Planning and Design Document

Jack McCullough, Luca Orita, Maria Bracegirdle, David Yang Zhang, Rebecca French

1 Introduction

This document will detail how we design the system that will meet the requirements set out in the "Requirements and Analysis" document. The main focus of this system would be implementing a mood retrieval system that will make use of machine learning in the form of language processing. This input data will then be displayed to the presenter via a series of graphs and concise visualisations.

2 Glossary

NLP: Natural language processing is a subfield of computer science that concerns itself in the interpretation of the human language. This can involve speech recognition, language generation or in our case, sentiment analysis. This is usually done by tokenising lexicons and extracting the meaning in context.
DOM: The Document Object Model [1] is a programming interface for HTML and XML documents. It represents the page so that the program can modify the style and content of the webpage. This is done by representing every webpage element as nodes, for which the backend can individually alter.
Virtual DOMs: A virtual DOM is a concept where a virtual. representation of the UI is kept in memory and synced with the actual DOM. This allows us to increase efficiency by not having to reload all the components of the webpage.
Jest: Jest is a library that allows us to make use of virtual DOMs in testing and allows us to test out rendered components.
SQL: Structured Query Language, an industry for performing data operations on databases.
Agile Approach: An approach where the activities to be completed are planned incrementally and then the planned activities are completed, before planning the next set of activities, releasing the system in a series of versions.

3 Planning

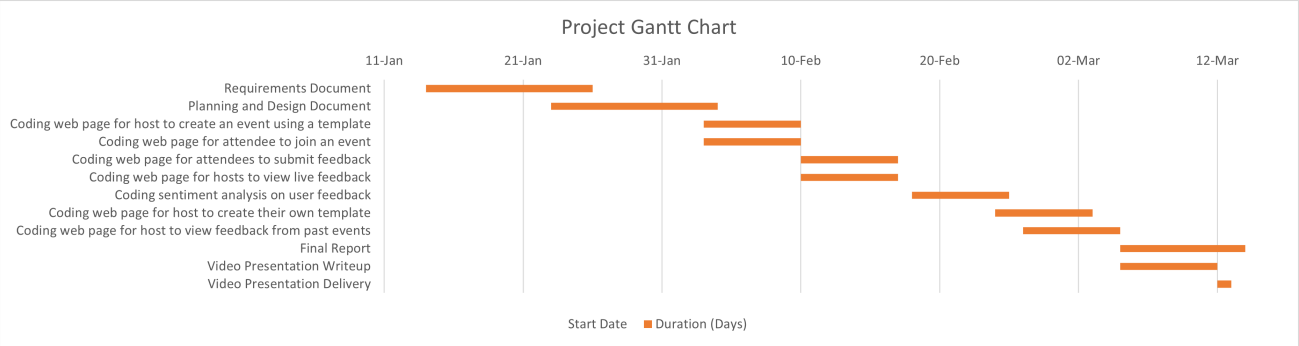

Figure 1: Gantt Chart showing our planned development process

As a group, we aim to meet at least once a week to monitor progress and communicate to ensure we know what each member of the team will achieve in the coming week, following the plan that we have created. We will utilise Microsoft Teams to communicate, using Git to work collaboratively on code,

ensuring that everyone uses an up-to-date version of the software. It also allows us to ensure proper version control so that if the code stops working, we can revert to a point where it did work. There were multiple instances where one meeting per week was not enough to resolve the problems encountered during development. Therefore, we organized multiple meetings, with part of the members for specific problems and the whole team when we had to rethink different aspects of the projects and make important decisions.

