Final Report

James Xu, Naveed Iqbal, Oliver Smith, Maxim Gorbach and Kartheyan Sivalingam

March 2021

1 Introduction

The purpose of this project is to develop a system for sending live feedback during an event/meeting. The host of the event is able to see the feedback and is also provided with an estimate of the current 'mood' of the attendees at their event. This document provides a description and evaluation of the developed system.

2 System Overview

2.1 Accounts

A user is able to create an account using a display name, an email and a password with the interface shown in fig.5. Having the user's email is useful as it would allow extra security features to be implemented in the future, such as having a secure password reset feature by verifying the user through their email. Once an account is created, the user can log in by using their email and password which will show the user all of their created and joined events. The email was chosen as the login method as it is more memorable and this allows multiple users to have the same display name, as users may share a full name, and being able to have your full name as the display name creates a more professional environment. Using your full name is not enforced in any way, but it is easier to do so because the display names aren't unique.

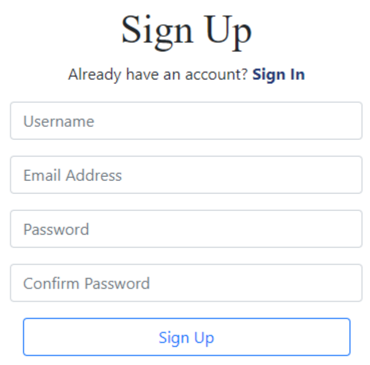


Figure 1: The account creation UI

2.2 Events

A user is able to create an event through the interface shown in fig.2 and giving the event a name. The user can choose to enable anonymous feedback for this event.

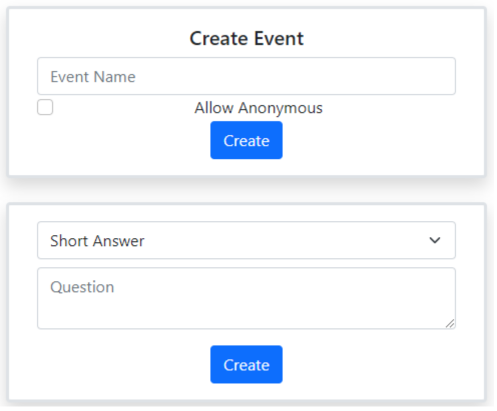
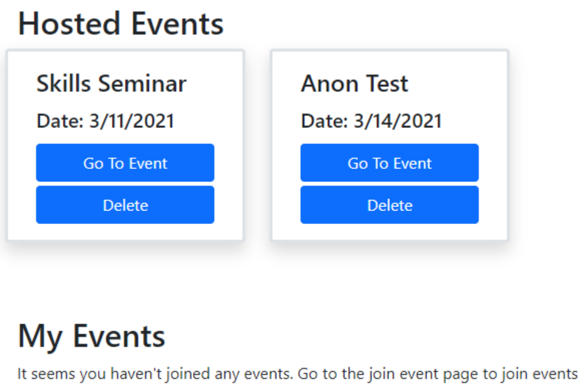


Figure 2: The event creation UI

The user is able to view all of their created and joined events once they have signed in, on the page shown in fig.3



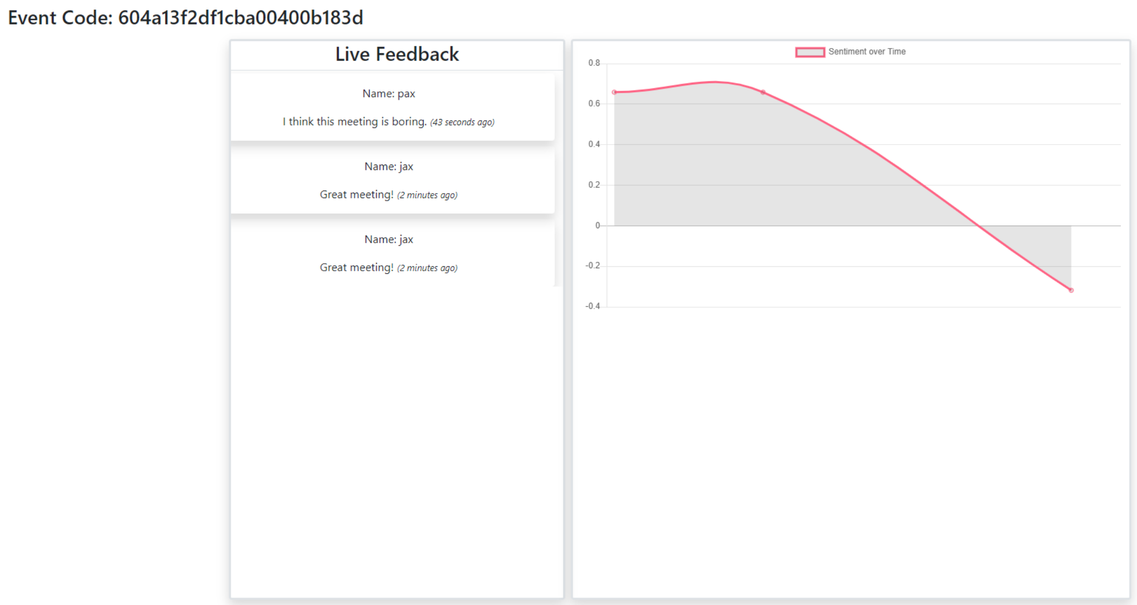
It seems you haven't joined any events. Go to the join event page to join events

Figure 3: View of users events

If the host has made a mistake when creating their event, they can easily delete the event, meaning they are able to recover from a mistake. An attendee is able to join an event by entering the unique event code which can be found by clicking on the relevant event. Once they have joined they will be able to submit feedback to the event through the same page.

2.3 Feedback and Analytics

By clicking on an event, a host is able to see all feedback as it is being submitted to this event live. The host's view is shown in fig.4. If the host has enabled anonymous feedback for the event, the attendees will have the option to submit feedback anonymously. The name on the host's view will appear as 'anonymous' if this is the case.


Event Code: 604a13f2df1cba00400b183d

Live Feedback

Name: pax

I think this meeting is boring. *(43 seconds ago)*

Name: jax

Great meeting! *(2 minutes ago)*

Name: jax

Great meeting! *(2 minutes ago)*

Figure 4: View of users events

When a user submits feedback a 'sentiment' value is automatically calculated for it based on the content of the feedback. It does this by using an existing database of words scored by human judges on how positive or negative they are. Any feedback is split into these individual words and each is assigned its sentiment value. As well as this, the word order and presence of words that 'intensify' a mood such as 'extremely' or 'slightly' are considered. This value ranges between -1 and 1. The sign on the sentiment (positive/negative) represents the polarity of the feedback; whether it was positive or negative feedback respectively, and the magnitude of the value shows the intensity of the feedback. The change in the sentiment throughout the event is displayed to the user as a graph, which is useful for seeing how the mood changed throughout the event to identify where issues occurred.

