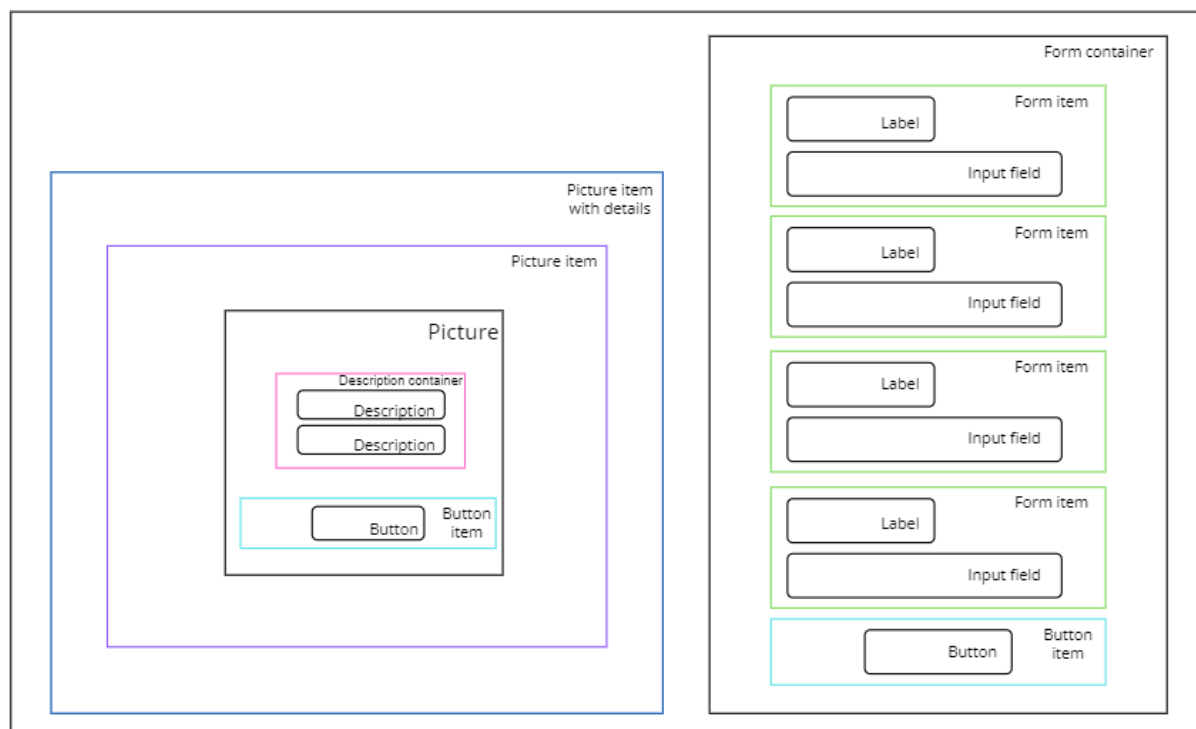


Figure A.1.