3.1 Team Organisation

After rigorous discussions, we have decided to adopt a plan-driven approach for this project. In a plan-driven approach, all of the process activities are planned, thus providing a more structured plan of work, processes being measured against the initial plan. We have some specifications from the Deutsche Bank and we will define the rest of them ourselves. Furthermore, many of us haven't worked on a project with such a big team and seeing the difficult circumstances in which we need to work, planning what each of us needs to do for the week ahead and being able to check our progress more easily would be beneficial, respecting the deadlines and ensuring more coordinated development. We decided against using an agile approach because it would require a lot of customer involvement which we don't have. The main advantage of an agile approach is the ability to accommodate changing requirements, at the cost of a less rigid plan and a more challenging development process. As we expect the requirements of this project to remain the same throughout, an agile approach brings few benefits so the waterfall model is preferable.

4 Design

This section will give an overview of how our system is going to look like and the reasoning behind our choices. We have outlined the use-cases, the way the system interacts with itself internally, and what types of languages/libraries we are going to ulilize.

4.1 Use-Case Diagram

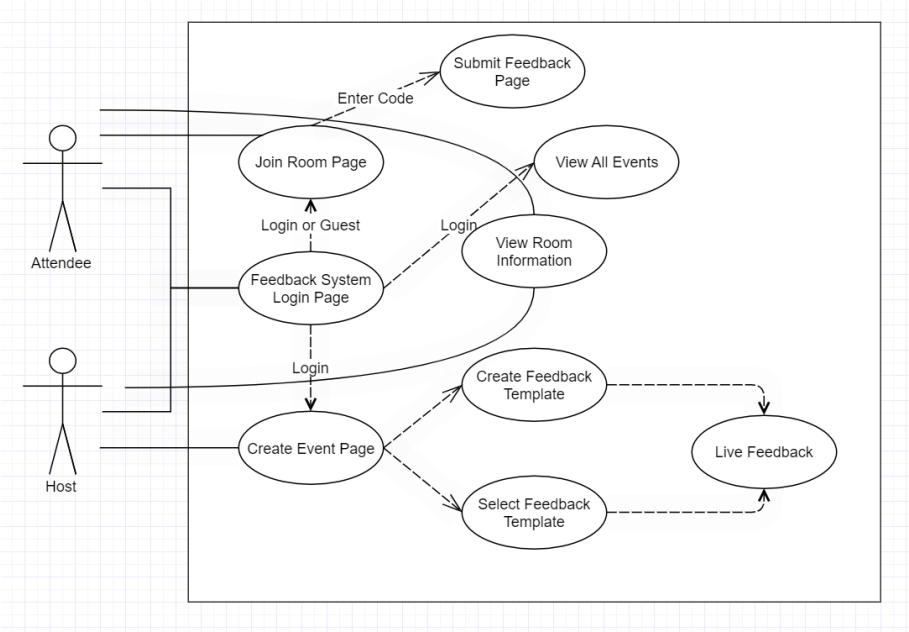

Figure 2: Use Case Diagram showing the intended functionality of our system

Two types of users will interact with the system, 'hosts', who create an event, and 'attendees', who deliver feedback for an event. An account is required to be a host so that any feedback templates they create can be accessed again by logging into their account. An attendee can create an account

to access the system, or they can access the system as a guest.

A host selects to create an event on the 'Create Event Page' and from there, they can either select a feedback template or create one. Once a feedback template is used, they will go to 'Live Feedback' as the event has been created. An attendee chooses to log in or be a guest. Once they have entered a room code on the 'Join Room Page', then they go to the 'Submit Feedback Page' where they can submit feedback. Both attendees and hosts can view information about the event room they are currently in, with the 'View Room Information' so long as they are either hosting or attending an event.

4.2 Activity Diagram

In Figure 3, an activity diagram displays the steps that a user can take using the feedback system. A user selects to log in or continue as a guest. If they chose to be a guest, then they will be an attendee. If they logged in, they can either create an event, making them a host or join an event, making them an attendee.

An attendee enters the room code of the event they wish to join. Once this is done, they can deliver feedback.