3 Requirements Changes

Due to the teams lack of experience with developing web application, a lot of time was spent learning how to use the necessary tools. The team also had other important commitments during the development process, so in the interest of the teams mental health, some of the original requirements were dropped. The aim was to reduce the pressure on the team and reduce the workload, geared towards reducing the amount of mistakes made during implementation.

3.1 Dropped Requirements

R1.1.2

This requirement was partially implemented, but there was insufficient time to complete it, so the requirement was dropped which helped to focus on completing more important features.

R1.5

Having multiple hosts is unlikely to be a useful feature; if the event is hosted in person the feedback could be displayed on a single device so that all hosts could see it. If the event was hosted online it would make sense for one of the hosts to monitor the feedback, having each host see it wouldn't be needed. Therefore the feature was dropped so that more time could be spent on the others.

R2.4

Tracking attendance wasn't the primary goal of the system and implementing this feature would require significant extra work, and there already exists other software to track attendance, such as BambooHR. Hence this feature is unnecessary and the time is better spent on improving the other aspects of the system.

R2.5

The primary use for this is for tracking attendance, since that requirement had already been dropped (R2.4), it makes sense to also drop this one.

R2.6

This is a quality of life feature that is unlikely to be used, as the only way to obtain an event code is for the host to share it with you, and guessing the event code is difficult as it is long. This means that malicious users joining an event is unlikely, so this feature was cut in order to spend time on other parts of the system.

R3.2

This requirement was partially achieved, but there was a lack of time to complete it, so the requirement was dropped which helped to focus on completing more important features.

R3.3

The initial plan to achieve this requirement was ambiguous and this was a complicated feature to implement, so it was decided that the feature would be cut to save the time needed to plan and implement this feature.

R3.4

Users other than the host aren't able to see the feedback for an event, therefore removing feedback is unnecessary since the host is the only one who can view it.

R4.2

The requirement for tracking individuals moods and attendance was dropped, since this feature depends on that, it was also dropped.

R6.2

The overall mood for an event isn't as useful or detailed as the graph showing change of mood over time. The graph was implemented instead of the overall mood, because it is a more useful analytic for the host.

R6.3.2

The requirement for projects (1.1.2) was dropped, as this requirements is dependant on that one it was also dropped.

R6.3.3

This requirement makes sense for tracking individual team-members moods throughout a project. Since the requirement for projects was dropped (1.1.2), spending time on fulfilling this requirement for events was unnecessary as it wouldn't be useful to the host.

R6.4

Dropped due to the dependence on the dropped requirement 3.2.

R6.5

Dropped due to the dependence on the dropped requirement 2.4.

4 Discussion of Development

The tight deadlines along with our intended scale for our project, meant our development process had to be very well structured to avoid wasting time and resources. In this section, we will go over our development process from start to finish, including discussions on the positive and negative aspects.

4.1 Development Tools

4.1.1 Version Control - Git/GitHub

As common with most software projects we used version control software, in this case we used Git and GitHub, this comes with many advantages allowing us to rollback and keep track of changes made. During our development, we rarely had to rollback the entire codebase as we tended to take a continuous integration approach where we would push small and incremental changes to the repository rather than the approach where developers would push large feature releases which had a higher chance of breaking the current build. To most effectively utilise version control software it was important to follow common/standard practice such as meaningful commit messages. As for most of us this was our fist time working on a large team based project, writing meaningful commit messages was something that we all improved on as time progressed. Something that tends to go well with the continuous integration approach we took was continuous testing, which could also be automated with the CI/CD (Continuous Integration/Continuous Deployment) tools provided by GitHub. Although this would have been useful we felt due to our decision to take a non-agile approach to development, it would be unnecessary to invest the time into this continuous testing.

4.1.2 Virtualization/Containerization - Docker

A relatively recently popular development tool, virtualization allows multiple developers working on different platforms to share a unified and single development environment. We decided to use Docker. Virtualization comes with two main advantages, the first being we don't have to waste time setting up different development environments, such as installing packages, setting up databases, etc. This also ensured we are using consistent versions of packages and frameworks, this meant we were able to not waste time and quickly start working on developing our product. Another advantage of virtualization is we won't encounter platform/environment specific errors. In summary, virtualization overall saved us a lot of time in the development of our project, by reducing the overall set up time for our developers as well as reducing our need to debug and fix environment-specific errors

4.1.3 Bootstrap

Bootstrap is a framework to help developers design websites faster and easier. It includes HTML and CSS based design templates and gives support for JavaScript plugins. We decided to use it for user interface development because it is fully customisable and provides responsive layouts. Customisation allows us to build an interface that fits clients' needs as we can modify most components to perform required tasks. Support for responsive layouts ensures that we can build the system that users can use on any devices. So, we are certain that Bootstrap will help us work efficiently.

4.2 Development Progression

4.2.1 Structure of codebase

MVC Pattern
The Model-View-Controller (MVC) is a sotware design pattern where the application is split into 3 distinct areas: the Models; representing our data entities, the Views; representing our front-end and user facing code and the Controllers; the backend which can also be seen as the connection between the data models and

the views. We had decided early on that we would be adopting this pattern to develop our web application, this allowed us to keep a consistent file structure throughout the development of our project, where code remained separated into models, views and controllers.

- Models
  - User.js
  - Event.js
  - Template.js
  - Feedback.js
  - Project.js
- Views
  - css
  - js
  - ...
- Controllers
  - user-ctrl.js
  - event-ctrl.js
  - template-ctrl.js
  - feedback-ctrl.js
  - project-ctrl.js

Another advantage of this consistency and this separation of ideas allowed members of our team who were working on different areas of the project to work on their areas simultaneously without being hampered or having to wait to until another unrelated section was finished. For example, the front-end developers were able to develop without waiting for development of the API, or backend developers can develop the models and controllers simultaneously. This was especially useful in our situation due to our relatively small development team and tight deadlines, where for the majority of the time everyone was working on distinct areas and without this structure we would be forced to wait on other areas of codes to be finished. As much as distinction and separation of these areas is useful, it is impossible to develop an entire project without having some cross over, this can lead to wastage of time as developers who worked on different areas have to read large areas of code to get small bits on information, for example when our front end developers wanted to see how certain API endpoints would format data, they knew where they can find it and due to the highly modular nature of our codebase, they don't have to waste time reading the entire codebase.

4.2.2 Planning the structure of the API

The first steps we took after setting up our file structure and configuration files (Dockerfiles, database files, etc), was to create our API endpoints and plan the structure of the returned JSON data. In hindsight, this is something that could have been done within our initial planning and design stage, and although we were able to quickly accomplish this initial plan. As the controllers and the functionality of these API endpoints were implemented, changes were made to the structure of the returned JSON data this caused minor disruptions as these changes also had to be reflected in the frontend. This is an area of improvement, where more thorough planning of the API could have avoided problems and saved us time.