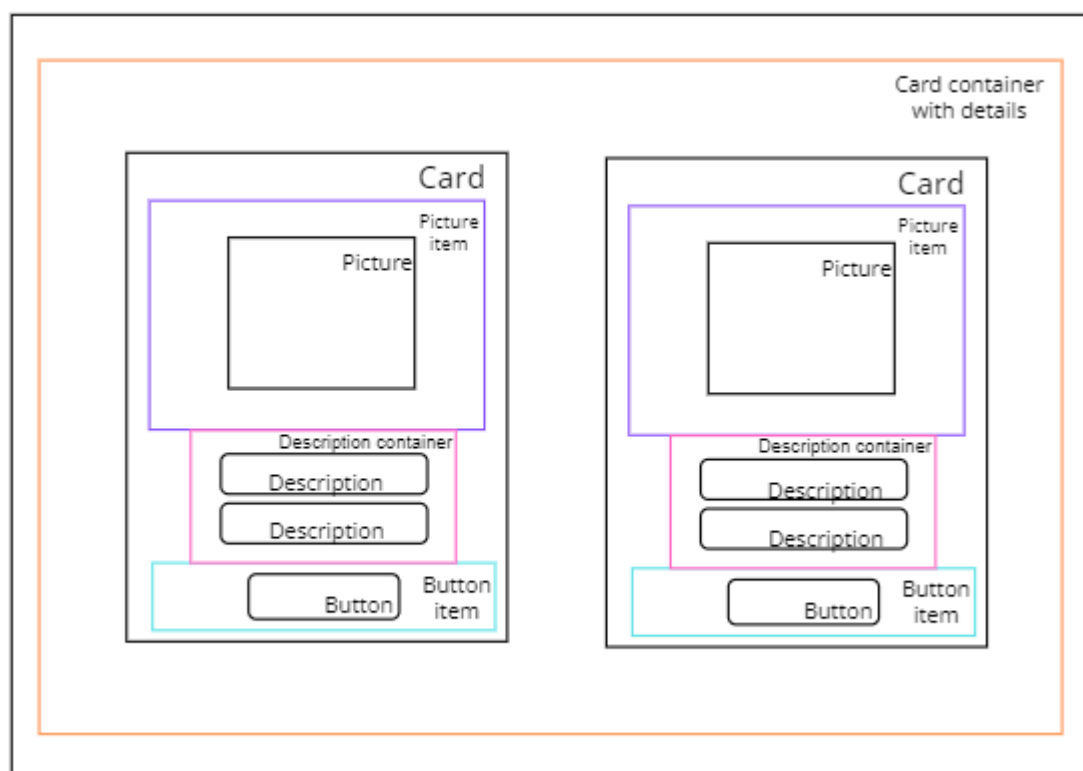


Figure A.2.