A host selects a template or opts to create a new one. If they choose to create a new template, they will input how often feedback should be taken, the different questions on the form, and a name for the event. Then, this template will be saved to a library of the user's templates. If the user selected a template, they will just be prompted to enter a name for the event. Once the event has been created, live feedback will be displayed to the host. If the 'Finish' button is pressed by the host, indicating that the event is finished, then a feedback summary over the whole event will be displayed. Otherwise, if attendees should supply feedback, then they will be prompted to do this.

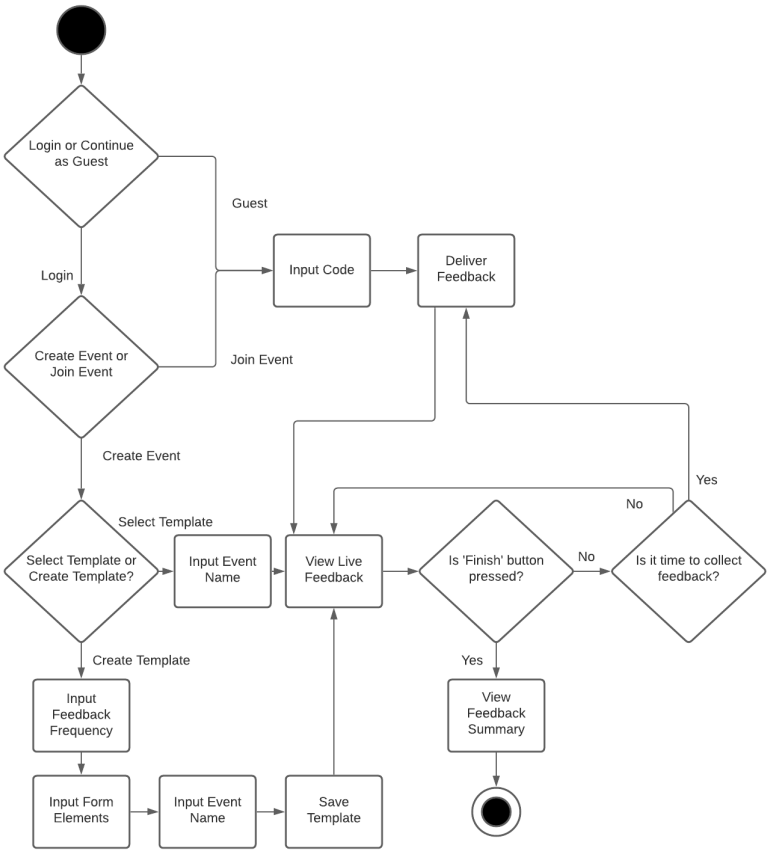

Figure 3: Activity Diagram showing the planned workflow of our system

4.3 Project Classes

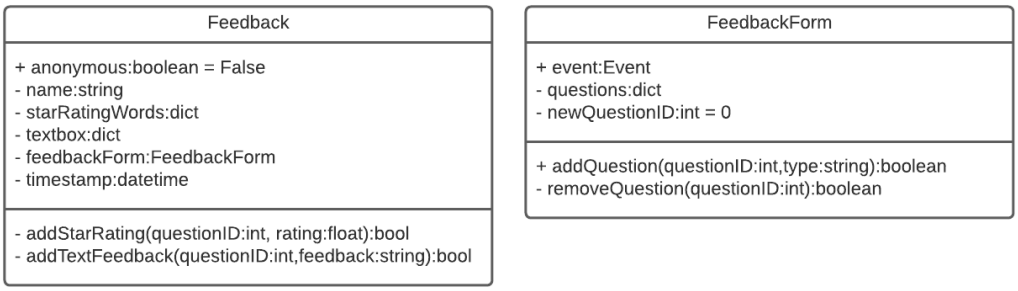

Figure 4: Class diagram representing the structure of the feedback class and feedback form

FeedbackForm
A class called 'FeedbackForm' stores the details of the feedback form that is filled out by attendees. A FeedbackForm object has a corresponding Event object. Furthermore, it stores a dictionary of all the questions. The key values for this are the question ID and the values are the type of question which can be either: 'starRating' or 'textBox'. The variable 'newQuestionID' indicates the next available question ID. It is initialised to zero. When 'addQuestion' is called, this value is incremented by one.

