

# Final Report

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

## 1 Abstract

The primary purpose of this report is to document the implementation and evaluate the feedback system prototype against the initial requirements provided to us by the customer. This report will provide an outline of our group organisation and will give an in-depth explanation of the methods driving our implementation of the various elements of our system prototype. The testing of our system will be described in detail. Finally, we will evaluate our system concerning both the customer's and our original requirements. From here we will suggest areas for the future development of the software.

## 2 Introduction

The customer, Deutsche Bank, hosts many large events which need to engage their audience and ensure critical information is communicated effectively. Typically, those who attend the events deliver feedback once the event is over. However, this means that the event cannot be adjusted for the current audience to address the issues raised by attendees. The adjustments can only be made for future events. Therefore, the ideal solution would be to allow attendees to deliver live feedback during the event, which can then be acted upon immediately.

It is also valuable to be able to process feedback for long term projects, allowing the organisers to access the effectiveness of their project, address specific issues and to see how any changes they make affect the feedback given by the participants.

To aid the strong feedback culture of the customer and in-line with the customer requirements, we have developed a prototype system that can be used to provide live event feedback as well as feedback for long term projects. Event hosts can view the feedback after it has been submitted. Our system also extracts a sentiment value from any text feedback given allowing the host to see the general sentiment of individuals submitting feedback.

## 3 Glossary

- **“Bag-of-words” Model: Bag of Words (BOW)** is a method to extract features from text documents. BOW is commonly used for machine learning, natural language processing, and information retrieval. In layman's terms, it's a way to collect words with no regard for the order of the words. BOW first removes any stopwords (words that don't have enough significance for our algorithm) from the sentence, as well as tokenising the sentence (breaking down a sentence into its elements). Finally, we vectorise the tokenised sentence.
- **MultiNomial Naive Bayes Classification** [1]: Naive Bayes is a learning algorithm known for its implementation simplicity as well as being very computationally efficient. Two event models are commonly used:
  - Multivariate Bernoulli Event Model
  - Multinomial Naive Bayes

For this project, we used the Multinomial Naive Bayes method. This method is based on the Bayes' theorem where the probability of elements in a dataset are mutually independent of each other (i.e. the order or usage of the word does not change the probability of the word being of a certain feature.)

- **NLTK**: Powerful Python package used to tokenise, analyse and understand written text.
- **Affordances**: UX design terminology used to actualize the knowledge and experience people already have to simplify the interaction flow.
- **Nielsen's Usability Principles**: Broad UX rules to help users navigate through a web page or application.

## 4 Group Organisation

Throughout our project, our requirements remained mostly unchanged. Any changes that were made to the requirements were discussed and agreed upon as a group. As a group the time in which we wished to do certain parts of the project altered from our original plan as some features, for example, the sentiment analysis took a shorter amount of time to complete than we expected, whereas other features, such as linking the login and registration pages to the database, took longer than we expected, and we had to adjust our schedule accordingly. Furthermore, we had to make some changes to our initial design as we worked on the project. To accommodate for the changes in the schedule and the design, our group held biweekly meetings to discuss the progress that had been made in the previous week, what we would like to achieve in the coming week, and any changes that were made to the initial design. When working on more challenging parts of the project, we worked in smaller groups of 2-3 people to complete these. We did this so that members of the group could spot any syntax or semantic errors in the code and also so that multiple people in the group have a better understanding of the system and can then develop it further.

Initially, we planned to use a Waterfall model to complete the project. Although we kept to our plan for the tasks to complete, changes were made when it came to the timings, and who worked on what areas of the project.

We came to realise that a plan driven approach was the most time efficient methodology as opposed to agile which would require more time and commitment. However we did have multiple branches on our Github containing incompatible features when it came to presenting the live feedback to the host of an event, since we had to choose between these two versions.

Our team worked well by having regular meetings, where we could discuss the progress we had made and raise any issues we were having when coding. Furthermore, it allowed us to compare our progress with our schedule and ensure that we would finish the project on time. Furthermore, we assigned roles for everyone in the team, so we would all worked on a certain area. This ensured that everyone could contribute and would know what part of the code they would develop in the time before the next group meeting.

Our primary form of version control was GitHub. For some members of the team this was the first time they had used a shared version control system. This lack of familiarity created some version inconsistencies that slowed the development process, we should have taken some more time to familiarise ourselves with this software and to discuss how we were going to use it as a team.

### 4.1 Progress Monitoring

The following Gantt chart displays the progress that was made throughout the course of the project.



Figure 1: Gantt Chart showing our Progress

Overall, we completed almost all of the tasks that had been planned, in time for the project deadline.

Two of the major tasks to complete for the project were completing the page for filling out the feedback forms for attendees to fill out and the live feedback page displayed to the host. To complete this, we split the team into two groups where three team members, Luca, David, and Jack, worked on displaying live feedback to the host and two team members, Rebecca and Maria, worked on displaying the feedback form to the attendee to be filled in. We decided to have more team members work on the live feedback as it was a more complex task.

As for the backend code, David was in charge of the sentiment analysis for which the system would pull data from the database, analyze it and would push the output back to the database. Jack worked on the database which served as a link between the frontend and backend. There was also maintenance done to the database if we noticed any redundancies or inconsistencies.

## 5 Implementation

After coding the HTML for each page in the feedback system, additional features were then implemented and the application was linked to the database using Python and Javascript. The implementation of all of the features was completed by separating tasks according to which page of the application they occur on.

### 5.1 Requirement Changes

One major change was made to our original requirements. Upon discussion during development, we concluded that instead of using alphabetic codes for the event codes, we would instead use numeric codes. Numeric codes were much easier to generate and process than alphabetic codes would have been, and we also found that the numeric codes were easier to recall. Therefore numeric codes would be better for hosts to distribute to event attendees since they are more memorable. Any other changes made were related to the priority of the requirements - these will be highlighted in the Evaluation.

### 5.2 Main Page

Initially, we created HTML displaying three buttons on the main page, allowing users to go to a page that will allow them to complete the following actions:

- Login with an email and a password
- Register an account
- Attend an event as a guest, without logging in

Each of the buttons is programmed to redirect to the corresponding page once a user presses them. We developed CSS [2], creating the formatting of the page, which can then be used on all the other pages of the feedback system so that they all have similar styling. The application's CSS is modular in order to have good design and it allows the page to be re-scaled for smaller devices.

