Final report - Group 40

Tutor: James Archbold
PM: Aidan Filby, BA: Rushabh Shah, Callum Wells, Kai Meller, Artem Grigor

March 15, 2021

1 Abstract

Throughout many industries different events are held with the purpose to engage many different audiences and effectively provide information to them. An important part of creating well received events involves receiving feedback. This is often acquired after an event and may help improve future events however feedback could also be collected during an event to allow more immediate changes if necessary. The client, Deutsche Bank, requested a system to track the "live mood" of attendees of such an event therefore allowing for feedback to be enacted upon quickly. Our system was designed to meet this goal through the analysis of targeted feedback given by users via a web application which could them be reviewed and analysed by an event organiser to help gauge general interest and to help improve the event. We managed to create a prototype of our planned project with code demonstrating the key features that our system would entail and a basic mock up of the system giving an idea of what our system would look like with these features fully implemented into it.

2 Introduction - The problem

Our project was designed to solve the problem which arises by only gathering feedback after the completion of an event. Such an approach is employed by many different companies today and all sorts of different events. This becomes a problem as this feedback cannot be enacted on until after the event. Therefore the desire for a system that allows gathering of feedback live during an event is strong, as this system could greatly benefit many of these companies. The base goal of the system is to provide some idea of the "live mood" of an event during the event but expanding from this the system can be used to provide targeted feedback to the leaders of an event to allow for quick responses to be made to this feedback and the mood throughout a meeting to be reviewed afterwards.
Our system was required simply to allow Hosts to set up events accessible by Attendees who would be able to give some form of feedback during the event. These attendees must also have the option of anonymity while doing so which could encourage those otherwise worried about how their feedback would be received to submit some. There also had to be some flexibility in the use of the system, allowing the Host to personalise the event, and Attendees to give context to their feedback. We solved these problems with the use of Tags which could be added to feedback.

3 System overview - The solution

Our system as a whole is basically made up of 2 parts due to the lack of integration of a lot of the back end system with the front end. Both the more functional back end and the actual mock up full system are worth considering when evaluating our solution.

3.1 Back end

The interactions between the front end and back end of our system are very limited but the back end itself has been well tested and provides the majority of the initially planned functionality including all the most important elements. While the functionality for user actions exists, the lack of a fully integrated front end means that users cannot easily interact with our system. The back end exhibits the following functionalities:
Firstly, User objects are able to be created alongside an associated Host and Attendee object and stored by the main system.
Users can use the createEvent method to generate a new blank event for which they will become the Host. All these events are again stored by the main system. As the Host, they are then able to toggle and change certain features of the event like it's name and whether it can be joined.
Hosts can also add Tags (small strings that can be added to feedback to reference specific topics e.g. communication) to the event which they can use to filter feedback. They can also delete events which recursively deletes all anonymous users/Guest objects associated with it, removes it from the list of events for all users with access to it, and removes it from the main stored list of events.
Users can alternatively join events via the event IDs or a link generated for the event (using these IDs) which finds if a User is the Host or an Attendee of the target event and then allows them to join by setting the currentEvent variable for the corresponding object. When an Attendee joins an event for the first time via a link they'll be added to the list of Attendees for the event and their own list of events they have access to will be updated. All users have a list of events they have access to and whether they are a Host or Attendee of that event.
Once Attendees are linked to an event, they are able to submit feedback to it consisting of 3 parameters: base mood, text and tags. The base mood represents an explicit mood value that would be chosen, the text is the actual feedback, and the tags are selected by the Attendee out of a list provided by the Event, and are designed to be used to specify which topics their feedback concerns.
The overall mood of individual feedback objects is calculated by running a component of the machine learning based library, Stanford CoreNLP [1], on the text component of the feedback, mapping this value to the same range as the explicit values, and then averaging this with the explicit mood value given. We decided to do this to make sure that the mood of the feedback is as accurate as possible (e.g. the attendee could submit a mood that doesn't correlate with their textual feedback). In case the user does not submit either the base mood or textual feedback, the respective mood is taken to be 50%.
After this value is calculated, the feedback is added to the event, and then stored both in the respective User and Tag objects.
The average mood of the event over a time period is calculated as follows:
Over the time period, an average mood is found for each user. This involves finding all feedback the user submitted during that time period and averaging the mood of these. Then these averages are averaged across all the users. Users that have submitted no feedback are ignored.
While not implemented on the front end, the back end also provides functionality to filter submitted feedback by Users or Tags, which is a useful feature for the Host to see feedback specified towards

certain areas or from specific Users. It also would work well with some of the cut features of the system like the ability to ignore feedback as the Host.

3.2 Front end

Not all of these desired functionalities have been fully implemented on the front end, but we do have an accessible working system that can be run via localhost.
In general this system runs on some test data loaded via localhost:*port*/init. The system is then used as follows, starting from the /login page:

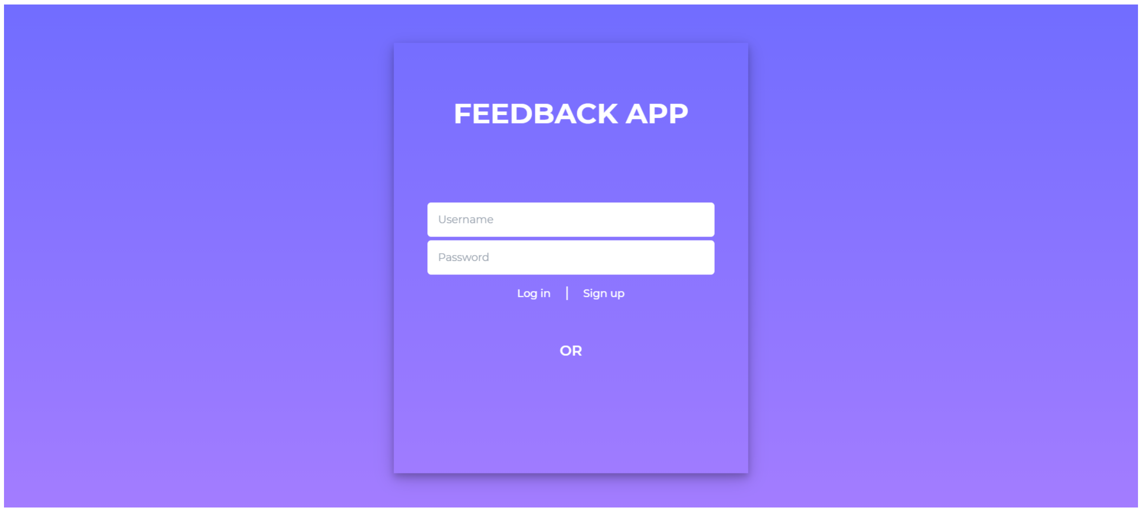


Figure 1: Implementation plan

Firstly the main login page is given. From this page a user would usually be able to join an event via an ID or could choose to login or sign-up. Here the only option is to login, which does not require a password, as the current system lacks any security system. Simply entering the Username of a user within the system and clicking enter will take you to the next page.