Feedback
A class called, 'Feedback' will store details of the feedback attendees give. Each question that exists on the feedback form is given an ID. A dictionary will be used to store the attendee's response to each question for star ratings and for input text, where the key is the question ID and the value is the feedback. A boolean value store whether the user has opted to be anonymous. If 'anonymous' is set to False, then the string 'name' will store the attendee's name. If not, this variable will be set to None. Furthermore, the variable 'timestamp' records the time at which the feedback was submitted so the host can view how the sentiment changed over time throughout the event using a datetime data type [2].

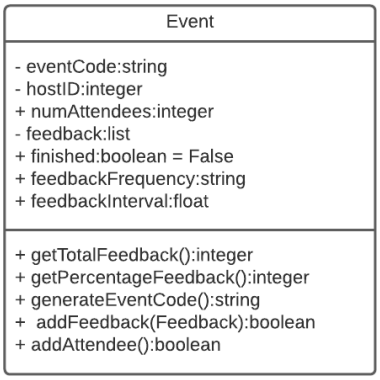

Figure 5: Class diagram representing the structure of the event class

Event
A class called, 'Event' will store details of an event that a host creates. Each event will have an associated 'hostID' which uniquely identifies the host. Furthermore, each event will have a variable 'eventCode' which is the string that attendees use to access the event. The code is generated using the function 'generateEventCode'. The number of attendees can be accessed using the variable, 'numAttendees'. A list of Feedback objects called 'feedback' stores all feedback that has been submitted for that event. The boolean variable 'finished' is False if the event is ongoing and True if the event

has finished. The variable, 'feedbackFrequency' determines how often attendees are prompted for feedback and will be either 'whenPrompted' or 'interval'. If this variable is set to 'interval', then the variable 'feedbackInterval' will be set to the number of seconds between how long it should wait before prompting attendee's for feedback again. If not, the variable 'feedbackInterval' will be set to None.

4.4 NLP method

NLP can be implemented in two ways: there's a lexicon-based method and the machine learning method [3]. The lexicon-based method relies on assigning a value to certain words depending on its sentiment.[3] The overall sentiment of the text would be the summation of all these values. In addition to the sentiment value generated, the local context of the word is also taken into account, which could either negate or intensify the result. The main advantage of this approach would be its simplicity compared to an ML implementation, however, its reliance of a predefined labelled dataset could potentially bottleneck the accuracy of such system.

4.5 Sentiment Lexicon

Each word from the lexicon is assigned a value from -100 to 100 (from negative to positive). In addition to the value, a probabilistic element is also taken into account, since some positive/negative words might have a different sentiment depending on the context (ie. "I'm happy that the presentation ended"). One potential way of calculating the sentimental probability would be to go through a dataset without the probability element enabled and count the number of times a supposedly "positive" word has had a negative connotation (and vice versa). The accuracy of such an approach can be increased by giving it a bigger dataset to analyse.

$$P(\text{ positive or negative } \mid w) = \frac{P(\text{ positive or negative } \bigcap w)}{P(w)} = \frac{\#w_P}{\#w} \tag{1}$$

"$\#w_P$ being the number that of positive/negative words in the sample, and # w being all the words in the sample."

As for negating the value of a lexicon, one potential approach would be to reverse the value of the word. However, real-life negation is not as clear cut since there are multiple levels of sentiment in between. For example, take the sentence "I don't hate my job". Semantically, the reverse of "hate" would be "love", which is not what the user intended to communicate. A solution to this would be inserting pairs/multiple words into the dictionary and assigning a predefined value to it.
Finally, combining all the sentence fragments into one single value will require the use of another universal formula. Unfortunately, we cannot simply sum all the sentiment values and output it to the user. For one, summing multiple positive/negative portions might result in a value higher than 100/-100 which would render our sentiment analysis program useless since there's nothing to compare it against. One simple solution could be taking the average of all the segments' values:

$$\frac{\sum_{i=0}^{n} X_i}{n} \tag{2}$$

We need to take into account, however, that the more segments we have, the more intense the overall sentiment should be. In other words, we should assign more/less value depending on how descriptive the user is. For example, "The presentation was informative and the presenter was good" should have a higher sentiment value compared to "The presentation was informative" since the user lists more good qualities. Our previous equation, however, would need to be amended.

