Planning and Design

Group 25
Finn Butler, Ion Orins, Petru Biriloiu, Shivakumar Mahesh, Zerui Shan Chen

1 Purpose

This document describes the planning and design of the live feedback system developed for Deutsche Bank, the stakeholder. It serves as an initial plan to outline the development team's approach to fulfilling the requirements set out in the Requirements Analysis document provided beforehand. Planning and design choices will be explained in detail with justification for each as well as diagrams showing the development phase and interactions within the system.

2 Glossary

Backend - the data access and manipulation layer of a server application
Deep learning model - a statistical learning model involving artificial neural networks with multiple layers of hidden neurons
Frontend - the presentation layer of a server application
Message-broker software - a computer program that intermediates the comunication between other software applications
NoSQL - describes any database storage mechanism designed to avoid the limitations of relational databases
RESTful API - design pattern involving HTTP requests, meant to standardise APIs
Remote procedure calls - a programming pattern that allows a subroutine to be executed by another program
Responsive web app - A web app that uses responsive design, i.e. tries to accommodate as many screen sizes and types of devices as possible

3 Technical Description

3.1 Third-Party Libraries

As we have chosen to borrow elements from the Reuse-oriented Software Engineering approach, a few third-party utilities will be used: *FastAPI* is a modern backend framework written in Python. It will be used to implement the business logic and manage the interactions between frontend, the database, and the emotion analysis module. *MongoDB* is the chosen database. Its NoSQL approach makes it very flexible, enabling fast prototyping. *React* is a JavaScript library for building fluid and responsive user interfaces. *DeepMoji* is a deep learning model that analyses the nuances of natural language in terms of emotions. It is a key component in the design of our emotion analysis module. *RabbitMQ* is an open-source message-broker software, which will be used for communication between backend components, specifically the FastAPI-emotion analysis tool and emotion analysis tool-DeepMoji interactions. *Docker* is a software service that allows the virtualisation of the development environments in packages called "containers".

3.2 Architecture Overview

We have chosen the Model-View-Controller architecture for following two reasons:

1. by splitting the system into small components, it enables the parallelisation of a considerable amount of tasks;

2. it makes the search for suitable tools easier, as most modern web development frameworks are based on the MVC architecture.
   The view component (mostly written in React) will display the model to the user and will interact with the model through a RESTful API implemented in the controller. The model will be defined in the backend component (FastAPI) and stored in the database (MongoDB). It contains the objects defined in the class diagram. The controller shapes the interaction between the view and the model. It will contain the RESTful API implemented in the backend and the mood analysis tool. The whole architecture will be containerised, using Docker, in order to:

   - speed up development, by side-stepping the need for every developer to install and configure the dependencies;
   - make deployment easier;
   - make bugs more reproducible, and hence testing simpler;
   - increase security, as the components can't interact in an unexpected manner with the systems outside the container.

