Team: SponsorLah!
Members: Clement Tee Lian Zheng, Siew Hui Zhuan
Project Title: Sponsorr!
Level of Achievement: Artemis

Orbital Milestone 3 Submission

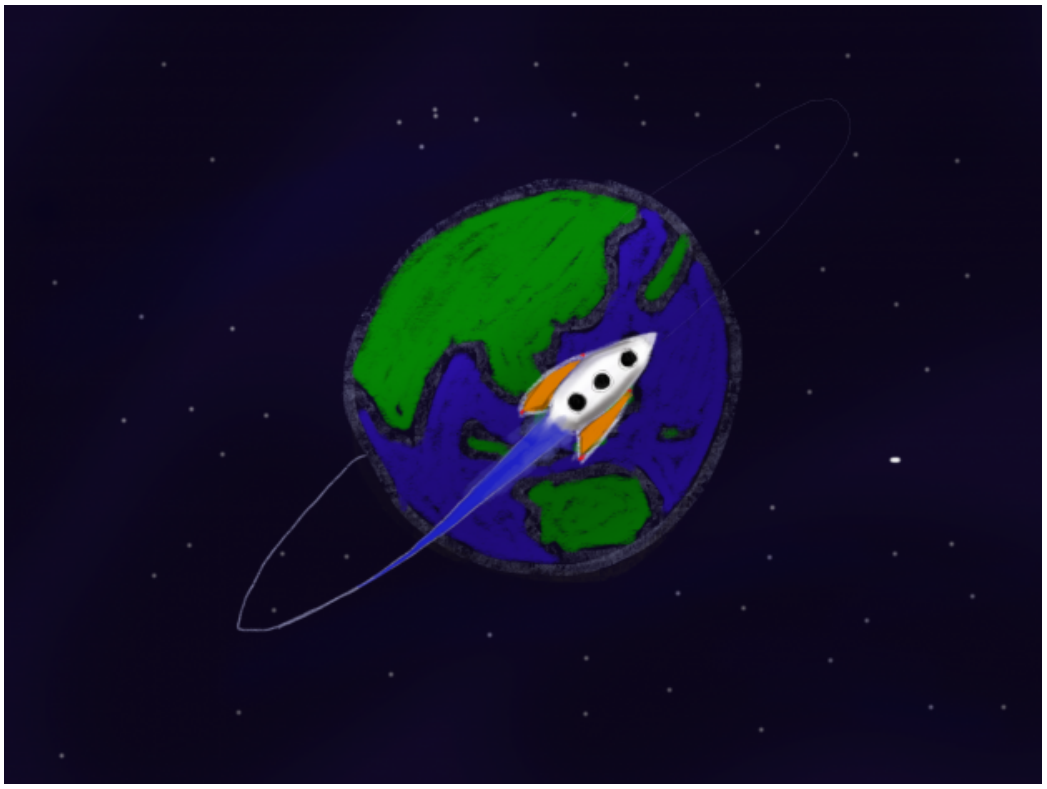


Table of Contents

1.0 Motivation

Event organisers often find themselves sending cold emails to potential sponsors to request sponsorship, but not getting any replies. You are also running low on time and have a target number of sponsors to reach out to or a target amount to achieve.

Individuals, businesses, and charitable foundations receive sponsorship requests from organisations that either do not fit their beneficiary criteria or do not align with their values and brand culture. Smaller businesses that are eager to promote their brand often find themselves overshadowed by larger, better-known competitors.

2.0 Aim

We want to solve the problem of matching the right sponsor for the right event. This can be done by building a platform that streamlines the process of sourcing for sponsorship.

When built, this platform aims to provide SMEs with an opportunity to promote their brand, service, or product to a specific target market by partnering with events organised by a third party.

This platform also helps corporations to be matched with charitable events and/or organisations that align with their brand image and core values in order to fulfil their corporate social responsibility (CSR).

In addition, this platform should also reduce the time and effort spent by event organisers to source sponsors, and instead be able to better plan their events with less logistical overhead.

3.0 Getting Started

3.1 Test Account

Users will need to sign up with a real email account in order for the application to work. Users can create a test account with temporary email via this site and all emails including confirmation and match notification will be redirected to this temporary inbox.

3.2 Resources

Here is the collection of resource that the team used throughout the project

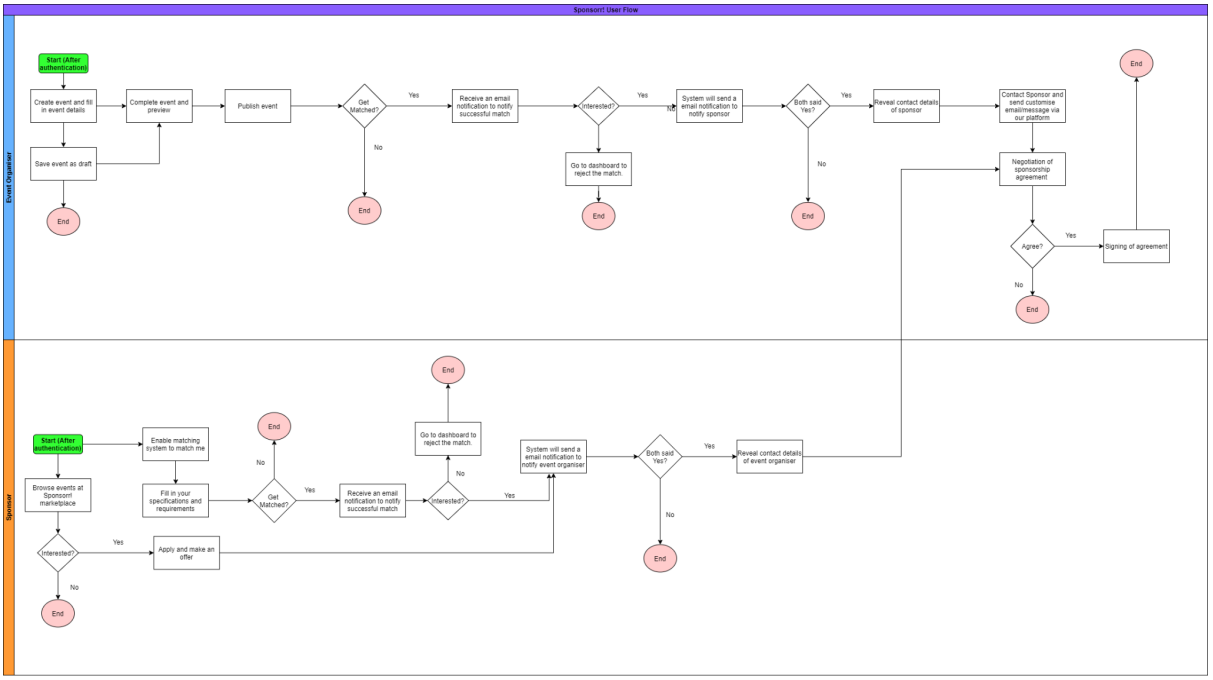
Resources	Link
Event Website	https://sponsorr.me/
Frontend Codebase	https://github.com/clement0010/sponsorr-frontend
Backend Codebase	https://github.com/clement0010/sponsorr-backend
Project Management	https://github.com/clement0010/sponsorr-frontend/projects/1

4.0 Core Features

4.1 User Stories

Roles	User Stories
Event Organiser	● I want to be able to create an organiser's profile so that I can manage multiple events as the same organiser. ● I want to be able to create an event quickly and publish it. ● I want to be able to fill in details for the event so that my event's needs can be met. ● I want to be able to attach proposal documents to my event. ● I want to be able to update the event's details on the go as the event's requirements evolve over time. ● I want to be able to remove an event once it is over or if it is cancelled. ● I want to be able to view my past matches. ● I want to be able to activate and deactivate email triggers whenever an event sponsor agrees to sponsor my event. ● I want to be able to accept or decline a sponsorship. ● I want to be able to communicate with matched sponsors.
Event Sponsor	● I want to be able to create a sponsor's profile so that I can respond to different sponsorship requests for the same sponsor. ● I want to be able to search and filter events to sponsor. ● I want to be able to change my specifications on the events to sponsor to get better, personalized matches. ● I want to be able to view my past matches. ● I want to be able to view the sponsorship status of events. ● I want to be able to activate and deactivate email triggers whenever there are available matches. ● I want to be able to leave messages to the event organiser whom I am matched with. ● I want to be able to accept or decline events. ● I want to be able to communicate with event organiser's whose events I agree to sponsor. ● I want to be able to have my brand/service/product shown to as many people as possible. ● I want to be matched to an event that aligns with my brand image and values.

4.2 Overview

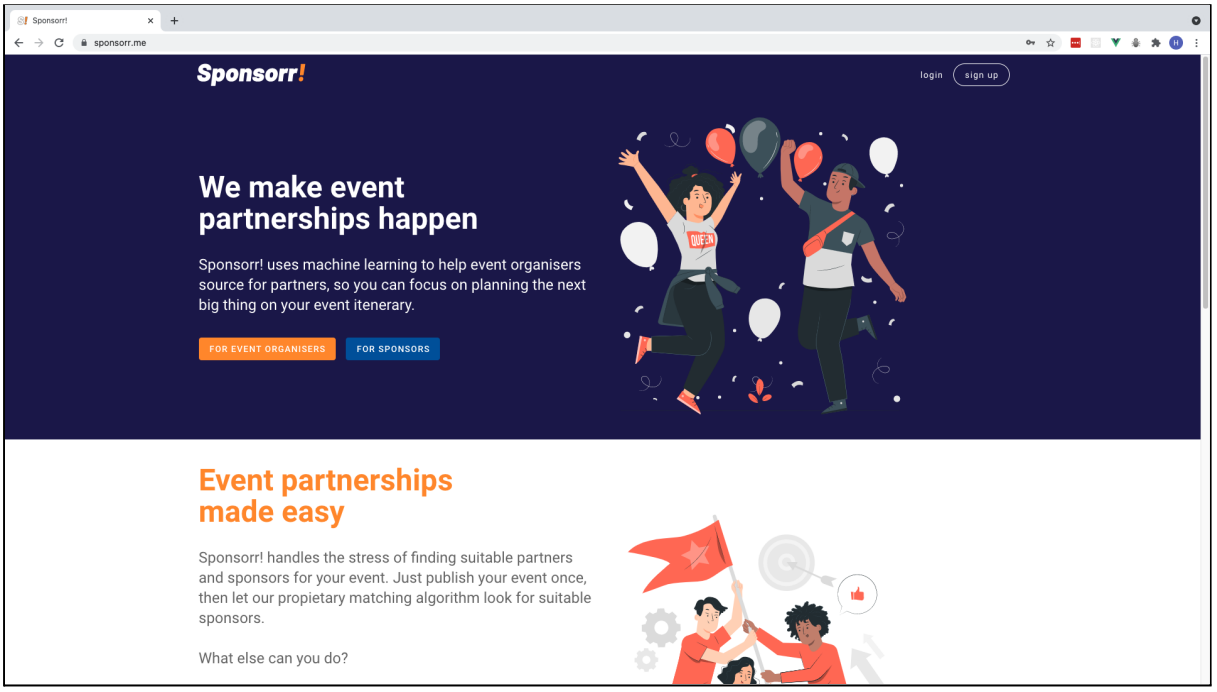


This diagram describes the user flow once they have signed up and logged into our system. Use this link to access the user flow diagram with higher resolution.

4.3 Features Breakdown

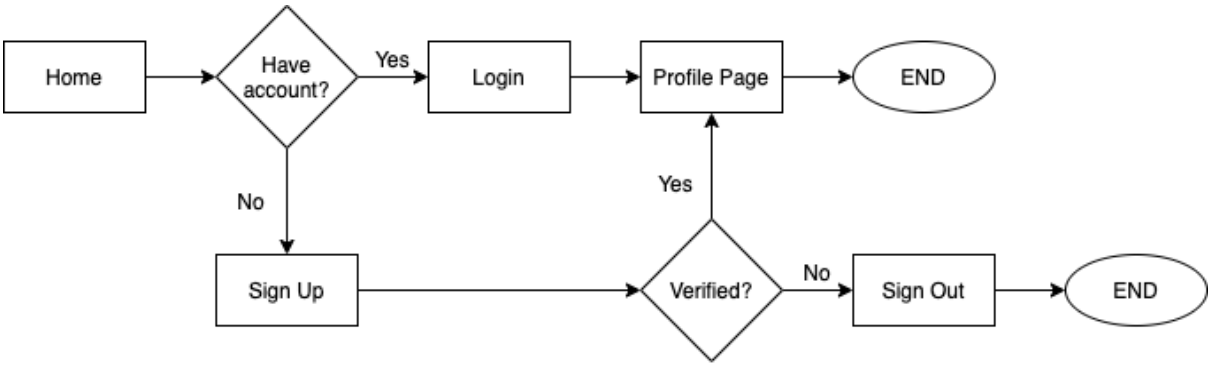
4.3.1 Landing Page

A refreshed landing page serves as the entry point to the application. Updates include element transitions, animated SVG images and programmatic scrolling.

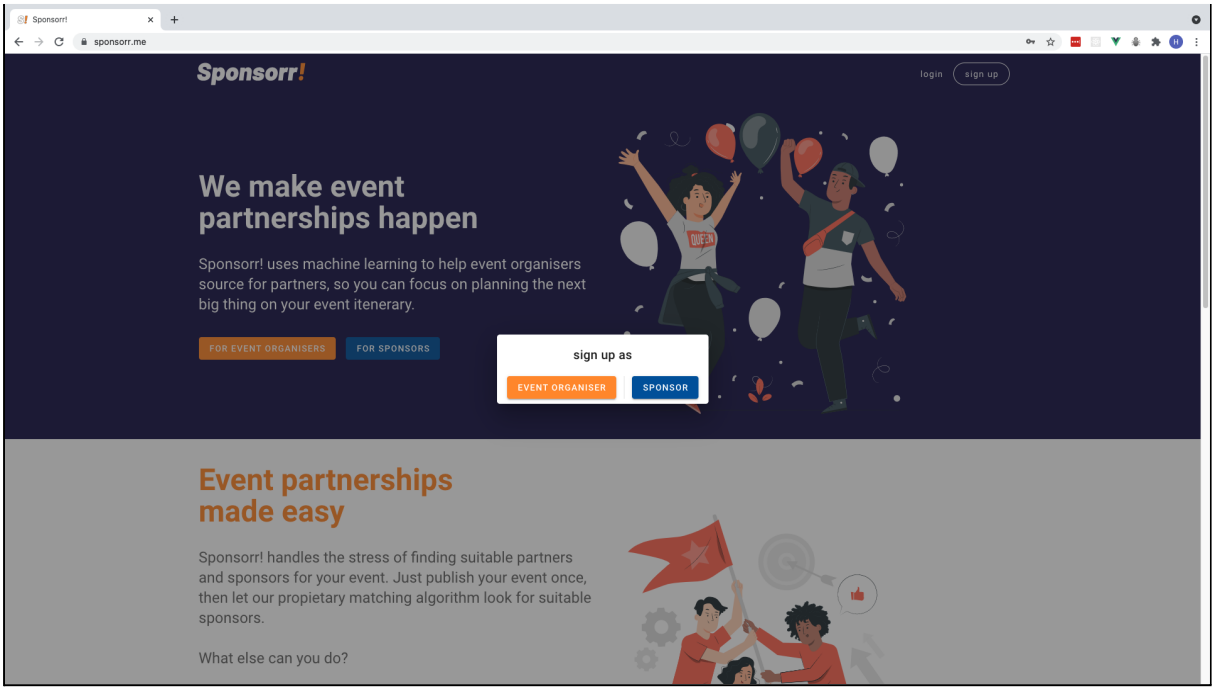


4.3.2 Authentication Flow

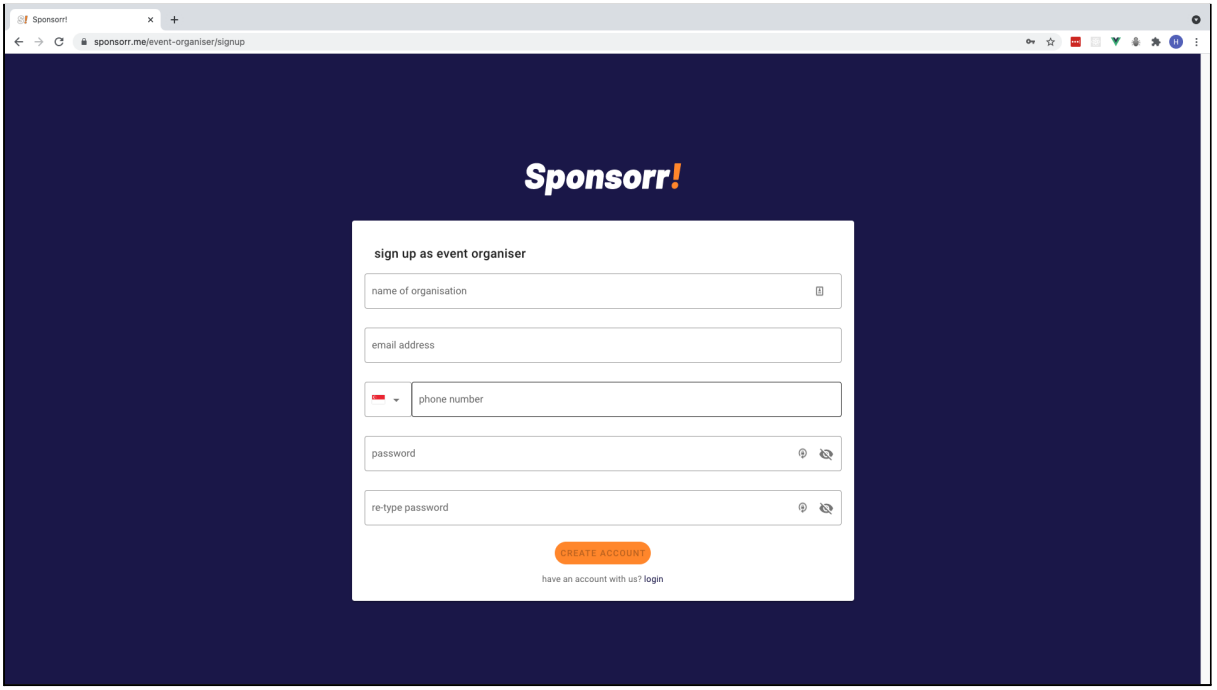
User flow diagram for authentication and verification



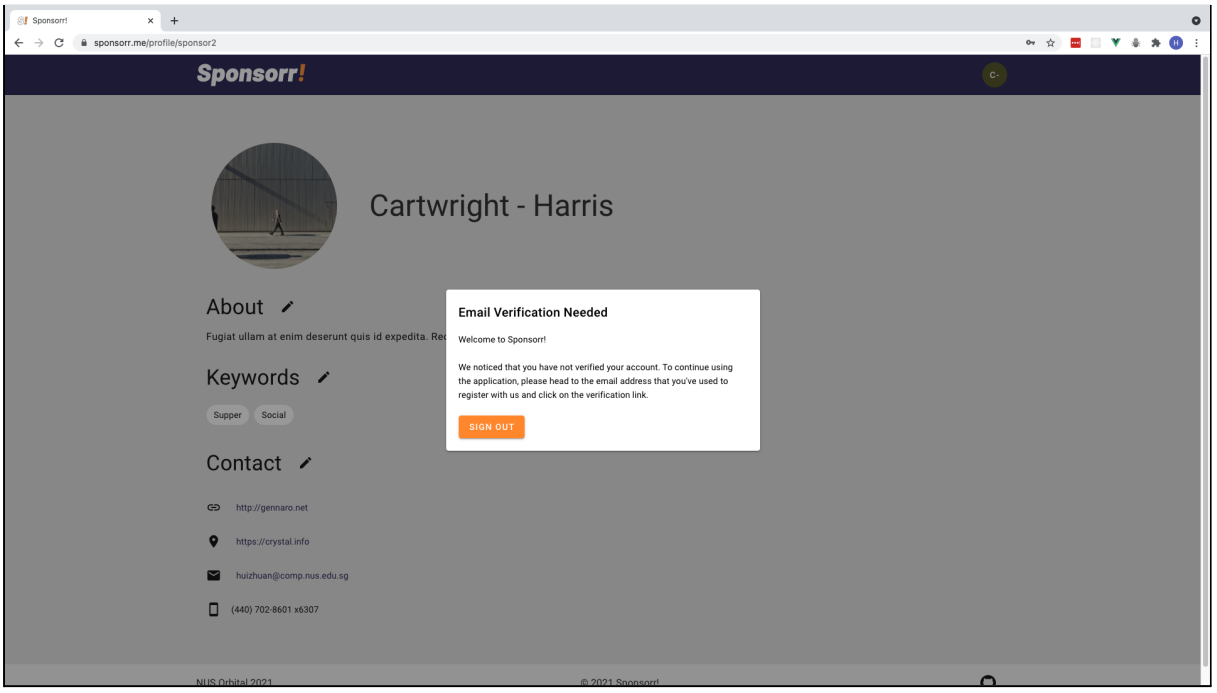
Users can sign up as an event organiser or sponsor.