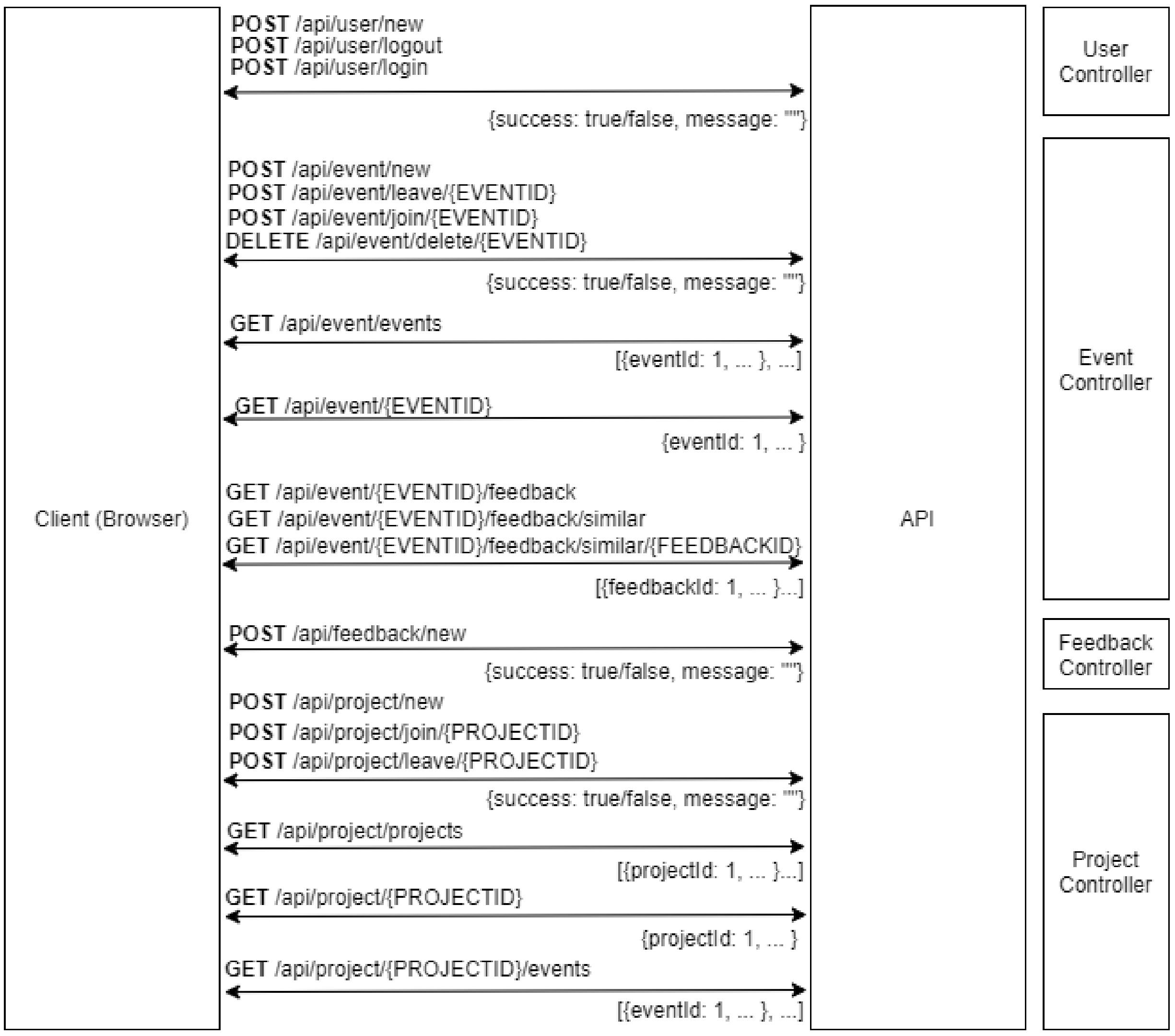

Client (Browser)
POST /api/user/new
POST /api/user/logout
POST /api/user/login
{success: true/false, message: ""}
POST /api/event/new
POST /api/event/leave/{EVENTID}
POST /api/event/join/{EVENTID}
DELETE /api/event/delete/{EVENTID}
{success: true/false, message: ""}
GET /api/event/events
[{eventId: 1, ... }, ...]
GET /api/event/{EVENTID}
{eventId: 1, ... }
GET /api/event/{EVENTID}/feedback
GET /api/event/{EVENTID}/feedback/similar
GET /api/event/{EVENTID}/feedback/similar/{FEEDBACKID}
[{feedbackId: 1, ... }...]
POST /api/feedback/new
{success: true/false, message: ""}
POST /api/project/new
POST /api/project/join/{PROJECTID}
POST /api/project/leave/{PROJECTID}
{success: true/false, message: ""}
GET /api/project/projects
[{projectId: 1, ... }...]
GET /api/project/{PROJECTID}
{projectId: 1, ... }
GET /api/project/{PROJECTID}/events
[{eventId: 1, ... }, ...]
API
User Controller
Event Controller
Feedback Controller
Project Controller

Figure 5: API diagram

As the backend developers were more familiar with the API, we decided it would be best if the frontend JS AJAX calls were all collated into a single file 'api.js' which also take in success and fail callbacks, allowing the front end developers to focus solely on implementing the actual user interface. This ended up saving a lot of times because as previously mentioned the API didn't have a detailed plan, so as development progressed many changes were made to the API, having all the API calls in one file, minimised disruption to front end development.

4.2.3 Developing the models

As discussed within the planning document, developing the models was an important first step in the development of our project. The detailed plan we had created for these models was useful, as this ensured no major changes were made to these models during development. We did however decide not to implement all the models all at once but rather focus initially on the core models: User, Event and Feedback. Once these core models were developed, we worked on implementing core features and controllers rather than the other models (Projects and Templates). Eventually, as the controllers were implemented we went back to the other models, however due to focusing on other features we decided to skip the Template system, and although in the backend we did end up implementing the functionality of Projects it was never completed in the front end (the opposite ended up being true for Templates; where it was implemented in the front end but not the backend).

4.2.4 Developing the user system

The user system is an important system in the development of the application, however one we were wary of not focusing to much development time on. The decision to not use a role-based user system in which hosts and attendees are distinct users ended up significantly simplifying what needed to be implemented. As we had already implemented the models, the backend developers can focus on implementing the controllers and the frontend developers can work on the user interface (views). Unlike the other systems however, implementation of the backend of the user system was not solely based on the controllers. Although, code related to the authentication (logging a user on) was part of the controller, the majority of the user system code was about authorization (ensuring user has permission to access a specific resource), this involved writing custom authentication middleware and token generation/storage.