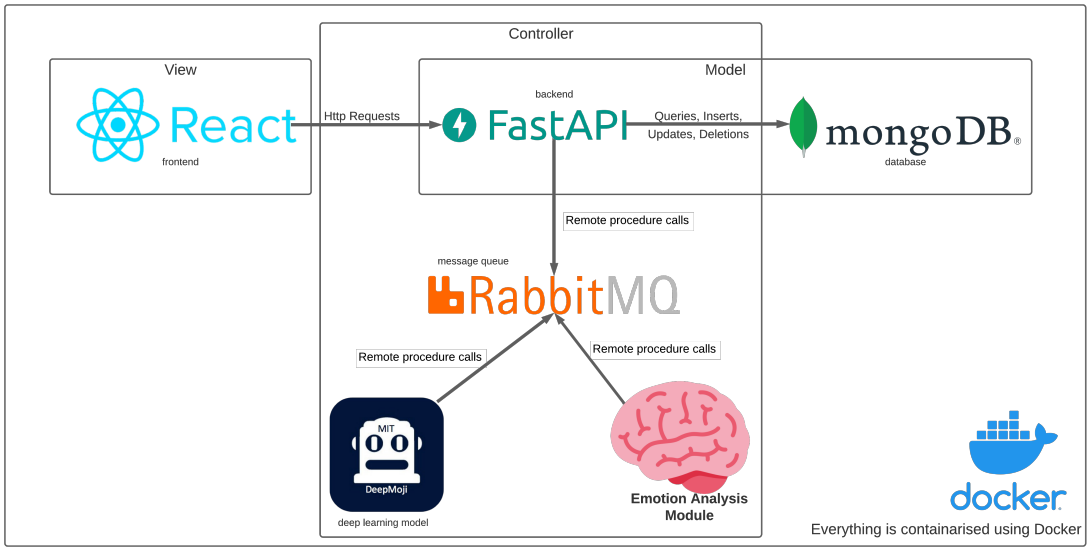

Figure 1: Diagram containing the overall system architecture

3.3 Frontend

3.3.1 Format

During our initial meetings we debated whether to create a desktop-friendly website with corresponding mobile application or solely a responsive web application that would function well on a range of browsers and devices. We chose to move forward with designing a responsive web application as we believed this would be the most efficient implementation whilst keeping all key functionality and aesthetics. This design choice was strengthened by the frontend team having existing experience creating such web applications and the backend team also having experience creating APIs that can be integrated with these web applications. Further benefits of this design choice include: greater compatibility with a multitude of devices in place of having to create multiple apps for different

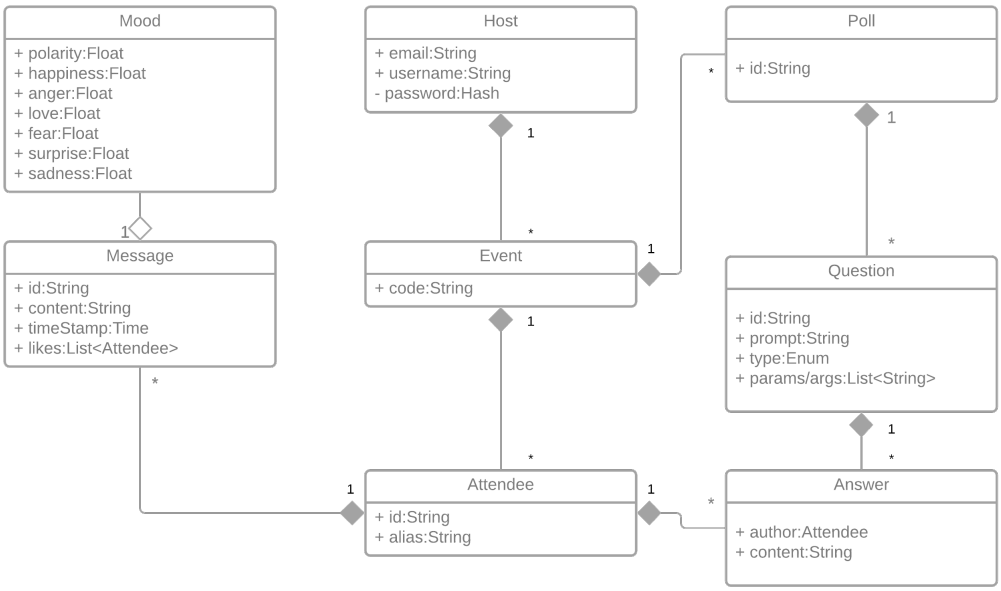

Figure 2: The class diagram of the system

operating systems, increased focus and time on one application as opposed to splitting time over multiple applications, easier testing and prototyping.

3.3.2 Language and Tools

For the web application we chose to use React. React is an open-source frontend JavaScript library for building user interfaces. It is a popular library for creating responsive web applications that are designed to be used on different sized devices such as phones and tablets. As well as this, there are many additional open-source libraries available for use that provide pre-built components for UI design such as navigation bars, forms, dropdowns, overlays etc. Furthermore, the primary frontend developer had previous experience creating applications using React so no research or learning period was needed and development could start earlier.
We used Facebook's "Create React App" to create our development environment with a template app. This provided us a multitude of tools including an integrated test runner, a live development server as well as premade build scripts. Create React App uses Node.js as the server environment for development, which allows the developer to see a live version of the application with changes being reflected instantly.

3.3.3 Branding and Design

The Head of Product position was given to the primary frontend developer as they would be creating all visual components of the application. During initial meetings the team decided to utilise the concept of emoji as a key design feature throughout the application. Emoji, from the Japanese 絵文字, are designed as a method to reflect mood without having to use text for description. They are visual representations of how humans perceive mood and emotion, hence their use in this mood-based feedback application. We utilised the concept of emoji with the concept of a feedback system to focus on the ballot box emoji: 🗳, an emoji designed to represent the concept of a vote or response system. This spawned the branding of our application with the name BallotBox and the specified emoji as the base for our logo.
For the primary colour palette we used a colour scheme generator to generate some example palettes, though in the end we decided to use a simple pairing of two colours: a light magenta and sky blue.