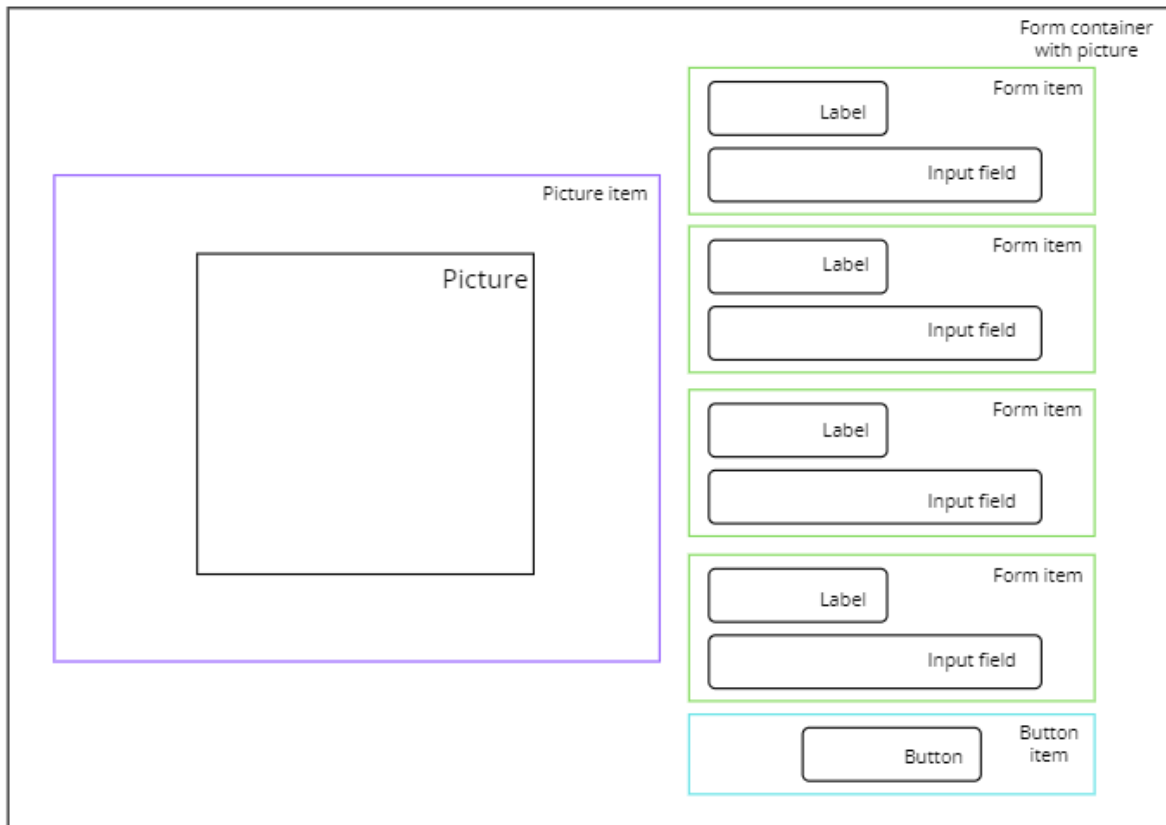


Figure A.3.

## 9.1.2 Appendix B – Wireframes

The wireframe shows a web browser window for 'website.com'. The header includes the site name 'Cisse UK' and navigation links: 'Events', 'About us', 'Log in', and 'Sign up'. The main content area features a 'Log in' form with the following elements:

- Log in** (Section Header)
- Email** (Label) above a text input field containing the placeholder 'Type your email address'.
- Password** (Label) above a text input field containing the placeholder 'Type your password'.
- A link: [If you forgot your password please click on this](#)
- A **SUBMIT** button.

Figure B.1.

website.com

☆ Cisse UK

EventsAbout usLog inSign up

Sign up

First name

Type your first name

Last name

Type your last name

Email address

Type your email address

Confirm email address

Confirm your email address

Password

Type your password

Confirm password

Confirm your password

Organisation

Type your organisation

SUBMIT

Figure B.2.

The image shows a web browser window with the address bar displaying 'website.com'. The browser's title bar includes a star icon, the text 'Cisse UK', and navigation links: 'Events', 'About us', 'Log in', and 'Sign up'. The main content area features a centered box titled 'My details'. Inside this box, there are six text input fields, each with a label above it: 'First name' (placeholder: 'Type your first name'), 'Last name' (placeholder: 'Type your last name'), 'Email address' (placeholder: 'Type your email address'), 'Password' (placeholder: 'Type your password'), 'Confirm password' (placeholder: 'Confirm your password'), and 'Organisation' (placeholder: 'Type your organisation'). Below these fields is a single button labeled 'Update'.

Figure B.3.

← →

website.com

×

☆ Cisse UK

EventsAbout usLog inSign up

## Edit an event

Image

Alt

Title

Date

Duration

Location

Description

Update

Figure B.4.