4.6 Frontend

For our frontend web development, we will be using React JS [4]. One of the main advantages for using the React framework would be its ability to create virtual DOMs, [5] which will allow us to

only update the changed components when the user interacts with the page. Due to the number of interactive elements that will need to be executed, this virtual DOM will greatly improve the overall efficiency of the system. Furthermore, having a component-based framework will allow us to easily test out subsections of the code and will help in any future debugging/maintenance. Finally, using ReactJS will help with our native app development, since we could make use of the design patterns and create a more consistent UI.

Design-wise, modularity would be crucial for future development. To make it easier to update and debug, the frontend of the system will send the data to the database which will then be read by the backend. This will allow future developers to make changes to the frontend without also needing to modify the backend.

4.7 Data Storage

For data storage and recall, we will be using SQL. We chose this because SQL is an industry-standard and as a result has extensive documentation, tools and resources available. SQL is also capable of carrying out the data operations we will need quickly and efficiently while being more reliable than custom written alternatives.

We will be using SQLite [6], a lightweight open-source relational database management system. This system is capable of having multiple concurrent read and write processes without them interfering with each other [7], this is useful for us as our website needs to be able to write and retrieve data for multiple concurrent sessions.

4.8 Generating Event Code

Room codes need to be unique and of adequate length to prevent users from joining rooms they have not been invited to. To ensure the uniqueness of the room codes, they will be generated by the database system that is used to store them. In SQLite, it is possible to place a uniqueness constraint on the room codes ensuring this property. It is also possible to generate random numbers in a range this range will be appropriate to ensure that the room code had an adequate length.

4.9 Backend

The backend will be implemented using Python because of its readability and rapid development. Another reason we have chosen Python instead of a more efficient programming language like Java or C++ is the range of machine learning and natural language processing libraries at our disposal. We will specifically be using the "Natural Language Processing Toolkit" (NLTK) Python library. This library will aid us in tokenising our text (segmenting the text and removing unnecessary words) [8], analyze word frequency and create custom classifications for our words. Furthermore, NLTK already comes with a built-in sentiment analyser which could serve as a basis to build upon. This can be done using the "scikit-learn" classifier that comes with NLTK. This subpackage will allow us to train the dictionary by extracting and evaluating keywords that aren't inside the built-in analyser.

The output will be sent to the database and then retrieved in ReactJS.