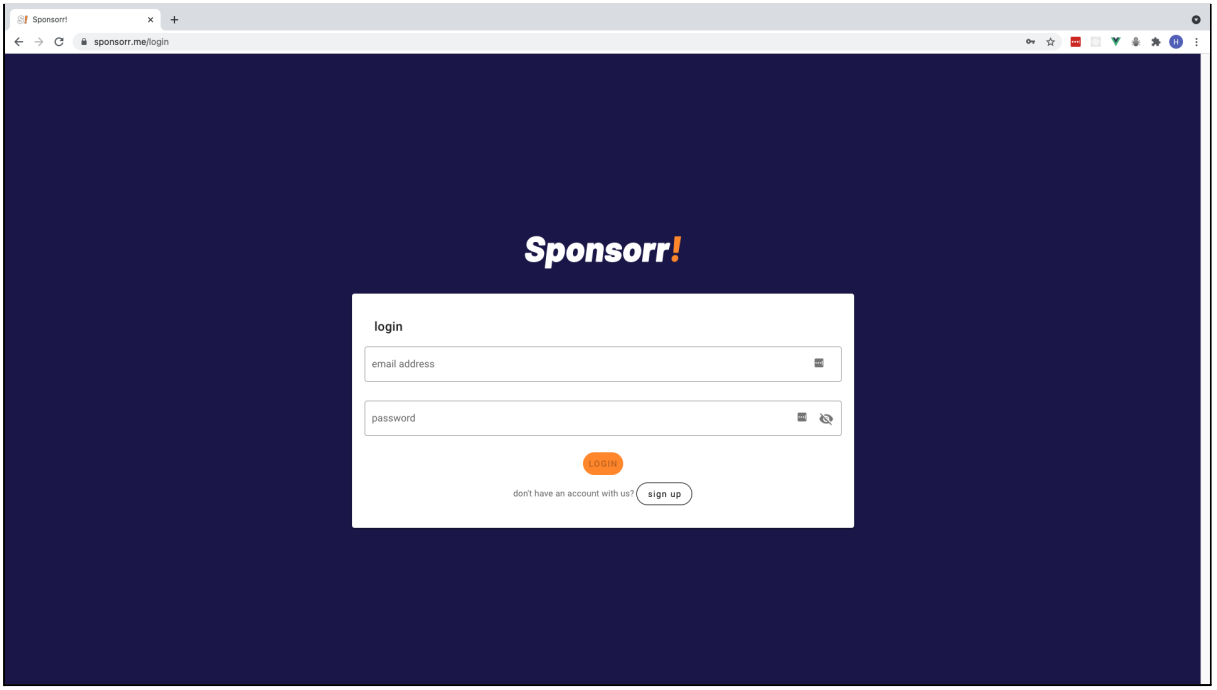
Users enter their account details to create their accounts. Upon successful registration, users are required to verify their email.



Users are prevented from using any function of the application without clicking on the verification link sent to the email they registered with.

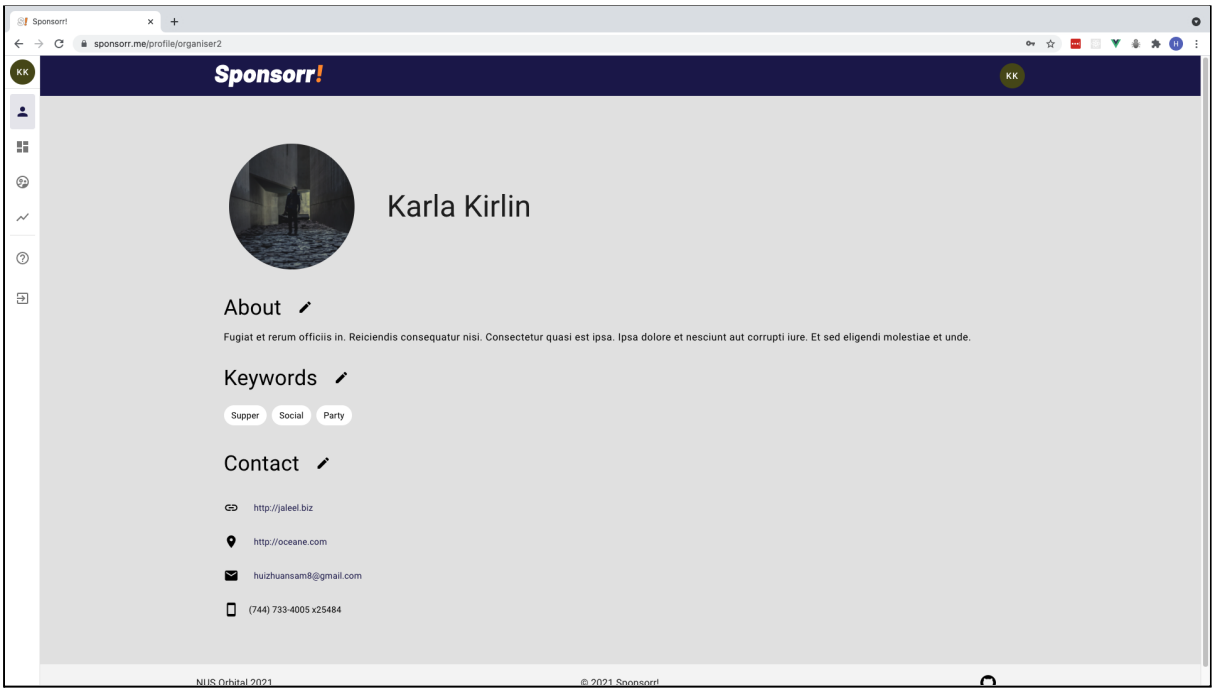


Users enter their account credentials to log in.

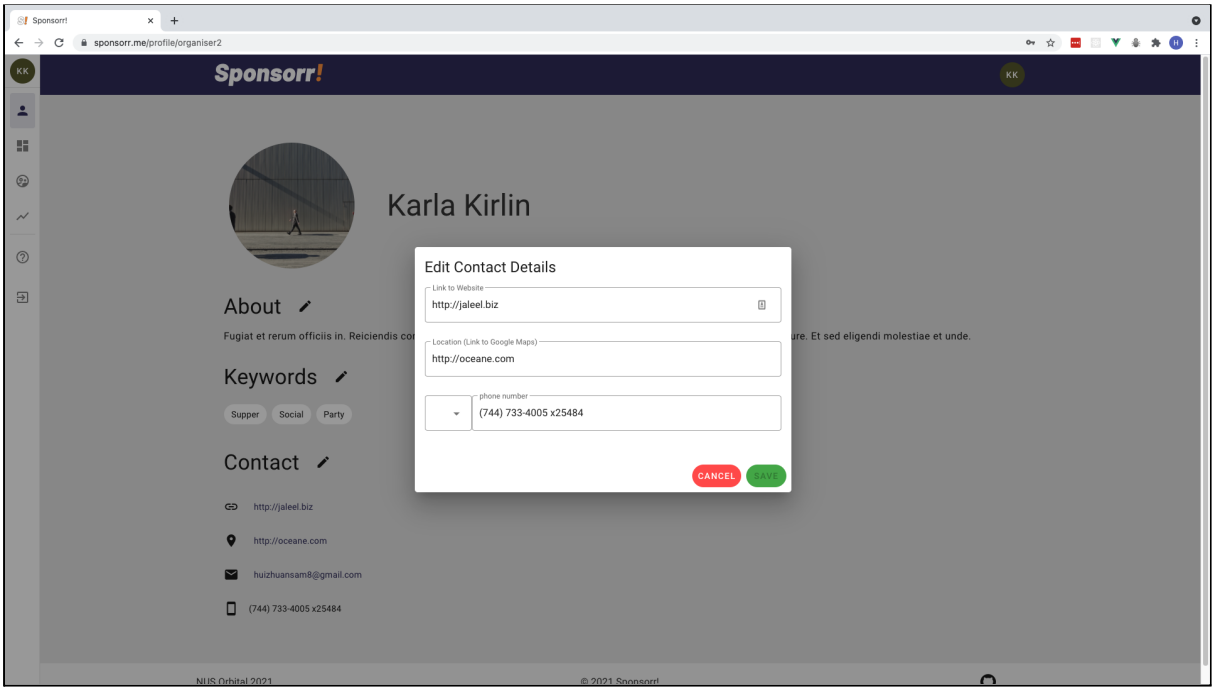


4.3.3 Profile Page

Users are able to see and edit their organisation details from the profile page.

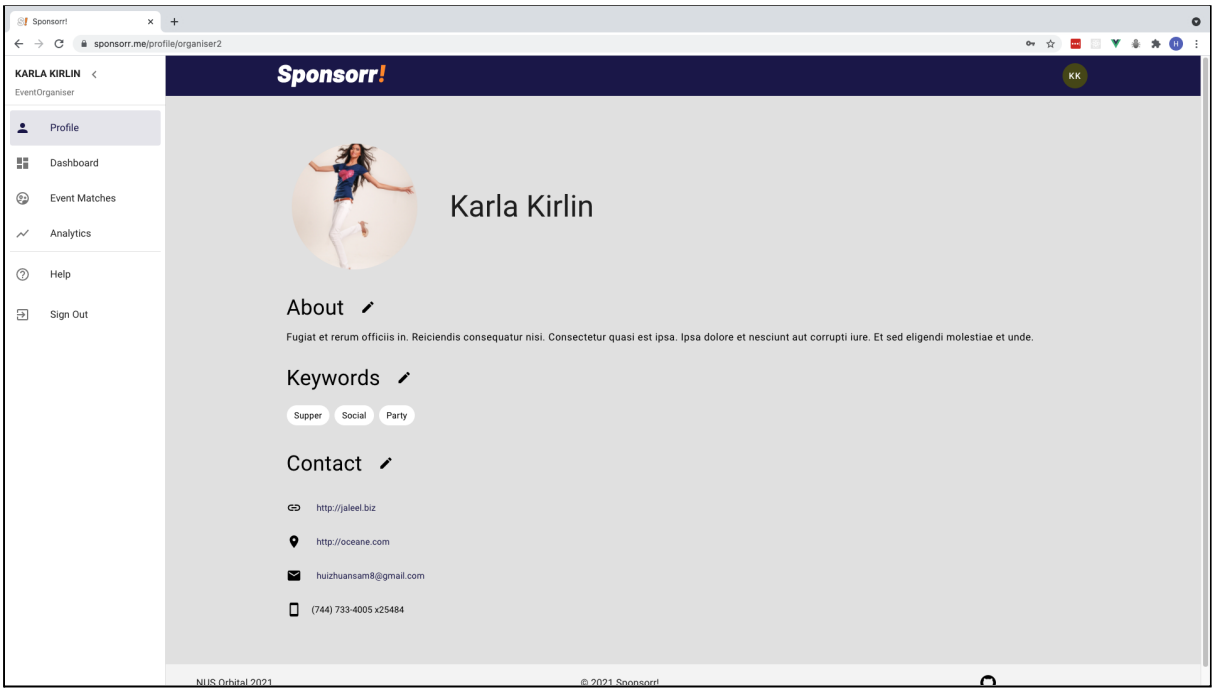


Users can edit their profile details simply by clicking the pen icon to update their profile.

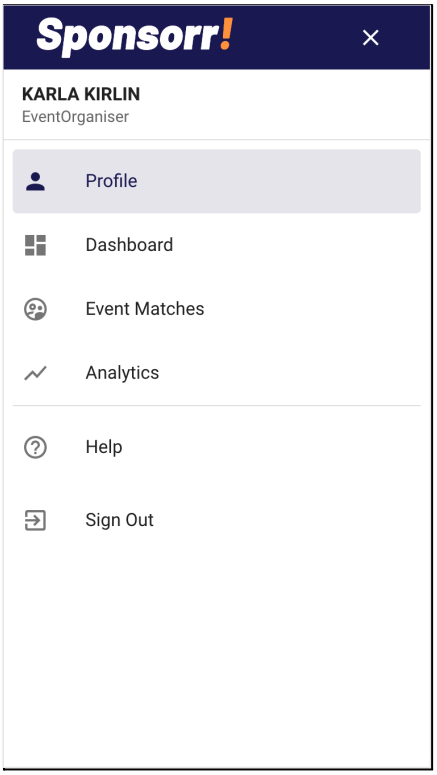


4.3.4 App Navigation

Application drawer (expanded). The application drawer provides users with a quick and intuitive way to switch between different sections of the application and can be minimised to increase the available screen real estate.

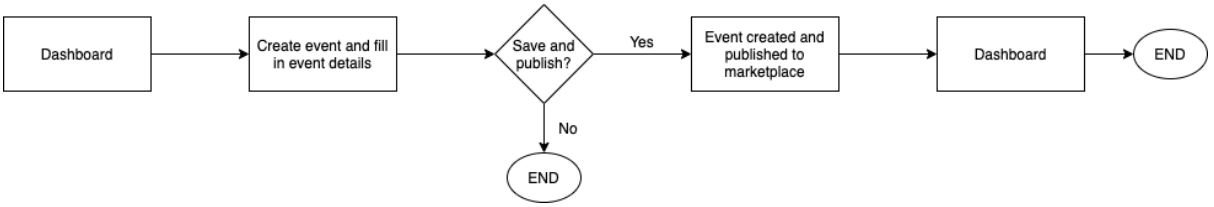


On mobile devices, the navigation method becomes a drop-down menu:

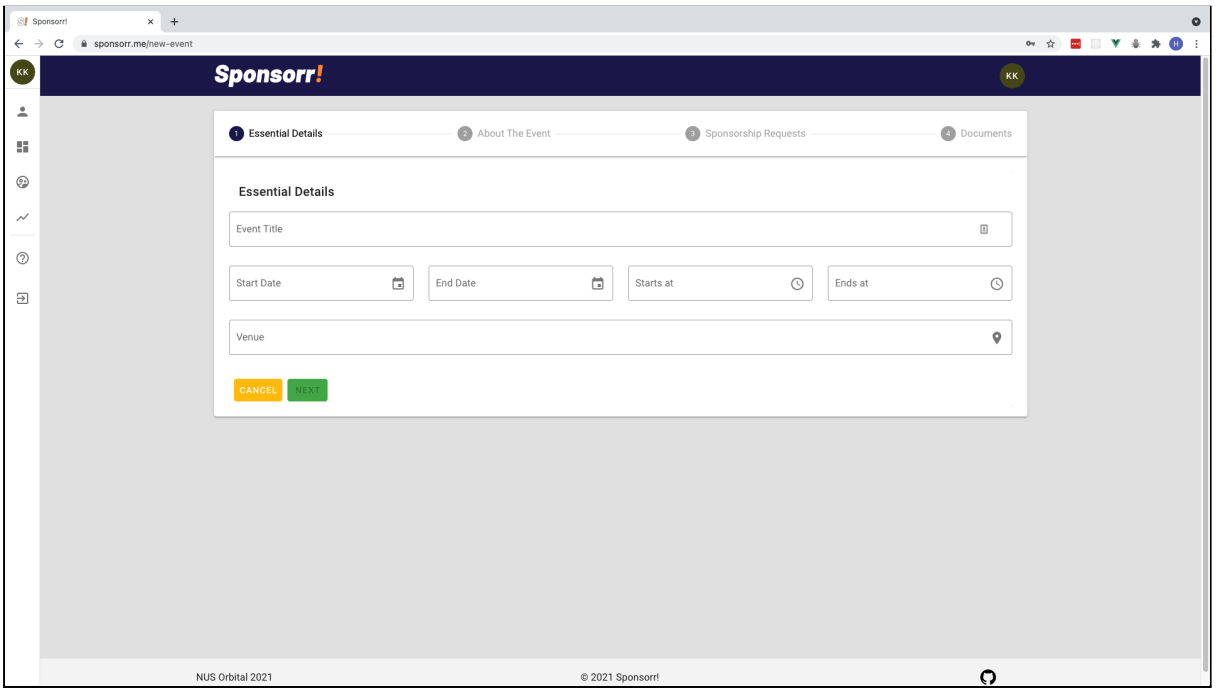


4.3.5 Create New Event

User flow diagram for event creation as an **Event Organiser**

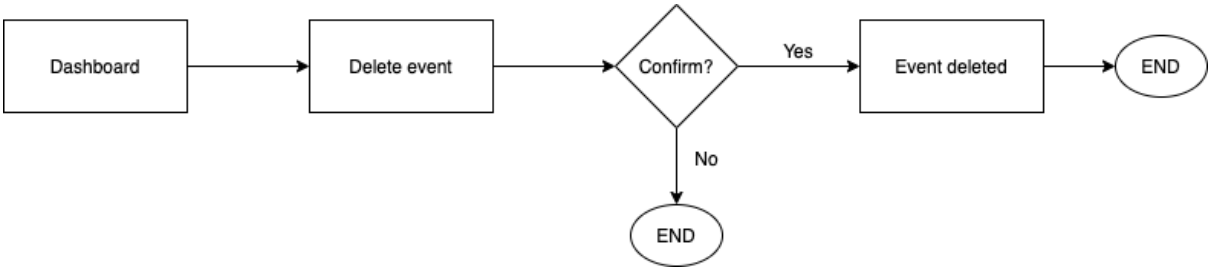


The event organiser creates a new event by filling in the create new event form. Here, the application collects event details from the user and uploads them to the database.



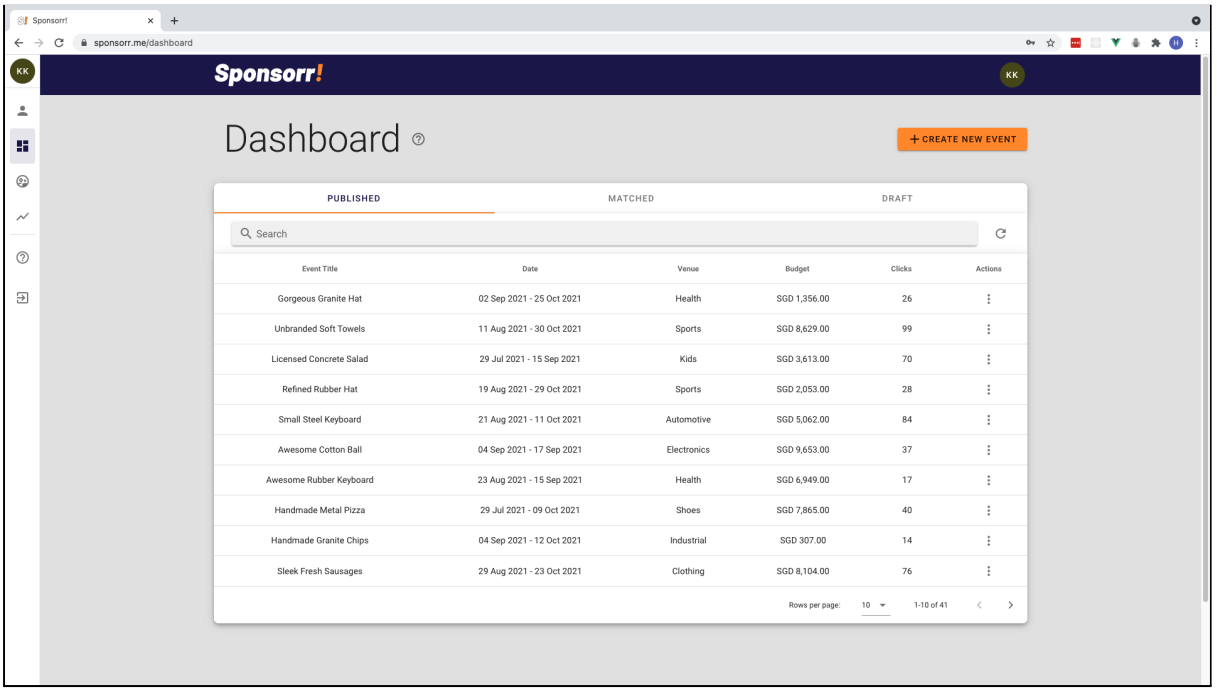
4.3.6 Event Organiser Dashboard

User flow diagram for delete event as an **Event Organiser.** The user has to unpublish the event prior to deleting the event. This is a **preventive measure** to ensure matched events can never be deleted to avoid confusion from sponsors.

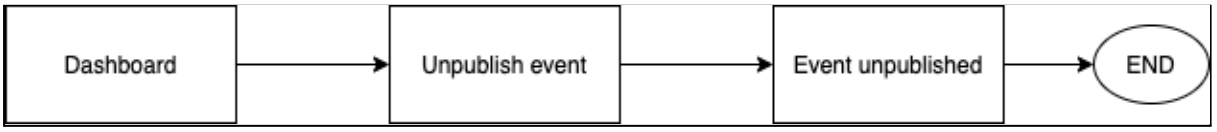


The dashboard displays all the events owned by the user and is categorized into 3 distinct categories (tabs): published, matched, draft. Users can click on the action buttons to publish, unpublish, view, and delete the events.

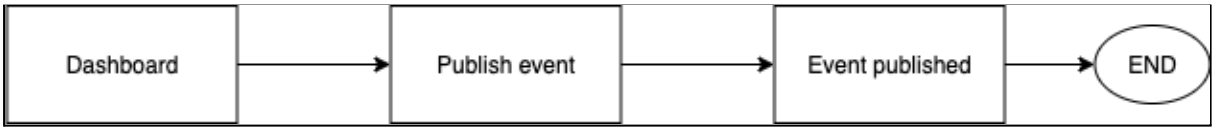
Adding to that, users are also able to search the event by title and re-fetch events to see their latest matches. This adds great value to our end-user as it allows a better user experience in controlling and managing their events.