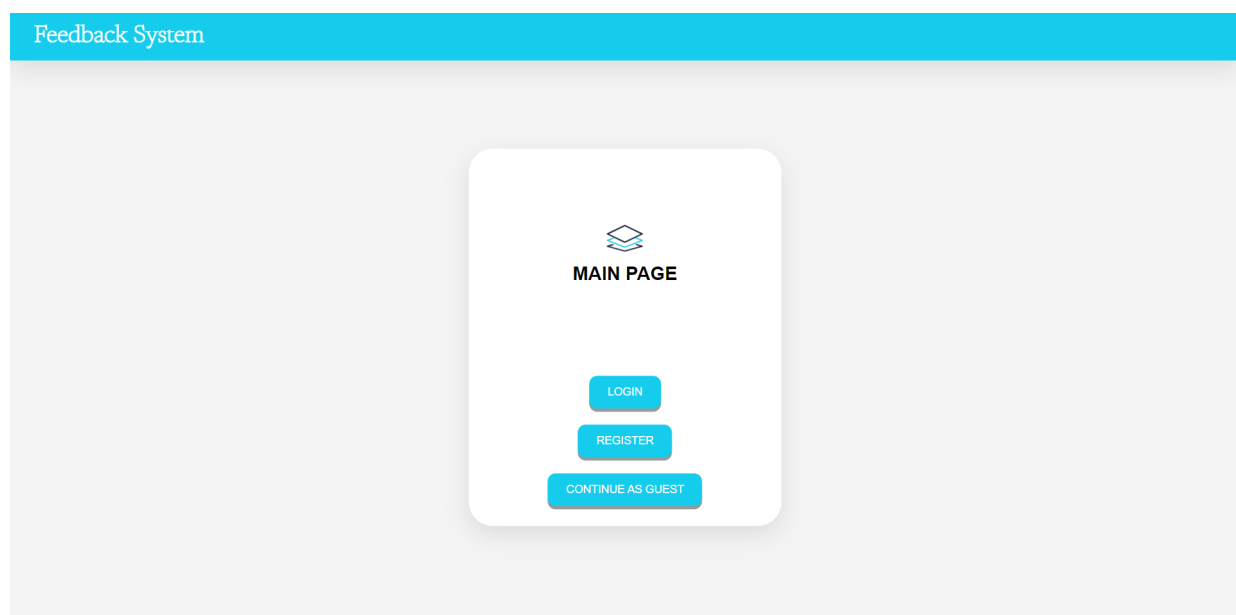


Figure 2: Main Page

### 5.3 Login

Initially, HTML for the login page was created so that a user can input their username and password. Then, pressing the Login button takes users to their profile.

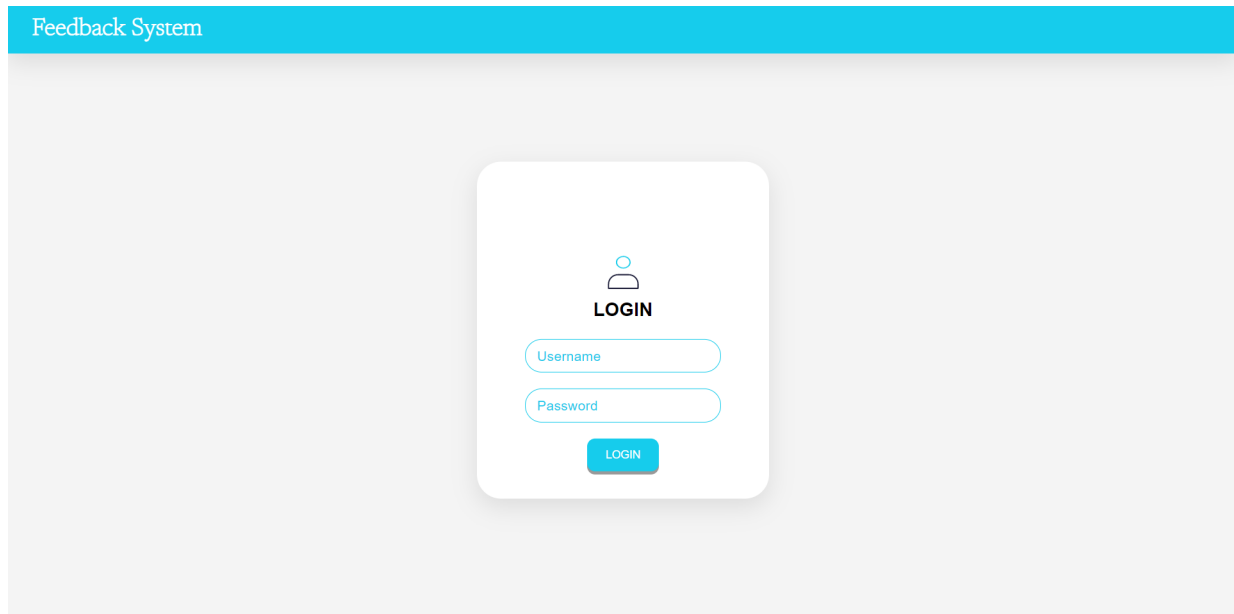


Figure 3: Login Page

Once a user presses the login button, the page should redirect the user to the corresponding profile, ensuring that:

- The user exists in the database
- The user and password combination match in the database

To verify all of these conditions were met, the program had to interact with the database once the login button was pressed. In our initial design, we planned to use ReactJS to achieve this. However, upon doing further research into ReactJS, we concluded that it was not appropriate for interacting with the database, as ReactJS is used for the frontend. Therefore, we did further research on what we could use for this and decided to use Python Flask [3] to be used along with SQLite3 [4]. Python Flask allowed the program to detect when a POST request [5] was made by the user when pressing ‘Login’ and SQLite3 then allowed the program to interact with the database, using SQL statements.

To create the Python Flask application, `app.py` [6] was created to have a login page and detect when a POST request was made. The form details are read [7], and then SQL is executed within the Python Flask application to interact with the database to check that the user exists and the password matches.

The `DBConnection` class stored in `dbConnection.py` is used to interact with the database. By calling the `confirmLogin` method within this class, this method returns True followed by the user ID if authentication is successful, and False followed by None otherwise.

The password is not stored in plaintext in the database but is instead the SHA256 hash of the user’s salt concatenated with the user’s password. Initially, the salt is retrieved from the database by using the following SQL statement:

```
SELECT salt FROM users WHERE email = ?;
```

The ‘?’ indicates the email that the user input on the login form. If a salt is not found for the corresponding user in the database, this means that the user does not exist in the database and the login is unsuccessful. However, if a salt is found, the user exists in the database and the salt is stored in a variable. The program then checks whether the email and password entered by the user correspond to what is stored in the database.

```
SELECT * FROM users WHERE email = ? AND password = ?;
```

The first ‘?’ indicates the email entered on the login form and the second ‘?’ indicates the concatenation of the user’s salt and the password entered on the login form that has been hashed using SHA256. If a result is returned from the database, this means that the password that the user entered on the login form was correct and they are successfully logged in. Therefore, they are redirected to their profile page. If not, this means that the email and password did not match, and the login is unsuccessful.

## 5.4 Registration

Once the HTML for the login page was completed, the HTML for the registration page was completed, allowing users to enter their: first name, last name, email, password, and a confirmation of their password to ensure that the user typed it correctly.