4.10 Internal Architecture

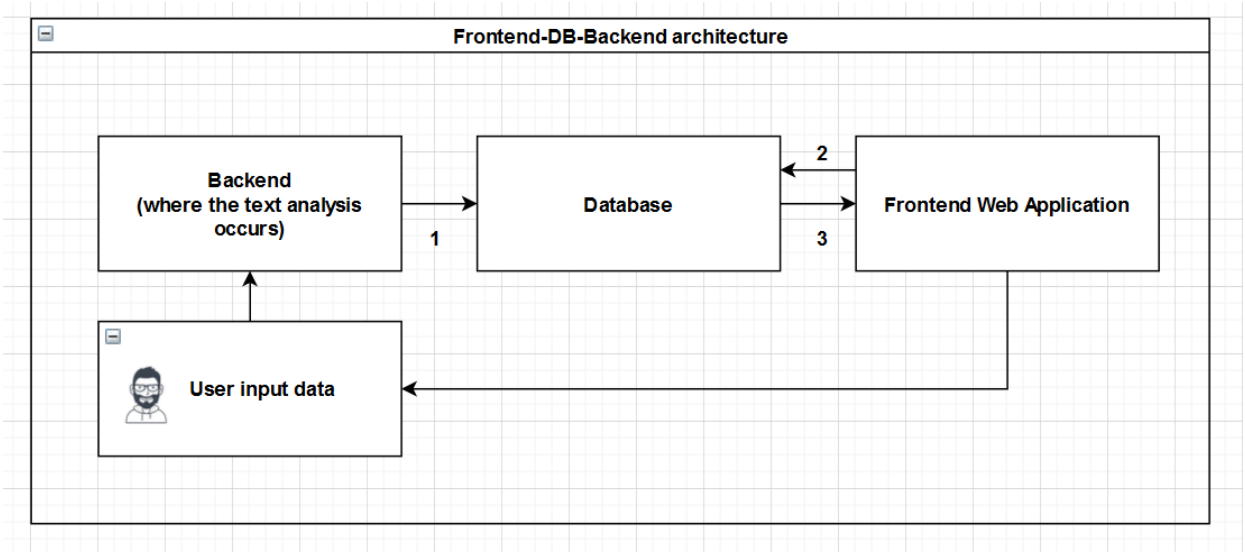

Figure 6:
1. The output of the sentiment analysis
2. User Account information, not the actual analysis text
3. Once stored in the DB, the sentiment analysis output is displayed to the user

5 User Interface Design

The flowchart overleaf in Figure 7 illustrates how the various menus in the system flow. This flowchart is not exhaustive - only the screens which are vital to the primary function of the system are shown. It is also important to note that the positioning and scale of the various elements on each screen will not necessarily dictate how they appear in the actual system - the flowchart simply shows which elements and content should feature on each screen and their approximate locations. Figure 7 is colour coded as follows:

- Blue outlined, blue filled box - Textbox
- Green outlined, green filled box - Button
- Solid red outlined, unfilled box - Area in which an error message should appear
- Large orange outlined, unfilled box - Pop-up window
- Purple outlined, unfilled box - Checkbox
- Red dashed outlined, unfilled box - Area in which content will appear when certain options have been selected

It is important that the host can view the feedback and how it changes over time, both during the event and after it. We concluded that pie charts were the best format to display the feedback to the host because it is clear to see at a glance how the attendees are responding to the event. Also, the host can see the average rating and mood submitted by the attendees - this simply gives the host an additional perspective to view the feedback in.

Once the event is over, the host can view individual answers to the questions asked, as well as the username of the attendee who submitted them - this is not available while the event is still active because it may be too distracting or off-putting to the host to see comments and ratings appear live. A checkbox was used to allow the user to submit feedback anonymously so that it is clear to an attendee that this is an option that they can select. The username of an attendee who wished to remain anonymous will appear to the host as ‘Anonymous’.