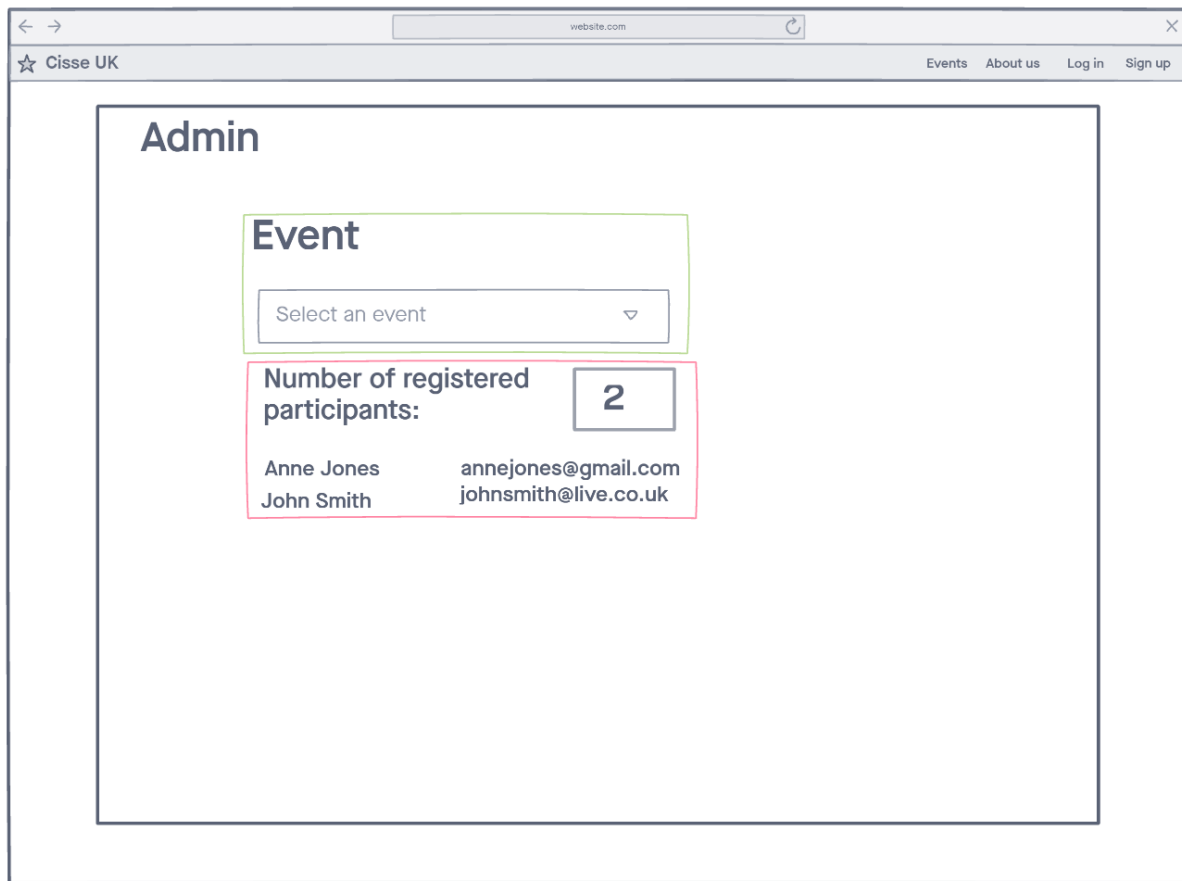


Figure B.5.