Authentication

The authentication procedure we followed is fairly standard, and although this was a MVP we felt it was important to follow basic security practices as such all passwords stored in the database are hashed and never stored in plain text. We used bcrypt which is standard for the majority of JS applications, and it offers a great balance between efficiency and security. As with most authentication protocols, it is important to use a protected salt with your hashing function. In our case, we felt it sufficient to hardcode this salt within our code, however in any production environment this salt would need to be stored in either a separate configuration file or environment variable.

Authorization

For authorization we also followed a fairly standard approach (sessions), we used a cookie based token which is generated by the server and sent by the client along with every request. This token is then validated by the server by ensuring it is in the database. This token will have an automatic expiry date, when it is removed from the database and the user will have authenticate themselves again. Although, this method of authorization is fairly popular it does not scale due to the need to store the tokens for every session in a database and have a database lookup for every request. The alternative method which we felt was not necessary for our MVP, is to use tokens signed with a secret which will be sent along with every request, however this signed token does not have to be stored and looked up to authorize the user. ExpressJS supports the use of middleware so to simplify the process of protecting API endpoints that required authorization, an authentication middleware was created this was then added to endpoints of the API which only authorised users can access.

User Interface

The user interface for the login and sign up pages were not prioritised and started off very simple and was improved gradually as more of the important features were completed.

4.2.5 Developing the events system

The events and feedback system formed the core of the problem as such it was prioritised in both front end and backend development. Development of the backend of the events system was just about implementing the controller, as the models had already been implemented.

Implementing the controllers

As we had already developed the models for events, the backend developers had to develop the controllers. This involved fairly simple database manipulation, MongoDB's powerful ORM abstraction made interacting with the database fairly simple. Most functions from creating an event to deleting an event were fairly standard and followed mostly the same structure.

Identifying similar feedback within event

In order to identify similar feedback, the StringSimilarity and SentenceSimilarity libraries were used. The StringSimilarity library is based on Dice's coefficient [2] which is used to gauge the similarity between two statistical samples. The SentenceSimilarity library calculates 4 similarity scores: the number of exactly matching words; the 'score' of the words that are matched partially or exactly, for which the StringSimilarity library is used; a score based on the order of the words in the two sentences; and a 'size' score which compares the sizes of the two sentences. In order to get a standard score in-between 0 and 1, the exact, order and size scores are all considered. For a selected feedback, all other feedback in the same event is compared to it. Any feedback with a score above a set tolerance is taken as 'similar' feedback. This tolerance was set to 0.4 after experimenting with different values, as a lower tolerance leads to feedback that isn't relevant, and a higher tolerance results in too few matches. Although this feature was implemented in the back-end, it wasn't completed in the front-end, so it isn't present in the final system.

User Interface

The user interface for the events system similar to the interface for the user system started off simple and was improved after other features were implemented. The UI involved more complicated front-end JS when compared to the user system, such as dynamically populating the events elements based on the events returned by the API. We made the early decision not to use front end frameworks such as Angular or ReactJS as we felt the learning curve would be too great, so we instead decided to use a more lightweight JS library for DOM manipulation - jQuery. These front-end frameworks would have helped us in developing this section of the code, as template engines typically support built in directives/functionality to reduce hard coding HTML within JavaScript which is how we achieved the events display. We did feel however feel overall we made the right decision in not using a front-end framework, as sections like this weren't common in our application. The only other section that required this was in the live feedback section, where again the feedback HTML had to be hardcoded within JavaScript.

4.2.6 Developing the feedback system

The feedback system was one of the more complex subsystems that took up the majority of our time.

Calculating mood value

For sentiment analysis we used a third party JavaScript library called vaderSentiment, it uses an efficient rule based model for sentiment analysis. It takes into account things like negations (e.g. "not good"), contractions (e.g. "wasn't very good") and even takes into account punctuation and whether your writing in all caps; things like exclamations to increase the positive or negative value or writing in all caps to signal emphasis on certain words. It even supports inferring sentiment from emojis (useful as the system was designed to be used on mobile). Rule based models of sentiment analysis tend to work well for short texts, which is what we expect the majority of live feedback to be. We felt using ML (machine learning) models or neural nets overkill to analyse sentiment for short texts and will have an impact on the real time nature of hosts receiving feedback. We also had a choice of calculating the mood value on the client or the server, we decided to reduce unnecessary additional strain on the server which could add additional latency for hosts receiving feedback, to just calculate the mood value on the client and then send this along with the message to the server. This however does add the additional risk of malicious users being able to modify the JS on the client and send whatever mood value they deem fit.

Live Feedback

In developing the live feedback functionality, we had two choices on how this could be implemented in the backend. The better; but more time consuming option, was to create a separate WebSocket server, which the clients (browsers) connect to and then this WebSocket server would then subscribe to MongoDB's change streams and alert the client of any changes in the relevant tables. The second approach; the one we ultimately decided to take was a simple polling approach where the client polls (via HTTP requests) the API at set intervals and the server will then respond with the feedback. Choosing the rate at which the client polls the server is a balance between efficiency and performance (how "real-time" the feedback is). We felt ensuring the feedback is real time is more important (consider feedback such as "This example is bad" or "I didn't understand the last thing you said"), that is why we decided on an interval of 2.5s which is fairly regular. The system supports anonymous feedback, when the host creates an event he has the ability to support anonymous events this choice will be stored as part of the Event model. When the user then submits feedback, if the host has allowed it the user has the ability to send his feedback anonymously, the browser will then send this choice along with the message to the server, who in turn doesn't store any user related data within the Feedback entry.

User Interface

To dynamically populate the feedback messages returned from the server, we had to hardcode HTML within JavaScript, which isn't inherently negative from a performance aspect but does impact how readable our code is or how easily we are able to make changes to the design. As discussed within the user interface section of the events system, the use of front-end frameworks could have simplified this section. Another

complex user interface task within the feedback system was presenting the host with a graph that shows a history of the mood values over time, this is discussed in detail in the next section.
Charts
We decided to present the live sentiment of attendees to hosts via a live updating line graph. We used the popular charting library 'charts.js', it was one of the more easier to use charting libraries with great documentation however as we started using it we started to realise there is not much room for customisation. For example, we wanted to have the x-axis labels update as data was coming in (i.e. for each feedback entry, the x-axis label would be 2s ago or 1 months ago) and although it supported live updating the data itself, the labels had to be fixed so we just decided to remove the labels of the x-axis all together. Although, we initially had plans to generate a variety of different graphs presenting the host with lots of real-time useful information, we ended up only having time to support one of these graphs.

4.2.7 Developing the templates system

The templates system was something that was in development but was not fully completed, the user interface of this section was however fully completed and functional. The backend including implementation of the model was not finished.
User Interface
The user interface for the templates system was based of Google Forms, however unlike Google Forms which supports a variety of different question types, we support two "Short Answer" and "Multiple Choice". We tried finding third party libraries that may support this template generation, but as this was not something we were able to find. So we instead had to write the front-end JS to support this, using jQuery helped in this task as it mostly involved DOM manipulation, such as appending and removing elements based on the user input.