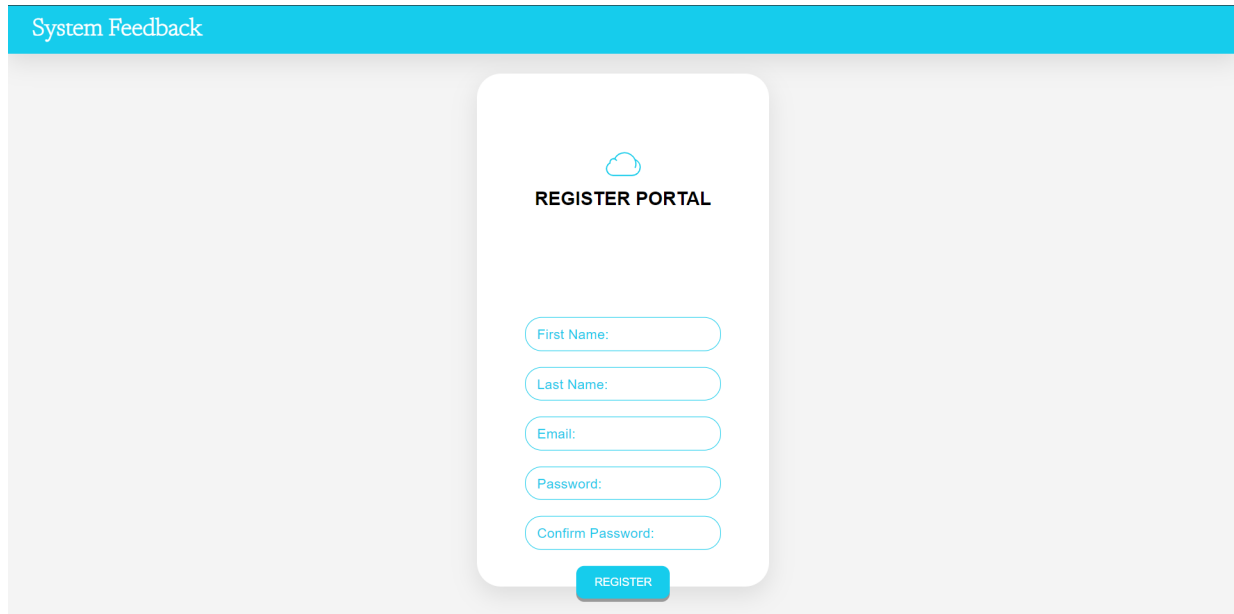
The image shows a web interface for a registration portal. At the top, there is a blue header bar with the text "System Feedback". Below this, the main content area has a light gray background. In the center, there is a white rounded rectangle containing the registration form. At the top of this rectangle is a blue cloud icon followed by the text "REGISTER PORTAL". Below this, there are five input fields, each with a blue border and a label: "First Name:", "Last Name:", "Email:", "Password:", and "Confirm Password:". At the bottom of the white rectangle is a blue button with the text "REGISTER" in white capital letters.

Figure 4: Registration Page

Once the user has input all of the information, the following conditions should be satisfied before the user is created:

- The email should not already exist in the database
- The ‘password’ and ‘confirm password’ input fields should match

Once a POST request is made, indicating that the user pressed the ‘Register’ button, the program first checks the ‘password’ and ‘confirm password’ input fields are the same. If they are, then the program calls the `addUser` method in the `DBConnection` class. First, SQL is used to determine whether the email already exists in the database using the following statement:

```
SELECT * FROM users WHERE email = ?;
```

The first ‘?’ will store the email entered on the input form. If any results are returned from executing the statement, then the email already exists in the database, and the function returns False indicating that the registration was unsuccessful.

However, if no results are returned from the SQL statement, then a salt [8] is generated by encoding 16 pseudorandom bytes into Base64. The salt is concatenated with the input password and then hashed using SHA256. Then, all these values are inserted using the following SQL statement:

```
INSERT INTO users (salt, password, firstName, lastName, email) VALUES (?, ?, ?, ?, ?);
```

where each ‘?’ represents: the generated salt, the hashed password, the first name, last name, and email respectively.

## 5.5 User's Profile Page

Each user has a unique menu upon logging in, which shows them the events that they are a part of - this includes events in which they are the host and events in which they are an attendee. For each event, the user can click the corresponding button which will take them to the relevant page. If they are the host of the clicked event, they are redirected to view live feedback for their event. If they are an attendee of the event, they are redirected to the feedback form for the event. Furthermore, a user can opt to create an event or join an event from this page, which will redirect them to the corresponding page.

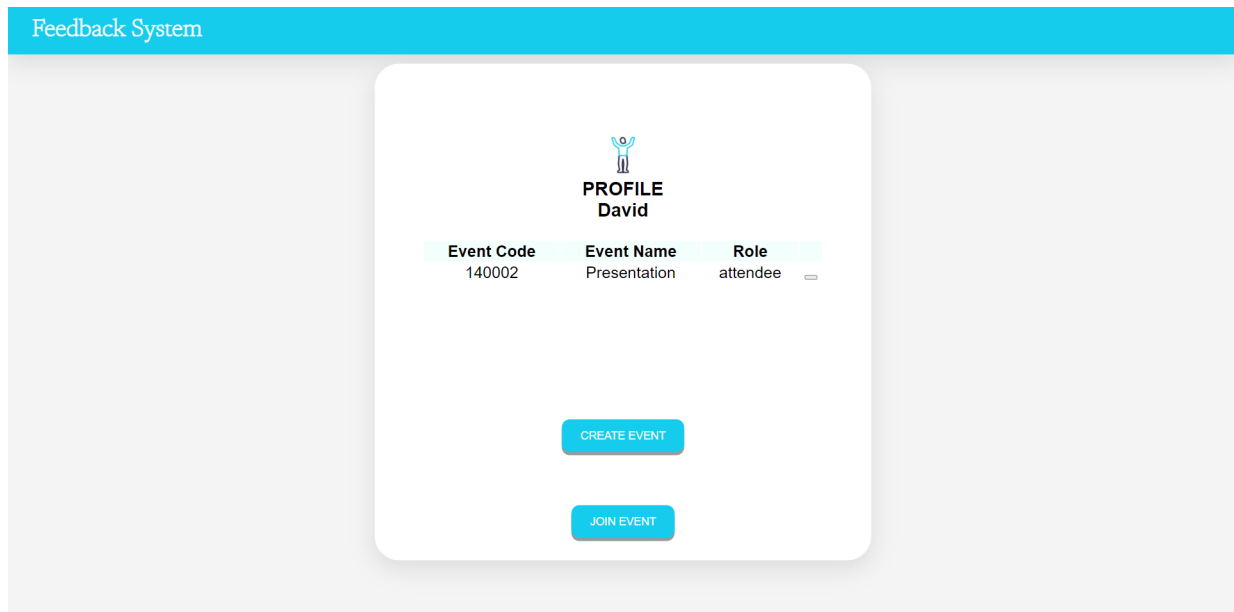


Figure 5: Profile Page

In order to return the events relevant to a user, the `getUserEvents` method in the `DBConnection` class is called. The user ID of the user who is currently logged in is passed as a parameter to the function. Inside the function, the following SQL statement is run to return events being hosted by the corresponding user ID:

```
SELECT roomcode, eventName FROM events WHERE hostUserID = ?;
```

The '?' represents the user's user ID. The results from this are then appended to a variable, `hostRows`.

Once this is completed, the following SQL statement is run to return active events that the user with the corresponding user ID is attending:

```
SELECT events.roomcode, events.eventName FROM events INNER JOIN event_members ON  
→ events.roomcode = event_members.roomcode WHERE event_members.userID = ? AND  
→ events.active = 1;
```

The '?' represents the user's user ID. The results from this are then appended to a variable `attendeeRows`. The function then returns the variables `hostRows` and `attendeeRows`, detailing events that should be displayed to the user on their profile page.

The Python string is converted into a JSON string [9] and then parsed in Javascript to be stored in a Javascript array. Once the events are stored in the Javascript array, the `createTable` function in `viewEvents.js`. This function will find the table with ID `thisTable` and create the table containing the: event code, event name, and role (either attendee or host) of the user. Also, there is a corresponding button for each event in the table that the user can press that redirects them to the corresponding event.

## 5.6 Creating Events

On the create event page, a user will be prompted to input an event name, select from a list of templates, and select a feedback frequency that they wish to use for their event, the options being: when prompted by the host, when the attendee chooses to, or at set intervals. If the feedback frequency selected is 'At Set Intervals', they

will be prompted to input the number of hours and minutes in an interval. This is done by using Javascript to display a section for inputting these values only when the, ‘At Set Intervals’ option is selected [10]. In order to meet the deadlines of the project, although the feedback frequency can be input, it is not implemented, and all feedback forms will act as if the feedback frequency is set to when an attendee chooses to deliver feedback.