### 9.1.3 Appendix C – Sequence Diagrams

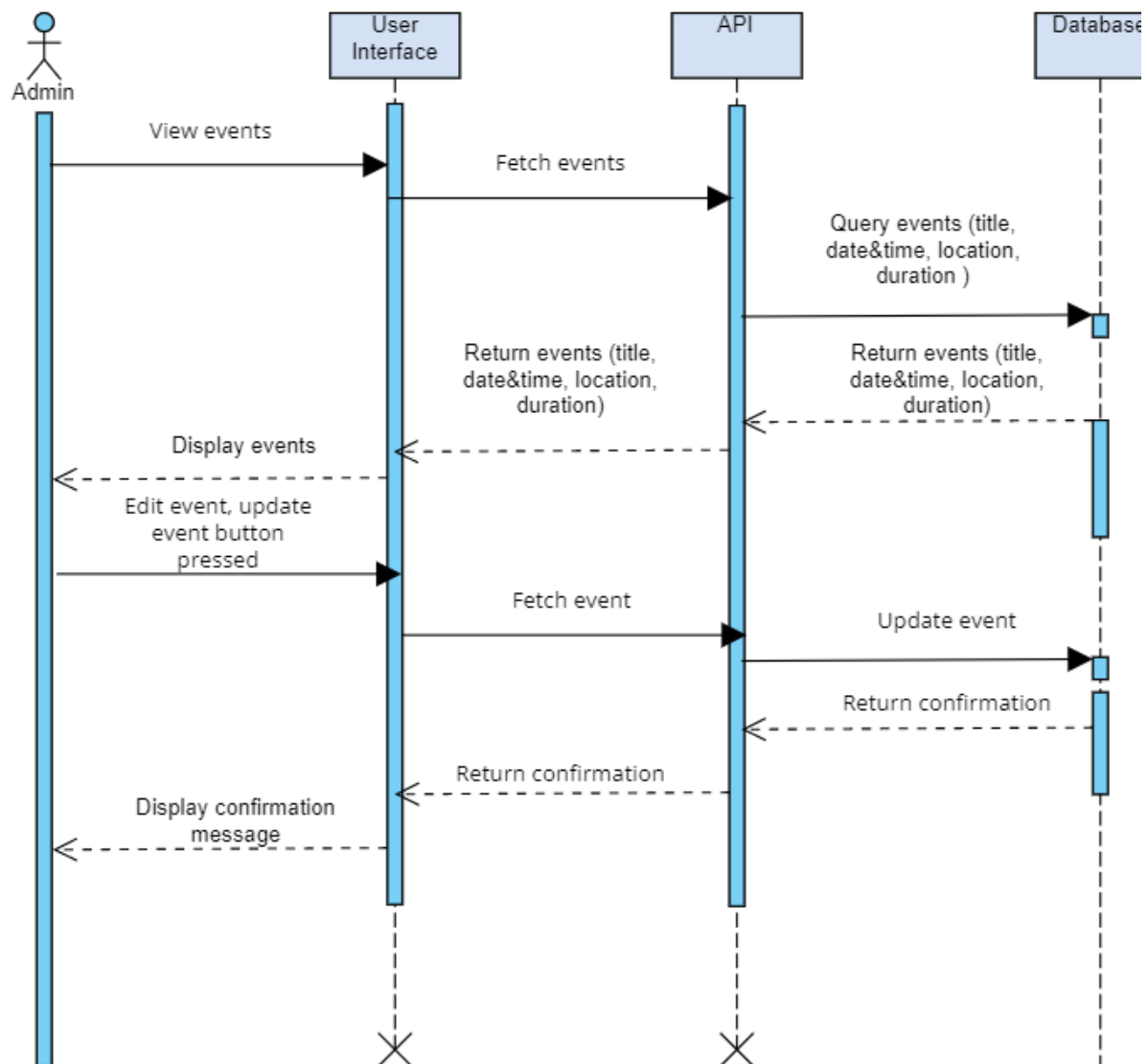


Figure C.1. Admin updating an event

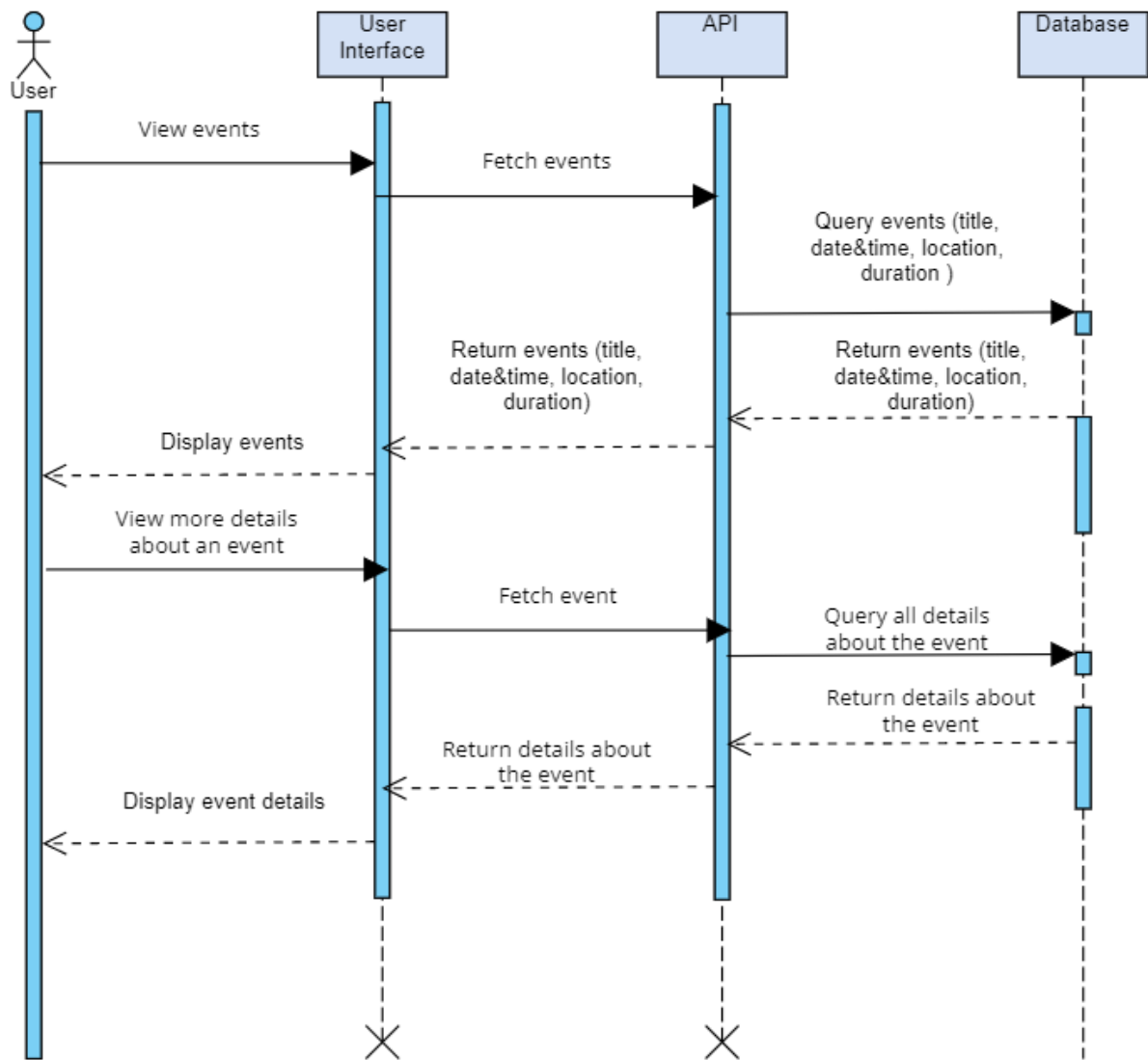


Figure C.2. User - view an event



## 9.1.4 Appendix D - GraphQL Schema

```

1  input CreateSecurityEventsInput {
2    id: Int!
3    img: String!
4    alt: String!
5    title: String!
6    eventDate: Int!
7    duration: Int!
8    location: String!
9    description: String!
10 }
11
12 type SecurityEvents {
13   id: Int!
14   img: String!
15   alt: String!
16   title: String!
17   eventDate: Int!
18   duration: Int!
19   location: String!
20   description: String!
21 }
22
23 input UpdateSecurityEventsInput {
24   id: Int!
25   img: String!
26   alt: String!
27   title: String!
28   eventDate: Int!
29   duration: Int!
30   location: String!
31   description: String!
32 }
33
34 type Attendee {
35   AttendeeID: Int!
36   UserID: Int!
37   EventID: Int!
38 }
39
40 input CreateAttendeeInput {
41   AttendeeID: Int!
42   UserID: Int!
43   EventID: Int!
44 }
45
46 input UpdateAttendeeInput {
47   AttendeeID: Int!
48   UserID: Int!
49   EventID: Int!
50 }
51
52 type Role {
53   RoleID: Int!
54   RoleName: String!
55 }
56
57 input CreateRoleInput {
58   RoleID: Int!
59   RoleName: String!
60 }
61
62 input UpdateRoleInput {
63   RoleID: Int!
64   RoleName: String!
65 }
66
67 type User {
68   UserID: Int!
69   FirstName: String!
70   LastName: String!
71   EmailAddress: String!
72   Organisation: String!
73   RoleID: Int!
74   BadgeID: Int!
75 }
76
77 input CreateUserInput {
78   UserID: Int!
79   FirstName: String!
80   LastName: String!
81   EmailAddress: String!
82   Organisation: String!
83   RoleID: Int!
84   BadgeID: Int!
85 }
86
87 input UpdateUserInput {
88   UserID: Int!
89   FirstName: String!