4.2.8 Developing the projects system

The projects system similar to the templates system was never completed, unlike the templates system however the backend was completed but the user interface was not.
Implementing the models and controller
The projects model was not implemented at the start like other models, this is one of the reasons development of this system was slower than the the other systems. The backend implementation of this system was simple, as it only involved mostly just simple database manipulation similar to the implementation of the controllers of events system.

4.2.9 Testing

The testing included two types of tests: endpoint testing and end-to-end testing. Endpoint testing allowed us to test each possible HTTP request and validate its response from the backend. End-to-end testing involved using browser automation to test each feature of our web application, to validate the requirements of our software. We automated our testing by writing test scripts using the libraries Jest, Supertest, and Selenium. Jest allows for multiple tests from multiple files to be run asynchronously which made running the test scripts very easy and quick. Supertest was used to send HTTP requests to the backend, and Selenium was used for browser automation.

The main challenges were getting to grips with the libraries, but after that the tests were relatively simple to write. However, there were also plans to write unit tests for the frontend, which turned out to be a bit more difficult due to the inline scripts used in our HTML files. Due to this, as well as time constraints, these plans were abandoned. Additionally, we planned to test our system for both Chrome and Safari, but we only tested for Chrome as we were running out of time. Also, tests were only written after the majority of development was completed, which meant that they were not available for our devs to use during development to help with debugging. These issues could have mostly been fixed by starting to write tests much earlier.

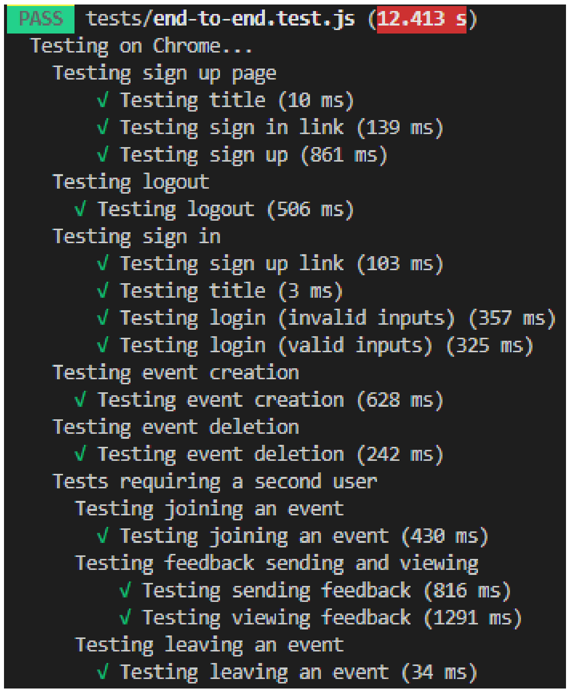
4.3 Evaluation

Overall, we think the development of the product was successful and went smoothly. As an area of improvement we felt more communication between front-end and back-end developers could have resulted in more fully completed features. For example, if we consider the projects and templates system, where projects was fully implemented in the backend and templates was fully completed in the frontend; more communication between the developers could have resulted in everyone focusing on one of these two which might have resulted in at least one of these tow being completed. Another area of improvement was the lack of initial planning of our API, as this was an important part which both front-end and backend development relied upon and ended up resulting in a lot of wasted time as code was changed back and forth as development progressed. Our biggest area of success would be our highly structured and modular approach in development, by adopting the MVC pattern we were given an implicit step-by-step development plan.

5 Product Evaluation

5.1 Testing and Validation

In order to validate our system and show that we have met our requirements, we ran browser automation tests to make sure all the features work. Here is the test log:

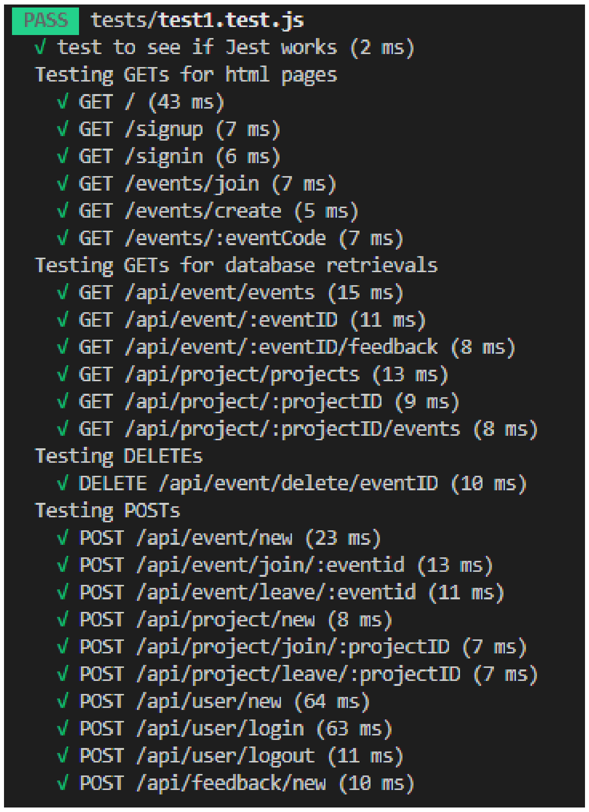


```
PASS  tests/end-to-end.test.js (12.413 s)
  Testing on Chrome...
    Testing sign up page
        √ Testing title (10 ms)
        √ Testing sign in link (139 ms)
        √ Testing sign up (861 ms)
    Testing logout
      √ Testing logout (506 ms)
    Testing sign in
        √ Testing sign up link (103 ms)
        √ Testing title (3 ms)
        √ Testing login (invalid inputs) (357 ms)
        √ Testing login (valid inputs) (325 ms)
    Testing event creation
      √ Testing event creation (628 ms)
    Testing event deletion
      √ Testing event deletion (242 ms)
    Tests requiring a second user
      Testing joining an event
        √ Testing joining an event (430 ms)
      Testing feedback sending and viewing
          √ Testing sending feedback (816 ms)
          √ Testing viewing feedback (1291 ms)
      Testing leaving an event
        √ Testing leaving an event (34 ms)
```

Figure 6: Selenium Browser Automation Test Log

All features that were implemented end to end were tested. All tests completed in less than a second except the test for viewing feedback. But this is because the page for viewing live feedback polls for updates every 2.5s, so 1291ms is acceptable. This means that requirement R8.1 is met, as all button inputs receive a response in ¡ 1s. R5.1 is also met, because feedback is received in less than 1 minute, though it was only tested locally. Other requirements are validated by their respective features being functional, apart from those that were not implemented.