Figure 6: Create Event Page

Once a `POST` request is made, when the user presses the ‘Submit’ button, this is detected in `app.py`. If the chosen template was ‘Create’, indicating that they wish to create a new template, then the user is redirected to the template creation page. Otherwise, the event is created by calling the `createEvent` method in the `DBConnection` class. This function will first generate a pseudorandom room code between 100000 and 999999, using the Python random library [11]. It then checks that the room code is not already in the database, using the following SQL:

```
SELECT * FROM events WHERE roomcode = ?;
```

where the first ‘?’ indicates the room code that has been generated. If any results are returned from the SQL function, then the room code already exists, so it is not valid. While the room code is not valid, the program will repeatedly try new codes until a valid one is found. The probability that a room code must be generated again because of a collision is very low due to the large range of numbers. Once a valid room code is generated, the event is added to the database with this room code using the following SQL statement:

```
INSERT INTO events (roomcode, eventName, feedbackFrequency, hostUserID, date, active,
↪ feedbackFormID) VALUES (?, ?, ?, ?, ?, ?, ?);
```

Once this function has successfully executed, the user is redirected to the live feedback page to view the room code and live feedback for the event that they have created.

## 5.7 Creating Templates

Before the user can create an event, they will have to choose between creating a template or using a pre-existing one. If the host chooses to create their template, they get redirected to the create template page. They are then prompted to name their template and will have the option to add questions to the template. With the “Add question” button, a new entry will appear for which the host can create the entry’s prompt and assign the type of question that it will be. A drop-down menu will appear and the host is given two options: “Text” and “Star rating”.

Figure 7: Create Template Page

In order to add or remove questions from the template feedback form using the buttons, two functions: `addRow` and `removeRow` are called in the `createTemplate.js` file [12]. The `addRow` function will insert a new row at the bottom of the table using Javascript. The `removeRow` function will use a for loop to iterate through each row. Any row that has been checked with the check box on the left-hand side will be removed, so long as this leaves at least one question in the table. If a user attempts to remove all questions from the template, this will not be allowed, displaying an alert to the user warning them that they cannot delete all rows.

Once a POST request is detected in `app.py`, indicating that the user pressed the ‘Create Form’ button, the program checks that each of the questions on the feedback template are assigned a prompt and a question type. If any are not, the feedback form will not be created and the user is redirected to the ‘Create Template’ page again, ensuring that an incomplete template cannot be created. Otherwise, the template feedback form is complete, and is added to the database by calling the `addTemplate` method in the `DBConnection` class. This method will insert the feedback form into the database using the following SQL:

```
INSERT INTO FeedbackForm(templateName, overallSentiment) VALUES (?,?);
```

Furthermore, each question on the feedback form is added to the database by executing the following SQL for each question:

```
INSERT INTO Question(questionNumber, type, content, feedbackFormID) VALUES (?, ?, ?, ?);
```

Once this function has completed and returned the feedback form ID, the event will then be created using this feedback form ID. This is done by calling the `createEvent` method in the `DBConnection` class, as described in section 5.6, on the Create Event page. Once the event has been created, the user is redirected to the live feedback for the given event, showing live feedback and the event room code.

## 5.8 Providing Feedback

Providing feedback can be done in either text form or the number of stars. The question type will depend on what the host chose in the create events page. If the host chose a question with a question type of star rating, a drop-down list will be displayed to the attendee, prompting them to select an option from 1 to 5 stars [13]. If the host chose a question type that required text, a text box is displayed, prompting the attendee to type a response.



Figure 8: Provide Feedback Page

Submitting the information triggers a **POST** request to the Flask backend. This first checks whether the attendee has filled out all of the questions on the feedback form. If they have not filled out every question, then nothing is added to the database, and an alert is shown to the user, using Javascript, stating: ‘Please complete all questions on the feedback form’. However, if all of the questions on the feedback form are answered, then the **addFeedback** method of the **DBConnection** class is called. If the user is attending as a guest (i.e. not logged in) or has opted to be anonymous, their user ID will be set to **NULL** in the database. This is done by executing the following SQL:

```
INSERT INTO feedback (anonymous, timestamp, feedbackFormID, roomcode, sentiment) VALUES
↳ (?, ?, ?, ?, ?);
```

However, if the user is logged in and has chosen to not be anonymous, the following SQL will be executed:

```
INSERT INTO feedback (userID, anonymous, timestamp, feedbackFormID, roomcode, sentiment)
↳ VALUES (?, ?, ?, ?, ?, ?);
```

Once all of the correct information is inserted into the **feedback** table, the answer to each individual question is then inserted into the **feedbackQuestions** table. If this is the user’s first feedback form they are filling out for the event, the following SQL is executed:

```
INSERT INTO feedbackQuestions (questionID, feedbackID, answer) VALUES (?, ?, ?);
```

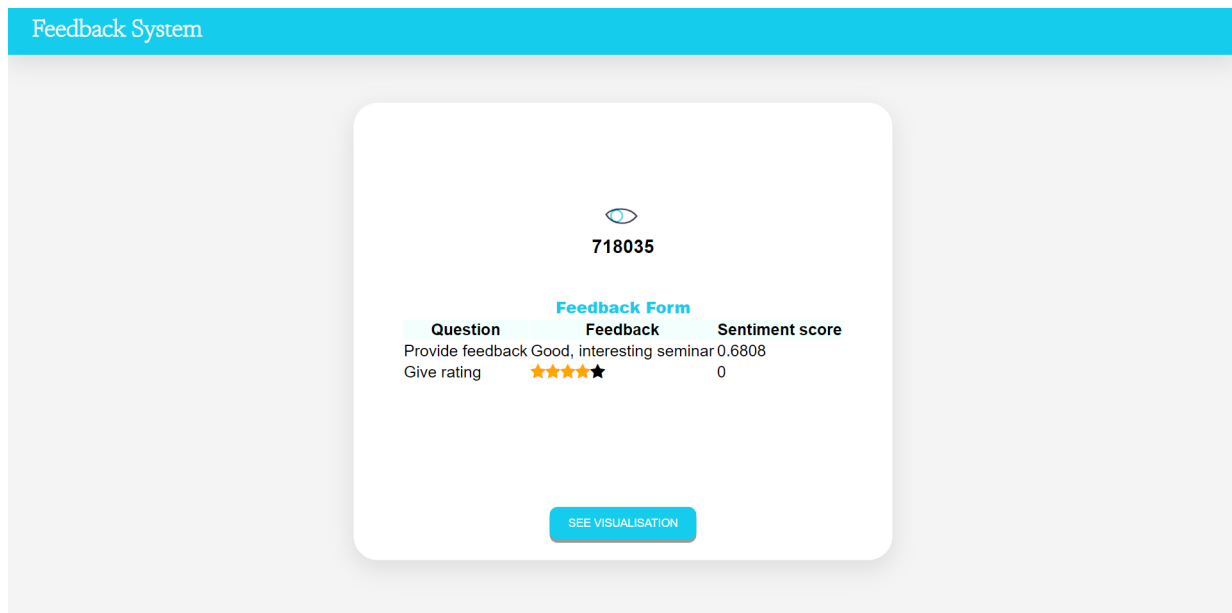
Otherwise, if the user has previously submitted a feedback form, their feedback will be updated using the following SQL:

```
UPDATE feedbackQuestions SET answer = ? WHERE feedbackID = ? AND questionID = ?;
```

Now that the database has been updated with all of the relevant information for the submitted feedback form, Javascript is used to display an alert to the user indicated that their response has been submitted successfully.