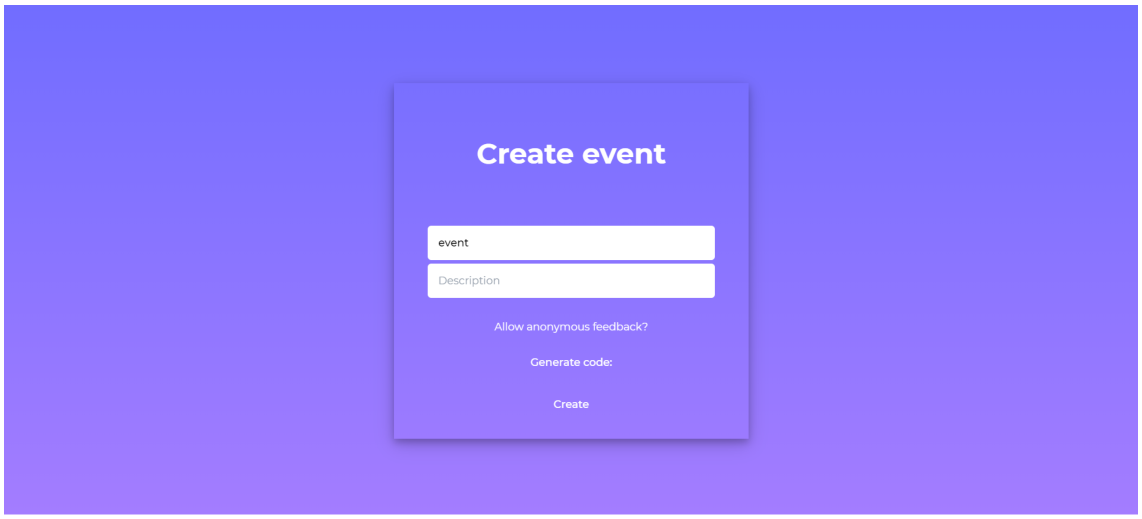
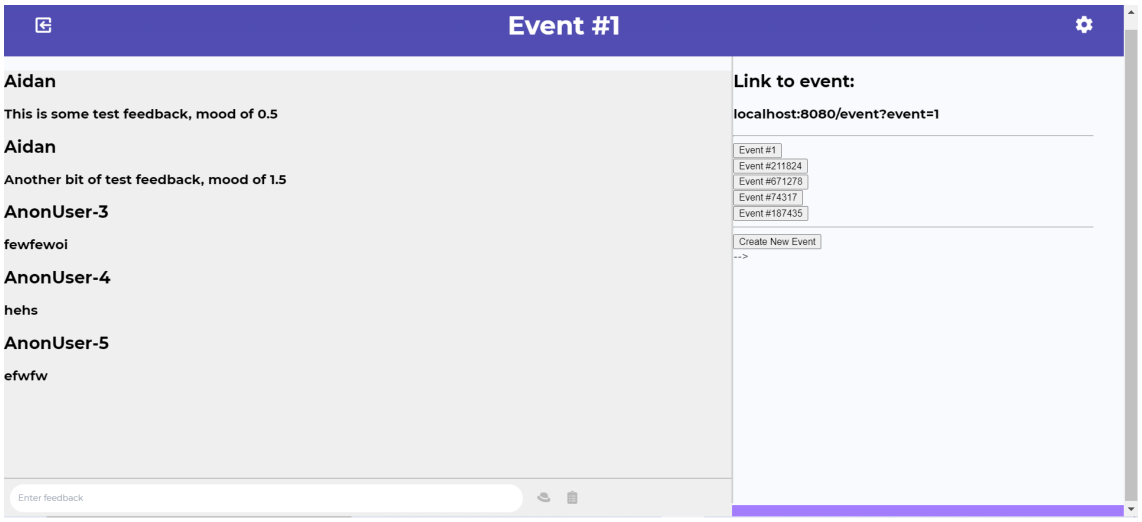


Figure 2: Implementation plan

This page is the event creation page. Here a user can simply enter an event name to create an event with that name and a randomly generated ID.



Aidan

This is some test feedback, mood of 0.5

Aidan

Another bit of test feedback, mood of 1.5

AnonUser-3

fewfewoi

AnonUser-4

hehs

AnonUser-5

efwfw

Figure 3: Implementation plan

Doing so takes us to the main event page. Down the right side is a list of events for the user which can be moved between and a link which can be shared and then used to join the current event. On this page feedback can be submitted to an event, though it is automatically treated as if it were from a random anonymous user and lacking any mood value when it is displayed on the screen. Additionally there is the ability to return to the event creation page where new events can be created.
Additionally a link on this event page can be used to access a graph of the general mood of feedback coming in over time. The data used by this graph was not fully integrated with the base system but still provided an idea of how mood data would be displayed to the Host.
This system is buggy and does not exhibit all the functionality of the back end as we unfortunately

ran out of time to do this, but we are glad we at least managed to achieve a semi-functional system in the end to demonstrate how navigation through the completed system would occur.

4 Changes from original plan

Many small features were changed from the original plan and many features were unfortunately dropped due to the time constraints of the project among other things that are discussed in more detail later. However a couple of major changes that did occur were the handling of anonymity and the approach taken when connecting up the system.

4.1 Anonymity

Initially, anonymity was planned to use simply an optional parameter for a user which could be toggled. When a user was anonymous they would have a temporary username. This username would be directly linked to an event preventing it from changing and so feedback from the same anonymous user could be tracked.

However, we then had to consider how this information would be stored, including the temporary usernames for each event and the feedback submitted which needs to be sorted by User. It's worth noting that usernames were not forced to be unique in our system although likely would be if we were to expand the system to feature an actual account system.

Users were given unique identifying IDs, although the generation and handling of these IDs is something that could be improved. For example, more random IDs could be used rather than using the incremental approach we chose, which could be bad for security reasons even though users would not typically be referenced directly by ID, at least not by other Users.

In order to successfully provide the desired functionality for anonymous users, additional user objects would need to be created for anonymous users so that these users would be referenced by the sorting of the feedback by user. These users would then share the ID of the base Attendee user and for each attendee a list of anonymous versions would be stored referencing the event they are associated with.

To achieve this a Guest class was created extending the Attendee class. The constructor for this method takes a User object as a parameter which the Guest object is then linked to via ID. It calls the constructor of the Attendee class using the ID of the User and the User object passed to it as parameters but then follows this by generating a new randomly generated name for the otherwise almost identical new User.

All such names are of the form ”temp” followed by 6 single digits and are unique pulling from the same pool of IDs as events (though the system could be adjusted such that it uses its own pool).

When a User first joins an event, a Guest object for them, for that event, is created and then this is used any other time they join the event in future and become anonymous. The Guest for the current event an Attendee is in is updated whenever they access an event by checking if one already exists, using it if so, but creating one otherwise.

In the process of creating this system for anonymous users the COULD requirement of having Guest users was also satisfied. This was envisioned to work such that users would start as Guests when accessing the system, with those Guests objects linked to a default/basically blank, user object of ID 0. These Guests could access events like normal but not create any. By logging in (functionality not included) or signing up (this was included to some degree) the Guest could become a main user instead. All this functionality was tested and worked on the back end.

4.2 REST API

The idea of using a REST API came up relatively early but exactly how we were going to use it changed over time. First, we discussed making both a mobile app and web-app, and using a REST API to enable data transfer between them. This idea was put on hold as something that we could consider if we were ahead of schedule, as developing a mobile app would take a long time.
The next time we considered a REST API was when we were deciding on a framework to use. We had to choose between using either React, or Backbone. One of the points we discussed when deliberating over the decision was that Backbone had RESTful services. We thought that we could integrate RESTful services with the JavaScript functionality that Backbone would give. We initially decided to use Backbone, however changed our decision early on in the implementation stage to use the previously mentioned React framework.
The main use that we had for the REST API was connecting our back end Java logic to the HTML and JavaScript objects, as well as mapping the HTTP requests to the web-app HTML files. We utilised a service called Spring[2], which allowed us to create an MVC-style controller module that would receive HTTP requests, returning HTML pages to the user's browser, activating the back end logic in the process of doing so. This controller was written in Java, and ran as a web-server using SpringBoot[2]. As a result, a state could be kept on the server side that stored the data created by the users. Receiving GET and POST requests was handled by the REST API, allowing for data to be passed between HTML pages as well as into the logic methods. This was handled by very simple notation with the Spring API, which was very helpful in simplifying the implementation process. The API allowed for the return of Strings to the users browser, which we used to send HTML pages compiled into a single string to display the information to the user.
The decision to use a REST API was a vital one in the implementation process. Any significant functionality, as well as testing was blocked until we could connect together all of the components of the back end to their front end equivalents.