In addition to these tests, we also conducted endpoint tests to validate the functionality of the backend in isolation, in order to be thorough. Here is the test log:



```
PASS tests/test1.test.js
  √ test to see if Jest works (2 ms)
  Testing GETs for html pages
    √ GET / (43 ms)
    √ GET /signup (7 ms)
    √ GET /signin (6 ms)
    √ GET /events/join (7 ms)
    √ GET /events/create (5 ms)
    √ GET /events/:eventCode (7 ms)
  Testing GETs for database retrievals
    √ GET /api/event/events (15 ms)
    √ GET /api/event/:eventID (11 ms)
    √ GET /api/event/:eventID/feedback (8 ms)
    √ GET /api/project/projects (13 ms)
    √ GET /api/project/:projectID (9 ms)
    √ GET /api/project/:projectID/events (8 ms)
  Testing DELETEs
    √ DELETE /api/event/delete/eventID (10 ms)
  Testing POSTs
    √ POST /api/event/new (23 ms)
    √ POST /api/event/join/:eventid (13 ms)
    √ POST /api/event/leave/:eventid (11 ms)
    √ POST /api/project/new (8 ms)
    √ POST /api/project/join/:projectID (7 ms)
    √ POST /api/project/leave/:projectID (7 ms)
    √ POST /api/user/new (64 ms)
    √ POST /api/user/login (63 ms)
    √ POST /api/user/logout (11 ms)
    √ POST /api/feedback/new (10 ms)
```

Figure 7: Supertest Endpoint Test Log

This shows the robustness of our backend. However, our system was not as fully tested as it could have been. It has only been tested for Chrome, though we intended to also test it on Safari. We don't know the validity of the system on Safari. We could have tested on Safari by changing the web driver used in the test script and running the tests on a macOS system or a VM running macOS.

5.2 System Evaluation

Our system delivers on the basic functionality required by the customer and the majority of the features we intended to implement. As shown in the test logs above, the requirements whose respective features were implemented were met. Non-functional requirements have also been met.

One of the major strengths of our system is our modular approach. This makes it simple to add new features, and it also makes testing much easier. However, the testability of the frontend could be improved, as inline scripts are used, which are quite difficult to unit test. To fix this, the scripts should be kept in separate js files and referenced in the HTML.

Another strength is that our backend is robust and reasonably fast. Furthermore, while it is missing some features, our user interface aptly fulfils its purpose. It has been evaluated in more detail in the section below.

The major flaws of our system are that some important features have not been implemented. The most notable being projects and templates. We also intended to include additional features that we came up with, such as attendance tracking and moderation systems, which did not get implemented either. Without these features, our system feels bare and basic, and does not have the uniqueness we hoped it would have.

With more time and resources, we would be able to implement all of the features that we are missing. We would also be able to conduct more rigorous and extensive testing. This would result in a more complete system with its own identity.

Overall, our system delivers on the basics, but feels like an unfinished product. It does not have much of an identity and we couldn't say that there is much to set it apart from similar systems.

5.3 User Interface Evaluation

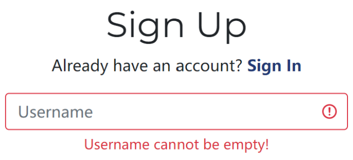
This section evaluates the user interface. Although creating and sending template feedback were not implemented, the interface for this part will still be evaluated to give a complete evaluation of the designing ideas.

5.3.1 Visibility Of System Status

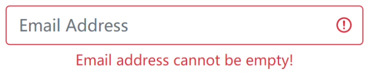
Objective: The system should always keep users informed about what is going on.

Input fields for passwords, email addresses, option text and question text are designed to show warning or error mes-sage indicating that the user input might be invalid. This design fulfils the objective to inform users about their input. See Figure 8.

(a) Username



(b) Email Address



(c) Password

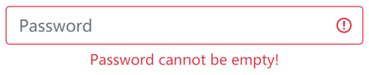


Figure 8: Input Warnings

Most interactions are expected to happen in at most 1.0 second (the limit for the user's flow of thought to stay uninterrupted[6][1]), in which case no special feedback is necessary. However, there still could be quick and immediate changes on the colours of buttons or short animations indicating that some action is confirmed and being performed if later users feel that it would be smooth to have these effects.

The design gives users crucial information regarding proper interactions and executions but leaves out the supplementary information as future development.

5.3.2 Match Between System And The Real World

Objective: The system should speak the user's language, with words, phrases, metaphors and representations familiar to the user.

For buttons on sign up and sign in pages, we used conventional labels to avoid any ambiguity, namely "Sign In" and "Sign Up". See Figure 9.

(a) Facebook

(b) Our System

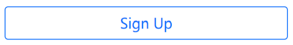


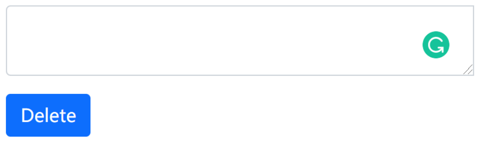
Figure 9: Sign Up Buttons

Although there are some buttons with the same label on different pages such as "Delete" and "Create", they do not perform the exactly same action (i.e. deleting the exactly same object) and only delete the container or box where the button is placed. We did not write whole sentences on those buttons, e.g., "Delete This Event" or "Delete This Template", as this is less concise and makes the button bigger than it needs to be. See Figure 10.

We did not use many icons throughout the system as too many of them might cause confusion since every user might have different interpretations of some icons. We only used the cross symbol "X" as a simple "button" to delete an item in a list. See Figure 11.

The design mostly uses simple labels with commonly seen words which should be familiar to users. It avoids using too many metaphors and only uses them in very straightforward situations.

(a) Button For Deleting The Template Question



(b) Button For Deleting The Event

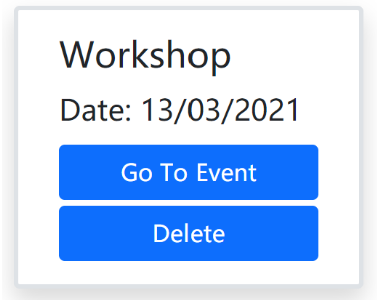


Figure 10: Delete Buttons

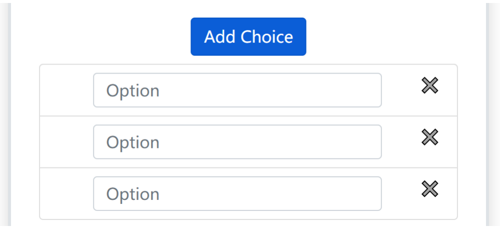


Figure 11: Cross Buttons

5.3.3 User Control And Freedom

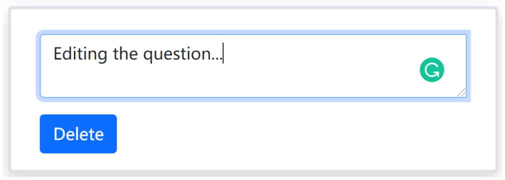
Objective: Users can easily leave out or redo the unwanted action without having to go through an extended process.

Users might encounter more mistyping or mis-clicking when creating an event and feedback templates compared to other interactions as it involves more typing and clicking. The system is designed so that for each multiple question, a user can delete options and change the question at any point of editing. See Figure 12.