These created a visually appealing gradient while keeping a professional feel to the application. Initial page design mockups were created using an online UI designer. These were then replicated in React to get a feel for how the interface would function in browser and on mobile. This will work as a basis for further functionality that will be added in during the development process, linking to the backend API. The pages will be designed such that any extra components can be easily added and existing components or properties such as colour can be modified with ease.

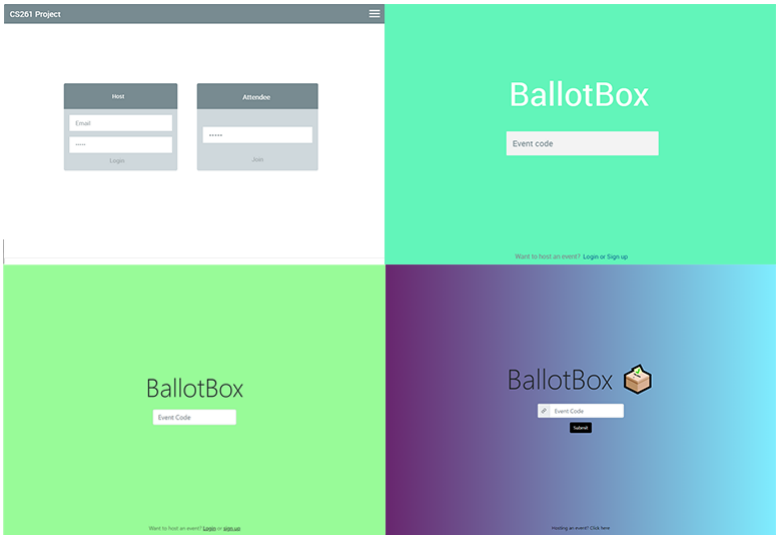

Figure 3: Iterations of UI mockups

3.3.4 Interaction

3.3.4.1 Host

We will create an authentication system for hosts, requiring them to input an email address and password. This is such that hosts can link events to themselves and record all past events for data purposes. They will then be able to create new events or access previously created events. Once in an event, hosts will be shown a control panel allowing them to create polls, see the comment wall and view live feedback data. This aspect will be mostly designed for desktop or larger tablet devices as the complexity of creating a poll is not something we would expect to be undertaken on a mobile device.

3.3.4.2 Attendee

The attendee view will be designed mainly for usage by those who may not have much technical knowledge and who will be accessing the app on a mobile phone. We chose this method as this fits the specification for the usage of the application during a conference or other such events where attendees will only have their mobile phone on them, not a desktop computer. The initial screen will simply contain the branding along with an input for the event code. Alternatively they can skip this step by scanning a QR code which the host can share. Having entered the event, they will then be greeted by a simple interface showing the comment wall along with the option to complete any polls that have been created by the host. Attendees will be able to input text comments and react with likes to other people's comments on the comment wall. This will be fully anonymised unless the attendee chooses an alias to use. This aspect of the application is kept as simple and easy-to-use as possible to cater for all persons who may be attending an event.