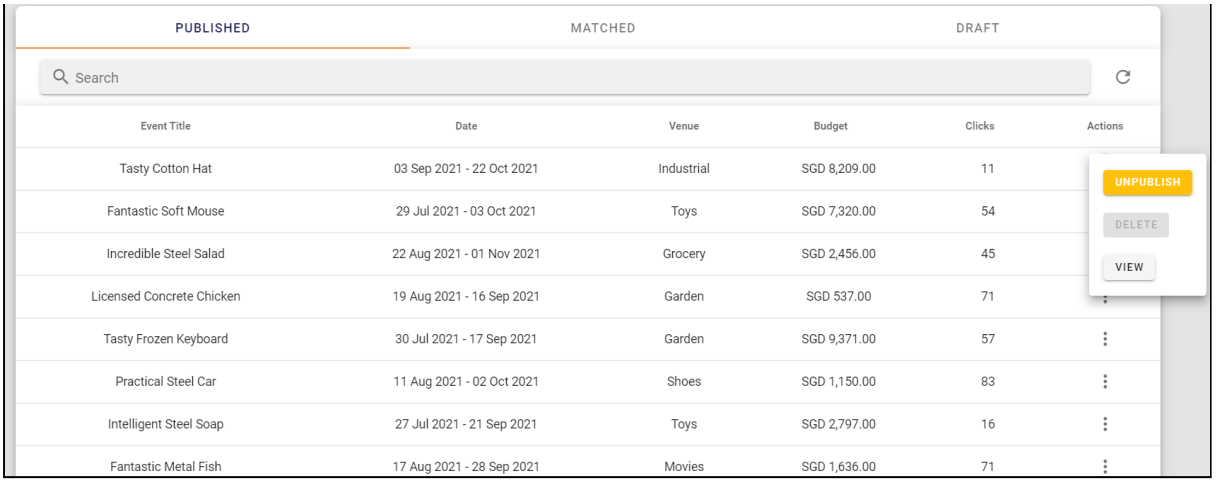
User flow diagram for unpublishing an event as an **Event Organiser**



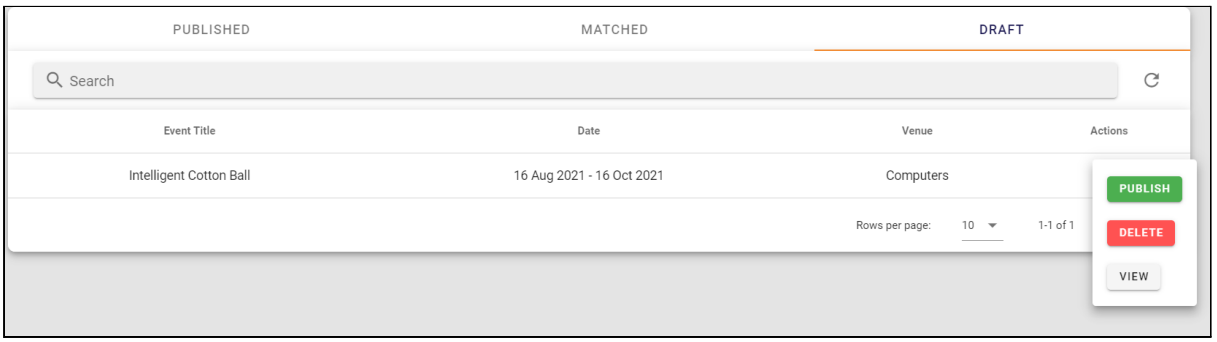
User flow diagram for publishing event as an **Event Organiser**



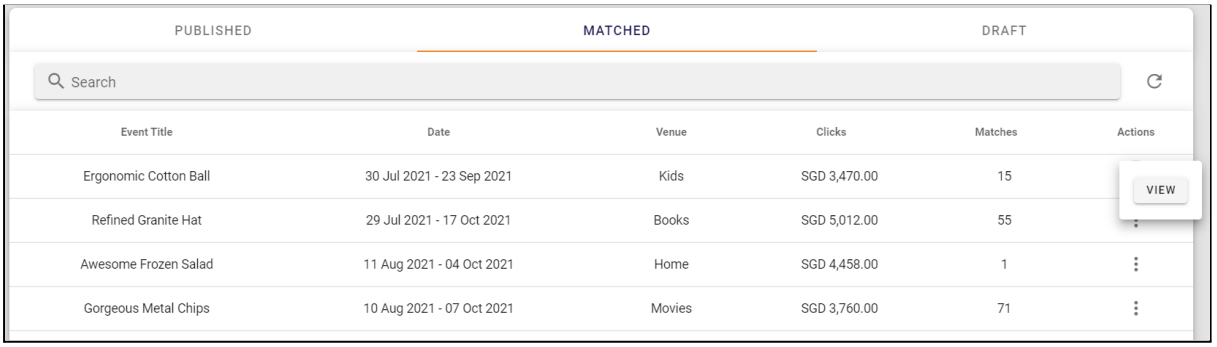
As mentioned, published events cannot be deleted. Diagram as shown below:



Only unpublished events (draft) can be deleted.

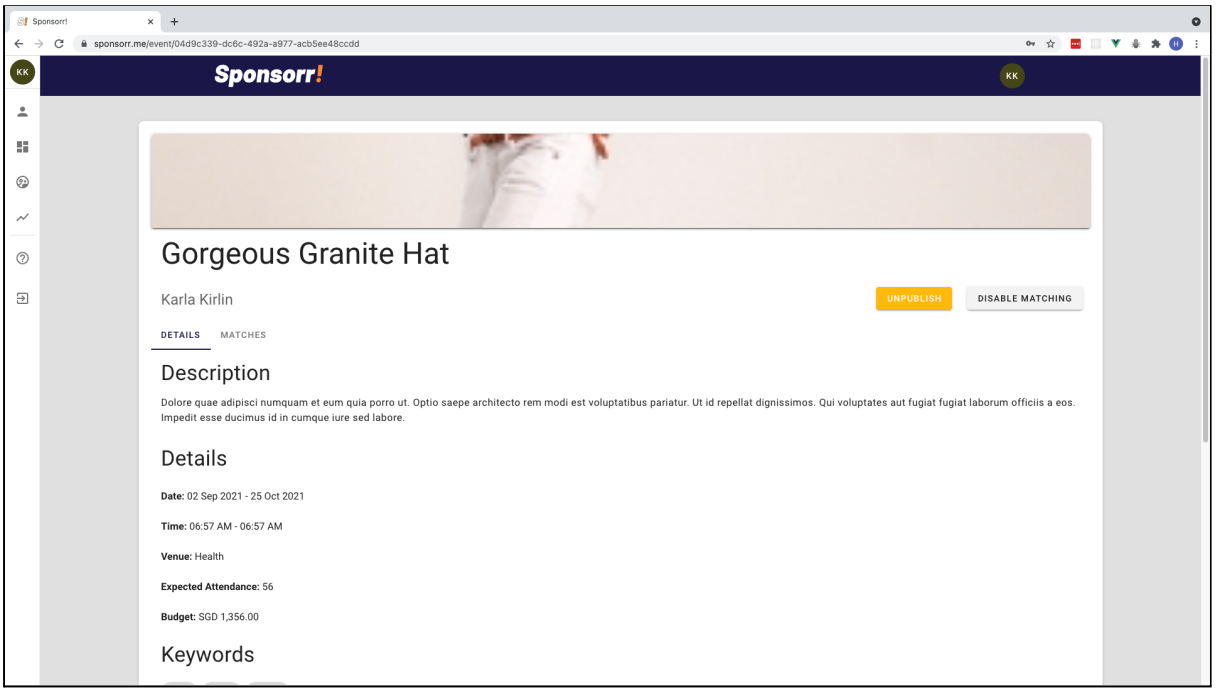


Similarly matched events cannot be **unpublished** nor **edited** to ensure event organisers fulfil their promises to sponsors.

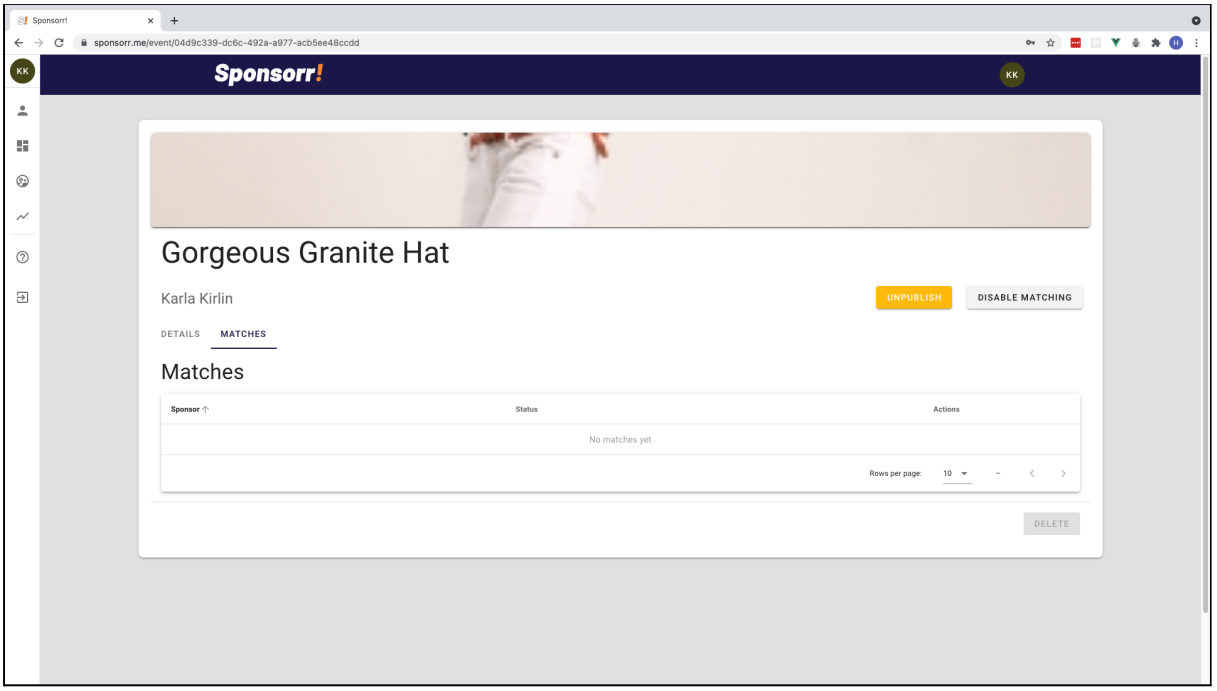


4.3.7 Event Page

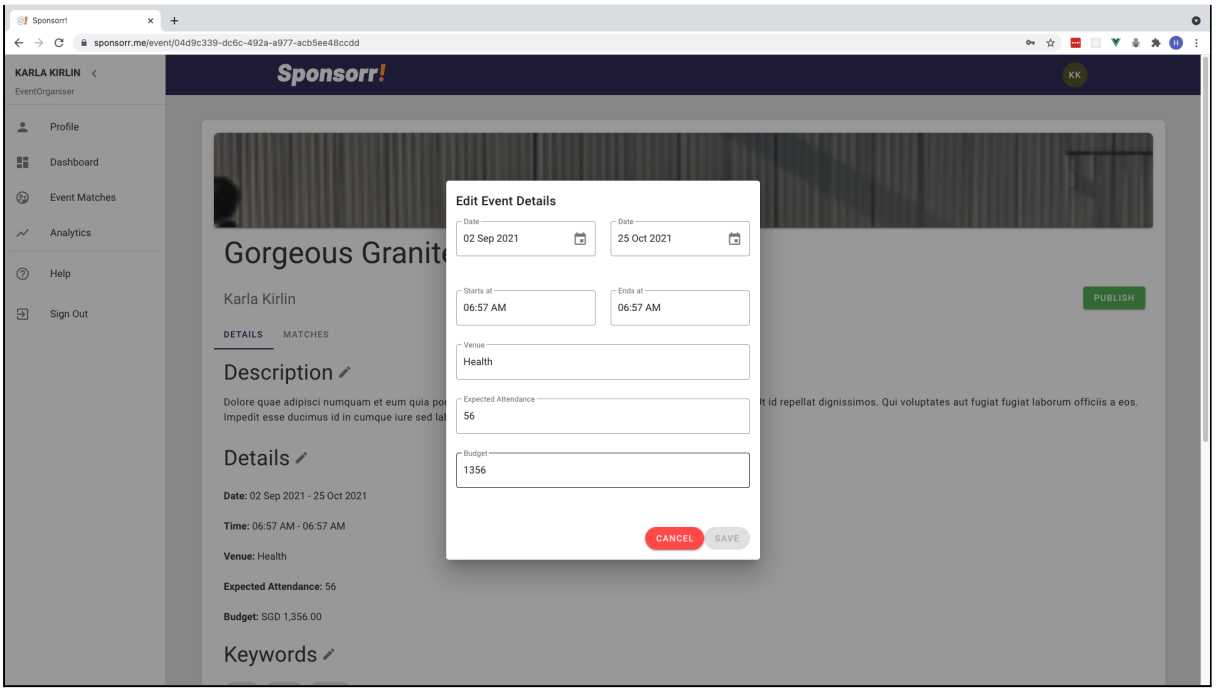
The event page (details tab) shows the details of the event, as well as action buttons to publish, unpublish, delete, enable matching, and disable matching. When users toggle matching, they are actually subscribing to our matching system.



The matches tab lists the matches belonging to the event. The organiser can view the sponsor, the offer, accept, or reject the match.

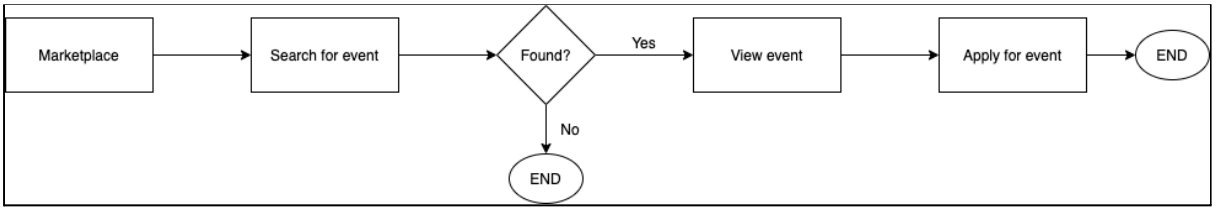


An event organiser can edit the event's details. The editing rules follow as mentioned in [section 4.3.6](#).

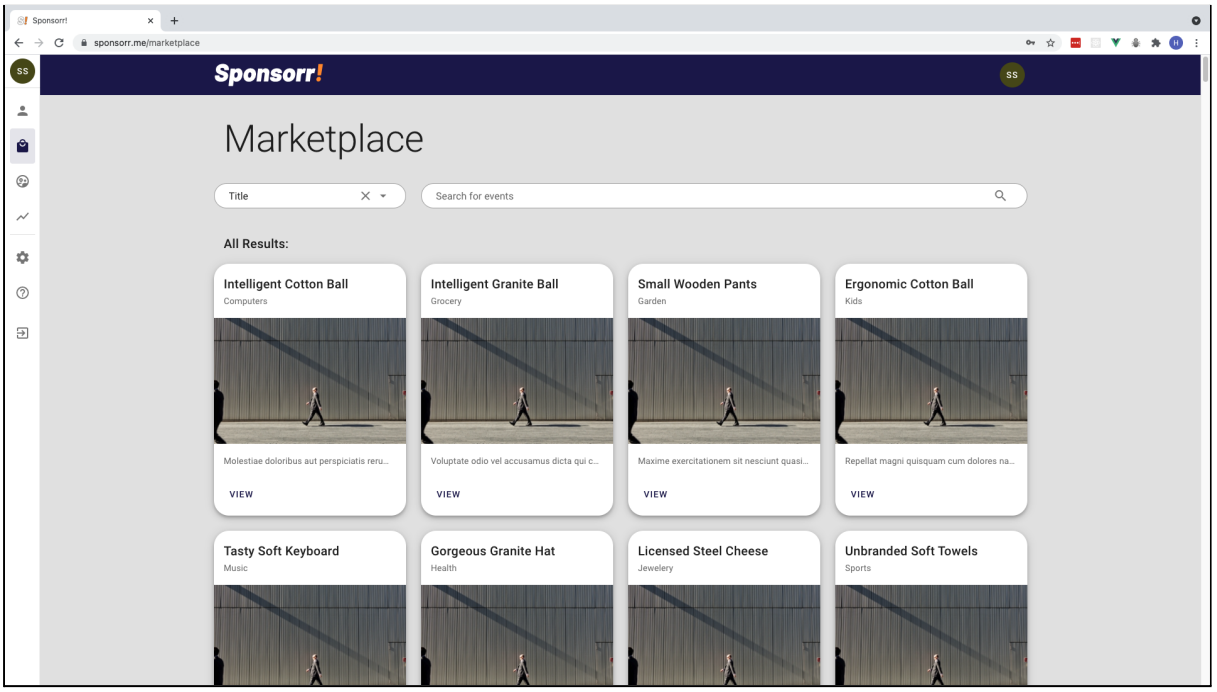


4.3.8 Marketplace

User flow diagram for applying for an event as a **Sponsor**

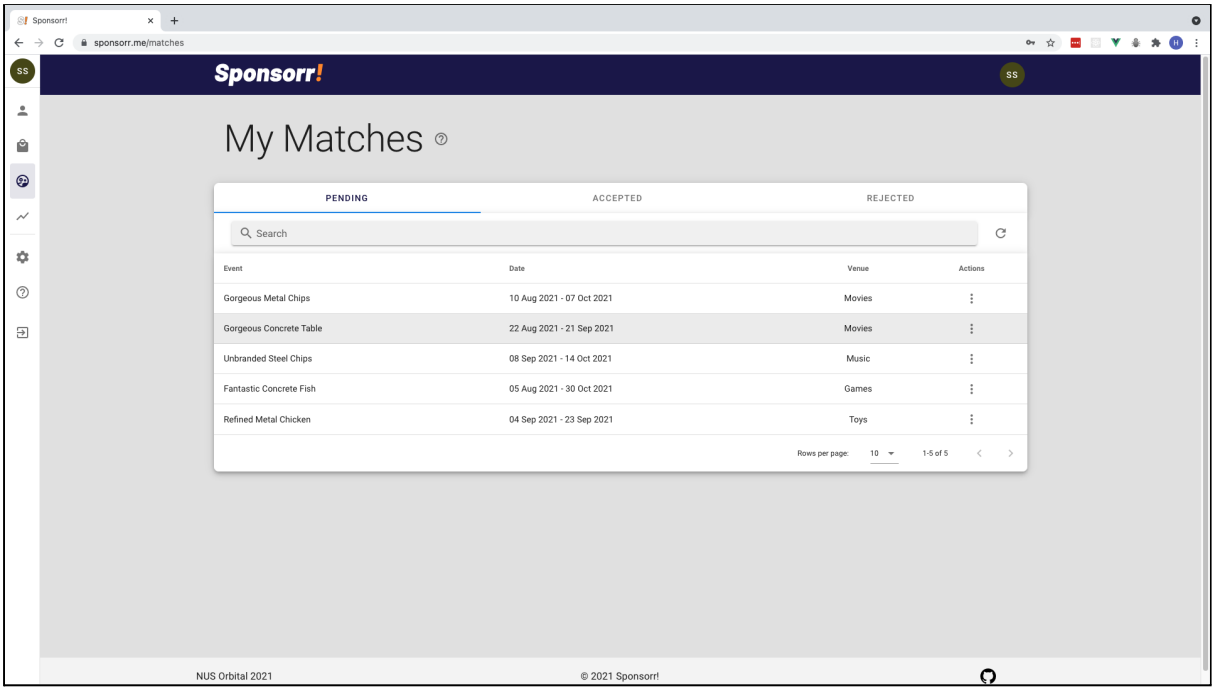


The marketplace allows sponsors to look for published events, either by browsing, or querying by titles, budget, or keywords.