90   LastName: String!
91   EmailAddress: String!
92   Organisation: String!
93   RoleID: Int!
94   BadgeID: Int!
95 }
96
97 type EventPresentation {
98   EventPresentationID: Int!
99   EventID: Int!
100  PresentationID: Int!
101 }
102
103 input CreateEventPresentationInput {
104   EventPresentationID: Int!
105   EventID: Int!
106   PresentationID: Int!
107 }
108
109 input UpdateEventPresentationInput {
110   EventPresentationID: Int!
111   EventID: Int!
112   PresentationID: Int!
113 }
114
115 type Location {
116   LocationID: Int!
117   Address1: String!
118   Address2: String!
119   Town: String!
120   Postcode: String!
121   URL: String!
122 }
123
124 input CreateLocationInput {
125   LocationID: Int!
126   Address1: String!
127   Address2: String!
128   Town: String!
129   Postcode: String!
130   URL: String!
131 }
132
133 input UpdateLocationInput {
134   LocationID: Int!
135   Address1: String!
136   Address2: String!
137   Town: String!
138   Postcode: String!
139   URL: String!
140 }
141
142 type Presentation {
143   PresentationID: Int!
144   Name: String!
145   Topic: String!
146   Duration: String!
147   Author: String!
148 }
149
150 input CreatePresentationInput {
151   PresentationID: Int!
152   Name: String!
153   Topic: String!
154   Duration: String!
155   Author: String!
156 }
157
158 input UpdatePresentationInput {
159   PresentationID: Int!
160   Name: String!
161   Topic: String!
162   Duration: String!
163   Author: String!
164 }
165
166 type PresentationMaterial {
167   PresentationMaterialID: Int!
168   PresentationMaterialName: String!
169   URL: String!
170   Added: Int!
171   Updated: Int!
172   PresentationID: Int!
173 }
174
175 input CreatePresentationMaterialInput {
176   PresentationMaterialID: Int!
177   PresentationMaterialName: String!