We did not design the system to have a double check confirmation window or box on every action evoked by users as this is not a requirement with high priority and does not affect the core functionalities of the system. This could be implemented in future development.

The design has undo and redo options when much editing is involved so users do not have to do extra work. We did not implement double check windows so the goal of giving users control and freedom was not fully fulfilled.

(a) Editing Short Answer Question



(b) Editing Multiple Choice Question

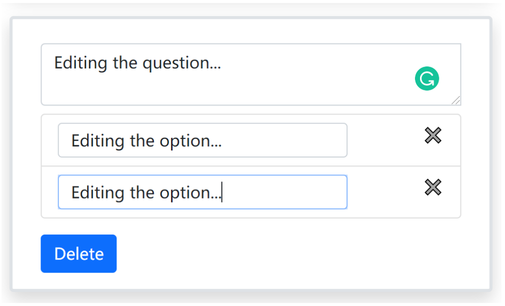
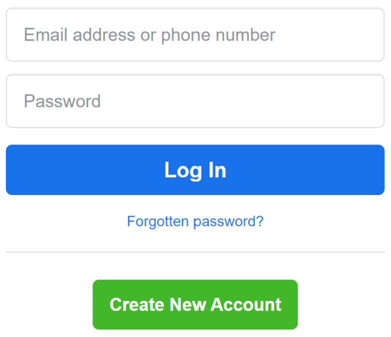


Figure 12: Editing Templates

(a) Facebook



(b) Our System

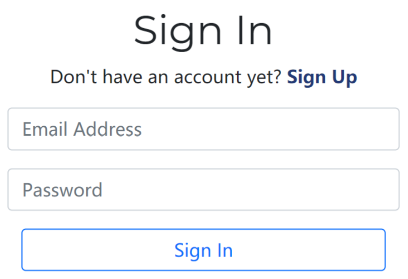
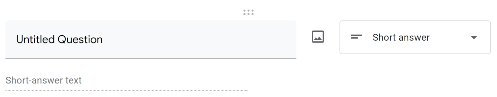


Figure 13: Sign In Pages

(a) Google Short Answer



(b) Our System's Short Answer

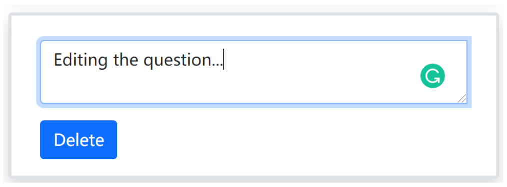


Figure 14: Short Answers

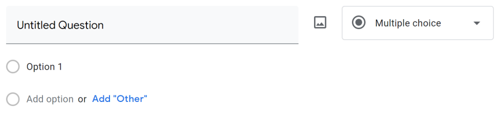
5.3.4 Consistency And Standards

Objective: Users should not have to wonder whether different words, situations, or actions mean the same thing. Follow platform and industry conventions.

We did not find explicitly stated conventions on sign up pages, sign in pages and questionnaire templates so the objective was not fully fulfilled.

However, we made sign up and sign in pages similar to Facebook sign in page. Our questionnaire template is based on Google Form. Both Facebook and Google Form are commonly seen and used so users should be familiar with the format. Even if an user had never used Facebook and Google Form, the system is designed to be self-explanatory and have clear labels hence should be easy to use. See Figure 13, 14, 15.

(a) Google Multiple Choice



(b) Our System's Multiple Choice

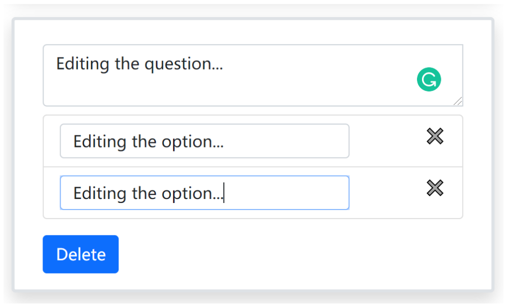


Figure 15: Multiple Choice

5.3.5 Error Prevention

Objective: Prevent a problem from occurring in the first place.

The sign up page checks whether the passwords repeated for confirmation are identical. See Figure 16.

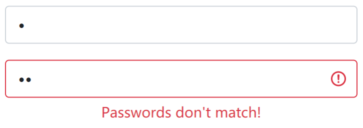


Figure 16: Passwords Do Not Match

Empty inputs will be detected and warning labels will appear and ask the user to give valid inputs. The frontend itself does not allow users to send invalid inputs (inputs will not be sent even if the button is clicked). The warning labels also catch users' attention before they attempt to click the button again without giving valid inputs. Hence the design prevents problems from occurring when users give invalid inputs. See Figure 17.

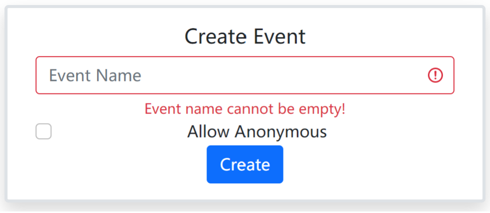


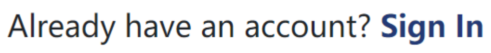
Figure 17: Empty Event Name

5.3.6 Recognition Rather Than Recall

Objective: Making objects, actions, and options visible. Make navigation visible and make it easy to go back, go to home page and quit.

The Sign up page and sign in page can be navigated from each other. After a user has logged in, the home page has a navigation bar to the pages for creating events, joining events, managing events, sending feedback and log out. This navigation is available on every page after login. Hence the design fulfils the objective. See Figure 18.

(a) Sign In Link



(b) Navigation Bar

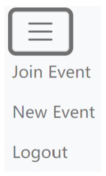


Figure 18: Navigation Bar And Link

5.3.7 Flexibility And Efficiency Of Use

Objective: Speed up the interaction for the expert users with accelerator. Experts prefer few screens with a lot of information and a lot of flexibility in the methods[5], while novice users prefer going through step by step sequences with little information.

We did not provide any shortcuts as this is not a requirement with high priority. Hence this objective is not fulfilled.

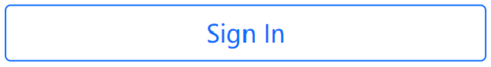
This however does not mean expert users are not important. Shortcuts could be provided when a large number of users had used the system, as compared to when the system was just released, in which case we might not be able to collect adequate user data and fully analyse users' frequent interactions with the system to provide shortcuts.

5.3.8 Aesthetic And Minimalist Design

Objective: Interfaces should not contain information which is irrelevant or rarely needed. Every extra unit of information in an interface competes with the relevant units of information and diminishes their relative visibility.

We kept the most pages as simple as possible. And we did not add too much animation. Only some hovering effects or focus effects are used to attract users' attention to the right place. See Figure 19.