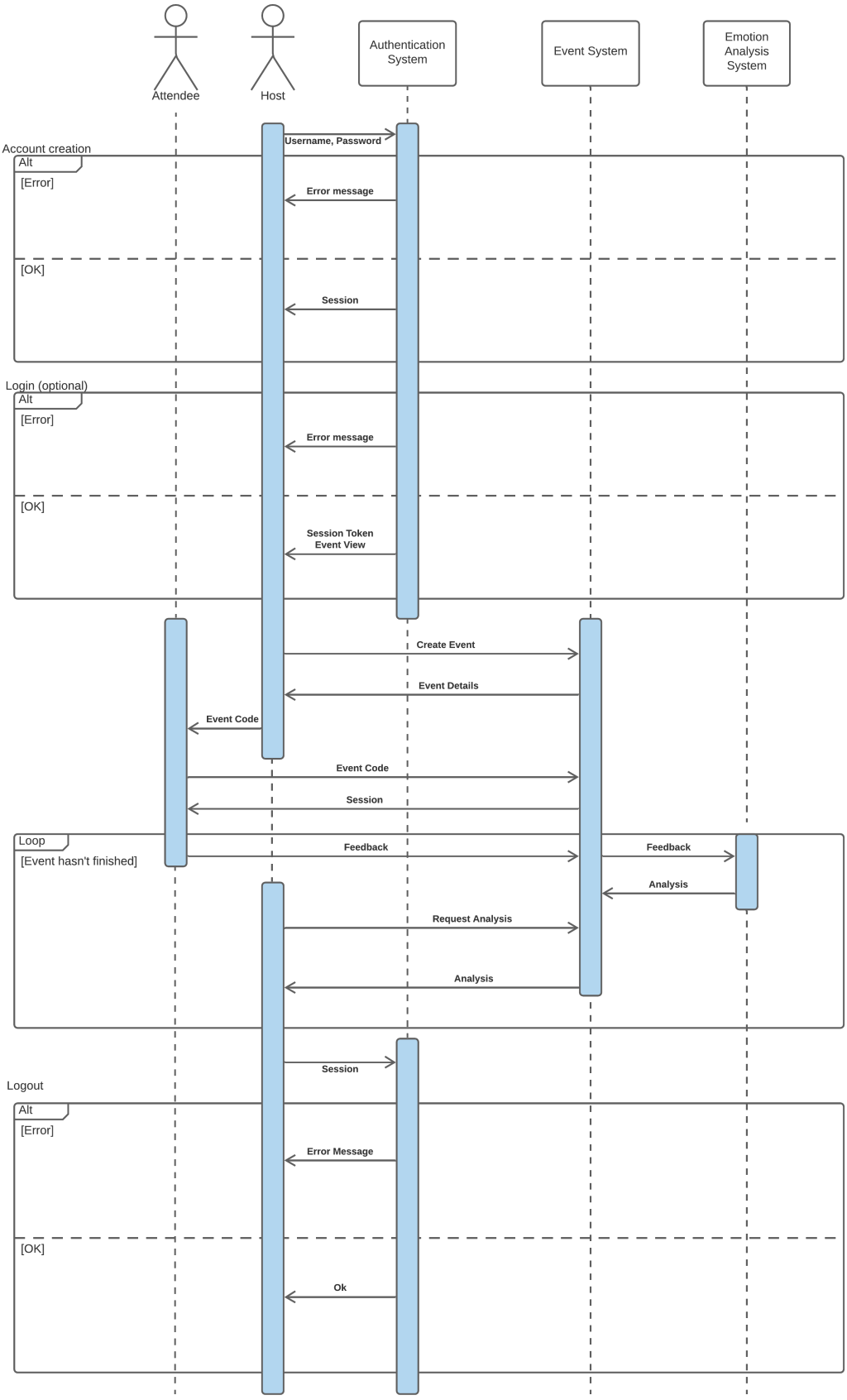

Figure 4: Sequence diagram of a successful event

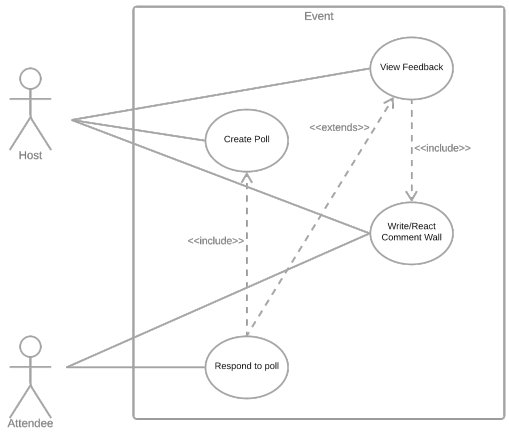

Figure 5: Use case diagram of the event creation and joining processes

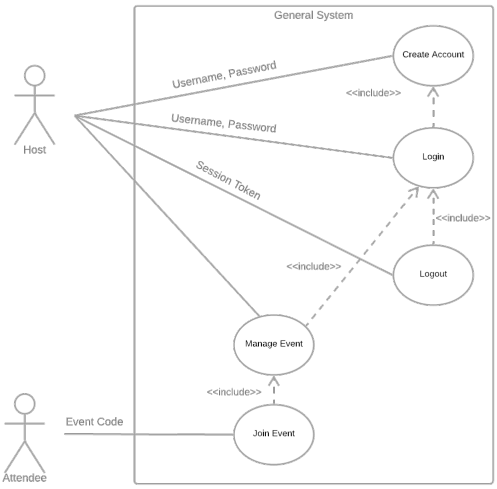

Figure 6: Use case diagram of the attendee-host-system interaction during the event

3.4 Backend

The backend will implement the authentication system and will expose an API to the frontend component that will allow it to query and modify the model. It will store the relevant data in the database and will query the emotion detection component through a message queue in order to complete the mood analysis requirement. Technology-wise, it will be written in Python using the FastAPI web framework. This framework automatically generates the documentation for the RESTful API, which can be used by the front-end developer when implementing the user interface.