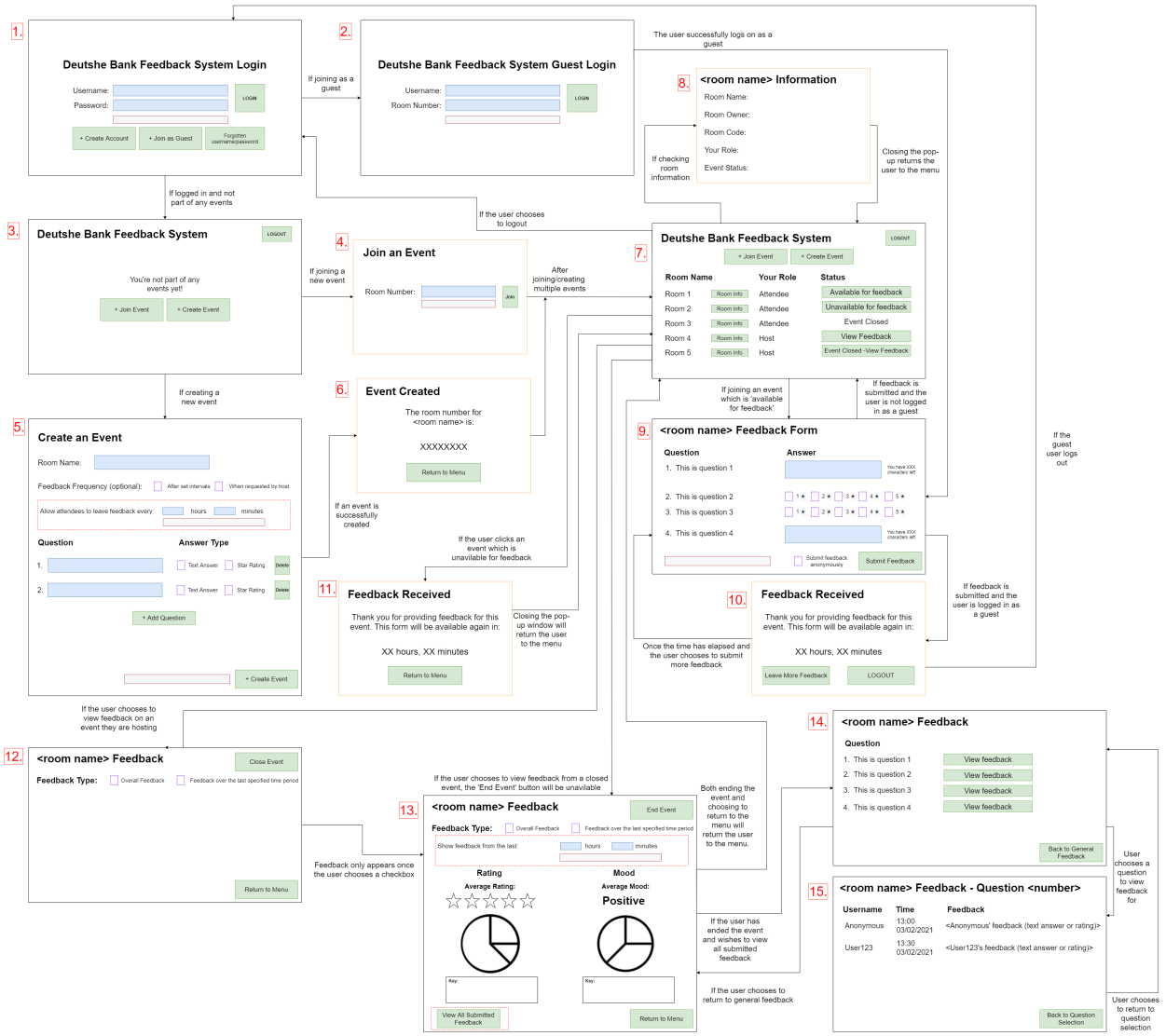

Figure 7: Flowchart of the user interface

6 Risks

The following table details the risks that would negatively impact the project schedule or the quality of the product being developed. Each risk has been summarised and a risk plan has been created to establish what to do should the risk occur. The risks are ordered such that the most significant risk is first and the least significant risk is last.

Risk	Summary	Risk Plan
Change to Requirements	As we develop the project, it may appear that one or more of our requirements are not suitable.	This is unlikely to occur as we do not expect the requirements to change throughout the project. However, if we decide that they do, we would decide as a team which requirements to change and when to complete these changed requirements.
Change to Design Plan	As we develop the project, it may become apparent that at least one aspect of the design is not suitable.	This would mean that we would have to alter our plans and have further discussion about the changes that should be made to the design.