## 5.9 Viewing Feedback

Feedback is presented to a host in two ways on two separate web pages. When a user chooses to view an event that they are hosting they will see a list of all the feedback received so far. In the case of star ratings, they will see the number of stars given by the user, for text feedback the submitted answer will be displayed in text form. This viewing method allows the user to see individual feedback giving them the ability to assess specific strengths and weaknesses in their approach.



The Live Feedback Page features a central white card on a light gray background. At the top of the card is an eye icon and the ID number 718035. Below this is a section titled "Feedback Form" containing a table with three columns: Question, Feedback, and Sentiment score. The table has two rows of data. At the bottom of the card is a blue button labeled "SEE VISUALISATION".

Question	Feedback	Sentiment score
Provide feedback	Good, interesting seminar	0.6808
Give rating	★★★★★	0

Figure 9: Live Feedback Page

However, it is also useful for the users to be able to see an overview of the feedback they have received so they can easily assess the overall success of their project or presentation. The web page has a singular button labelled “See visualisation” which redirects the user to another web page where their feedback is displayed visually. The web page displays two pie charts which represent the average mood and the average rating of the feedback, these pie charts were made using Google Charts [14]. The average mood representing the text feedback and the average rating the star rating feedback. The average mood is determined by the semantic analysis of the feedback text.



Figure 10: Visualisation

On top of sharing the information from attendees to the host, we also want to analyse the content itself and provide the user with a quantifiable number for the sentiment of the message. For this, we call on our sentiment analysis object, (which is discussed in further detail below) and get a dictionary containing the percentage of positive, negative and neutral words. In addition a compound score is given to the whole message. During our development, we changed the way this score was showcased to the user. At first, we only returned the compounded score, (i.e. for the sentence “I’m sad”, the sentiment analysis machine would only return -0.4767), however, this would cause problems when visualising the data, as it was difficult to showcase -0.4 in a pie

chart. Instead, we decided to have the pie chart represent the percentage of words, so for “I’m sad”, we’d have [compound: -0.4767, neg: 0.756, neu: 0.244, pos: 0]. This was much easier to represent since it didn’t have negative values to account for.

To make this general feedback more useful to the user there is an option to only view feedback from a certain period. This means that if the user made some sort of change to their presentation or project and wanted to see how this had affected their feedback they can choose to only view feedback submitted since this change was implemented.

The user is given the option to do this by selecting a time in hours and minutes in an HTML form. The pie charts will then be remade only containing feedback submitted within the given time frame. The visualisation of feedback shows one of the key advantages of semantic analysis, it allows for the visual representation of a large amount of non-numerical data. Without the semantic analysis, it would be very hard for the users to gauge general trends from a large amount of feedback.

## 5.10 Database

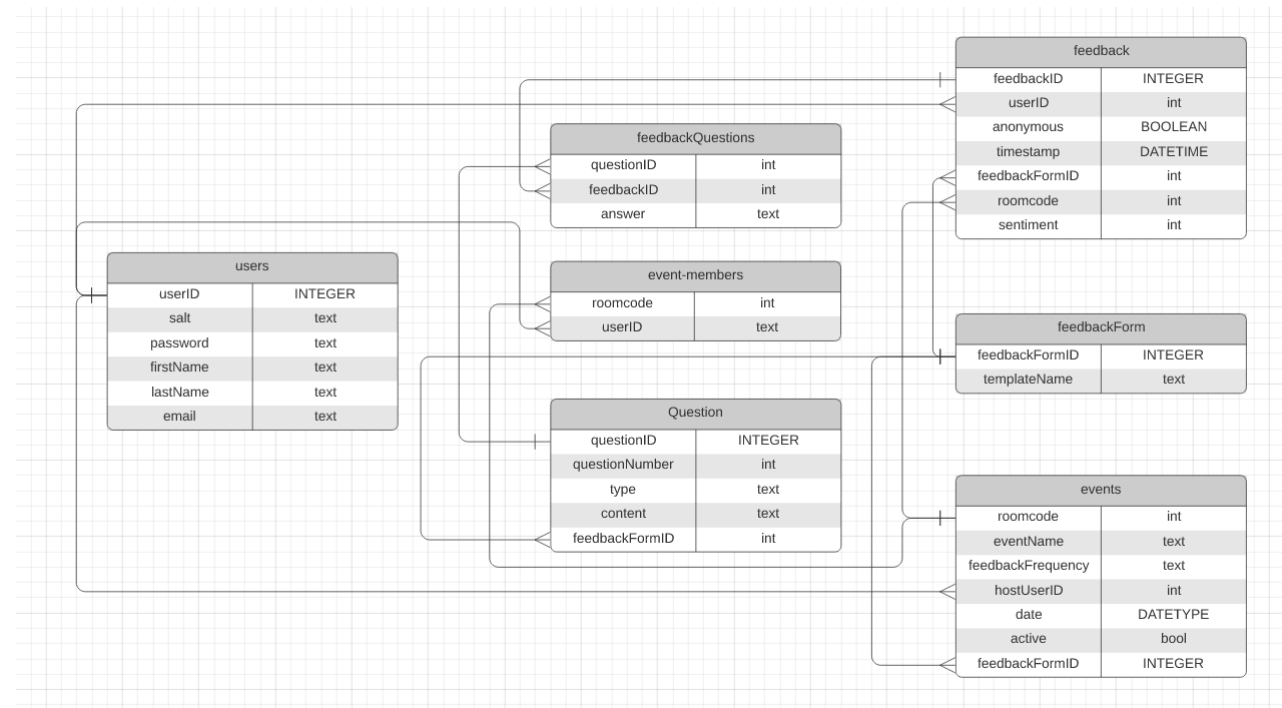


Figure 11: Crows Foot Notation Diagram of Database Structure

### Users Table

In the **users** table in the database, each user is given a user ID to uniquely identify them. Furthermore, an email is used to authenticate, as well as two fields, **password** and **salt**. The salt will be a pseudorandom selection of 16 bytes, generated using the Python `os.urandom()` method [15]. It is then encoded in Base64 using the `base64` Python library [16]. Then the password field will store the salt concatenated with the plaintext password, hashed using SHA256 using the Python `hashlib` library [17]. Doing this improves the security of the application because storing user passwords in plaintext increases the likeliness of user’s passwords being compromised.

### Feedback Table

When a user submits feedback this is stored in the **feedback** table. The user has the option to submit anonymously and this choice is stored. A timestamp is also made when the feedback is submitted, this can be used to compare the time of submission to that of other user feedback, this allows the system to display feedback from a specific period to a user.

### Question Table

When a user creates a feedback form it may contain many questions, which are stored in the **questions** table. Each question is identified by a unique integer and has a question number, a type and some text content which is

the question input by the user. Each question is also assigned to the feedback from the user is currently creating.

### Event Members Table

This table stores pairs of userIDs and room codes. The table is used to find which users are assigned to a particular room and which rooms a user is assigned to.

### Feedback Questions Table

The answers to individual questions on the feedback form are stored in the `feedbackQuestions` table. There are two formats for the question, star rating and text feedback, these two feedback types can both be stored in the same table. The star ratings are stored as the string form of their numerical value. It is possible to later distinguish the star feedback questions from the text-based ones as the `questionID` of the entry refers to a question in the Questions table which has its type recorded.

### Feedback Form Table

The system allows templates for feedback forms to be used multiple times, so when creating feedback from the users has the option to name the template they have created allowing for the template to be more easily identified and reused. This table stores the names of the created templates.