3.5 Statistics and Emotion Detection

3.5.1 Statistical Estimators of Mood

3.5.1.1 Polarity

The key aspect of sentiment analysis is to analyse a body of text in order to understand the ideas expressed by it. Typically, we quantify this sentiment with a value $p \in [-1, 1]$ which we call **polarity**. We denote the **overall sentiment** as $S(p)$ where $S : [-1, 1] \mapsto \{-1, 0, 1\}$ using the rule

$$S(p) = \begin{cases} 1 & 1/3 < p \leq 1 \\ 0 & -1/3 < p \leq 1/3 \\ -1 & 0 \leq p \leq -1/3 \end{cases}$$

3.5.1.2 Complex Emotion

We consider the human emotions, namely joy, sadness, anger, fear, surprise and love for the prediction of mood. There is much controversy in psychology over how many basic emotions are displayed by humans, so we simply pick the 6 most commonly used emotions for emotion detection.

We define the **emotion space** to be the set,

$$E = \{\vec{v} \in \mathbb{R}^6 : \vec{v} \cdot \vec{1}_6 = 1, \}$$

where $\vec{1}_6$ the vector in $\mathbb{R}^6$ with all co-ordinates 1. We say any $\vec{v} \in E$ is a **complex emotion** and say that any standard basis vector in $\mathbb{R}^6$ is a **raw emotion**. Hence joy, sadness, etc. are the 6 raw

emotions.

We consider complex emotions since there is an essential uncertainty in emotion that we can predict from the user. Indeed, E is the probability simplex on the 6 emotions. If we chose to predict raw emotion instead we may simply predict the complex emotion first and then predict raw emotion from complex emotion as $\vec{e}_j$ where $j = \text{argmax}_i\, v_i$, v_i is the i^{th} component of $\vec{v} \in E$ and $\vec{e}_j$ is the j^{th} standard basis vector of $\mathbb{R}^6$. The prediction of raw emotion from complex emotion in the way above is justified since it leads to the lowest prediction error.

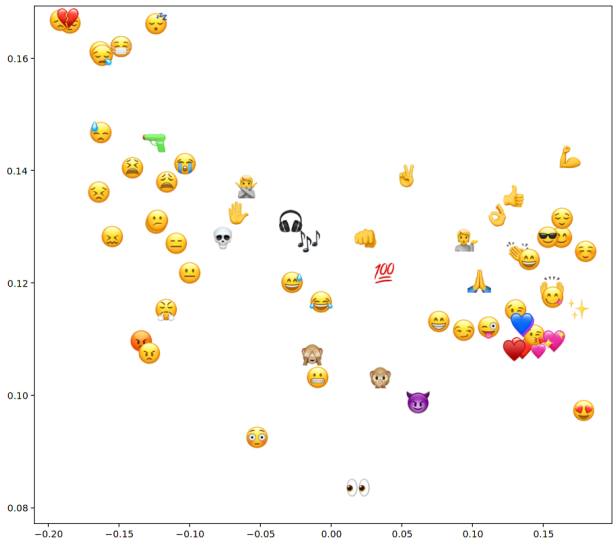

Figure 7: Principal components analysis (dimensionality reduction) on the complex emotion of each emoji estimated using Bayesian Statistics. The complex emotion of each emoji is a point in $\mathbb{R}^6$ and has been projected onto $\mathbb{R}^2$ using PCA.

3.5.2 The Problem

Let T to be the set of all bodies of text and let J be the set of 64 emojis that are used in this project. If $J' \subseteq J$ then the objectives are as follows:

1. Construction of a function $C_e : T \times J' \mapsto E_r$ that obtains a test accuracy greater than 99% when it's output is converted into raw emotion.

2. Construction of a function $P : T \times J' \mapsto [-1, 1]$ that obtains a test accuracy greater than 99% when the polarity is mapped into overall sentiment.