178   URL: String!
179   Added: Int!
180   Updated: Int!
181   PresentationID: Int!
182 }
183
184 input UpdatePresentationMaterialInput {
185   PresentationMaterialID: Int!
186   PresentationMaterialName: String!
187   URL: String!
188   Added: Int!
189   Updated: Int!
190   PresentationID: Int!
191 }
192
193 type Badge {
194   BadgeID: Int!
195   BadgeName: String!
196   BadgeCriteria: String!
197 }
198
199 input CreateBadgeInput {
200   BadgeID: Int!
201   BadgeName: String!
202   BadgeCriteria: String!
203 }
204
205 input UpdateBadgeInput {
206   BadgeID: Int!
207   BadgeName: String!
208   BadgeCriteria: String!
209 }
210
211 type Mutation {
212   deleteSecurityEvents(id: Int!): SecurityEvents!
213   createSecurityEvents(
214     createSecurityEventsInput: CreateSecurityEventsInput!
215   ): SecurityEvents!
216   updateSecurityEvents(
217     updateSecurityEventsInput: UpdateSecurityEventsInput!
218   ): SecurityEvents!
219
220   deleteRole(id: Int!): Role!
221   createRole(createRoleInput: CreateRoleInput!): Role!
222   updateRole(updateRoleInput: UpdateRoleInput!): Role!
223
224   deleteUser(id: Int!): User!
225   createUser(createUserInput: CreateUserInput!): User!
226   updateUser(updateUserInput: UpdateUserInput!): User!
227
228   deleteAttendee(id: Int!): Attendee!
229   createAttendee(createAttendeeInput: CreateAttendeeInput!): Attendee!
230   updateAttendee(updateAttendeeInput: UpdateAttendeeInput!): Attendee!
231
232   deleteEventPresentation(id: Int!): EventPresentation!
233   createEventPresentation(
234     createEventPresentationInput: CreateEventPresentationInput!
235   ): EventPresentation!
236   updateEventPresentation(
237     updateEventPresentationInput: UpdateEventPresentationInput!
238   ): EventPresentation!
239 }