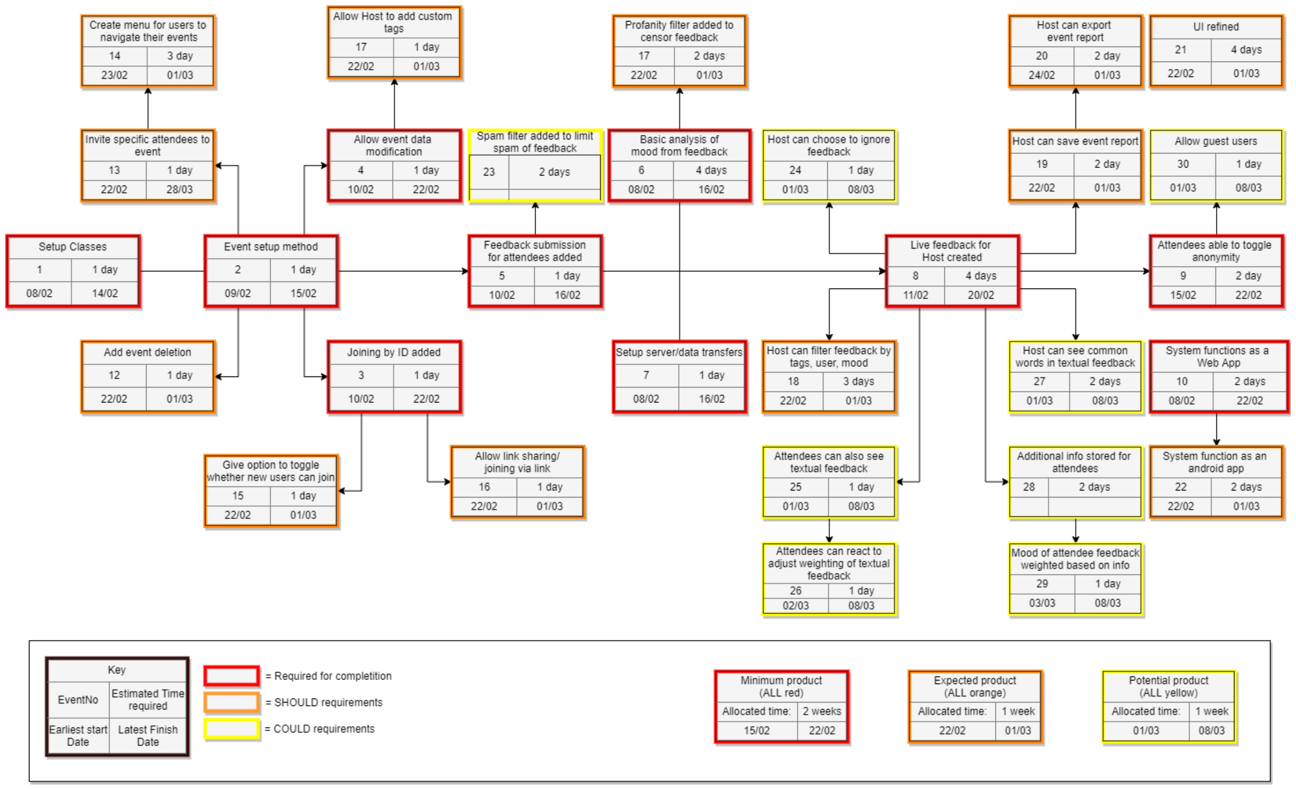
5 The development process

This was our first time developing a big group software project and hence we found ourselves learning how best to run the process as we went. While things mostly went well, we did encounter some hurdles during the process.

5.1 Initial plan and organisation

Our initial plan for completing this project was optimistic but designed to still give some leeway for failure to achieve certain parts of it. Requirements for the project were ordered by priority, with the 3 levels of priority being MUST, SHOULD and COULD allowing a clear indication of the importance of their completion.
We took an incremental software approach designed to cope with potential changes to the requirements while still ensuring a functioning project was achieved by the end of the 5 week development cycle. An Incremental software development involves completing the project in increments with integration and testing of completed components at the end of each iteration/development cycle. This allows the project to be completed in smaller increments while allowing for a fully connected system to be completed early on to which new features can then be continuously added in future cycles. It seemed suitable for this project due to the way that many of the smaller features we had considered for the system would not require major changes to the base system to add.

The initial plan was designed to give us a functioning product with MUST requirements completed 2 weeks into development, and with SHOULD and COULD requirements each requiring an additional week. This left the final week free allowing for any finishing touches or catching up alongside the creation of our final report and presentation of our product.
Within these weeks, the tasks to be completed were split up and an estimate of the work required for each component was considered as well as dependencies between then. This provided us with the following complete plan:



Create menu for users to navigate their events 14 3 day 23/02 01/03
Allow Host to add custom tags 17 1 day 22/02 01/03
Profanity filter added to censor feedback 17 2 days 22/02 01/03
Host can export event report 20 2 day 24/02 01/03
UI refined 21 4 days 22/02 01/03
Invite specific attendees to event 13 1 day 22/02 28/03
Allow event data modification 4 1 day 10/02 22/02
Spam filter added to limit spam of feedback 23 2 days
Basic analysis of mood from feedback 6 4 days 08/02 16/02
Host can choose to ignore feedback 24 1 day 01/03 08/03
Host can save event report 19 2 day 22/02 01/03
Allow guest users 30 1 day 01/03 08/03
Setup Classes 1 1 day 08/02 14/02
Event setup method 2 1 day 09/02 15/02
Feedback submission for attendees added 5 1 day 10/02 16/02
Live feedback for Host created 8 4 days 11/02 20/02
Attendees able to toggle anonymity 9 2 day 15/02 22/02
Add event deletion 12 1 day 22/02 01/03
Joining by ID added 3 1 day 10/02 22/02
Setup server/data transfers 7 1 day 08/02 16/02
Host can filter feedback by tags, user, mood 18 3 days 22/02 01/03
Host can see common words in textual feedback 27 2 days 01/03 08/03
System functions as a Web App 10 2 days 08/02 22/02
Give option to toggle whether new users can join 15 1 day 22/02 01/03
Allow link sharing/ joining via link 16 1 day 22/02 01/03
Attendees can also see textual feedback 25 1 day 01/03 08/03
Additional info stored for attendees 28 2 days
System function as an android app 22 2 days 22/02 01/03
Attendees can react to adjust weighting of textual feedback 26 1 day 02/03 08/03
Mood of attendee feedback weighted based on info 29 1 day 03/03 08/03
Key
EventNo
Estimated Time required
Earliest start Date
Latest Finish Date
= Required for completition
= SHOULD requirements
= COULD requirements
Minimum product (ALL red) Allocated time: 2 weeks 15/02 22/02
Expected product (ALL orange) Allocated time: 1 week 22/02 01/03
Potential product (ALL yellow) Allocated time: 1 week 01/03 08/03

Figure 4: Implementation plan

The work was delegated between the team members based on our strengths and what parts of the project we wished to focus on.
Rushabh took command of constructing the UI elements for the project.
Aidan and Kai focused on the back end logic of the system with Aidan focusing more on Users and User interactions and Kai focussing more on Events and Feedback analysis.
Callum and Artem were both originally slated to work on the front end and providing the interactions between the elements of the UI and the back end logic of the system with Artem also providing some support to the back end.