Size Underestimation	It is possible that we did not allocate enough time to a certain part of the project as it is larger and more complex than we initially realised.	As a result, we would alter our plan to allocate more time to the given task and have others contribute to this aspect of the project so that we can complete it in a reasonable amount of time.
Team Member Unavailable	Due to illness, internet problems, or other factors, a team member may be unavailable for a certain time.	As a result, we would consider the tasks that this team member has been assigned, and how we will allocate these tasks amongst the rest of the team members for the time that the team member is unavailable.
Tool Under-performance	As the project goes on, it may become apparent that a tool we have selected is not appropriate for the task.	As a result, we would refer back to research undertaken to decide on another tool we would like to use if we discover that the chosen tool is not appropriate for the task.

Table 1: Table showing issues that may arise during the project, and the course of action required

7 Test Plans

Knowing that at the Deutsche Bank there is a strong culture of product testing, we decided to make a thorough testing plan. As it is an important part of the development by giving us the opportunity to check our progress more rigorously, comparing it from week to week, we designed a testing plan before starting any development. This will also ensure that we will produce a high-quality product. We divided the testing plan into three major parts: for the first section, we will test parts of the code as it is written, including syntax, execution and logical errors that are to be found. This will be done by comprehensive testing of the code with normal data, exceptional data as well as extreme data in order to find any syntax problems undetected during the planning phase. After all the parts of the code are written and tested by themselves, a more complex test will involve checking the systems final code, with all of the parts combined, reassuring the functionality of the product. All of the testings will be done with documentation of any problems that are to be found as well as possible predicted outputs, to compare our expectations from the requirements with how the system actually functions.

The second testing section will consist of checking if the code is efficient and maintainable. This will be done with dry runs, documenting results through trace tables. By stress testing the product, we can further understand and document its limits and find any inefficiencies in the code that, if fixed, could provide a smoother running time, further refining our system.

The last testing section will consist of checking if the software is fit for purpose. We will ask different testers to use our system through a web interface to check if our interface is user friendly and easy to use. We also want to ensure that users find our products secure enough, regarding any issues with personal information and privacy. [9]

We will test our Python code using Unit testing since it allows us to spot any logical errors in the program. For the ReactJS code, we will use 'Jest' to access the DOM [5] [10] and mock code modules.

In addition to the correctness, we will also ask test users to rate and highlight the weak features in our UX. This could lead to small changes to the interface or system workings (i.e. increase response time). Finally, we will do an integration test to make sure all the code functions well together.

Once the functionality part of the code has been tested, we can start measuring performance, which could be simulated by bringing the system under heavy loads. This stress testing could be performed

by having multiple "rooms" opened at the same time or by increasing the number of inputs in the web app. Once we find out the limit of our app, we have to make sure that if it does crash, the recovery is as seamless as possible.

Although out of scope, future maintenance or iterations of this system might include security testing.

References

[1] DOM. DOMs. https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction. Accessed 2 February 2021.

[2] w3schools. Python Dates. https://www.w3schools.com/python/python_datetime.asp. Accessed 3 February 2021.

[3] S Jurek, M Mulvenna, and Y Bi. Improved lexicon-based sentiment analysis for social media analytics. *Security Informatics*, 9, 2015. Accessed 2 February 2021.

[4] ReactJS. React. https://reactjs.org/. Accessed 3 February 2021.

[5] Virtual DOM and Internals. https://reactjs.org/docs/faq-internals.html. Accessed 1 February 2021.

[6] SQLite. SQLite home. https://www.sqlite.org/index.html. Accessed 3 February 2021.

[7] SQLite. SQLite locking. https://www.sqlite.org/lockingv3.html. Accessed 3 February 2021.

[8] M Mogyorosi. Sentiment Analysis: First Steps With Python's NLTK Library. https://realpython.com/python-nltk-sentiment-analysis. Accessed 1 February 2021.

[9] Robert M. Groves and Brian A. Harris-Kojetin. Legal and Computer Science Approaches to Privacy. https://www.ncbi.nlm.nih.gov/books/NBK475780/. Accessed 3 February 2021.

[10] Jest. Jest testing. https://jestjs.io/docs/en/tutorial-react. Accessed 2 February 2021.