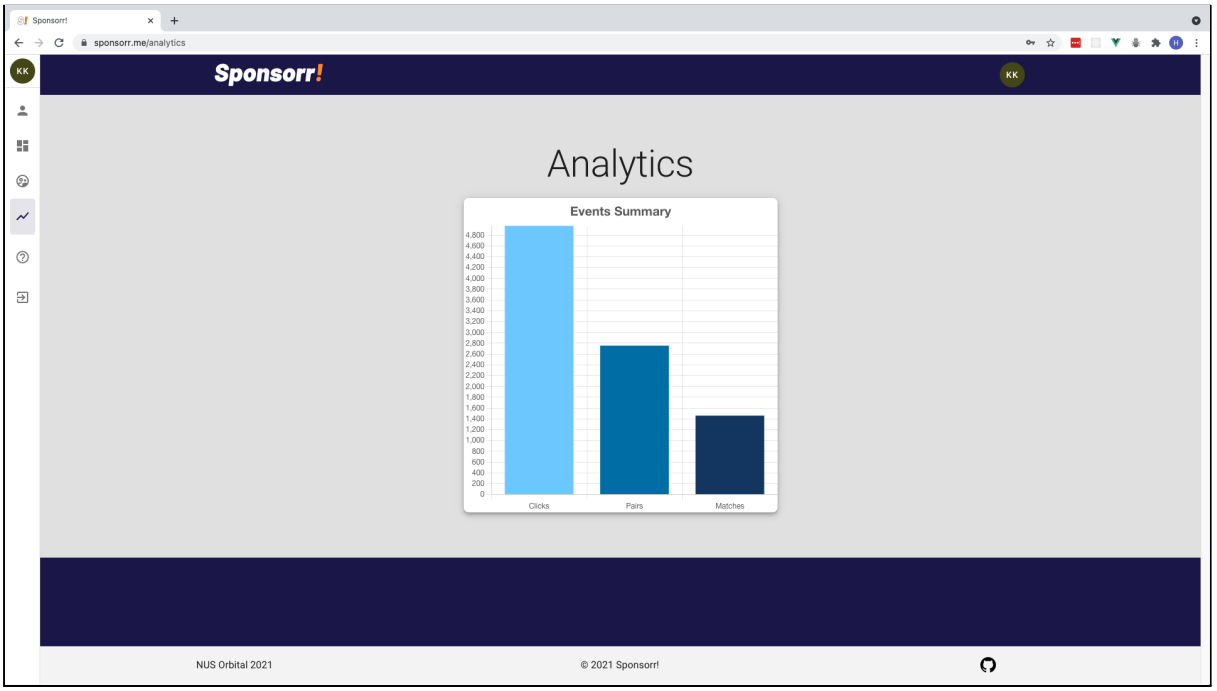
4.3.9 My Matches

Sponsors are able to view all of the matches that they have applied for or have been assigned to them by the system. The matches are categorised into 3 distinct categories (tabs): pending, accepted, rejected. Sponsors can click on the action buttons to accept, reject, or view the matched event.

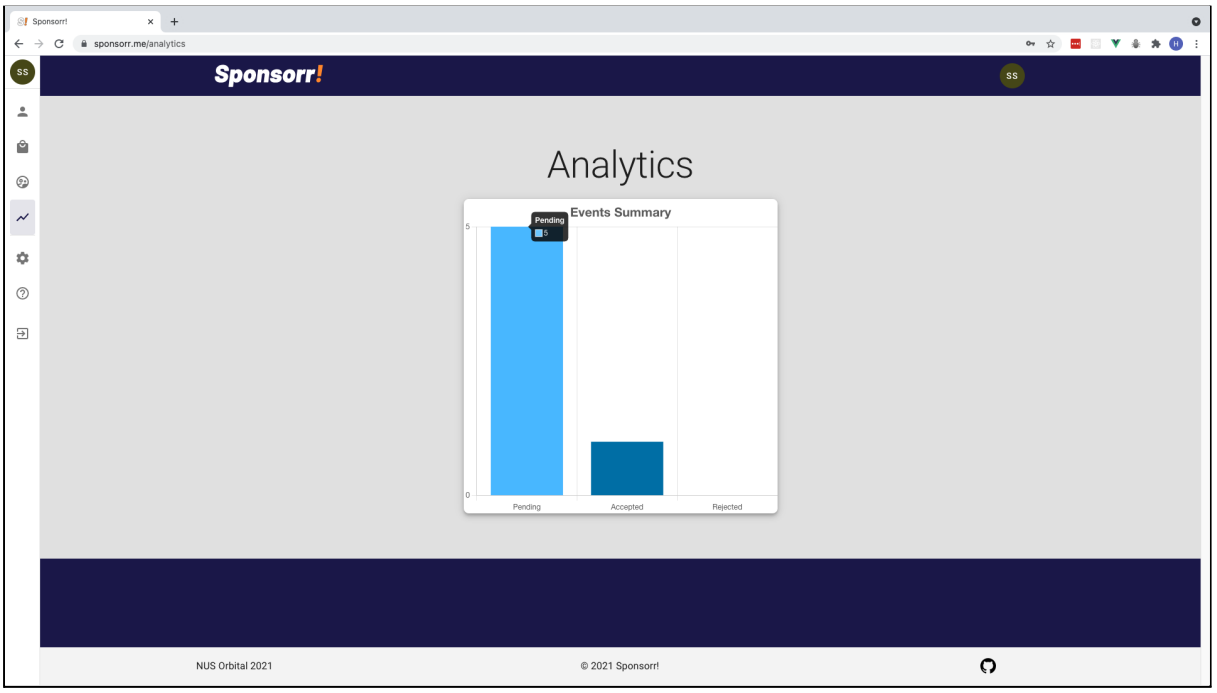


4.3.10 Analytics

Event organisers are able to see the total number of events, pairs, and matches represented in a bar chart.

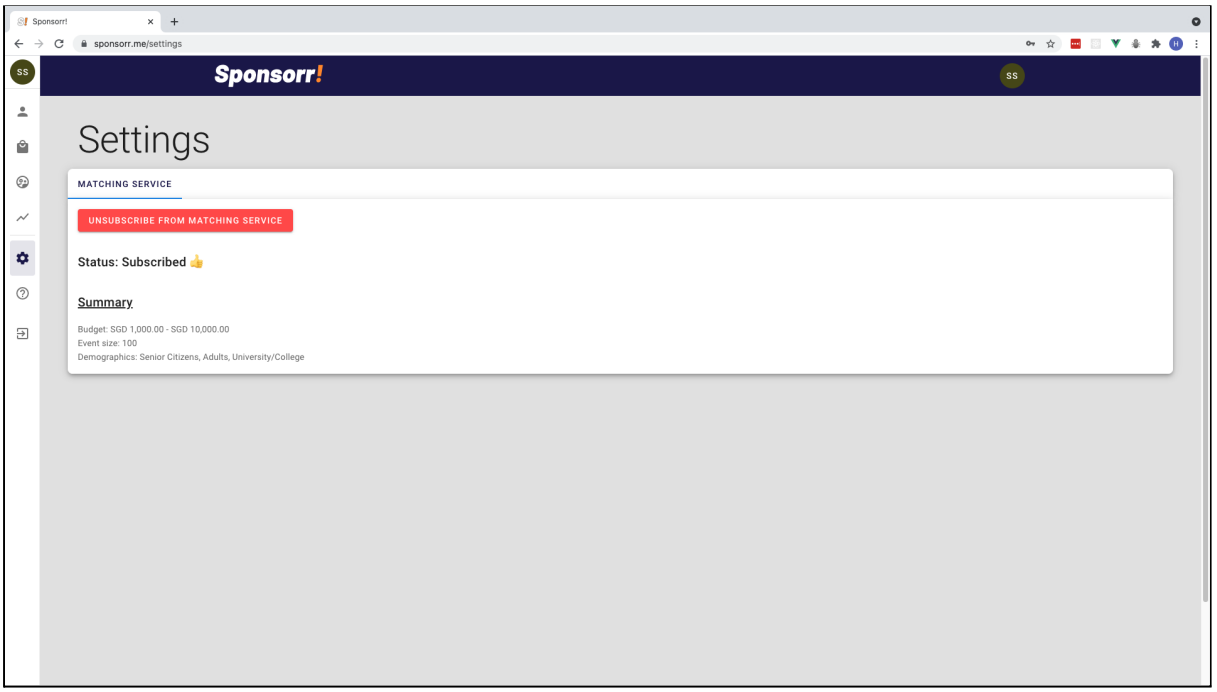


Event sponsors are able to see the total number of matches pending, accepted, and rejected in a bar chart.

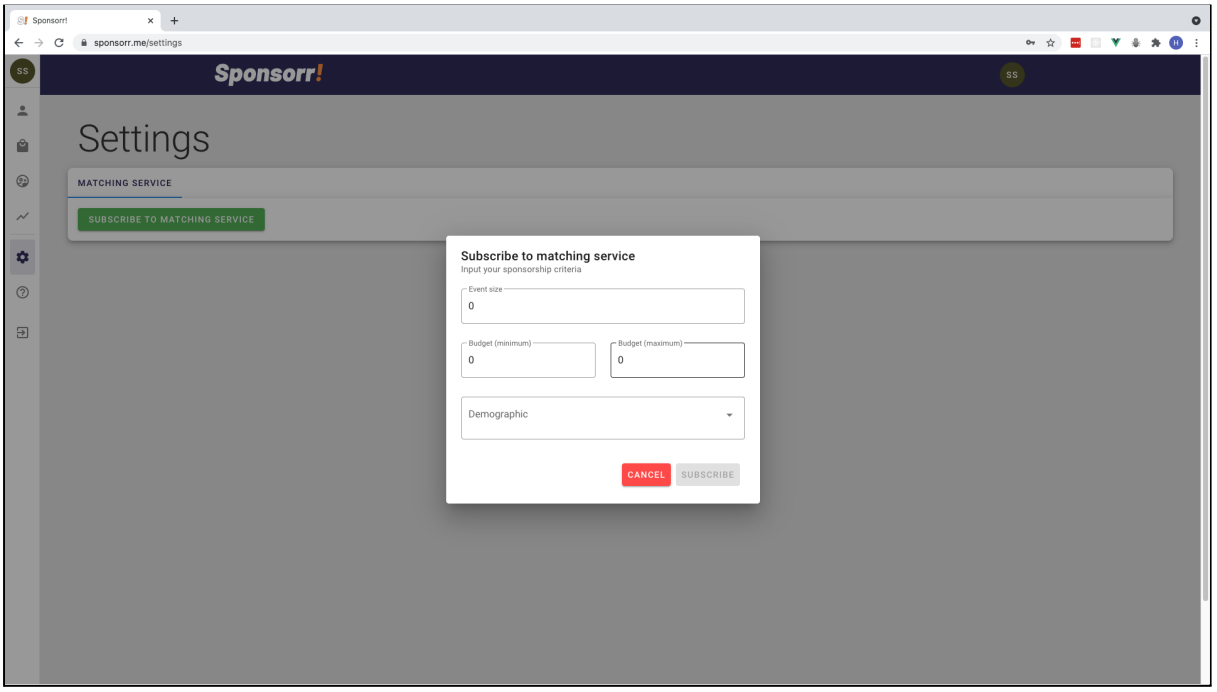


4.3.11 Settings

The settings page allows sponsors to submit their sponsorship preferences to the matching service, or unsubscribe from the service.

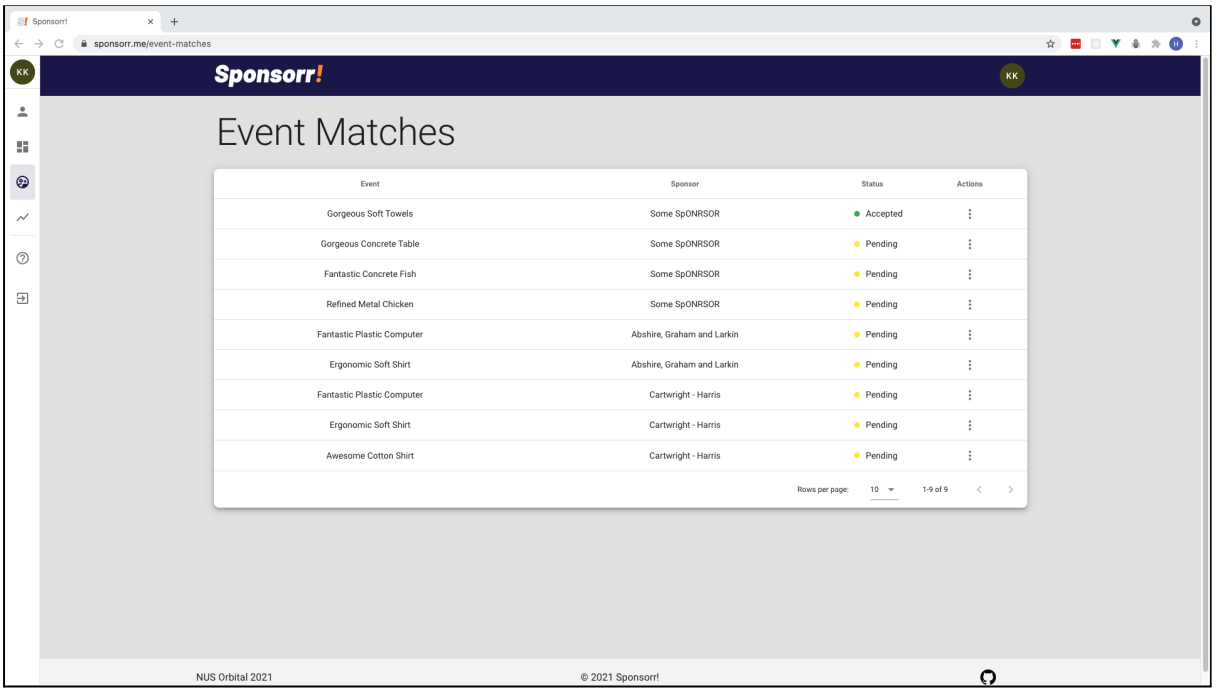


The matching service subscription form.



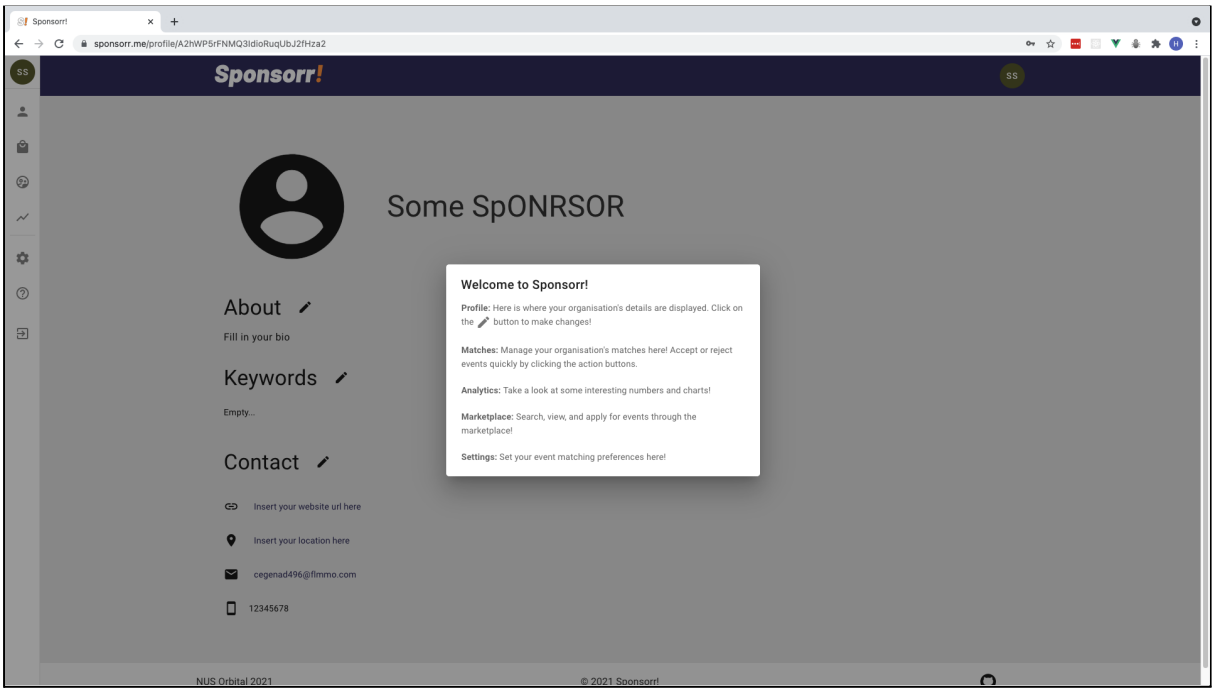
4.3.12 Event Matches

The event matches page allows event organisers to see a list of all the matches their events have gotten, as well as their statuses. Event organisers can accept, reject, view offers, or view the sponsor by clicking the action buttons.



4.3.13 Help Dialogs

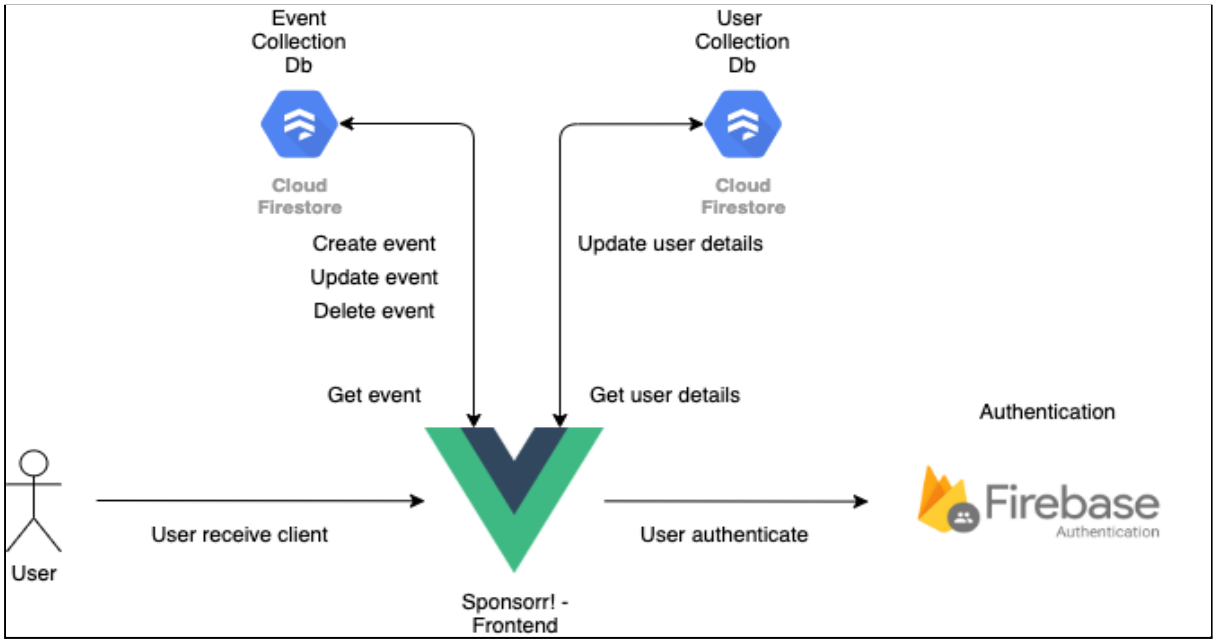
The help popup provides instructions on how to use the application and what each page does. The popup is accessible everywhere in the application upon login and verification.



5.0 Design

5.1 System Design (Server Side)

5.1.1 Authentication



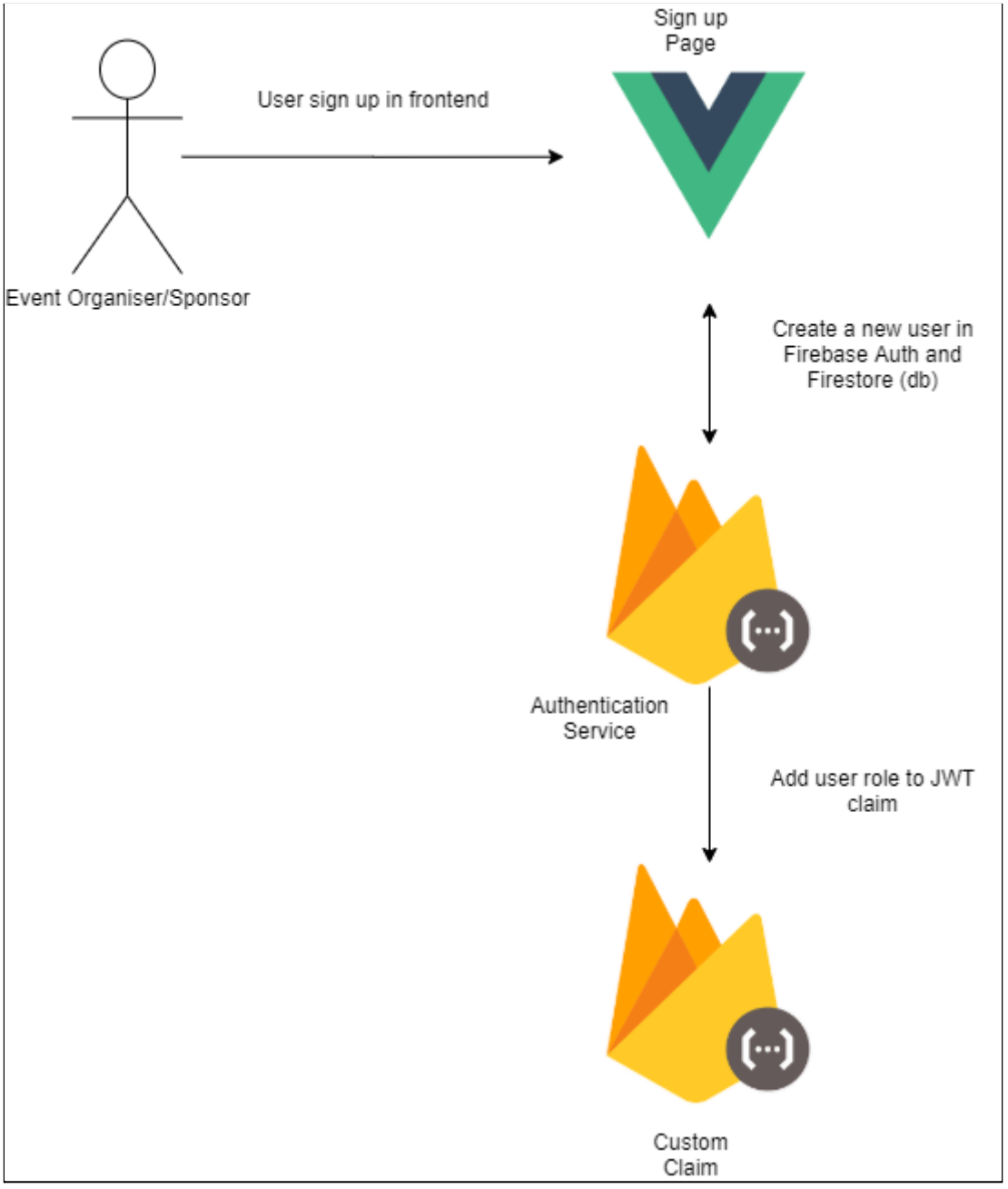
This diagram illustrates the integration between the client and the server of our application. Use this link to access the original diagram with a higher resolution.