5.2 Skeleton code

The first step in the development of the back end of our system required the construction of skeleton code for the project. This code closely followed the class diagrams constructed during our design process allowing for it to be written quickly and included the different classes required by the system and the data each required.

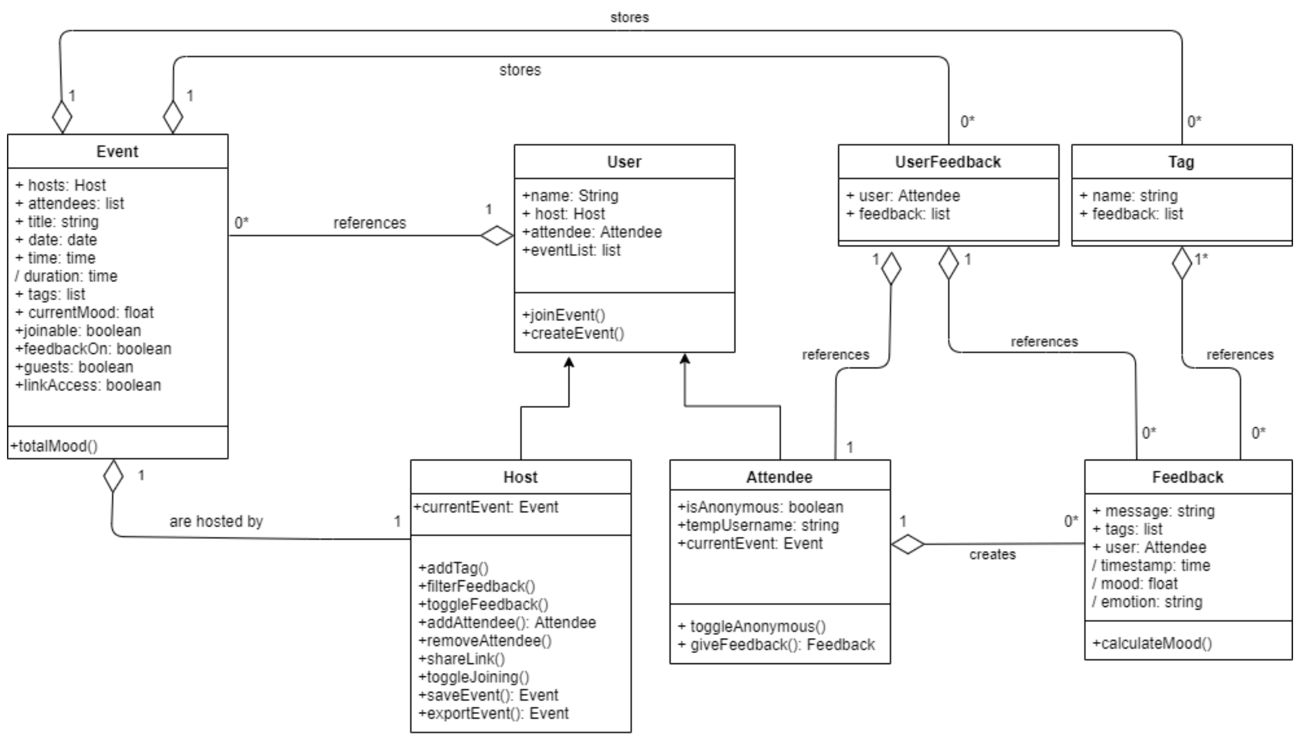


Figure 5: Class diagrams

During the process consideration was made of smaller methods that may have been missed in the design but would likely still be needed to achieve certain requirements or within other different methods. Comments were used throughout the skeleton code to provided a clear idea of exactly what each method would need to do and when/what action (possibly in the front end), may trigger it, as well as the priority of each method. Some larger methods were also given a brief outline of how they'd function using comments within the method and an additional main class was also created to store data for the system as a whole and to allow for easy testing of different back end components and methods as they were built.

5.3 Connecting up the system

Connecting the front end to the back end components very quickly became a prominent issue in the implementation phase. When designing the system, we had not focused very much on how we planned to make the connection; nobody in the group had much experience with linking Java into a web-app before, which led to ideas being very limited in the discussions in this regard. Going into the implementation phase we had assumed that we would be able to use a framework to call on the Java functions, and that we would be able to use JSON to pass all of the data between front end, JavaScript and the back end.
Once we began working on connecting the components together, it became clear that this did not work. Finding a solution that would allow us to connect the logic became the main concern as we looked to create a basic functioning model and was a focus point in the meetings we had at the time. We decided to ask our project supervisor if they had any recommendations, as we thought that due to our lack of experience in the error, we could be steered more in the right direction to look for a solution. They suggested that we should look into REST APIs, as it seemed like an

efficient way to fix the problem. REST had already come up earlier in the initial discussion of the project, however in that circumstance it was for linking the web-app to a mobile app if we decided to create both. Looking more into the idea, we decided that using such an API would be well suited to our existing code, and we started looking for a suitable API.
When looking for an API, we focused mainly on finding one that would be very quick to learn and could be implemented in a very fast and simple manner. Everyone working on the implementation needed to be able to simply understand the API, as everyone in the group would likely be working with it in some regard. After looking online for an API that would work with Java, we decided on using Spring[2], and more specifically Spring Boot, an API which allows for the creation of RESTful systems that follow the MVC model. Spring Boot fit the requirements we were looking for, as it was a popular method for creating RESTful systems, and so there was plentiful information online that helped with development. Additionally, when designing the app, we decided that we wanted to follow the MVC model, and as such we were able to implement our desired system model in a more integrated way.
At first we attempted to run the REST system separate to the web interface, with the front end making calls to the REST service, and then printing out the data returned. While this method of implementation could have worked, it was not very good for testing easily as it would require the app to be deployed directly to the internet. We hadn't discussed a way of hosting the web app at that point and it was decided that doing it this way would only delay connecting the components together. As such we looked for an alternative system that would work with code we had at the time. The solution we settled on was to run the REST controller as the main service, with accessing the app being done entirely through the REST API. The API would take HTTP requests to given URLs, and return an appropriate HTML page according to that request. This simplified the process of testing the program a lot, as calls to the website were effectively handled as the same process as the REST API, and so navigating to mapped requests became a lot easier. Implementing the API itself did not pose too many problems - the syntax for a simple implementation was quite easy to understand and use. The main problem that arose was keeping a storage state throughout the uptime of the website. Data was kept through Java in the back end, however the storage had to be initialised to a single static state whenever the API was initially booted. The Spring system primarily relied on JPA database to create repositories, however this would have required work on the back end components to make sure they functioned with the new storage method, and so we instead decided to find an alternative solution. We solved this by mapping a request (/init) which would initialise the needed data stores, which could be accessed once after starting the REST service. This also was very helpful as a way of loading in dummy data to the website, which made testing significantly easier.
Once the storage system had been finalised, connecting up the system became a much simpler task of wiring together each bit of functionality, with much of the implementation following a similar template. This led to a cascade of every component of the system coming together very quickly once this hurdle had been cleared.