```

```

240 deleteLocation(id: Int!): Location
241 createLocation(createLocationInput: CreateLocationInput!): Location
242 updateLocation(updateLocationInput: UpdateLocationInput!): Location
243
244 deletePresentation(id: Int!): Presentation
245 createPresentation(
246   createPresentationInput: CreatePresentationInput!
247 ): Presentation
248 updatePresentation(
249   updatePresentationInput: UpdatePresentationInput!
250 ): Presentation
251
252 deletePresentationMaterial(id: Int!): PresentationMaterial
253 createPresentationMaterial(
254   createPresentationMaterialInput: CreatePresentationMaterialInput!
255 ): PresentationMaterial
256 updatePresentationMaterial(
257   updatePresentationMaterialInput: UpdatePresentationMaterialInput!
258 ): PresentationMaterial
259 }
260
261 type Query {
262   getSecurityEvents(id: Int!): SecurityEvents
263   listSecurityEventss: [SecurityEvents]
264   getLocation(id: Int!): Location
265   listLocations: [Location]
266   getPresentation(id: Int!): Presentation
267   listPresentations: [Presentation]
268   getPresentationMaterial(id: Int!): PresentationMaterial
269   listPresentationMaterials: [PresentationMaterial]
270   getBadge(id: Int!): Badge
271   listBadges: [Badge]
272   getAttendee(id: Int!): Attendee
273   listAttendees: [Attendee]
274   getUser(id: Int!): User
275   listUsers: [User]
276   getRole(id: Int!): Role
277   listRoles: [Role]
278 }
279
280 type Subscription {
281   onCreateSecurityEvents: SecurityEvents
282   @aws_subscribe(mutations: ["createSecurityEvents"])
283 }
284
285 schema {
286   query: Query
287   mutation: Mutation
288   subscription: Subscription
289 }
290

```

Figure D.1. – GraphQL Schema

### 9.1.5 Appendix E. – Create table statements

```

CREATE TABLE Role (
    RoleID int(8) NOT NULL PRIMARY KEY AUTO_INCREMENT,
    RoleName varchar(50) NOT NULL
);

CREATE TABLE User (
    UserID int(8) NOT NULL PRIMARY KEY AUTO_INCREMENT,
    FirstName varchar(50) NOT NULL,
    LastName varchar(50) NOT NULL,
    EmailAddress varchar(250) NOT NULL,
    Organisation varchar(100) NOT NULL,
    RoleID int(8) NOT NULL,
    BadgeID int(8) NOT NULL
);

CREATE TABLE Attendee (
    AttendeeID int(8) NOT NULL PRIMARY KEY AUTO_INCREMENT,
    UserID int(8) NOT NULL,
    EventID int(8) NOT NULL
);

CREATE TABLE EventPresentation (
    EventPresentationID int(8) NOT NULL PRIMARY KEY AUTO_INCREMENT,
    EventID int(8) NOT NULL,
    PresentationID int(8) NOT NULL
);

CREATE TABLE Location (
    LocationID int(8) NOT NULL PRIMARY KEY AUTO_INCREMENT,
    Address1 varchar(250),
    Address2 varchar(250),
    Town varchar(40),
    Postcode varchar(8),
    URL varchar(250)
);

CREATE TABLE Presentation (
    PresentationID int(8) NOT NULL PRIMARY KEY AUTO_INCREMENT,
    Name varchar(50) NOT NULL,
    Topic varchar(50),
    Duration varchar(3),
    Author varchar(100)
);

CREATE TABLE PresentationMaterial (
    PresentationMaterialID int(8) NOT NULL PRIMARY KEY AUTO_INCREMENT,
    PresentationMaterialName varchar(50) NOT NULL,
    URL varchar(250) NOT NULL,
    Added int(10) NOT NULL,
    Updated int(10) NOT NULL,
    PresentationID int(8) NOT NULL
);

```

```
CREATE TABLE Badge (  
    BadgeID int(8) NOT NULL PRIMARY KEY AUTO_INCREMENT,  
    BadgeName varchar(50) NOT NULL,  
    BadgeCriteria varchar(50) NOT NULL  
);
```