5.1.2 Custom Claim

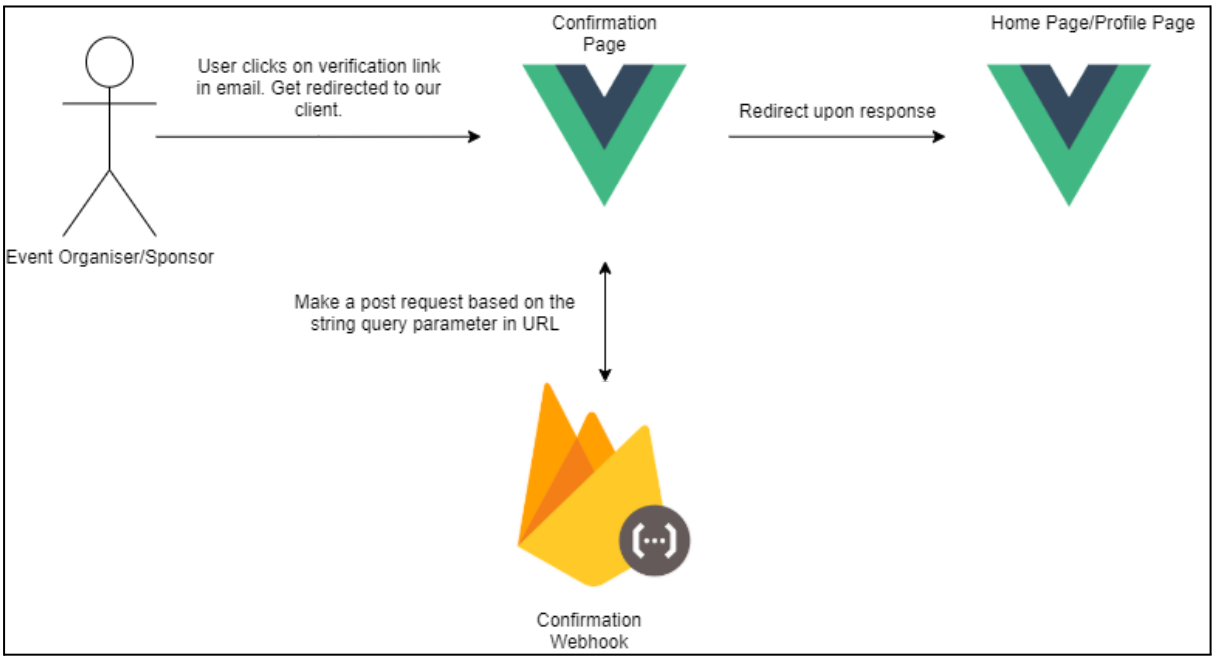
To enable role-based security authorisation to our database, we have to set a role in the JSON Web Token (JWT) that firebase auth is using to authenticate users. This is achieved when a new user signs up and a custom claim cloud function is triggered on the user creation event to add a custom claim to our user record.

Furthermore, this service also handles generating a webhook URL and sending it out as a confirmation email. See below for detailed elaboration.

More information about JWT will be discussed in section 7.5.2.



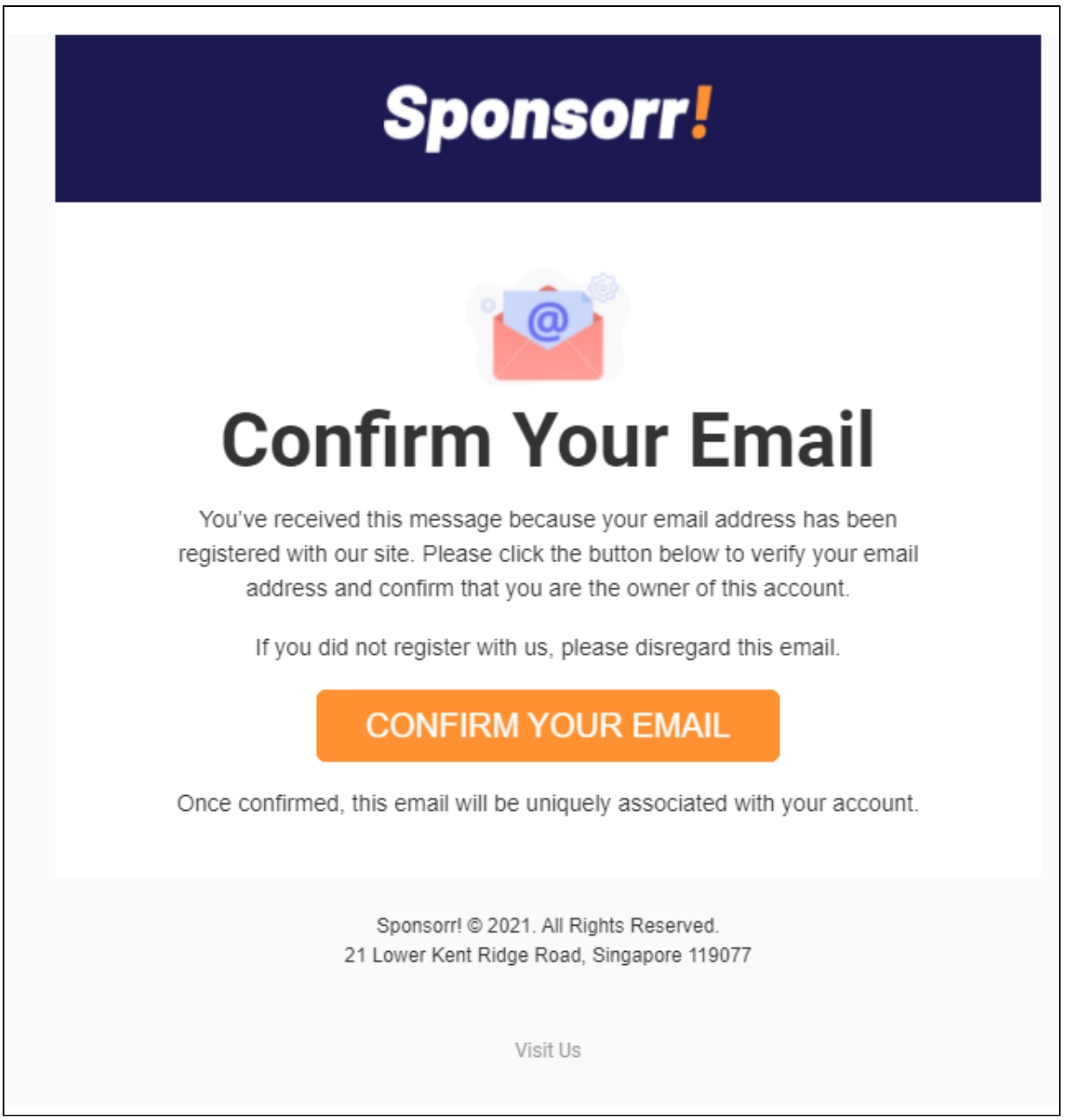
5.1.2 Account Confirmation



This diagram illustrates the system that handles the email verification process upon sign up confirmation. We utilised a webhook design to implement this system.

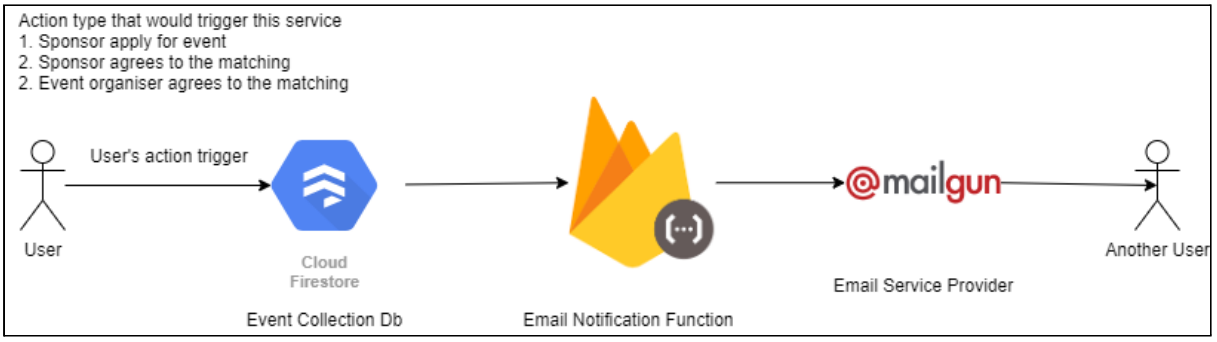
<u>How It Works</u>

1. User receives a customised email as shown below.
2. User has to click on the confirmation button and will be redirected to our client to trigger a webhook.
3. The confirmation button is a redirect URL with the following format *sponsorr.me/confirmation?hash=<signature>&emailAddress=<user email>&id=<user id>*. This information will be used as the body payload when calling the webhook and it is designed in a way to address security vulnerabilities which would be further elaborated in section 7.5.1.
4. Upon receiving a successful response from webhook, users will be redirected back to their profile page in an automated fashion.



Confirmation Email Design

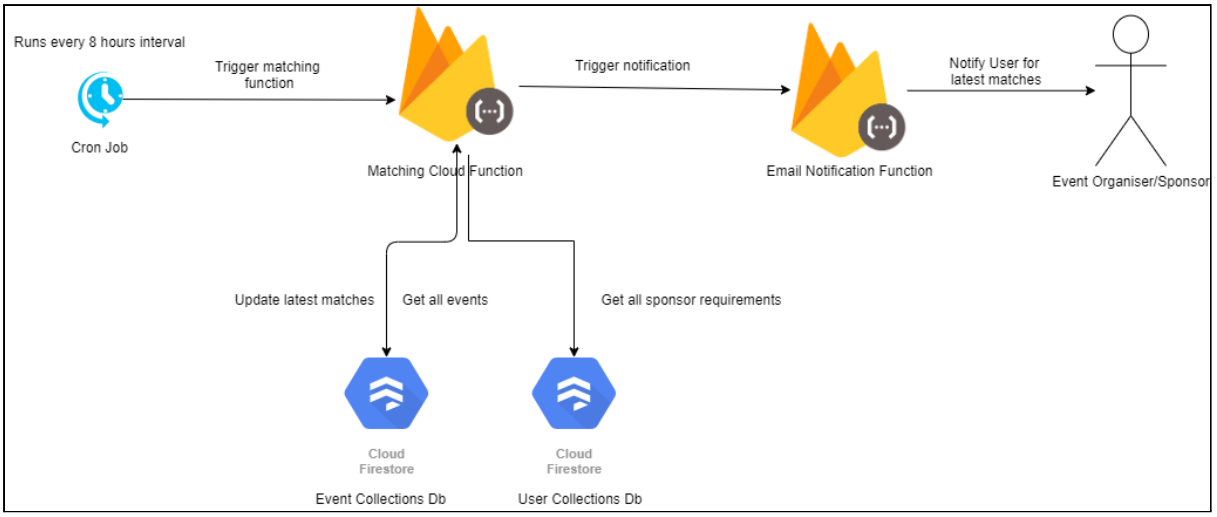
5.1.3 Email Notification System



This diagram illustrates the email notification system based on the user's actions or the cron job action. The email service provider is Mailgun and we use **SMTP** API to send the email out. Three are two services that will trigger this system -- Confirmation Email and the Matching System.

Note: We will also be CC'd in the email sent.

5.1.4 Matching System



This diagram illustrates Sponsorr!'s matching system using a cron job that runs every 6 hours. Use this link to access the original diagram with a higher resolution.

Design Consideration

1. The team has a plan to integrate Machine Learning to optimise the matching algorithm. Hence this system is highly extensible to integrate machine learning in the future.
2. The parameters used are budget, event size and target audience. Here are some of the criteria to get a successful match:
    a. Subscribed: Both event and sponsor must be subscribed to this matching system.
    b. Budget: We assume the event budget is quite flexible since they are being sponsored and have more room for negotiation. Hence we would have +/- 20% Tolerance for deviation. Generally, we would assume event organisers can organise as many events as they want. Adding on to that, the sponsor budget is stricter as it is very unlikely to increase the budget to sponsor.
    c. Demographic: The event demographic must be a subset of sponsors as sponsors do not want to target the wrong audience.
    d. Event Size: This parameter favour the organiser more as this is their event and the sponsor most likely has to accommodate for this.

How It Works

1. Every 6 hours, this service will be triggered and all subscribed events and sponsors will be eligible for the match.
2. Upon successful matching based on the criteria stated above, both parties will receive an email notification using our in-house mailing service. You can find the email design below.
3. What is more interesting about these systems, it is all **automated** without any need of human intervention.

Sponsorr!

Hey Bridget Kemmer, we found you a match!

Some SpONRSOR seems like a good match for Unbranded Steel Chips!

Head over to Sponsorr! now to view your match!

Accept the match...

... to continue communicating with the sponsor

Reject the match...

... if you don't like what you see

To Sponsorr!

© 2021 Sponsorr!

Matching Notification Email

5.1.5 Matching System (Manual Trigger)

The design for this system is similar in section 5.1.4, instead of a cron job it allows admin to trigger this matching system manually via a POST request. However, the team acknowledges that this endpoint would be exposed to the public.

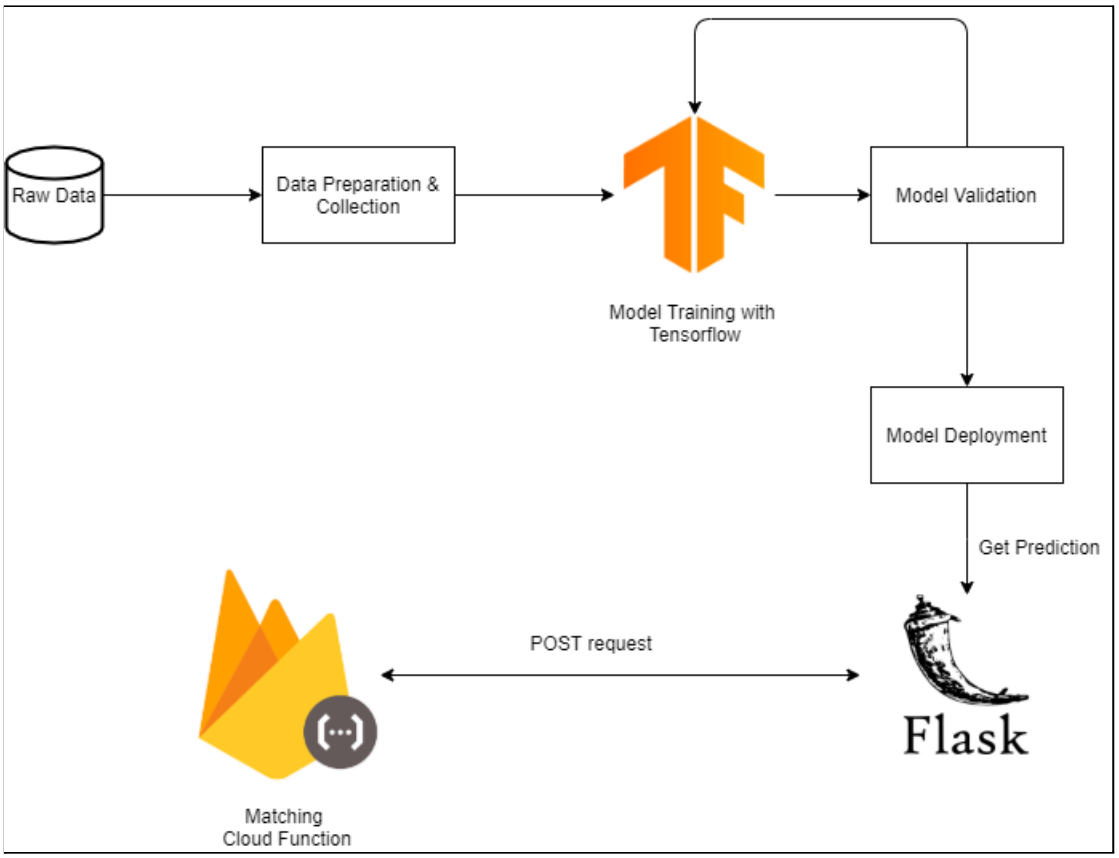
Hence, this endpoint would require an *x-api-key* in the header for validation before it runs. This API token is generated manually by us and is only available to us. Therefore, it provides a maximum-security unless the token is exposed to the public.

5.1.6 Machine Learning Pipeline

Due to time constraints, the team decided to push this extension to post-milestone 3. In the future, we wish to integrate machine learning to help our end-user get a better match and optimise their matching based on analysing the keywords, event descriptions and other static parameters such as budget, preference, etc.

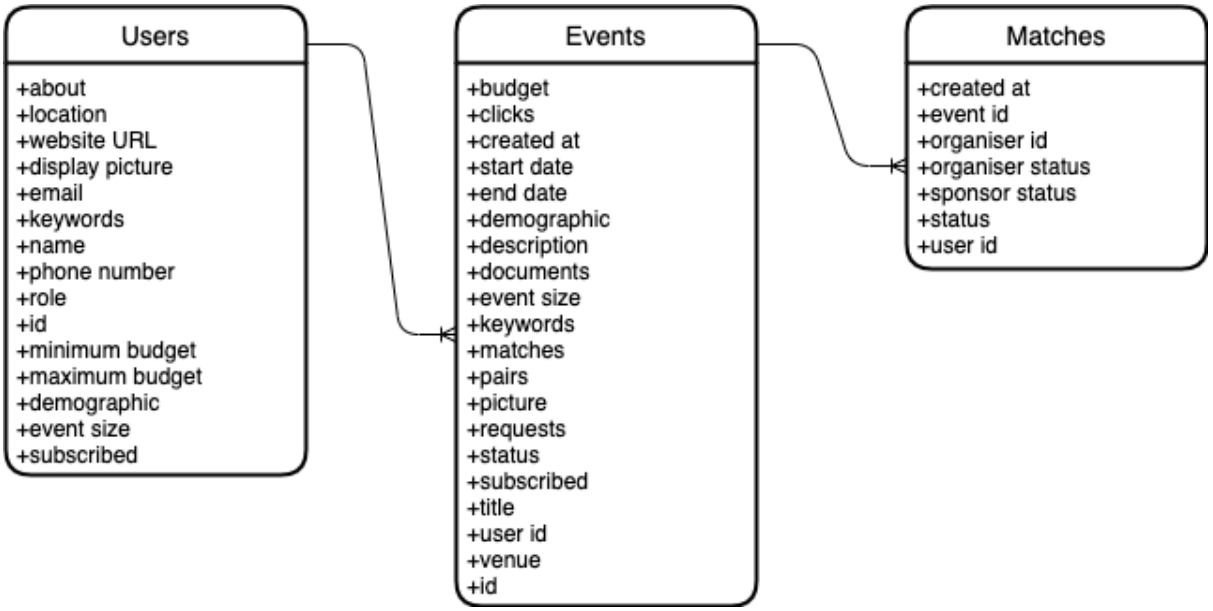
Potential Technology:

- Tensorflow with Keras API for machine learning
- Flask to serve the API for prediction



5.2 Database Schema

The diagram below is the updated schema from milestone 1. The reason for the change is for a better relationship modelling for our matching logic. Here we are reading organiser and sponsor response status to decide the final match status.