5.4 Was the plan/organisation followed?

Early in the development cycle motivation was low despite the intended deadline of a working base system by the end of the 2nd week and so development took a while to begin.
However twice weekly meetings were quickly organised to keep track of progress alongside many other means of communication which was done predominantly via discord both within a server and individually between team members for certain components. Trello was also used and is where records of each of the meetings including what was discussed and decided were kept and even what

was planned to be discussed so team members could have an idea of what was going to be discussed in the meeting and add what they were wanting to discuss so others were aware. Finally, we also used a GitHub repository to store the code for the project and regular commits helped update each other on what was being worked on.
We also almost immediately lost contact with a team member though thankfully we felt we had accounted for this due to the member in question also lacking in contribution to the original deliverables. As this result had been expected the team member was not assigned a very specific role in the development process but with room given for him to join if he chose to. When it became clear this was unlikely to happen the work was already nicely divided between the remaining members although we do believe he would've been useful for the more difficult components of the project such as connecting the system where more knowledge and help was clearly needed.
Despite this it was still very difficult to balance the workload with only 4 of the expected 5 team members and we quickly fell behind our initial plan and did not meet our initial deadline for a working project.
Development did begin to accelerate however and though deadlines continued to be missed this was mostly due to some major bottlenecks and difficulties discovered during the process. Eventually this did lead to certain elements of the project having to be cut including:

1. The use of a menu page to join events a user is a member of. Upon entering the system users would've been given the option to login/signup or just join an event via an ID/link. Once logged in, which would be done by some external account system, the user would've had access to a page showing events they have access to and also giving them the option to create events or join by ID as normal. These events would be labelled to show if the user was the Host or an Attendee and in some scenarios events could be adjusted so that joining was only allowed for existing members via this method and not via links.
   This part of the system was removed as it was considered unnecessary for our prototype and would require a considerable amount of further effort with both UI and back end functionality to produce. Also due to the lack of a structured account system it seemed unnecessary as Users could just join events as guests. The removal of this did also lead to the removal of the option of adding/removing specific users from an event though this could still exist without the menu as the majority of the need for this was to do with the different access levels of events that the menu would allow. As a whole, the change actually simplifies the complexity of the system quite a bit while not compromising too much in terms of functionality.

2. The ability for hosts to ignore items of feedback and hence remove them from consideration in calculations of mood (essentially deleting them). To implement a basic version of this would not have required too much effort on the back end though would've required the ability to clearly be able to select and remove feedback in the front end display. The difficulty with this feature would've lied in exactly how mood was effected and whether past mood values needed to be recalculated in case they were effected by the removal of the feedback object. This could require some intensive repeated calculations compromising the live element of the system.

3. Saving and Exporting event data. This is another option that may not have been too difficult to implement on the back end due to event data being clearly stored with event objects however we were unsure of how to structure the pdf report that would be presented to the Host in this case and the feature was decided to be mostly unnecessary as the user can get any information they want directly from the system without needing to export data, only losing it after they choose to delete the event.

4. The ordering of feedback by different parameters such as newest, oldest or most relevant (perhaps decided by user reactions to the feedback, themselves being an additional feature we considered but chose not to implement). This would've required the ability to sort feedback by such parameters. Also none of these options seemed like particularly sensible ways to sort the data and so just sorting data by newest first as its stored by default felt suitable.

5. Spam and profanity filters. These features would obviously be added if there was more time. A profanity filter especially is a very important feature for any system to have but we were unable to find a suitable library to achieve this in time. A spam filter would've been equally useful, providing a preventative measure against users trying to break/mess with the system.

6. Showing common words in feedback. This feature would've required some intensive analysis of the words used throughout all the feedback in the system and could likely have been slow. The use of tags in the system all mostly covered the need to direct feedback to specific areas though this feature still could've allowed a clearer indication of exactly what, related to the tags, was a problem. Tags in general were seen more as topics/areas towards which feedback would be directed (often neutral in nature) whereas keywords would be more specific (e.g. a tag could be 'communication' and a common keyword could simply be 'laggy').

7. Weighting feedback on user data. This feature very quickly was seen to be too complicated to implement and especially hard to do effectively. It was difficult to both consider exactly what to weight feedback on and what info to ask from and store for users. Additionally the effect of guests and anonymous users on this system had to be taken into account. There were no clear ideas of what and how much to weight feedback on and the feature was ultimately considered unnecessary to achieve the key requirements. This feature would likely have been scrapped even given the time to complete it's implementation.

8. Android mobile app. We initially planned as a SHOULD requirement to allow users to join via an Android app. Looking back, we realise that this was not a realistic requirement for the reason that it would roughly double the work load of making just a web-app as it would be a separate product. However, we knew to prioritise the web-app over the Android app in our requirements, and once we realised the scale of the project and the lack of time, we quickly discarded the idea, especially as it would only contribute minimally to the prototype.

9. Feedback templates. One idea that we mentioned in our requirements analysis was the ability for Hosts to tailor the format of the feedback to their needs. The idea was that Hosts could remove features from the feedback objects (e.g. removing textual feedback and tags from the template if they just want Attendees to submit a mood value). We would have liked to implement this feature as we believe that it would be useful to Hosts, and offers that extra degree of flexibility for them to use the app in the way they want.

Despite the removal of some of these elements other elements, even some originally of lower priority than those above were still included such as allowing Guest users which followed trivially from necessary changes to the handling of anonymity.

5.5 Major challenges

There were a large number of challenges faced during this process with all sorts of different errors occurring throughout the code. The majority of these ended up to be focused around the lack of experience in building a larger project, for example with a lack of knowledge of how to link

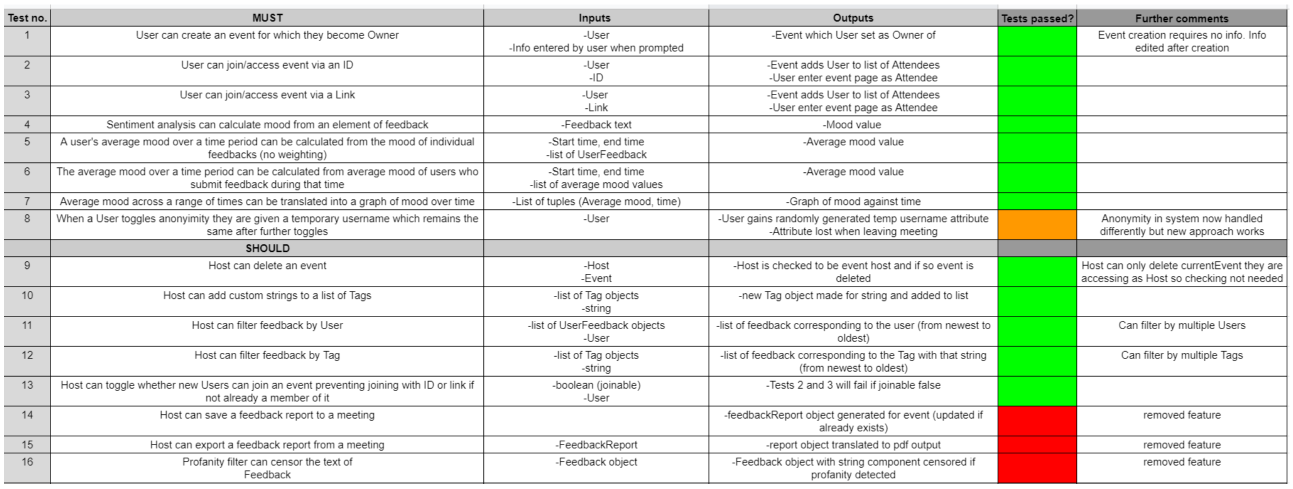
multiple java class files and how to adjust the classpath of a java program (where specified packages and classes used by the program can be found) so needed libraries could be accessed.
However the main challenge of the development was the complete lack of any experience and knowledge in the field of web development which was a necessary area to understand in order to create a system that could function as a web app. We were quite unlucky to end up in this situation but had no more than some basic HTML and CSS knowledge between us. This is part of the main reason why connecting up the system became such a huge bottleneck in the development of the project as we explored different approaches and lots of time was spent researching. Eventually we required prompting by our supervisor towards one of the approaches we had looked at, using a REST API.
In hindsight we would have done a lot better by deciding on an approach early on and sticking with it as we tried to do during planning where we initially decided to use the Backbone.js framework [3]. We also could've sought help from our tutor far earlier and maintained much better contact with them in general during the development process like we did during the first part of the project.
We were also badly effected by having only 4 of our initial 5 team members active during development as mentioned before and also found ourselves plagued by illness and other commitments which in some cases limited the contributions of other team members. In these situations there was a varying degree of communication of the potential issue and consequential picking up of the slack by other team members.