3.5.3 The Solution

3.5.3.1 Estimation of Complex Emotion

For the implementation we rely on **Bayesian Statistics** and novel classifier called **DeepMoji** that takes as input, text data and outputs a probability distribution on emojis (that are likely to be associated with that data).

First, we train a classifier called the *emoji learner* that takes as input $J' \subseteq J$ which is the emoji data (and a second optional parameter a probability distribution over J') and outputs a vector $\vec{v} \in E$. Second, we train a classifier that takes as input $t \in T$ (text data) and outputs a vector $\vec{v} \in E$. We call this learner the *text learner*.

Before building the two learners we require a transformed data set. The initial dataset contains two columns, text and it's target emotion "anger", "sadness", "happiness", etc. We take this dataset,

pipe the text column through DeepMoji to get a probability distribution over emojis for each row of text and use this new transformed dataset which has 65 columns, the first 64 are probability distribution over emojis and the last column is the emotional state target. Using this transformed dataset we perform bayesian estimation to arrive at a probability distribution over each emoji which reflects the degree of anger, joy, sadness, etc. expressed by each emoji, estimated from empirical data.
Hence for each emoji, we obtain a vector that expresses it's complex emotion. Precisely, if X is a random variable that takes values in the set $\{$"*joy*", "*anger*", "*surprise*", "*fear*", "*sadness*", "*love*"$\}$ and Y is a random variable on J the set of emojis,

$$P(X = \text{“}anger\text{”}|Y = 😠) \propto P(Y = 😠|X = \text{“}anger\text{”}) \times P(X = \text{“}anger\text{”}).$$

It is easy to estimate the likelihood and prior from the aforementioned transformed dataset.

For the *emoji learner*, given $J' \subseteq J$ we take the expected value of the complex emotion expressed by each emoji in J (the expected value is taken with respect to the probability distribution specified in its input, if one is not specified the mean is taken).

To build a *text learner*, we can pipe text data into DeepMoji which will output a distribution on emojis, and we take this distribution and pipe it through the emoji learner to receive a complex emotion.

Finally we use an Ensemble method to combine the *emoji learner* and *text learner* into one strong learner. Indeed, we can use a simple weighted average ensemble since both the initial learners are already strong learners and since both the learners return probability distributions over emotional states the weighted average of both distributions is also a probability distribution over emotional states. The model selection, i.e. the estimation of the best weights of the ensemble will be performed using K-fold cross-validation.

3.5.3.2 Estimation of Polarity

It is similar to the estimation of complex emotion and is also simpler. The difference is that the dataset required is one with sentiment labels and the parameters for the ensemble must be re-evaluated for this new task.

3.5.4 Empirical Classification Error

We use two different datasets of emojis and text data with appropriate labels for testing the two models.

4 Testing

4.1 Unit Testing

As we are using an agile methodology, programming and testing of each component will be performed concurrently in order to make sure that each section works individually. The software will be run in an effort to verify that it is performing according to the specifications set beforehand in the requirement analysis report. Results from the input sections will be compared against expected outputs and modifications will be done accordingly.

4.2 Integration Testing

After proving that each component works individually, we will make sure that these components work as a system, with the aim to expose defects in the interaction between them when they are integrated. The following interactions must be tested:

- fontend-backend;
- backend-database;
- backend-emotion analysis tool;
- emotion analysis tool-deep learning model.

Testing will be done to check the flow of data between modules and make sure there are no interfacing bugs.

4.3 System Testing

After the integration of every component, some verifications should be done to check that the system meets the specified functional and non-functional requirements. Our program should be tested on different browsers to check if the application is compatible with them; and tests should also be undertaken on different devices, making sure that the host's interface is compatible with larger devices (such as desktops and/or large tablets), and that the attendees' interface works on mobile phones.

4.4 Acceptance Testing

In order to verify the functionality of the system, we will present our application to a group of friends/family members. They will be provided with a series of tasks, specifically selected to help us determine if the system is easily usable from an end-user's perspective and if our product meets the functional and non-functional requirements written in the requirement analysis report. During this process, their actions will be monitored, and we will ask them to provide live commentary when, and where, they find it difficult to understand the intentions of the software or the UI. Our main intention is that the user should not require much technical knowledge in order to use our application. Once we record the results and we compare the expected data to the actual once we will determine the test results. If these match, the test case will be passed.

4.5 Stress Testing

To make sure the system is robust, a number of stress tests will be employed. They will target both individual components and the system as a whole. This kind of testing will be performed manually but also using scripts designed to push the system to its limits.