### Events Table

When a user creates a new event they give it a name, a feedback frequency and a feedback form template. These properties are all stored in the events table. The `events` table also stores the date of creation of an event, the host's userID and a boolean value to show if the event is still active or not. The room is also assigned a room code, used to access for access to the room, that is stored in the table.

## 5.11 Creating the Database

Running the file, `createDB.py`, creates an empty database file with the tables previously described. This Python file will utilise the SQLite3 library to create all of the tables with the corresponding fields and store this database in a file called, `database.db`. Initially, no data will be stored in the database. However, by interacting with the feedback system, the database will be updated using SQLite3.

## 5.12 Sentiment Analysis

As mentioned in the “Planning and Design document”, we have decided on using a lexicon-based method for sentiment analysis [18]. The lexicon-based method relies on assigning a value to certain words depending on their sentiment. The NLTK library was crucial for our project since it provided us with a foundation to build upon. This Python library allowed us to get a head start in development due to its finely tuned pretrained model. To improve its accuracy, however, we performed Multi-Nomial Naive Bayes Classification using scikit-learn and the dataset “Twitter US Airline Sentiment” [19] which used Twitter data scraped from February 2015. A group of contributors were asked to classify positive, negative and neutral tweets, including explaining why they chose those classifications. A drawback of using this dataset was the lack of sentiment labels for the tweets. Given that there are only three possible options to choose from, the classifications for the aforementioned tweets weren't very precise. Besides, the reasons given for the classification were sometimes vague and unhelpful. Some users ranked the tweets out of a “gut feeling” and would not provide a quantifiable reason. Two examples of this are shown in Figure 12 below.

5.7E+17 negative	1 Bad Flight	0.7033 Virgin America	jnardino	0 @VirginAmerica it's r #####	Pacific Time (US & Canada)
5.7E+17 negative	1 Can't Tell	1 Virgin America	jnardino	0 @VirginAmerica and i #####	Pacific Time (US & Canada)

Figure 12: Two examples of improperly justified classifications

Due to a large number of entries, we decided to go forward with this dataset and train our model using it. Now that we had our set of texts and their respective labels, we had to convert the information into numbers. We used the “Bag-of-words” model to extract and quantify the features from the text, which resulted in a matrix that counted the number of words in a given document. Code-wise, this meant importing the sklearn package and using the `feature_extraction` feature to vectorise and tokenise the text. We'd then have to specify the target (i.e. which word we want to vectorise) and the `test_set` size, (a flag to randomise records could also be used).

Finally, we can create our text classification model using the built-in `MultinomialNB()` function where we'd pass the previously mentioned data into a multinomial naive bayes classifier object. We then used the `predict` method to test our model for accuracy.

Unfortunately, the classification rate using our dataset and our code was only 32% which was considerably lower than the prebuilt score (around 78-80%). This score could have been influenced by the previously mentioned vague answers and lack of classifying options. On top of that, the tweets were ranked on the sentiment for the airline, not the overall sentiment, so tweets like “I enjoyed the flight but my vacation was horrible” would be considered to be a positive tweet about the airline, but in reality, the sentiment of the tweet would either be neutral or negative.

We concluded that it would be best to use the prebuilt NLTK sentiment machine, whose accuracy far surpassed ours.

---

**Algorithm 1:** Model building for language processing

---

**Result:** Evaluation of model

```

Import packages (pandas, NLTK, SKLearn);
data = Parse dataset;
wordMatrix = [];
/* This matrix will keep track of the number of repetitions for each unique word in
   the given dataset */
for row in data do
    Tokenise row to remove unwanted signals;
    Add to document-term matrix when encountering a word;
end
readjustedMatrix, target = train_test_split( matrix.size, data["Sentiment"], test_size=0.3,
    random_state=1);
/* the "target" comes from the sentiment given to the tweet by the users */
model = MultinomialNB(readjustedMatrix);
predicted= model.predict(target);

```

---



---

**Algorithm 2:** NLTK prebuilt sentiment analysis

---

**Result:**

```

Import NLTK;
Import SentimentIntensityAnalyzer from NLTK;
obj = SentimentIntensityAnalyzer();
db = Database;
scores= [];
answers= Get all the entries from db for specific user and roomcode
for answer in answers do
    score = obj.polarity_scores(answer);
    compoundedScore = score["compound"];
    scores.append(compoundedScore);
    ...
end
return scores;

```

---

### 5.13 User Interface

In this section, we will discuss how we made an effort to satisfy good UI principles to make the user experience simple and aesthetically pleasing. One of the main techniques we followed was “Gestalt Laws of Perceptual Organisation”. The idea behind this was that individual things in an environment would appear whole to the user, as the mind would fill in the gaps for any missing information.

For instance, we used the law of proximity to divide our page into subsections. When creating our template, the page can be perceived as being split into three parts: the prompt where the host would input the template name, the main section where the user would add/remove questions from the template and then create a form at the bottom. This layout allows the user to easily navigate through our page. Furthermore, the buttons on all the pages are placed in a way so that the law of continuity applies. All the submit, create, and join buttons are the same colour and shape, as well as being in the same place. This means that the user’s brain will subconsciously look at similar-sized and placed buttons when they want to submit information.

With affordances in mind, we designed our webpage so the user would be familiar with the interactive elements. One example of an explicit affordance would again be our buttons since the user (theoretically) should have had experience with them. An example of a more implicit type of affordance would be the dropdown menus when selecting templates/question type.

A conscious effort was made to also follow all of Nielsen's principles, but due to time constraints, we were not able to fully realise all of them. One such example is the "Visibility of system status" principle that states users should be kept informed of the system status by providing appropriate feedback to determine the next steps. Our system would return errors on the command line so that we could properly debug any issues, but we weren't able to display them to the user via HTML. This could greatly worsen the user's experience as they might be stuck for quite some time figuring out what the issue was.

We did, however, try making our site as error-prone as possible, both by limiting unnecessary information as well as sanitising any of the input prompts (ie. allowing unusual characters).

## 6 Testing and Validation

Our testing method consisted of a static and dynamic part, for which we used different tools to meet different requirements. For our static testing, we used "pylint" to detect common bugs. In addition to this, we used PyCharm to make sure we followed Python guidelines to improve readability for future development. As for the more complex logic mistakes, team members would take turn reviewing the written code.

The dynamic structural testing was primarily done using the package "pytest", for which we would assert certain conditions and expect a boolean result. To have a large dataset to test with, we used a random data generator [20] to create a CSV file containing some valid and invalid users. The app\_test file would then go through the entries checking for issues in the database. As for the functional testing, we ignored the code and only cared about the results of the system. For this kind of testing, we tried replicating how a user would go about on the website as well as trying to break it using edge cases. We would then assess the results and evaluate which requirements the feature met.

To test the HTML is valid, the W3 Markup Validation Service [21] was used, to ensure the HTML of the page to ensure the HTML is valid and is of high quality. This makes sure our website will look the same across all platforms and eases any future maintenance.

To view the contents of the database, we used, SQLite Viewer [22], a tool for viewing the contents of an SQLite database. This allowed us to test out the insertion and deletion of elements easily and aided us in seeing the link between tables more clearly.

No.	Test Purpose	Expected Outcome	Result
1	Once an event room is created, it can be joined by an attendee by typing in the room code	Once an event code that exists in the database has been entered by an attendee, the attendee is added to the attendees for that event in the database and the feedback form for the corresponding event is displayed to the attendee	The expected outcome is achieved, the attendee is taken to the correct feedback form page and they are added to the database table, 'event_members' for the corresponding event they joined.
2	An attendee can opt to be anonymous by selecting an option on the feedback form	If an attendee opts to be anonymous on the feedback form, their user ID will not be stored in the database with their feedback. Otherwise, the corresponding user ID will be stored in the database when the attendee submits their feedback	The expected outcome is achieved. If the user opts to remain anonymous, 'null' is stored in the 'userID' field in the 'feedback' table. Otherwise, the corresponding user ID is stored here.