6 Evaluation of the system

Our final product was far from what we initially intended it to be but still shows off a number of interesting ideas of how the fully completed system we envisioned would work.

6.1 Testing

When it came to testing the system, unit testing was performed throughout to test individual components of the system on the back end. This made use of our test logs presented in the design and planning document. An extra 2 columns were added, 1 to record whether the test had passed, and 1 for any further comments about the feature or info on any changes to it. A green box indicates the test passed, a red box indicates failure and an orange box indicates some form of partial success for example where a major change was made to the feature being tested. Due to time constraints all tests based around COULD requirements failed/were not completed and so are not included in the final test log.



Test no.	MUST	Inputs	Outputs	Tests passed?	Further comments
1	User can create an event for which they become Owner	-User -Info entered by user when prompted	-Event which User set as Owner of	🟩	Event creation requires no info. Info edited after creation
2	User can join/access event via an ID	-User -ID	-Event adds User to list of Attendees -User enter event page as Attendee	🟩	
3	User can join/access event via a Link	-User -Link	-Event adds User to list of Attendees -User enter event page as Attendee	🟩	
4	Sentiment analysis can calculate mood from an element of feedback	-Feedback text	-Mood value	🟩	
5	A user's average mood over a time period can be calculated from the mood of individual feedbacks (no weighting)	-Start time, end time -list of UserFeedback	-Average mood value	🟩	
6	The average mood over a time period can be calculated from average mood of users who submit feedback during that time	-Start time, end time -list of average mood values	-Average mood value	🟩	
7	Average mood across a range of times can be translated into a graph of mood over time	-List of tuples (Average mood, time)	-Graph of mood against time	🟩	
8	When a User toggles anonyimity they are given a temporary username which remains the same after further toggles	-User	-User gains randomly generated temp username attribute -Attribute lost when leaving meeting	🟧	Anonymity in system now handled differently but new approach works
	SHOULD				
9	Host can delete an event	-Host -Event	-Host is checked to be event host and if so event is deleted	🟩	Host can only delete currentEvent they are accessing as Host so checking not needed
10	Host can add custom strings to a list of Tags	-list of Tag objects -string	-new Tag object made for string and added to list	🟩	
11	Host can filter feedback by User	-list of UserFeedback objects -User	-list of feedback corresponding to the user (from newest to oldest)	🟩	Can filter by multiple Users
12	Host can filter feedback by Tag	-list of Tag objects -string	-list of feedback corresponding to the Tag with that string (from newest to oldest)	🟩	Can filter by multiple Tags
13	Host can toggle whether new Users can join an event preventing joining with ID or link if not already a member of it	-boolean (joinable) -User	-Tests 2 and 3 will fail if joinable false	🟩	
14	Host can save a feedback report to a meeting		-feedbackReport object generated for event (updated if already exists)	🟥	removed feature
15	Host can export a feedback report from a meeting	-FeedbackReport	-report object translated to pdf output	🟥	removed feature
16	Profanity filter can censor the text of Feedback	-Feedback object	-Feedback object with string component censored if profanity detected	🟥	removed feature

Figure 6: Implementation plan

Tests were performed with a variety of values using the main method of the back end system. As far as integration testing is concerned, little of this was actually used due to the lack of a fully integrated system for much of production. A combination of some elements of the back end were however used together to assure certain features worked together as expected during development but these tests were not formally recorded.
As a whole the system is likely to contain some unidentified bugs and what exists of a fully integrated system already lacks needed functionality from the back end to prevent errors and allow the system to be used effectively. However the pre-planning during the design of the system means there are few known flaws with the system as a whole and the majority of the individual components of the system have been shown to function effectively with all sorts of inputs.
User acceptance testing was also not really possible to perform as we stated in our design however the system did not reach a state where it was ready for this kind of testing during development anyway.

6.2 UI evaluation

The UI makes use of a minimal interface to keep a smooth and focused design. User attention is drawn in with a simple yet effective colour scheme of purples, whites and greys, keeping the website simple and easy on the eyes. Buttons utilize principles ensuring that visibility of system status is met: by darkening when the user hovers over them, and changing colour to purple when activated (as can be seen with the toggle anonymous button). A fedora icon symbolises an 'undercover' mode, which allows users to input feedback anonymously when activated, and the button also becomes highlighted to ensure the user is aware when the mode is and is not active, as submitting anonymous feedback may be a significant feature in some events. Clicking the templates button brings up a modal box and causes the background to darken, identifying to the user that they should shift their attention to the contents of the box should they wish to use templates to give feedback. Clicking outside the box closes the template options. We chose to also include labels for the message buttons, to further ensure that the system is readable and easy to use.

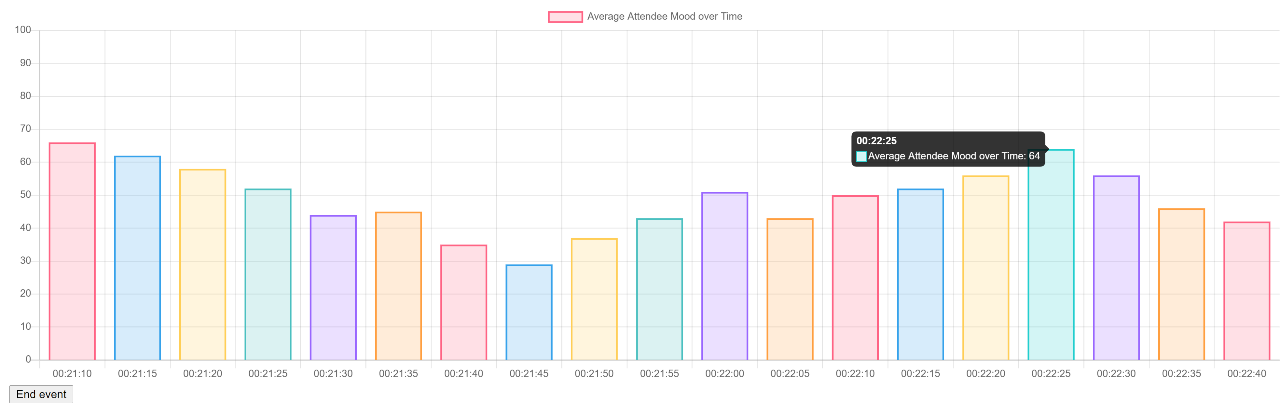


Figure 7: Average Mood Graph

The graph deviates from the general colour scheme and makes use of a range of more vibrant colours to allow new mood data to stand out to the host, and updates itself every 5 seconds with an average of all new mood data collected in this interval. Mood is calculated on a scale of 0 to 100, with 100 being positive. As you can see, we use a bar chart as opposed to a line graph as showing data in bars would mean more transparency about the true meaning behind the visual representations: in a line graph, a slope between two data points might be interpreted as a steady increase. The average mood of all feedback messages received in the last 5 seconds is calculated and output as a new bar in the chart. This approach proves to be more effective than say, adding a new bar for every single message, as the host can see a general trend in group mood as a whole compared to individuals that may be inputting anomalous feedback. We also considered the possibility that a multitude of messages may be sent in one interval, while only one message is sent in the next, resulting in different meanings behind the bars for these two intervals in the graph. However, this approach is still the most suitable for our project and ensures that excessive data is not flung at the host, a continuous scale is kept to allow the host to track feedback throughout the whole event, and the graph is updated frequently enough so as to not dissolve individual messages completely.