5 Organisation Tools

For communication, the platform Discord is used. It was chosen because it allows direct messaging, topic-based communication, voice communication and screen sharing. Text channels for different stages and processes of the system development have been created (such as frontend, backend, requirements and brainstorming), and additional will be created if the need arises. Git is used for version control and Github is used to host the repository. When coding, the developers should create a branch, implement the feature in the branch and then merge it into main. For document sharing and collaboration, or for multimedia content in general, Google Drive is used. Once a document is completed, it must be converted to LaTex. Task assignment and progress tracking is done in Trello. Boards for each software development process will be created. Each board will have three self-explanatory task queues: to-do, doing, done.

6 Planning and Risk Analysis

6.1 Planning

Primary development plans were drafted during the initial meetings at the start of this project. We decided to begin by assigning roles for each aspect of the project, aiming to split the workload evenly between each member whilst at the same time allowing members to focus on their strengths. We have produced a Gantt chart to detail the individual tasks required to complete this project coupled with the time periods in which these tasks shall be completed.

Our planning method is relatively fluid and can be easily extended or modified in case of a change in context such as a requirements change or disruption within the team. We decided that every team member should be kept informed about tasks carried out by other team members in order to allow possible crossover if someone gets stuck on a task and requires assistance. This also provides insurance should a member of the team be unable to complete their tasks in the future.

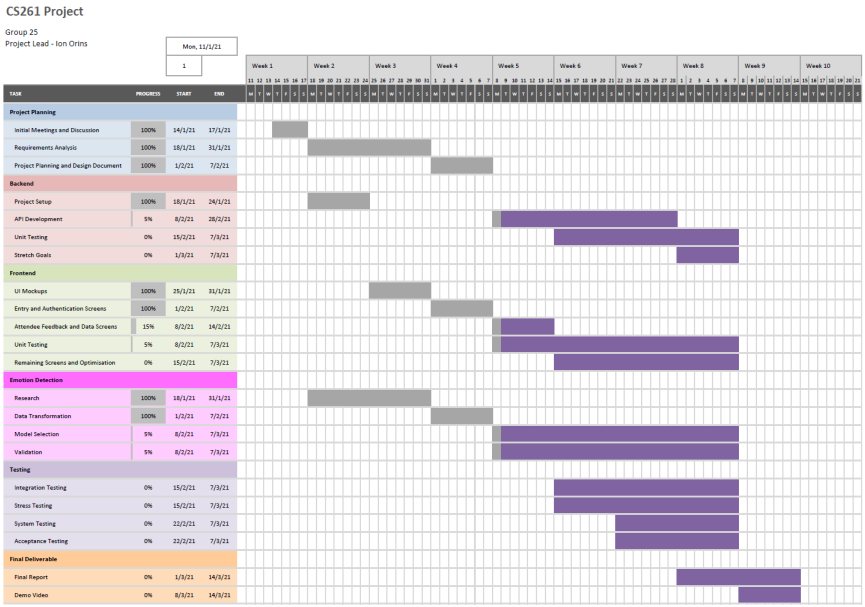

Figure 8: The Gantt chart of the project

6.2 Risk Assessment and Monitoring

We have identified a number of risks and developed a mitigation plan and a contingency plan for each of them. These have been described in the table below.

Risk	Probability	Impact Rating	Mitigation	Contingency	Residual Risk
Team member unavailability	High	Medium	N/A	Reassign roles to other members	Low
Change in requirements	Very High	Medium	Allow flexibility in planning and design	Update designs and plan, redevelop parts if necessary	Very Low
Data loss	Low	High	Use version control and store data in cloud	Try to recover data using various methods, rewrite code	Very Low
Hardware failure	Low	Medium	Keep all necessary project files updated on cloud	Repair or purchase new hardware	Very Low
Software incompatibilities	High	High	Research compatibility before deciding on a tool	Containerise tool and communicate with the rest of the system using http or message queues	Very Low
Slacking	Low	Medium	Keep team members motivated, allow room for delays, keep track of the progress	Ramp up development speed, redistribute tasks	Very Low
Project not finished before deadline	Low	Very High	Create a solid plan and execute it efficiently	Request extension	Low
Dataset-label unavailability	Low	High	N/A	Create dataset labels manually or by using some other machine learning model with high accuracy.	Very Low

Figure 9: Table analysing project risks and mitigation strategies