3	If a host chooses to have a star rating question on their feedback form, the attendee can answer this question, rating from 1 to 5.	The user should be able to select how many stars they would like to give for this star rating question and the database should be updated with this feedback once a feedback form is submitted.	The expected outcome is achieved as a user can select a star rating from 1 to 5 and the number of stars they selected is added to the 'answer' field in the 'feedbackQuestions' table in the database.
4	If a host chooses to have a text box question on their feedback form, the attendee can answer this question by typing feedback.	The user should be able to type in their feedback for the question.	The expected outcome is achieved as a user can type in their feedback and the feedback is added to the 'answer' field in the 'feedbackQuestions' table in the database.

Table 1: System Testing of Attendee Requirements

No.	Test Purpose	Expected Outcome	Result
1	A host should be taken to a create event screen upon pressing the 'Create Event' button	The user is redirected to the 'Create Event' screen where they are prompted to enter information about the event they are creating	The expected outcome is achieved as the user is successfully redirected to the 'Create Event' page
2	If the host chooses to create a template feedback form, they are taken to a screen where they can create one	Upon selecting the option to create a template, they are taken to a screen where they can: add questions or remove questions. Each question must be assigned a question name and a question type	The expected outcome is achieved, users can add and remove questions from the template and then it can be created and added to the database.
3	Confirm that a host is assigned an event code upon creating an event and that attendee feedback is displayed	The event room code that is added to the database should also be displayed on the screen when a host creates an event. If an attendee submits a feedback form, their feedback should be displayed to the host	The expected outcome is achieved. Initially, the room code is displayed on the screen, which matches the one that is added to the database file. When an attendee joins the room and delivers feedback, this feedback is correctly displayed on the live feedback page
4	Once a template has been created, a host should be able to reuse that template	Once a template has been created, when viewing the options for templates to use for an event, the template that has been created should be an option as a template. If this template is selected, the feedback form used for this event should match the template that has been chosen	The expected outcome is achieved. When a host creates a template, attempting to create an event will display this template as a template option for the host to select. If this template is chosen, the event is created using the correct template as a feedback form
5	For an attendee to join the event that they want, all room codes must be unique. Therefore, creating a new event should use a room code that is not already in use	When an event is created, the room code it is given should always be unique in the database	Because the room code is the primary key of the events table, meaning that it must be unique, and because the unique event room codes can be seen when creating multiple events, the expected outcome is achieved

6	On the live feedback page showing the overall trend of feedback, the host should be able to view feedback from a given time	When the host enters the past number of hours and minutes they want to view feedback from and press the 'Submit' button, only responses from this time frame should be displayed to the user	When entering a set number of hours and minutes to show feedback from, only feedback from this last period is included in the pie chart, which can be confirmed by viewing the timestamp of the feedback in the database
---	---	--	--

Table 2: System Testing of Host Requirements

No.	Test Purpose	Expected Outcome	Result
1	An attendee's feedback must be submitted to the database and viewable to the host in less than five seconds.	The submitted feedback should be in the database-less than five seconds after it is submitted.	The expected outcome is achieved. Once feedback is submitted to the database, the data can be pulled onto the host's 'live feedback' screen, all in less than five seconds.
2	The software must function the same, and the webpage elements must display the same or very similarly when opened in some of the most commonly used web browsers.	The software must function and view correctly on some of the most commonly used browsers.	The expected outcome is achieved. The software runs and displays correctly in the Chrome, Firefox, and Edge browsers, and also displays correctly when the browsers are put in their mobile versions.
3	The elements on the webpage must resize and move appropriately when the browser window's size is changed.	The webpage elements must resize appropriately when the web browser window's size is changed, and the webpages must still function the same.	The expected outcome has been met. The webpage elements resize appropriately when the window size is changed, and the functionality of the webpages does not change.

Table 3: Non-functional Testing

## 7 Evaluation

Now that both our implementation and testing processes are complete, we can reflect on choices made throughout the project, and evaluate our system prototype against the initial requirements.

### 7.1 Project Appraisal

Despite this project having to be done remotely, communication was not an issue - all team members regularly posted and explained code updates to keep everyone else up to date with any recent bugs, fixes, or added features. Splitting team members into two groups was also an effective approach - we found that both groups' work integrated well when required and that neither group was too heavily reliant on the other. This meant that our time was effectively used throughout development.

Although we deviated from our design plan by using Python Flask instead of ReactJS, we were still able to meet our aims in terms of development. This change was a good one, and we found that Flask was a more appropriate and easier to use option. Had we been able to spend more time researching implementation methods while designing the system, we could have avoided wasting development time on trying to make ReactJS work, however, given the deadlines in place, this was not possible.



## 7.2 Customer Satisfaction

Next we must evaluate our final prototype against our initial requirements, and therefore also evaluate the customer's satisfaction. We can see to what extent the initial requirements have been met by referring to the Testing section. The requirement numbers in Tables 4 to 6 correspond to the requirements numbers in the Requirements Analysis document [23].

Req. No.	Degree of Success
1	This requirement has been fully met. A host can create an event, can choose a template or create their own, and can name their event. This is shown in tests 1 and 2 in Table 2.
2	This requirement has been fully met. An event is created after the host has provided the questions for the attendees to complete. A unique code is assigned to this newly created event, and attendees can use this code to join the event. This is shown in test 5 in Table 2.
3	This requirement has been fully met. Hosts can view the feedback from the past x minutes or from all time throughout the event, where x can be modified by the host. This is shown in Figure 10, and in test 6 in Table 2.
4	This requirement has not been met. We underestimated how long the higher priority requirements would take, and decided that this feature was not a priority.
5	This requirement has not been met. We decided that the priority of this requirement should be changed to be lower during development because it is not essential for a working prototype, and we did not have enough time to implement it.
6	This requirement has not been met. Once again we decided during development that our priorities lay in submitting and viewing the feedback, and so the priority of this requirement was lowered.
7	This requirement has been partially met. Hosts are able to use previously used templates when creating new forms. However, we did not implement the ability to delete templates due to time constraints. This is shown in test 4 in Table 2. We decided during development that creating and reusing templates was a higher priority requirement for the prototype.
8	This requirement has not been met. It was a very low priority, hence we did not have time to implement it.

Table 4: Evaluation of functional host requirements

Req. No.	Degree of Success
1	This requirement has been fully met. Although we modified this requirement so that codes were entirely numeric, all codes issued to events were unique, as shown by test 5 in Table 2.
2	This requirement has been fully met. Attendees are able to fill out a feedback form created by the host, as shown in Figure 8, and in test 1 in Table 1.
3	This requirement has been fully met. Attendees are given the option to make their feedback anonymous, as shown in Figure 8, and in test 2 in Table 1.
4	This requirement has been fully met. Attendees can submit star ratings for the questions that require this. This is shown in test 3 in Table 1.
5	This requirement has been fully met. When the attendee is required to give text feedback, it is analysed using sentiment analysis. This is shown by test 4 in Table 1.
6	This requirement has not been met. This feature was not a priority in development and so there was not enough time to implement it.