6.3 Sentiment analysis

Sentiment analysis was a key component of this project even if it was handled by a library. Finding a library was the key problem but the Stanford CoreNLP [1] library provides an incredibly powerful toolkit for natural language processing and quickly became clear to be the best solution within Java. The system as a whole uses pipelines which take in input text, process it using what it calls NLP annotators before outputting a final set of ”annotations” of the inputted text, one of which is sentiment. We used a couple tutorials ([4]) to make use of the sentiment annotator to calculate a mood class for each piece of text which was then mapped to a numerical mood value.
The sentiment analysis is performed by a powerful deep learning model which was trained with an extensive data set known as the Stanford Sentiment Treebank, which is a collection of around 11,000 movie reviews, but is constantly being improved with new data. This style of approach has been proven to be far more effective than many rules based approaches and is far more effective that anything we would've been able to implement ourselves with the given cost and time constraints of this project.
Ideally we would have gathered our own data set of feedback given to events and used it to train our own sentiment annotator using CoreNLP. However this was totally unfeasible for the scope of

the project, as a data set of a similar size would be required. Despite this, the fact the annotator was trained on movie reviews was actually a positive, as the language used in movie reviews would not be too dissimilar from that of feedback from attendees at events. The model is not perfect but provides the best estimation of the mood of text that we could find.

6.4 Back end evaluation

As far as the back end of the system is concerned the majority of requirements (all those that were carried through to the project's completion) were met with only a few SHOULD and all the COULD requirements being dropped. This is demonstrated by our unit testing from earlier. However the system does not function as a coherent whole with a few tests within main being the only attempt to view how components function together. The tests have also not been performed with many extraneous or abnormal values and there are likely unexplored edge cases. In general care has been taken to prevent the appearance of null values or other inputs that may cause issues with the code. Certain potential errors have not been tackled however simply due to being unlikely to have any effect within a relatively small prototype such as not accounting for running out of event/guest IDs which could lead to an infinite loop and long wait times as the number of remaining IDs decreases. The sharing of the pool of random IDs between guests and events is also a bad decision that would be worth rectifying in future.
The navigation of the User class hierarchy as users move throughout the system has also not been properly explored and tested due to the lack of connecting up the whole system and similarly the regular updating and calculation of overall mood is not performed though the overall mood at any point can be calculated with a simple function call.
While our system did not use a database to store the info required we did put some thought into effectively storing the data so it could be retrieved and search quickly as needed. Initially the majority of the system used Arraylists to store events, users and feedback. However, we decided to change the majority of the data structures to Hashmaps instead for significantly better fetching of data. Because we didn't have the foresight to use HashMaps from the beginning, a lot of time was spent in editing code that was written when the data structures were still ArrayLists, and we found ourselves finding methods, that we had missed in our updating of the code, that were producing bugs because the calls to the two data structures are very different. In hindsight, we should have decided on data structures used in the design stage of the project, which would have stopped anything like this happening.

6.5 Full system evaluation

In the end we did manage to achieve a working navigate-able system that could be run via localhost. Some shortcuts were taken during the implementation of this and the base back end code which the system interfaces with is not the newest version of the back end code. Additionally not all features that have been implemented were implemented as they were intended to be/would be in a completed system. However, these shortcuts were ultimately worthwhile to provide at least a small example system that could be shown off to give an idea of how a completed system would look.
Unfortunately many of the system requirements were not implemented in this full system due to the time constraints and ultimately this prototype does not meet the customers needs in its current state.
The system also lacked long term storage however did make use of an init page to simulate this by

loading in sample data to the system. This was effective to visualise how the system would work with data already within it.

6.6 Potential additions

The obvious additions that would be made to our system next after fully implementing the back end functionality with the front end would be to complete the requirements that were removed in the development process. Our system should also be changed to use a proper dedicated database stored on some form of server in order to maintain the state of the system between uses and provide long term storage. This avenue of a system was not explored as part of the project as it was deemed too difficult and we had a lack of experience in the area. We also felt it unnecessary for a prototype system.
After this the system would finally be in a releasable state and could undergo user acceptance testing before being released. The system could then be deployed and combined with an existing more robust account/security system and even some timetabling/event scheduling system.
As well as maintaining the system, there are still more additional features that were considered during the planning stage of the project that could be reconsidered and added to the system such as allowing multiple hosts for easier monitoring of feedback quickly without distracting from the event.

7 Evaluation of our development

As a whole this project taught us a lot about making larger software projects and working as a team, allowing us to critically look back on the experience and learn from issues that arose for future projects.

7.1 The original plan

As a whole it is clear our original plan for both our product and the completion of it was incredibly unrealistic given the abundance of other responsibilities we all had and the effort that could be put towards this project. I believe we also underestimated the difficulty of the task and the skill level and knowledge required to complete certain tasks both of which we lacked in many cases. Particular difficulty was had, as was mentioned before, with the actual construction of the system as a whole and the connection of components which was done via REST API. The effort required to achieve this was severely underestimated due to the lack of prior experience throughout the entire team. While in general the days required for different components of the code were not far off what was actually required, the building of the system as a whole was barely accounted for. Additionally those timings assumed the system was going to be worked on for a few hours every day which was an unrealistic standard of engagement for an entire 5 week project. The plan also didn't account for some of the initial setup which everything else depended on such as setting up skeleton code and the GitHub repository which took some time to complete including the time spent familiarising ourselves with git and effectively using it throughout the project.
The plan was always believed to be unfeasible but this philosophy likely did not benefit our work as the plan was not incredibly well updated after we began to fall behind and once the first couple of deadlines were missed, motivation to hit any new deadlines quickly began to disappear.

7.2 The methodology

While we did not manage to stick to our planned methodology, the idea of an incremental software development cycle was likely a smart decision as it allowed us to identify the bottleneck caused by connecting up the system earlier than we may have otherwise. If we had not aimed to have a connected system within the first 2 weeks this issue may have arisen much later in the development process leaving less time to work on it. Admittedly more focus could've been given to this bottleneck earlier in the process, but even once it was identified we were naive as to exactly how much work would be required. A lot of time was spent researching different options rather than picking a potential solution and using it.
The incremental software approach also allowed us to be more flexible with the system and made it easier to remove requirements and functionality once it became evident this was required for the projects completion on time. In particular we also benefitted from visualising the dependencies of the system and carefully designing our initial system such that adding additional SHOULD and COULD components later in the development cycle would require minimal editing to the base system. While no changes were made to the requirements, our relatively flexible plan based approach would've made it easier to adjust to such a change than with a more heavily structured plan based approach.
As a whole having a well structured plan and design was very advantageous, even if elements of it weren't always followed and gaps were found in the design that needed to be addressed during development. The class diagrams were particularly useful in constructing skeleton code for the back end of the project and we benefitted greatly from discussing exactly how the mood would be calculated at any one time period in the design. While we naturally struggled to implement this in the final prototype due to issues with connecting up the system, the discussion of the UI and how the system as a whole would be navigated and function together was beneficial for any similar future projects.