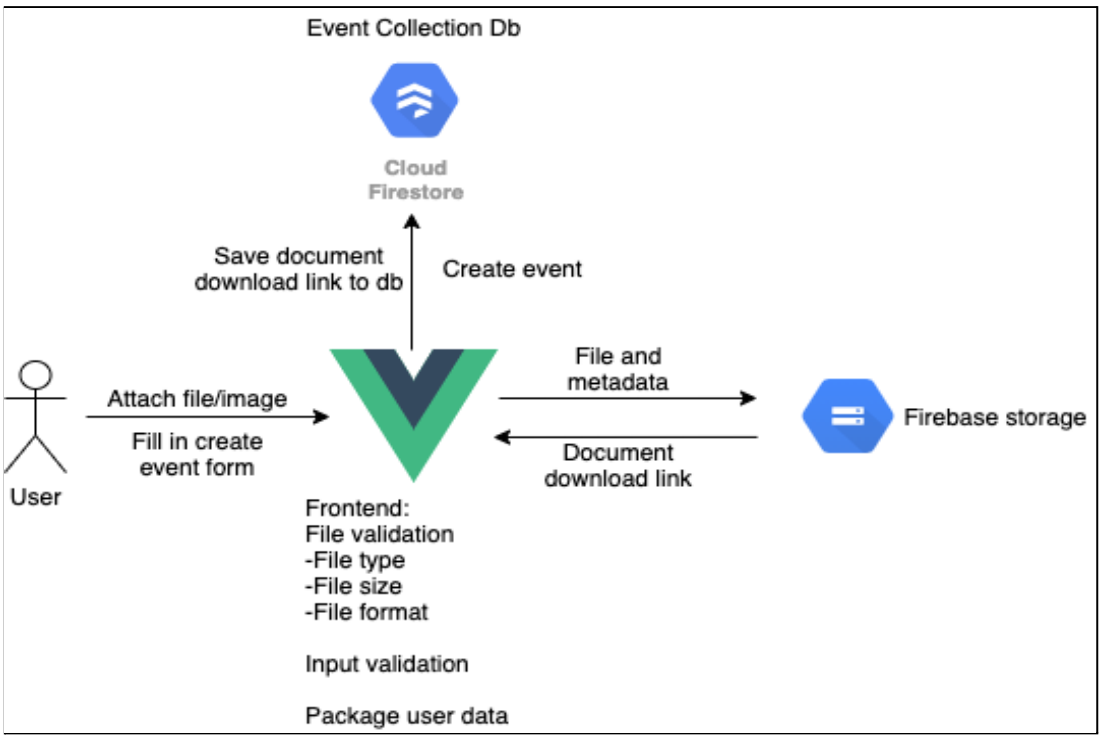


The database schema. Use this link to access the original diagram with a higher resolution.

5.3 System Design (Client Side)

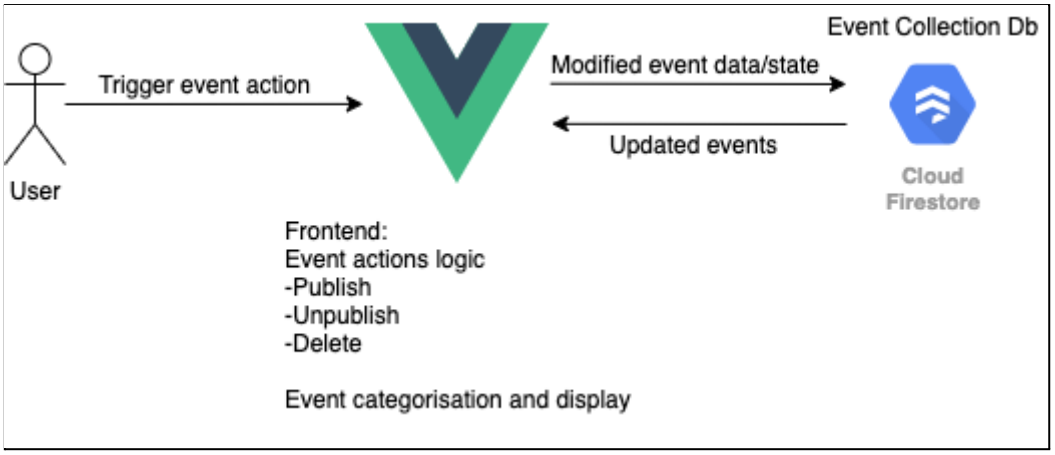
The following system designs are updated accordingly based on the milestone 2 development plan. (Refer to the milestone 1 submission)

5.3.1 Create Event and Upload Document/Images



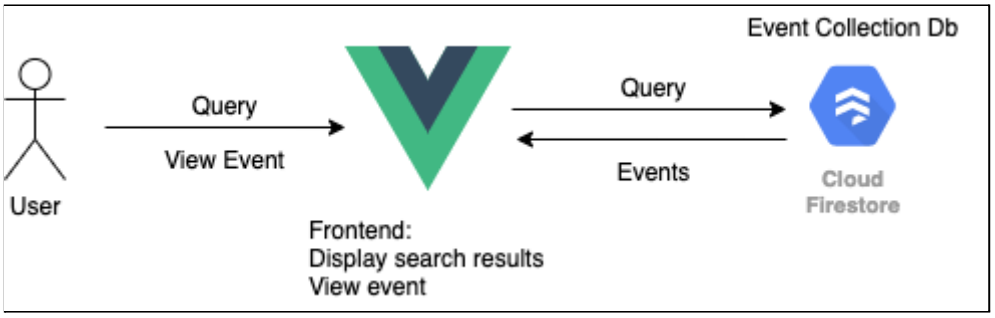
The diagram illustrates the event creation system. Use this link to access the original diagram with a higher resolution.

5.3.2 Dashboard



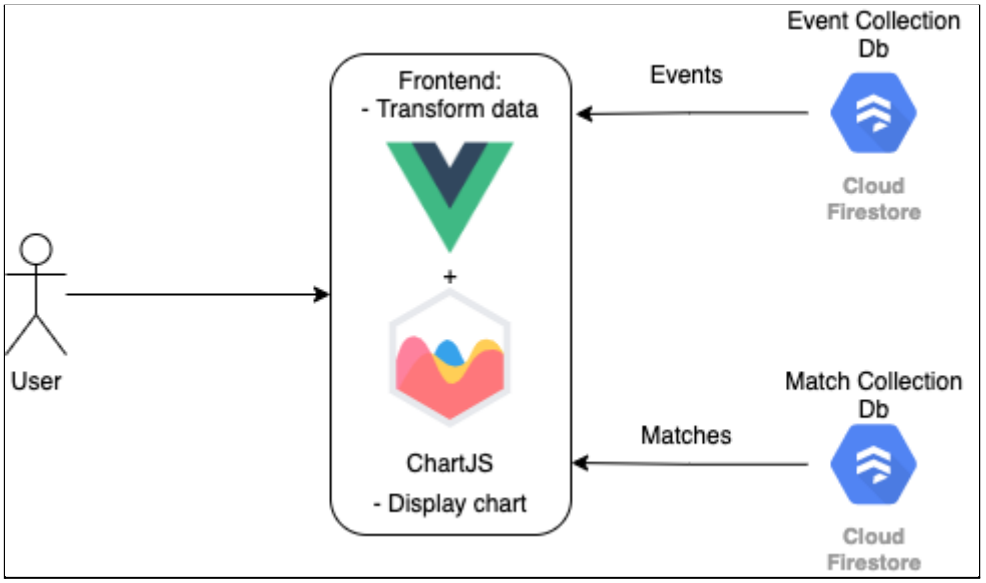
The diagram illustrates the dashboard's system. Use this link to access the original diagram with a higher resolution.

5.3.3 Marketplace



The diagram illustrates the marketplace system. Use this link to access the original diagram with a higher resolution.

5.3.5 Analytics Page



The diagram illustrates the analytics system. Use this link to access the original diagram with a higher resolution.

5.4 Services

As you can see from the way we structure our design diagram, we adopt a microservice architecture to build our application such that it's more **maintainable** and **scalable**.

Here is a list of running cloud functions in our backend.

Function	Trigger	Region	Runtime	Memory	Timeout
confirmation	HTTP Request https://asia-southeast2-sponsorr-prod.cloudfunctions.net/confirmation	asia-southeast2	Node.js 14 Beta	256 MB	60s
customClaimService	document.create users/{userId}	asia-southeast2	Node.js 14 Beta	256 MB	60s
eventStatusChangeServi…	document.write matches/{matchId}	asia-southeast2	Node.js 14 Beta	256 MB	60s
ext-firestore-send-…	document.write mail/{id}	asia-southeast2	Node.js 14 Beta	256 MB	60s
matchCronJobService	0 */6 * * *	asia-southeast2	Node.js 14 Beta	256 MB	60s
matchService	HTTP Request https://asia-southeast2-sponsorr-prod.cloudfunctions.net/matchService	asia-southeast2	Node.js 14 Beta	256 MB	60s
notificationService	document.create matches/{matchId}	asia-southeast2	Node.js 14 Beta	256 MB	60s
populateDbService	HTTP Request https://asia-southeast2-sponsorr-prod.cloudfunctions.net/populateDbService	asia-southeast2	Node.js 14 Beta	256 MB	60s

Function Name	Trigger Event/Access Method	Functionality
confirmation	HTTP POST request	Validate webhook URL and verify the incoming email address.
customClaimService	On new user document create	1. Add role in JWT as a custom claim to provide role-based security access 2. Generate a webhook URL token and send a confirmation email
eventStatusChange	On match status update	Update the event status to "matched" and increment the match count by one (1)
matchCronService	Cron Job	A cron job that runs every 6 hours to find good matches for subscribed users.
notificationServices	On new match creation	Send an email to notify sponsors and event organisers about the new match
populateService	HTTP POST request	Populate our database with seed data and mock users. It is protected with our custom API key.

6.0 Development Plan

6.1 Overview

Key Milestones / Tasks	Key Dates / Deadline	Todo
Milestone 1	31 May 2021	● Project Ideation ● Project roadmap planning ● Wireframing ● Setup development environment and workflow ● Learn technology stack ● Add features: ○ Home Page ○ Authentication ○ Sign Up ○ Login ○ Profile Page ○ Edit Profile ● Milestone report, poster, video, and peer review
Milestone 2	28 June 2021	● Consolidate peer review and draft action plan ● Bug squashing ● Add features: ○ Dashboard ○ Add Event ○ Edit Event ○ Event Page ○ Marketplace ○ My Matches ○ App Navigation ● Milestone report, poster, video, and peer review
Milestone 3	26 July 2021	● Consolidate peer review and draft action plan ● Bug squashing ● Codebase refactoring ● Add features: ○ Analytics Dashboard ○ Settings and subscription ○ Email verification ○ Matching service cron job + automated email

		● User testing and feedback ● UI revamp ● Milestone report, poster, video, and peer review
Final Rush Checkpoint	11 August 2021	● Tie up loose ends, optionally implement feature extensions
User Testing and Review	12-16 August 2021	● Project poster ● Project video ● End-to-End testing
Final submission	16 August 2021	● Submit project poster, video, and code
Splashdown	23 August 2021	● Presentation + END

6.2 Consolidated Timeline

Key Milestones/Tasks	Task Breakdown
Application Development 1	
Home Page	The home page will give a good overview of how the app works and instructions to use the application.
Authentication page (Login & Sign Up)	The team has designed the authentication flow of the application that supports role-based authentication and guarding unauthorised users against accessing protected pages.
Profile Page	The profile page allows users to customise their details such as display pictures, descriptions and contact details.
Application Development 2	
Event Creation	As an event organiser, the user is able to create and publish their event in a streamlined manner and pitch it to potential sponsors. This event creation flow is integrated with saving event data to our database and uploading documents, pictures to our storage system.
Event Management	We decided to go with a dashboard view to allow our users to manage their events all in one place. This allows users to have full control over the existing events. Users are able to do a simple query search by event title in the dashboard and reload to fetch the latest match updates.
Marketplace	The marketplace allows sponsors to find potential event partners with just a few searches away. The current marketplace supports client-side fuzzy search. Hence it has a wider coverage to find relevant events. Furthermore, sponsors can inspect the event with just a click away. This allows them to make a better decision when picking an event partner.
Matching Dashboard	This dashboard allows users to manage their current matches and decide to accept/reject the match. This will enable users to manage all matches in one place.

Application Development 3	
Matching Subscription Service with Email Notification Service	This service enables users to receive matches that fit their requirements and users will receive email notification when a match is triggered.
Matching Cron Job Service	This cron job service will run every six (6) hours to scan through all events and users database collections for potential matches.
Analytics Dashboard	As you can see from the database schema, the application is able to record the number of views and clicks based on user interaction with the application. This extension will display a simple analytics dashboard for the Event Organiser to review their event performance.
User Acceptance Testing	A user feedback session was conducted in this phase via a 1-to-1 interview method to observe user behaviour when using our application. We have since collected user feedback at the end of each interview and consolidated a report in section 7.0.
Application Testing Plan	The testing plan for our projects include but is not limited to 1. Unit Testing/Integration testing For unit testing and integration testing, it is mainly focusing on utility function, database methods and components with a single responsibility. Integration testing will include components that interact with more than two (2) components. 2. End-to-End Testing At this stage of testing, we will be simulating user behaviours including clicks and inputs. This will give us better test coverage for our application and sieve out as many bugs as possible before production. 3. API Testing As our application is using cloud functions for some of our services (email, matching). The team plans to use **POSTMAN** to simulate some of the processes and interactions with the cloud function. 4. Others Aside from the three plans mentioned above, the team will look into generating test coverage reports to evaluate

	the performance of our tests and increase coverage.

*Extensions are subjected to change based on the team's progress on other core features.

6.3 Future Extension

Due to time constraints, there are some features that the team has yet to implement. However, these would not affect the deliverables of our project as they are mostly optimisation or feature extensions.

Key Milestones/Tasks	Rationale/Task Breakdown
Machine Learning	As stated previously, the machine learning model can be easily integrated with our existing matching cron job service as it can be packaged as a separate API as designed in section 5.1.6.
SMS Notification	We realised that email notification is a great value-add to our application to notify our users when they have a match. However, this is less optimal as compared to an SMS notification as users are less likely to check their email from time to time. The team will be looking at SMS service providers to leverage their service and integrate it into our existing system.
Event Categorisation	While our marketplace provides all events to our end-user. However, it would be great to have a categorised view to improving our sponsor's user experience. This would require a slight modification in our database schema and an additional view in our client.

7.0 Technical Aspect

7.1 Technologies

Stack	Description/Rationale
Frontend - VueJS with Typescript	The team is using VueJS as a frontend framework with typescript enabled for type-safety checking and most importantly to leverage the code completion and IntelliSense feature. It helps us to avoid unnecessary bugs during run time.
Backend - Firebase	The team is using Firebase for authentication and authorization service and Firestore as a database. ● Smooth integration with the client and support for role-based authentication ● Flexibility in defining user access permission with firebase rules ● Cloud Function ● Blazing speed when integrated with VueJS
Wireframing - Figma	The team used Figma to do the wireframe for the entire application.
Version Control - Git	The team is using Git for version control. Some frequent commands that we use are merging, rebasing and reverting commits.
Project Management - Github	The team is using Github Projects as a tool to manage the project. We leveraged the kanban board to keep track of our application development tasks and progress. Future plan for the team is to integrate CI/CD pipeline via Github Action.

7.2 Production Application

A production version of the web application has been deployed using Firebase Hosting via CI/CD GitHub Action:

You may find the latest version here https://sponsorr.me/.

In milestone 1, visitors can explore the landing page, create an account with a role and sign up, log in, view your profile page, edit the contents of the profile page, and sign out.

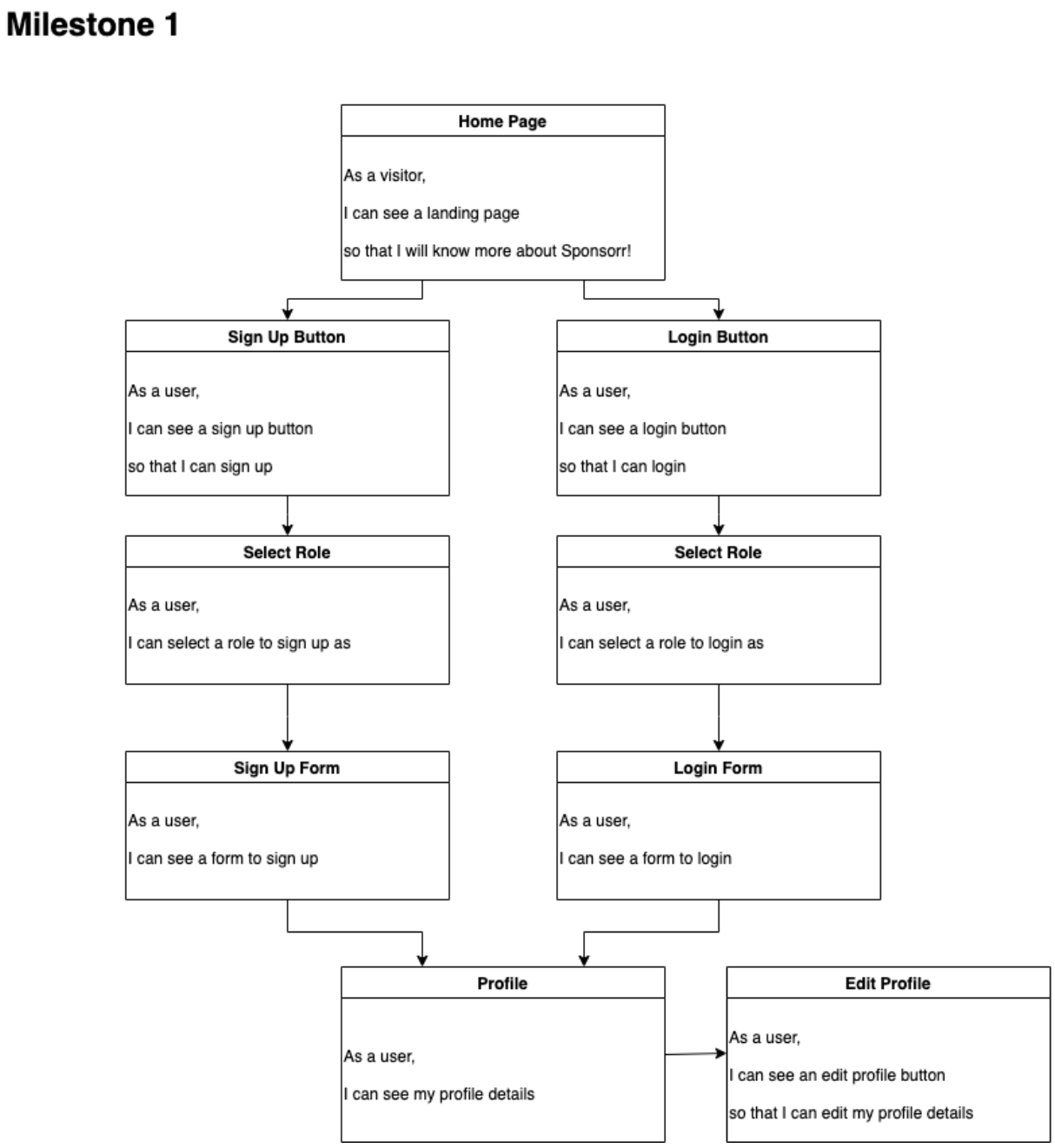
In milestone 2, event organisers can create an event, upload documents and pictures, manage events via the dashboard and publish/unpublish/delete their events. While sponsors can search events via the marketplace through client-side fuzzy search to view and apply.

In milestone 3, the team has extended most of the features in milestone 1 and 2, these include verifying email, analytics page, matches view and revamped home page design. The team also did acceptance testing with 8 different users and has made further optimisation and changes based on the feedback and results. You may find a detailed user feedback report in section 10.0.

7.3 System Documentation

7.3.1 Milestone 1

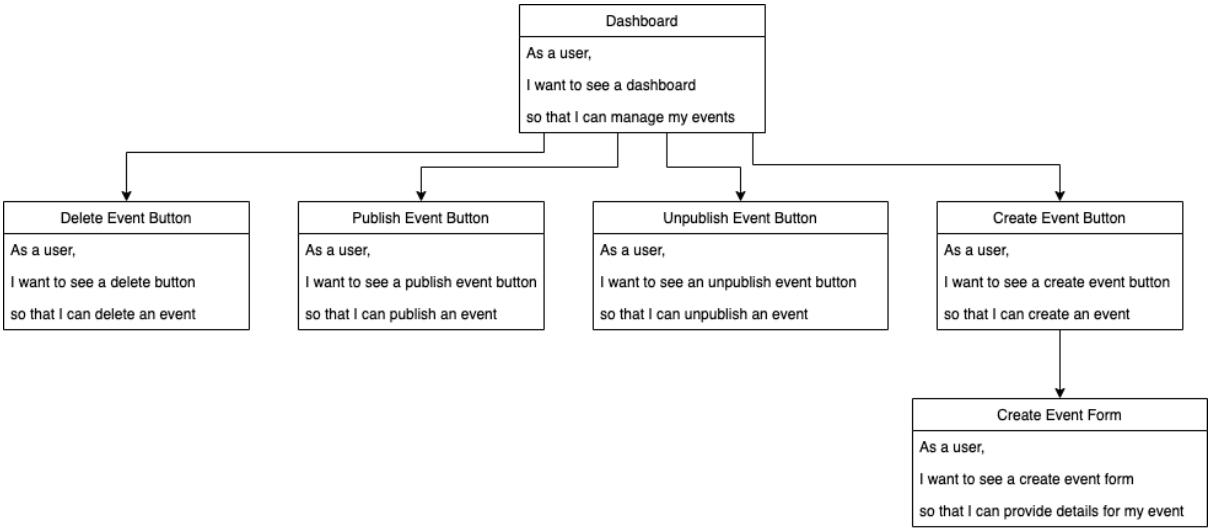
Diagram below shows our current user authentication system



7.3.2 Milestone 2

Event Dashboard

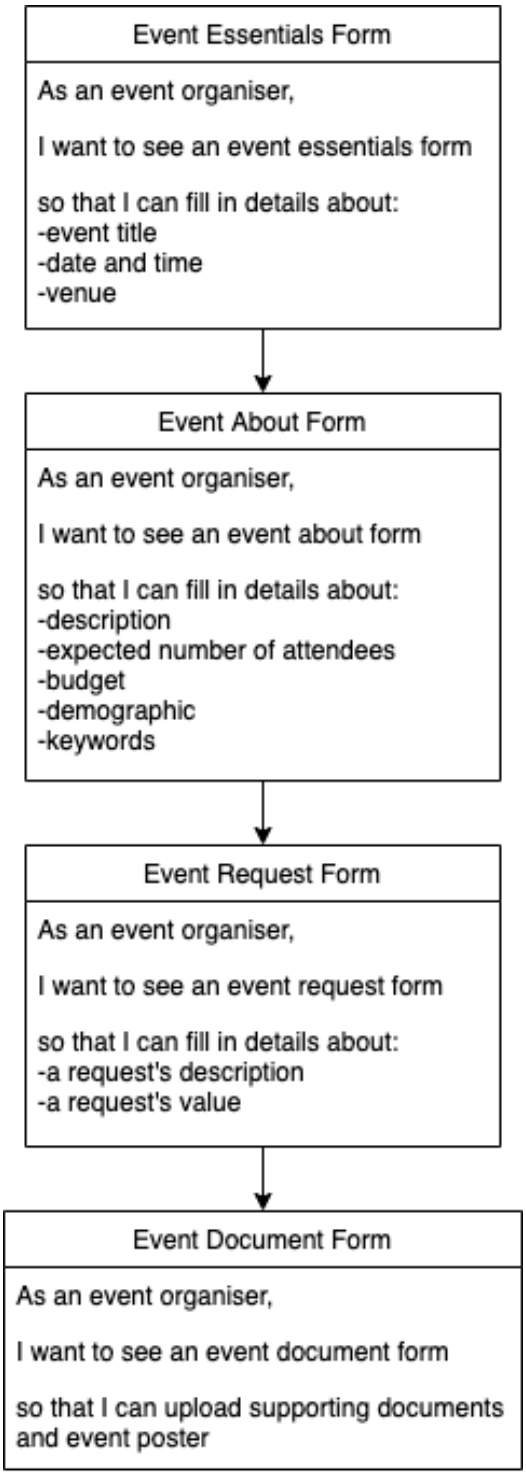
Diagram below shows our current progress for event management via the dashboard



Click the link to see the original diagram with a higher resolution.