Table 5: Evaluation of functional attendee requirements

Req. No.	Degree of Success
1	This requirement has been fully met. Attendee responses are submitted in less than five seconds - this is shown by test 1 in Table 3.
2	This requirement has been partially met. The system is accessible from different web browsers (as shown in test 2 in Table 3), and is also displayed correctly when the browser is switched to mobile mode. Making the software accessible from web browsers was the priority, and so we did not test it on a mobile or try to make a mobile application.
3	This requirement has been met. The web page elements will resize according to the size of the browser window, and all elements will remain accessible. This is shown in test 3 in Table 3.

4	This requirement has been fully met. Feedback is accessible to the feedback analysis system to be viewed in under 5 seconds - this is shown in test 1 in Table 3.
---	---

Table 6: Evaluation of non-functional requirements

After looking at the testing, the majority of the initial requirements have been met. Seven requirements were not fully met, however these requirements are not crucial to the core functioning of the software. Therefore we can conclude that this prototype has been successful since the most prioritised/fundamental requirements have been fulfilled.

## 8 Future Development

With our prototype evaluated against the initial requirements, we are now able to suggest areas for future development if this project were to continue.

The most crucial area of improvement for this software is the security of the data stored in the database. We have not included verification of the format of the user's email - this could be done by using regular expressions. A simple way to verify users' identities would be to send a verification email to the account they gave and asking them to click a link within the email. Furthermore, password rejection could also be implemented to improve account security. In other words, instead of allowing any length and permutation of characters, we could force the user to choose a long password containing symbols, numbers and upper case letters.

At the moment, all questions on the feedback form are required, so the attendee has to answer every question before they can submit their feedback. In the future, this could be developed further so that a host can decide which questions are mandatory and which questions are optional. Another useful feature for hosts would be the ability to group users together to form groups, and be able to send an email or notification to the members of the group when there is a new event to join. This would make event code distribution easier for the host.

Although a minor change, adding the event name at the top of the live feedback and the feedback forms would make it clearer to the user which event they are currently viewing or providing feedback. Additionally, changing the way that the user inputs their star rating feedback could be changed so that they could click on one of a group of five images of stars. Logged in users may appreciate being able to filter the events that appear on their menu, for example, they could filter by their role in the event, or by whether or not an event is available for feedback. Another way the prototype could be improved is by adding a logout button for users to be able to log out of their accounts on the profile page once they are logged in.

Another way in which our prototype could be improved would be to optimise the software for use on mobile devices since this would make the feedback system more accessible to attendees of an event - they may not always have their laptop open when they want to give feedback. We could do this by either improving the layout of the website or by developing a mobile application that works in tandem with the web version of the software.

## 9 Business Analysis

From a business point of view, due to the covid-19 pandemic, many markets have risen sharply in the past year and a half. We are mostly interested in online communication, as working from home has become something usual in our days. Moreover, the online communication platform and tools have been adopted by more industries, becoming more used in the business sector as in the academic, real estate, banking and many others. Therefore, applications such as Skype, Zoom, Discord, Microsoft Teams and many others have risen record high numbers.

The global video conferencing market was estimated at 4.8 Billion USD in 2019 and is expected to reach 9.2 Billion USD by 2027 and the market revenues from collaboration software have been increasing in recent years, from around 7 billion USD in 2015 to over 12 billion in 2019 and are forecasted to keep increasing, reaching about 13.58 billion USD in 2024.

As we established that the market for online video communication is rapidly growing and expanding, we would like the Deutsche Bank to furthermore profit from this project. That is why, for the future development of the application, we plan to design the sentimental analysis and feedback visualisation as an easy to use tool that could be adapted to all online communication platforms and collaboration software, with many more features

than presented this far.

This can become a sustainable and recursive cash inflow source for the Deutsche Bank, with a fast return from their investment. Future development can be sustained by the income generated by the tool itself, as well as sustain a complex marketing strategy.

## 10 Conclusion

We have created a live feedback system prototype to cater to the strong feedback culture of the customer. Using this system will provide a way for participants of many events, workshops, or small group projects to give feedback to the event host or group leader, which can be acted on immediately instead of for the next cohort.

We believe our project has been a success because we have fulfilled the requirements of the customer, and all of our prioritised requirements. Furthermore, our project could be developed in the future to add additional functionality and offer more meaningful ways for feedback to be delivered.

## References

- [1] Multinomial naive bayes explained. <https://www.mygreatlearning.com/blog/multinomial-naive-bayes-explained/>. Accessed 12 February 2021.
- [2] lordicon. Icons. <https://lordicon.com/icons>. Accessed 1 March 2021.
- [3] Pallets. Python Flask. <https://flask.palletsprojects.com/en/1.1.x/>. Accessed 1 February 2021.
- [4] Python Software Foundation. Sqlite3. <https://docs.python.org/3/library/sqlite3.html>. Accessed 1 February 2021.
- [5] w3schools. HTTP Methods. [https://www.w3schools.com/tags/ref\\_httpmethods.asp](https://www.w3schools.com/tags/ref_httpmethods.asp). Accessed 2 February 2021.
- [6] Flask Login Page. <https://prettyprinted.com/1/YX2>. Accessed 3 February 2021.
- [7] Processing incoming request data flask. <https://www.digitalocean.com/community/tutorials/processing-incoming-request-data-in-flask>. Accessed 24 February 2021.
- [8] OWASP. Password Storage, Salting. [https://cheatsheetseries.owasp.org/cheatsheets/Password\\_Storage\\_Cheat\\_Sheet.html#salting](https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html#salting). Accessed 2 February 2021.
- [9] Stack Overflow. JSON. <https://docs.python.org/3/library/json.html>. Accessed 21 February 2021.
- [10] How can i show a hidden div when a select option is selected in javascript? <https://www.tutorialspoint.com/how-can-i-show-a-hidden-div-when-a-select-option-is-selected-in-javascript#:~:text=To%20show%20a%20hidden%20div,display%E2%80%9D%20to%20block>. Accessed 16 February 2021.
- [11] Python Software Foundation. random. <https://docs.python.org/3/library/random.html>. Accessed 18 February 2021.
- [12] Dynamically add/remove rows in html table using javascript. <https://www.viralpatel.net/dynamically-add-remove-rows-in-html-table-using-javascript/>. Accessed 25 February 2021.
- [13] Javascript drop-down list. <https://stackoverflow.com/questions/17001961/how-to-add-drop-down-list-select-programmatically>. Accessed 28 February 2021.
- [14] Google charts. <https://developers.google.com/chart>. Accessed 6 March 2021.
- [15] Python os.urandom(). <https://www.geeksforgeeks.org/python-os-urandom-method/>. Accessed 4 March 2021.
- [16] Python base64. <https://www.base64encoder.io/python/>. Accessed 4 March 2021.
- [17] Python hashlib. <https://docs.python.org/3/library/hashlib.html>. Accessed 4 March 2021.
- [18] Scikit and nltk model generator. <https://www.datacamp.com/community/tutorials/text-analytics-beginners-nltk>. Accessed 18 February 2021.
- [19] crowdflower. Kraggle dataset. <https://www.kaggle.com/crowdflower/twitter-airline-sentiment>. Accessed 12 February 2021.
- [20] Random data generator. <https://www.mockaroo.com/>. Accessed 30 February 2021.
- [21] Markup validation service. <https://validator.w3.org/>. Accessed 8 March 2021.
- [22] SQLite Viewer. <https://inloop.github.io/sqlite-viewer/>. Accessed 3 March 2021.
- [23] Jack McCullough Luca Orita Maria Bracegirdle David Yang Zhang Rebecca French. Requirements analysis document.