7.3 Working Online

A big factor in the development of this system was the fact that we were working online. This obviously had a detrimental effect on communication with both each other and the department in comparison to being able to work together in the same room and talk face to face with our tutor. It became much harder to hold team members accountable, and meetings in general suffered from not being face to face. While there were attempts to get everyone on camera during such meetings to provide a better feeling of connectivity this was not really achieved. Additionally the absence of team members at both meetings and overall could've been easier to prevent with in person contact. I also would argue that working together on code in the same room would have saved a lot of time, both in faster communications and lack of doubling up work, which would likely have meant the overall product would be more developed by delivery, as well as boosting morale. We attempted to tackle this issue by having a voice chat which team members would occasionally drop into while working in order to communicate with other team members while doing so. Viewing exactly what others were working on and helping debug each others code was however much harder in this situation than it would've been in person. Additionally I believe team members asked for help with simple problems much less than they otherwise would have. Detecting problems was made much harder by this as elements of the project were not always as close to being completed as communication through messages came to suggest.

7.4 Communication

Communication in general was handled well despite the difficulties of online work. During the development process twice-weekly meetings were held which were mostly well attended by the 4 active team members. Additional meetings were also held in the final week to help complete the process. Meetings were recorded with regular meeting logs which were created before the meeting to give members an idea of what may be discussed and updated after to give the results of the meeting. Some of these logs did not include a lot but the discussions had in the meetings proved very important. More care could've been taken to get an idea of exactly how much progress had been made at each stage and to motivate team members but progress was very slow for the first few weeks and very little was getting done despite some smaller deadlines being given, if not incredibly clearly presented.
Communication of the plan and info discussed in these meetings could also have been improved but engagement with reminders and larger messages of important information was limited and much more progress seemed to be made by individually contacting and working with different team members. Regardless regular communication was maintained with all active team members to get an idea of what they were doing and to give an idea of what they could do next, including maintaining regular contact with the missing team member to provide updates and opportunities to join in again. A trello page was also presented to help track progress and also to store different files and the meeting logs but this also experienced poor engagement so progress had to be tracked more directly through asking.
The GitHub was useful to track progress especially on the back end with regular commits giving an idea of changes being made though some of these lacked detail occasionally. In general while team members were encouraged to give regular updates of progress this was perhaps unrealistic to expect as it would've taken some time off the work itself to give these regular updates. However especially with the main bottleneck of the front end to back end connection it was not always clear what progress had been made as this code was not shared until very late in the process.
Communication with our project supervisor could also have been improved. During the first portion of the project regular emails were sent to update our project supervisor. Additionally we had a meeting directly with the supervisor and sent off draft documents for early feedback.
However during the development process contact with our supervisor was much more limited with only 1 more in person meeting being held during the time and much less frequent email updates. We also did not manage to submit any drafts for feedback. This was partly due to a miscommunication of responsibilities between our project manager and business analyst and also just the feeling we had little of importance to mention and that we had to work out certain things for ourselves.

7.5 Skeleton code

Creating skeleton code was a very good decision to streamline the development process on the back end. While it was not considered in our initial plan and thus caused some delay, as a whole having the skeleton code likely sped up future development on the back end making it easy to just fill in different methods. The class system as a whole proved to meet the needs it was designed to and while a few additional methods were added the majority of methods used by these classes had been considered before they were implemented. The only major change to the system included the addition of the Guest class but otherwise the system proved to be pretty robust. Additionally the general functioning and data stored by the system and the moving between classes within it, which was not very well considered by the initial class diagrams, did slow progress but came together nicely with the other components and proved useful to help test the different components of the system without a connected front end.

7.6 REST API bottleneck

As mentioned before, during the implementation phase, we experienced a bit of a bottleneck when it came to continuing towards a working solution. There was a need to connect the front end of the app to the back end logic, and while this remained a problem, we were unable to develop any further. We were quite slow in responding to this particular issue at first, as there was little communication in the periods between meetings. Communication would still occur inside of the front end and back end teams, however this meant that the issue went unnoticed as there was not any need at that point to discuss linking the two ends. While this was eventually resolved by discussion with our supervisor, where we talked about the idea of using a REST API to solve the issue, there was still plenty of work to do after this discussion and more time and resources in the team could've been allocated towards this. Once we knew our approach, we began working towards implementation immediately. Since it was different to the original plan, however, we had not had time to practice with the API much, which caused the implementation of the REST API to take a large portion of our time. This bottleneck did hamper our ability to achieve many of our COULD requirements, which should be used as a lesson on how to prepare for future projects.

7.7 Management

Both the project manager and business analyst roles were treated quite loosely with the business analyst mostly just being responsible for sending out invites for meetings. The project manager took care of meeting logs and the general management of the whole team which mostly consisted of being in regular contact with them and additionally assigning tasks to each of them. The project manager could have been more proactive with setting deadlines for specific tasks to speed up development even if those provided weren't always met. In general, care was taken to keep up to date with everyone's situation and what they were working on, at least after the first couple of weeks of development. The project manager did also help make the final decisions a lot of the time, especially as the deadline drew close and certain things needed to get done.

7.8 Changes to the plan

Our plan was changed considerably from quite early on in the process simply due to the unfeasible initial plan and the lack of consideration of the fact that worth ethic would slowly increase throughout the 5 weeks. Once the plan was lost, it was almost abandoned altogether with only smaller deadlines being communicated via messages to the group. Whether these messages were seen or read was not always clear. A much better job could've been done of adjusting the plan by taking the initiative to remove certain potential components from the finished product at an earlier stage rather than at the last possible moment as many were. The removal of such features was considered in earlier meetings and if it had gone through then may have helped make the project seem less daunting and more manageable and hence encouraged completion of what was left.
The eventual removal of originally planned elements was ultimately a good decision though and helped us to have a semi working prototype by the project deadline. More care should've been taken to plan exactly when certain elements needed to be finished, after initial optimistic

deadlines were missed. While formal adjustments to the plan were not used, we adjusted well to changes due to work being more than expected and due to other commitments and various other smaller issues we faced.

7.9 Report and Presentation

The report was originally planned to be developed alongside the main system as components were developed. Due to the more rocky development this was ultimately scrapped as an idea as it would've been hard to write a report while we were unsure of exactly how much of a product we would have at the end of development. Indeed some parts of the report were still written before this was known and later had to be edited to match the actual product produced. This issue was believed to be unlikely to cause a problem when the writing of the report was first considered due to how most projected newer components were unlikely to effect earlier ones significantly. But due to the lack of bringing the system together until near the end of development, things did have to be rewritten.
The presentation also suffered from this as it could not be properly produced until an idea of exactly what would be demoed was produced. The initial goal to have only minor parts of the product to do coming into the final weeks of development was not met and so this meant the time to produce the demo was limited. As with multiple other elements our progress was hampered here by a lack of experience within the group in recording and editing presentations. The presentations were created by recording using OBS [5] and using Shotcut to edit [6]. Both pieces of software were unfamiliar to us but turned out to be simple and intuitive to use.

8 Final thoughts

Overall, while we did not produce the project we originally intended to, we were able to learn a lot from the process and still produce a simple prototype of what we originally planned. We faced a lot of challenges due to a lack of members, experience and knowledge but have now filled some of these gaps and will be able to work far better in future projects.

9 References

[1] *Overview - corenlp*, `https://stanfordnlp.github.io/CoreNLP/`, Accessed: 13/03/2021.

[2] *Spring*, `//https://spring.io/`, Accessed: 14/03/2021.

[3] *Backbone.js*, `https://backbonejs.org/`, Accessed: 14/03/2021.

[4] *Day 20: Stanford corenlp – performing sentiment analysis of twitter using java*, `https://www.openshift.com/blog/day-20-stanford-corenlp-performing-sentiment-analysis-of-twitter-using-java`, Accessed: 13/03/2021.

[5] *Open broadcaster software — obs*, `https://obsproject.com/`, Accessed: 14/03/2021.

[6] *Shotcut - download*, `https://shotcut.org/download/`, Accessed: 14/03/2021.