Event Creation Form

The diagram below shows our current progress for event creation

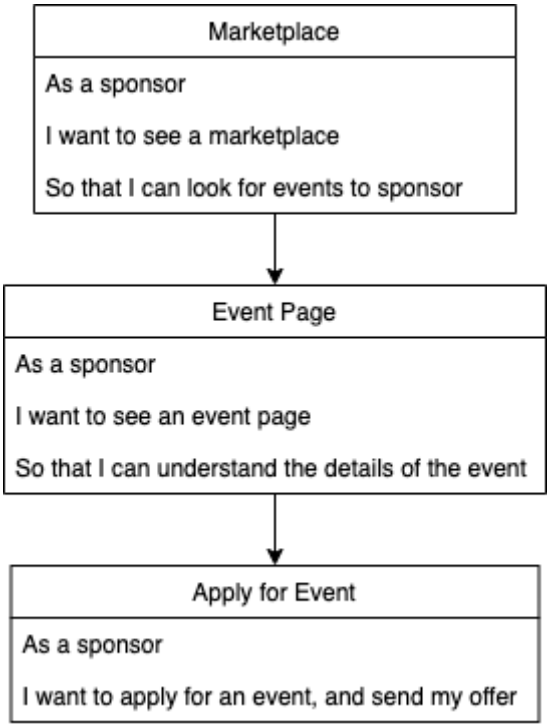


Click the link to see the original diagram with a higher resolution.

7.3.3 Milestone 3

Event Application

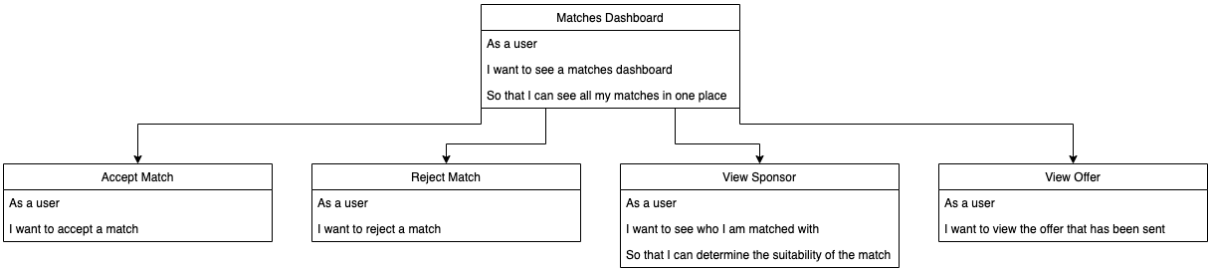
The diagram below shows the process for sponsors to apply for an event



Click the link to see the original diagram with a higher resolution.

Match Actions

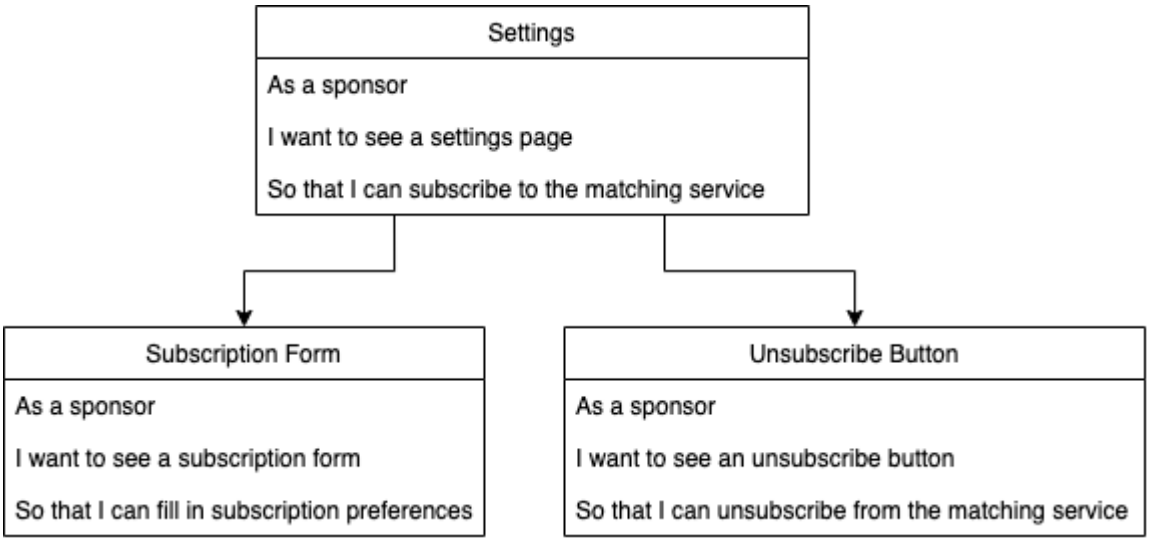
The diagram below shows the process after receiving a match



Click the link to see the original diagram with a higher resolution.

Subscribe User to Matching Service

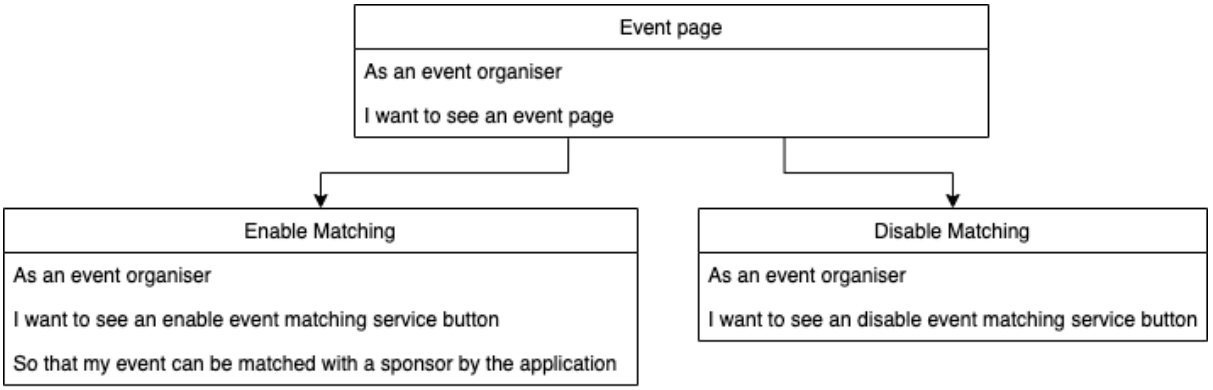
The diagram below shows the process of subscribing to and unsubscribing from the matching service



Click the link to see the original diagram with a higher resolution.

Enable Event Matching Service

The diagram below shows the process of enabling and disabling an event's matching service

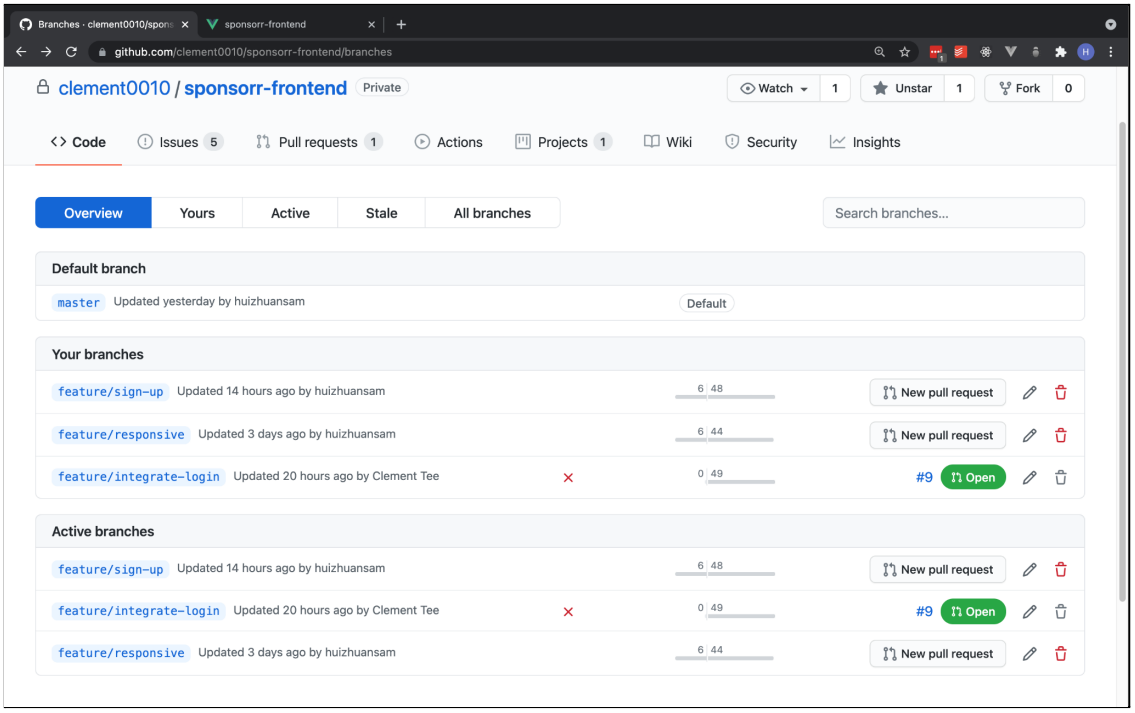


Click the link to see the original diagram with a higher resolution.

7.4 Demonstration of Best Practices

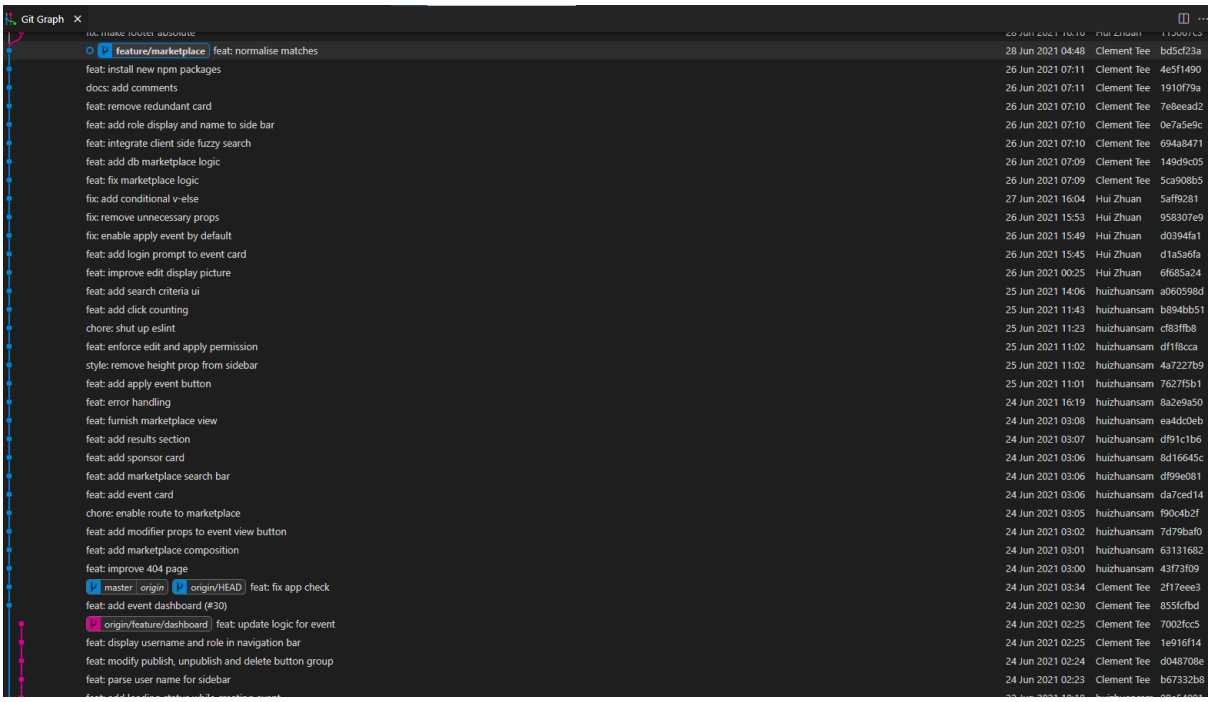
7.4.1 Version Control

- The team has been using the git branching to develop features, where each new feature has its own branch and follows a specific naming convention, i.e. feature/<name of feature>.
- Once a feature has been completed, a pull request is filed and the team reviews the feature. Once the feature is deemed completed, it is then merged into the master branch on the remote Github repository.
- The team uses git rebase on the master branch for a cleaner and more understandable commit history.



The screenshot above shows the branch naming convention we have adopted

We got a very clean tree as we follow the rebasing methodology to keep the commit records clean and formatted in a single timeline.



The screenshot above shows the git graph of our project.

The standard workflow for the team:
Run this in bash/terminal before running anything important:

- *git status*

Working with the master branch

- Before Starting
  git pull --rebase
- Pushing Changes
  git add [filename]
  *git commit -m "[Commit message]"**
  git status
  git push

Note*: Refer to section 7.4.5 for commit message requirement

Working With Other Branches

- Creating New Branch
  git checkout -b feature/[name of feature]
- Moving Between Branches
  git checkout [name of branch]

SOP After Making Changes
Important: Create a new Pull Request. Once all conflicts and issues have been resolved, proceed to merge.

Alternative Merge Method
In your master branch:

- *git pull --rebase*

In your development branch:

- *git merge master*

In your master branch:

- *git merge branch*

Note: Only do this when you are absolutely sure that there will be no merge conflicts. Otherwise, always do a pull request. This is for convenience and to prevent confusion.

Resolving Merge Conflicts

1. Notify all parties affected
2. Resolve conflict together using Github's online **Web Editor**
3. Delete redundant files. This is also an effective way to resolve merge conflicts

Deleting Branch After Merge
Three choices for deleting development branches:

1. Delete via Pull Request
- When confirming merge to master using pull requests, select the option to delete a branch

2. Delete via Terminal
- To delete remote branch:
    - *git push origin --delete development*
- To delete local branch:
    - *git branch -d development*

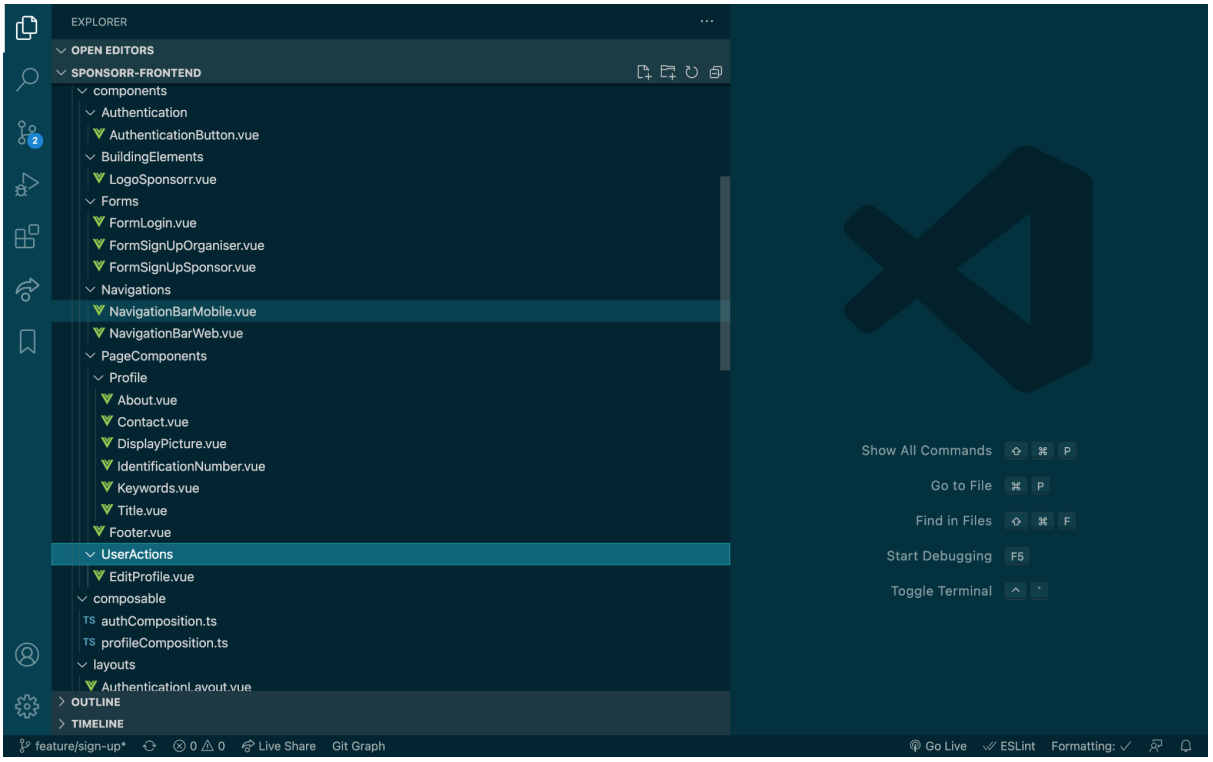
Development Workflow

1. File an issue (see Project board)
2. Open a new branch and work on the issue (follow branch naming and commit message conventions)
3. Pull request and code review
4. Merge and close issue*

Note: Branch can be deleted safely after merging

7.4.2 DRY

The project's frontend design focuses on creating reusable components and UI, resulting in writing less repetitive code. The team also uses composable functions that are reusable across the application.



The screenshot above showcases part of the project's file structure. Individual components and features are written in their own separate files and can be composed and reused together to create another component or view.