(a) Not Hovering



(b) Hovering

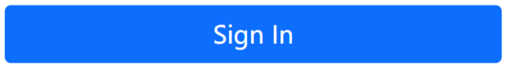
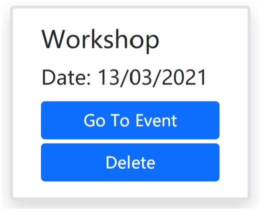


Figure 19: Hovering Effect

Events might have a lot of information, e.g., name, time, feedback and member list, etc. On the home page after login, each event is only displayed with essential information, e.g., name and time, which should be a good reminder for users. All the information provided to users are necessary hence the objective is fulfilled. See Figure 20.

Figure 20: Event



5.3.9 Manage Errors

Objective: Error messages should be expressed in plain language (no error codes), precisely indicate the problem, and constructively suggest a solution.

As explained above, the sign up page and sign in page will have red labels indicating the user gave some invalid inputs. The page for creating an event and feedback templates does not allow empty input so displays red labels if there are any empty input. The system will not show any error code which most users might not be able to understand. Hence the design fulfills the objective

5.3.10 Help And Documentation

Objective: Use prompt and contextual help related the task, allow easy search.

The system overall is fairly self-explanatory. This report also gives some hints as to how the system works. However, it is agreed that users still need a document full explaining the system and containing tutorials that should be easy to search. This might be developed in future work and hence the objective is not fully fulfilled.

5.4 Overall Changes On The User Interface

We had some changes on the interface, mainly on the layouts and colours. We wanted to make the system look succinct and light so only used light colours compared to the initial design. There are some buttons or tabs that have deeper colours. But these are to show contrast as they are quite important.

We mainly used blue as evidence yielded from some experiments has suggested that viewing blue is beneficial for efficient performance. Blue was also posited to be "restful" and to produce calm and stable actions[3][7]. We used red labels as warning as it is suggested to carry the strongest reaction of all colours[4].

6 Process Evaluation

6.1 Methodology

The adopted methodology was iterative, prototype based. The main idea was to focus on implementing increasingly functional prototypes of the system, in order to spot changes that needed to be made more rapidly. This approach was practical for the team as we had little previous experience with developing software as a group and creating websites. Consequently we were unsure of what issues could occur during the process, so taking an iterative approach allowed us to evaluate our product during development and make necessary changes.
However, the planned order in which development would be carried out meant that joining the front-end and back-end would be the final step. This contradicted our methodology since the prototype wouldn't be functional until the final stage of development. It also meant that it was more difficult to test our work at each stage , leading to less thorough testing during development because it was more time-consuming to do so.
In conclusion, the choice of methodology was good, however it was not carried out as effectively as it could have been.

6.2 Communication

The communication in-between the front and back-end teams on the structure of the server, and how to access data from it, was effective. As a result little time had to be spent in the final stage of development connecting the two, and there were no issues as a result of this. The current progress was also effectively communicated, so everyone was aware of the stage of development.
On the other hand, the communication internally within the front and back-end teams was less effective; which member was doing what work wasn't always communicated and this lead to some overlap in the work. Consequently time was wasted implementing some features twice or scrapping work that was unneeded. The process was complicated by the fact that the team was spread out geographically and meeting in-person was not possible. This made it more difficult to organise meetings because of the spread of time zones and some ideas were more difficult to explain in an online environment.
Overall, the communication went well in-between the teams, but it could've been improved within each team.

6.3 Planning

The separation of the system into multiple subsystems in the design was advantageous in the implementation stage because it made separating the work much easier. In addition testing the system was simpler because the independent subsystems could be tested separately, making it easier to track down the cause of bugs in the system. This reduced the time spent on debugging.
Ambiguity in the original design plans meant that some decisions were left to be made during the development process which wasted time. For example, the design for the template system was unclear in the design, which meant that it ended up being cut from the final product, as there was insufficient time to plan out how to create them. This also meant that the standards were unclear for the front and back-end teams; it wasn't explicit what work would be done client-side or server-side and it wasn't obvious how data would be communicated between the two. Although these issues were resolved, it slowed down the development, thus fewer features were in the final product than there could've been if the planning and design was more thorough.

6.4 Management

The team manager succeeded in creating a positive, stress-free environment by being understanding and supportive of the team-members. Subsequently the team maintained good mental health and a positive attitude throughout the project. This improved the final product as everyone was productive and had a good work ethic, so the work that was done was of a higher quality.
The order in which work was carried out in the front-end and back-end was an issue. Because of the way the work had been planned out, some features were implemented in the back-end when they hadn't been

started by the front-end team and vice versa. For example, the functionality was in place on the server for a user to create projects that consisted of multiple events and to manage these, however there was no way for a user to do this on the website. This was mitigated by the good communication between the teams, as mentioned previously, because the teams communicated well. Although the work was planned inefficiently, the teams were able to quickly connect the front and back-end development. Despite this, more features could have been completed if the work order had been planned out more effectively. In order to improve this, the work should've been planned so that both teams were working towards the same goal at the same time then connecting them once they were done, and only if one team were ahead of the other would they move on to another task. This would mean that even if the same amount of work was done, the final product would be more complete. Another problem stemmed from the lack of concrete deadlines for intermediary stages of development. As a result work, was sporadic and it was not obvious that the implementation was behind schedule. To improve on this, more deadlines with less time in-between should have been set. This would make it obvious how much work needed to be done in what time, and would also have made issues clearer to the team at an earlier stage. In summary, the management was successful at maintaining a positive working environment, however the planning of how work was carried out lead to a less complete final product.

References

[1] Stuart K. Card, George G. Robertson, and Jock D. Mackinlay. The information visualizer, an information workspace. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '91, page 181–186, New York, NY, USA, 1991. Association for Computing Machinery.

[2] Lee R. Dice. Measures of the amount of ecologic association between species. *Ecology*, 26(3):297–302, 1945.

[3] Andrew J. Elliot and Markus A. Maier. Color psychology: Effects of perceiving color on psychological functioning in humans. *Annual Review of Psychology*, 65(1):95–120, 2014. PMID: 23808916.

[4] Waldemar Karwowski. *International Encyclopedia of Ergonomics and Human Factors, Second Edition - 3 Volume Set*. CRC Press, Inc., USA, 2006.

[5] Yea-Seul Kim, Mira Dontcheva, Eytan Adar, and Jessica Hullman. Vocal shortcuts for creative experts. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, CHI '19, page 1–14, New York, NY, USA, 2019. Association for Computing Machinery.

[6] Robert B. Miller. Response time in man-computer conversational transactions. In *Proceedings of the December 9-11, 1968, Fall Joint Computer Conference, Part I*, AFIPS '68 (Fall, part I), page 267–277, New York, NY, USA, 1968. Association for Computing Machinery.

[7] Jacob S. Nakshian. The effects of red and green surroundings on behavior. *The Journal of General Psychology*, 70(1):143–161, 1964. PMID: 14115829.