7.4.3 SOLID

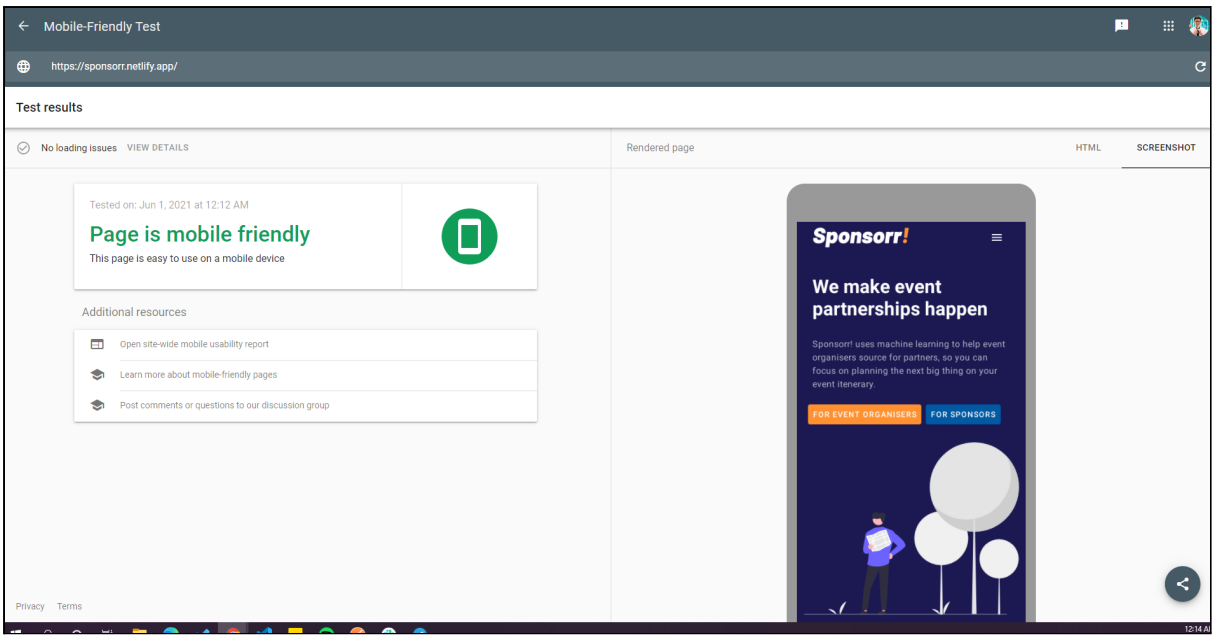
Similarly to Milestone 1, the team uses the single responsibility design principle when working with the CRUD logic for the event. As you can see here the *fetchUserEvent* is only responsible for fetching the user event from the database; *createEvent* is responsible for creating the user event in the database. The usage of this principle leads to a more maintainable and extensible codebase.

```
export default function useEvent() {
  const loading = ref(true);
  const error = ref(false);

  const event = ref<SponsorEvent>();

  const createEvent = async (newEvent: SponsorEvent): Promise<void> => {
    try {
      loading.value = true;
      await createEventToDb(newEvent);
    } catch (err) {
      console.error(err);
      throw new Error(err);
    } finally {
      loading.value = false;
    }
  };

  const fetchUserEvent = async (eventId: string): Promise<void> => {
    try {
      loading.value = true;

      const userEvent = await getEventFromDb(eventId);

      if (!userEvent) {
        throw new Error('Failed to fetch event');
      }

      event.value = userEvent;
    } catch (err) {
      console.error(err);
      error.value = true;
    } finally {
      loading.value = false;
    }
  };
```

7.4.4 Mobile-first design approach/Responsiveness

The team prioritises mobile users and strives to make sure the UI components are compatible across all screen sizes so that their experience is seamless. The rationale behind this approach is that we want our end-user to be able to access our application from anywhere and anytime.

We use google mobile-friendly test to check whether we meet the requirement and the result is as follows:



7.4.5 Linting

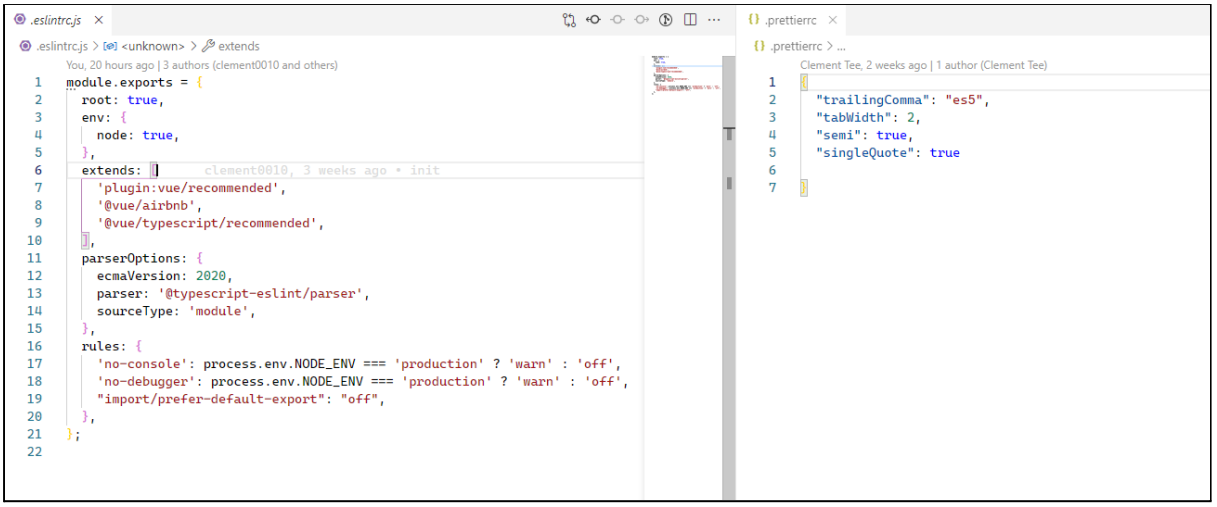
1. Commit linting

The team has decided to use [commitlint] as an avenue to streamline our commit message to avoid unnecessary confusion. The accepted commit message is of this format: **type(scope?): message**. Here are some of the types that we use for clarity:

- feat: Use this when developing features and adding functionalities
- style: Use this for code styling
- fix: Use this for bug fixes
- refactor: Use this when refactoring codebase or removing codes
- chore: Use this for configuration of project settings

2. Prettier with ESlint

The team acknowledges that everyone has a different code style. We decided to use prettier paired with ESlint for code formatting to make sure that we follow the convention coding style.



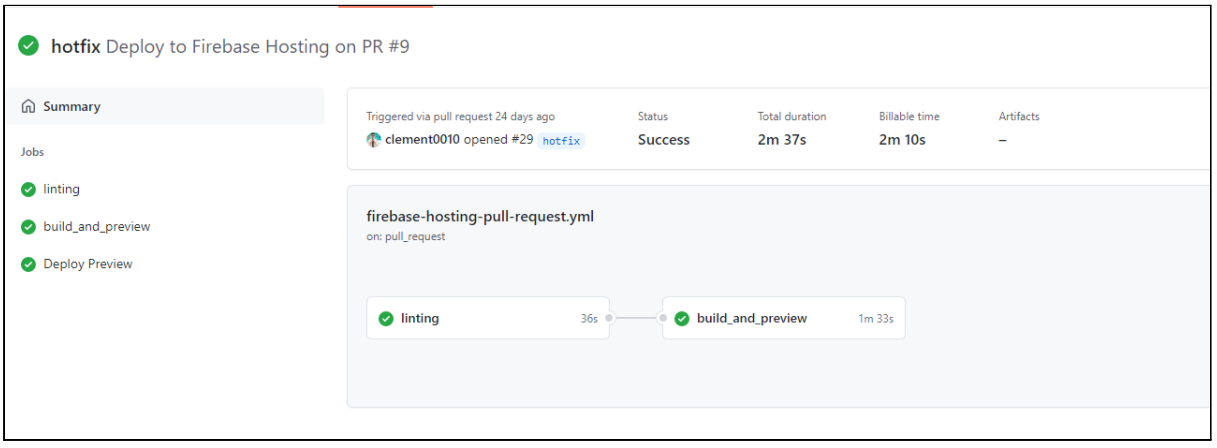
The configuration file of ESlint and Prettier linter

7.4.6 Continuous Integration and Deployment Pipeline

The diagram below shows the existing workflow we have by Github Action. It can be broken down into 2 pipeline

1. Deploy and preview on a pull request (PR)
2. Deploy on a merge with the master branch

The following GitHub action is triggered when a PR is created. It first runs a code linting check before building a preview version of the application.

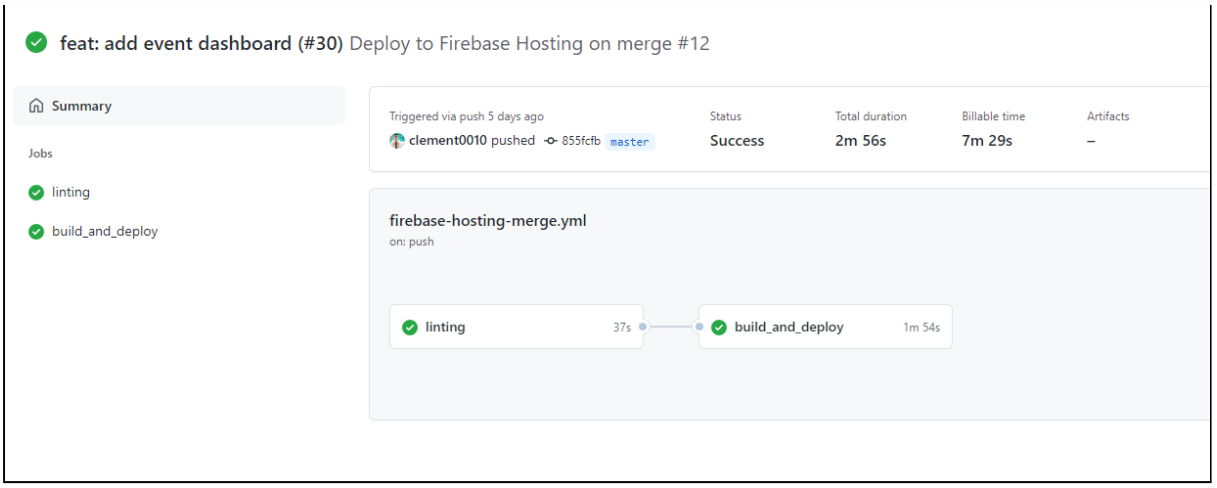


Pipeline when a PR is created

The following YAML file describes the workflow of the pipeline

```yaml
name: Deploy to Firebase Hosting on PR
'on': push
jobs:
  linting:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2
      - run: npm ci && npm run check:style
  build_and_preview:
    if: github.event_name == 'pull_request'
    needs: [linting]
    runs-on: ubuntu-latest
    env:
      VUE_APP_FIREBASE_API_KEY: '${{ secrets.VUE_APP_FIREBASE_API_KEY }}'
      VUE_APP_FIREBASE_AUTH_DOMAIN: '${{ secrets.VUE_APP_FIREBASE_AUTH_DOMAIN }}'
      VUE_APP_FIREBASE_PROJECT_ID: '${{ secrets.VUE_APP_FIREBASE_PROJECT_ID }}'
      VUE_APP_FIREBASE_STORAGE_BUCKET: '${{ secrets.VUE_APP_FIREBASE_STORAGE_BUCKET }}'
      VUE_APP_FIREBASE_MESSAGE_SENDER_ID: '${{ secrets.VUE_APP_FIREBASE_MESSAGE_SENDER_ID }}'
      VUE_APP_FIREBASE_APP_ID: '${{ secrets.VUE_APP_FIREBASE_APP_ID }}'
      VUE_APP_MEASUREMENT_ID: '${{ secrets.VUE_APP_MEASUREMENT_ID }}'
    steps:
      - uses: actions/checkout@v2
      - run: npm ci && npm run build
      - uses: FirebaseExtended/action-hosting-deploy@v0
        with:
          repoToken: '${{ secrets.GITHUB_TOKEN }}'
          firebaseServiceAccount: '${{ secrets.FIREBASE_SERVICE_ACCOUNT_SPONSORR_PROD }}'
          projectId: sponsorr-prod
```

Yaml file for GitHub Action on PR

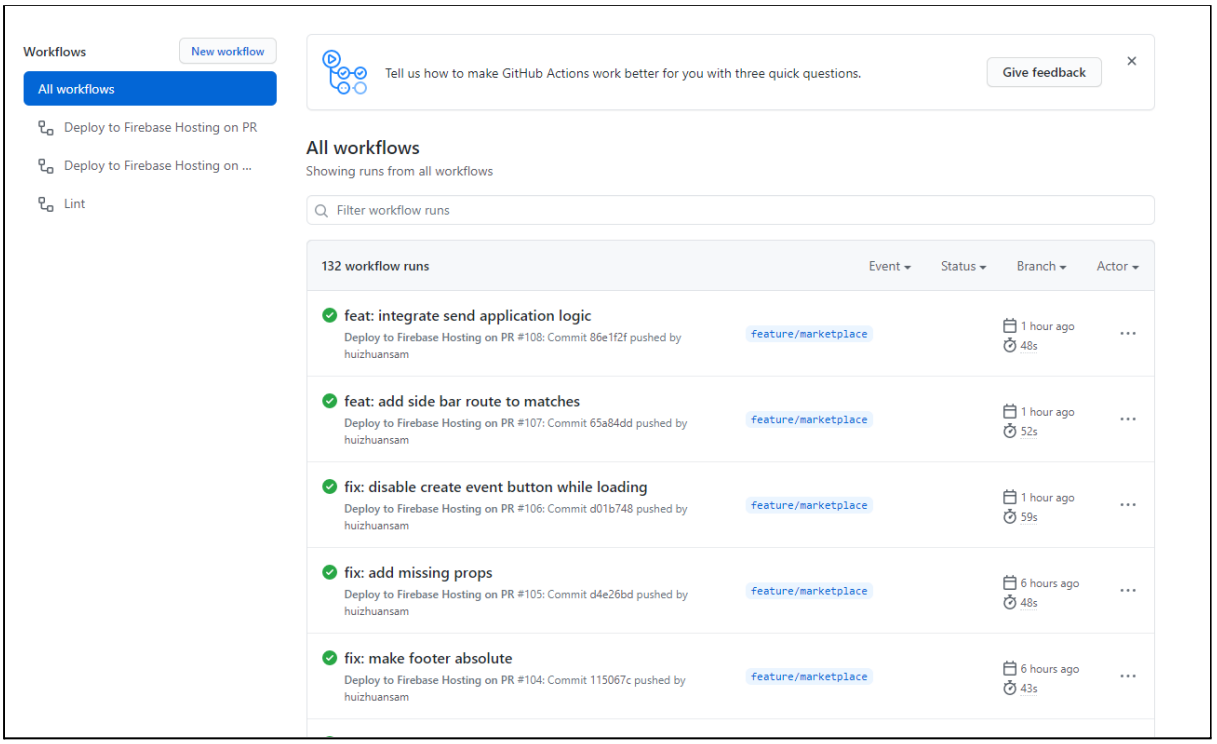


Pipeline when a branch is merged with master

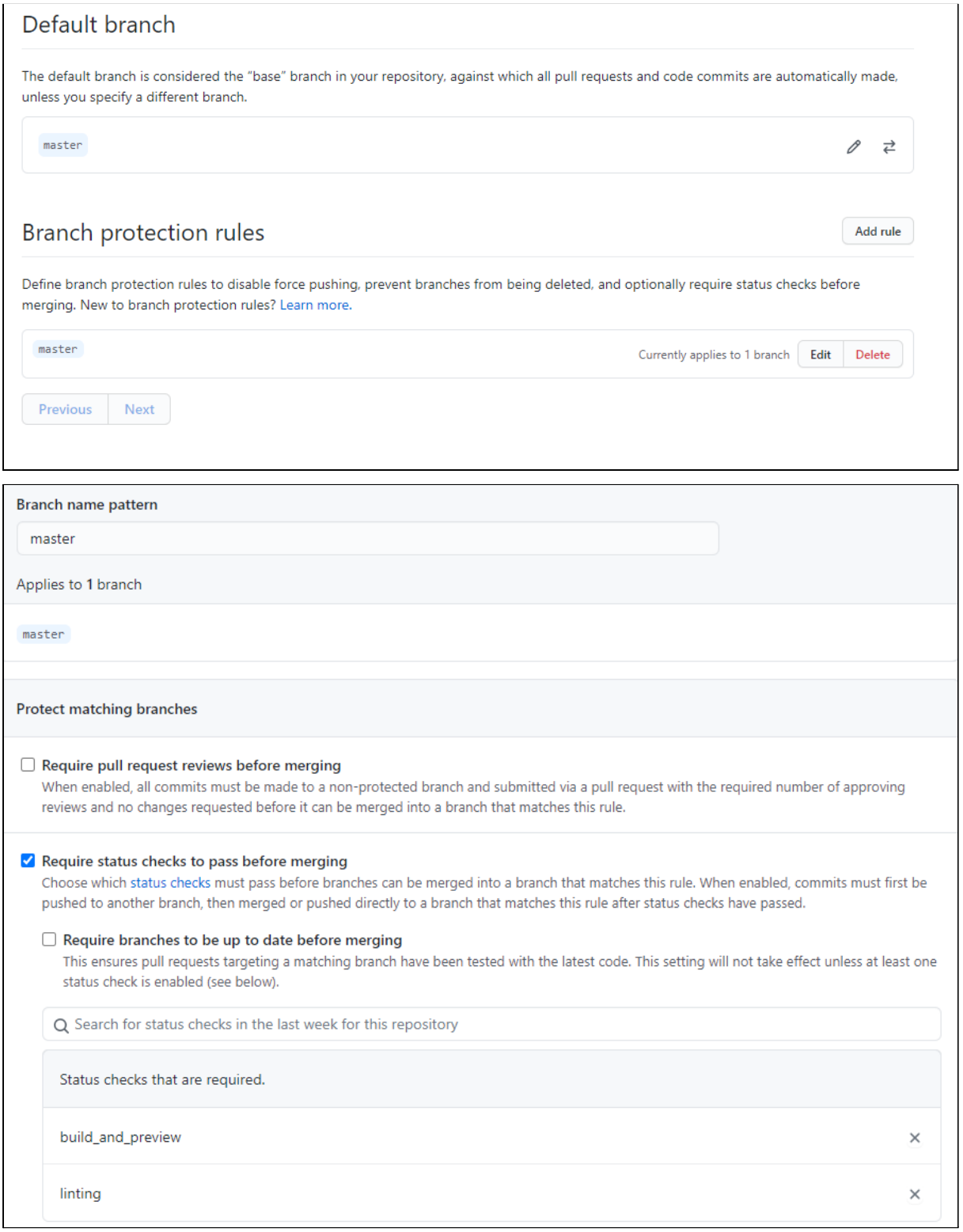
The following yaml file describes the workflow when a merge happens on the master branch.

```yaml
name: Deploy to Firebase Hosting on merge
'on':
  push:
    branches:
      - master
jobs:
  linting:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2
      - run: npm ci && npm run check:style
  build_and_deploy:
    needs: [linting]
    runs-on: ubuntu-latest
    env:
      VUE_APP_FIREBASE_API_KEY: '${{ secrets.VUE_APP_FIREBASE_API_KEY }}'
      VUE_APP_FIREBASE_AUTH_DOMAIN: '${{ secrets.VUE_APP_FIREBASE_AUTH_DOMAIN }}'
      VUE_APP_FIREBASE_PROJECT_ID: '${{ secrets.VUE_APP_FIREBASE_PROJECT_ID }}'
      VUE_APP_FIREBASE_STORAGE_BUCKET: '${{ secrets.VUE_APP_FIREBASE_STORAGE_BUCKET }}'
      VUE_APP_FIREBASE_MESSAGE_SENDER_ID: '${{ secrets.VUE_APP_FIREBASE_MESSAGE_SENDER_ID }}'
      VUE_APP_FIREBASE_APP_ID: '${{ secrets.VUE_APP_FIREBASE_APP_ID }}'
      VUE_APP_MEASUREMENT_ID: '${{ secrets.VUE_APP_MEASUREMENT_ID }}'
    steps:
      - uses: actions/checkout@v2
      - run: npm ci && npm run build
      - uses: FirebaseExtended/action-hosting-deploy@v0
        with:
          repoToken: '${{ secrets.GITHUB_TOKEN }}'
          firebaseServiceAccount: '${{ secrets.FIREBASE_SERVICE_ACCOUNT_SPONSORR_PROD }}'
          channelId: live
          projectId: sponsorr-prod
```

Yaml file for GitHub Action on merging with master



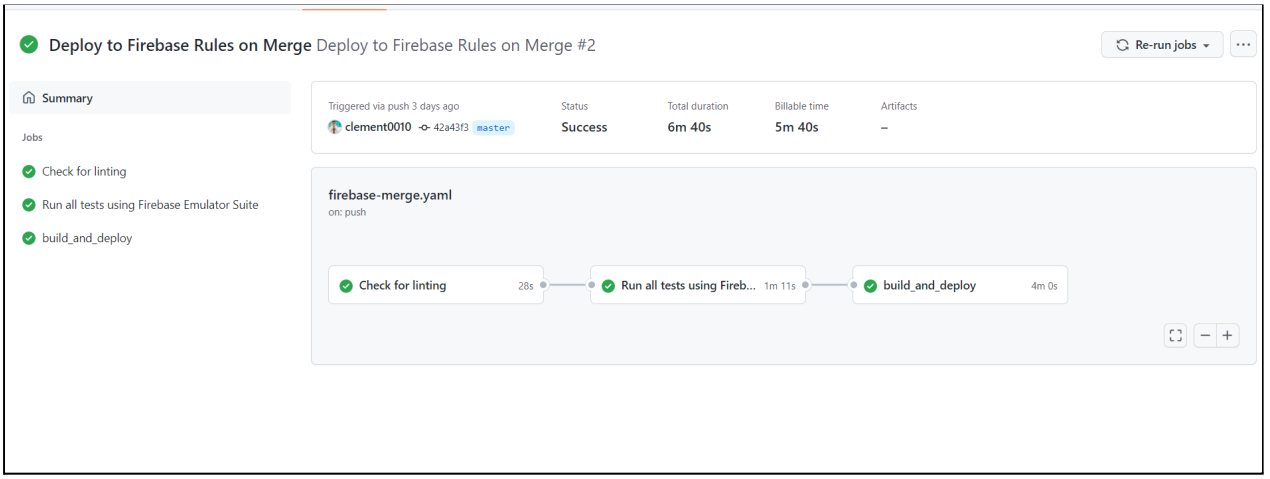
Aside from that we have enabled branch protection to protect against broken commits from merging with the master branch. Broken commits here are those who failed the pipeline. The following displays the branch protection for our master branch.



Branch protection for the master branch

Another main benefit we get out from this CI/CD pipeline is the hassle free process to deploy our application for production. The team does not have to worry about production build as it is being automated in the workflow.

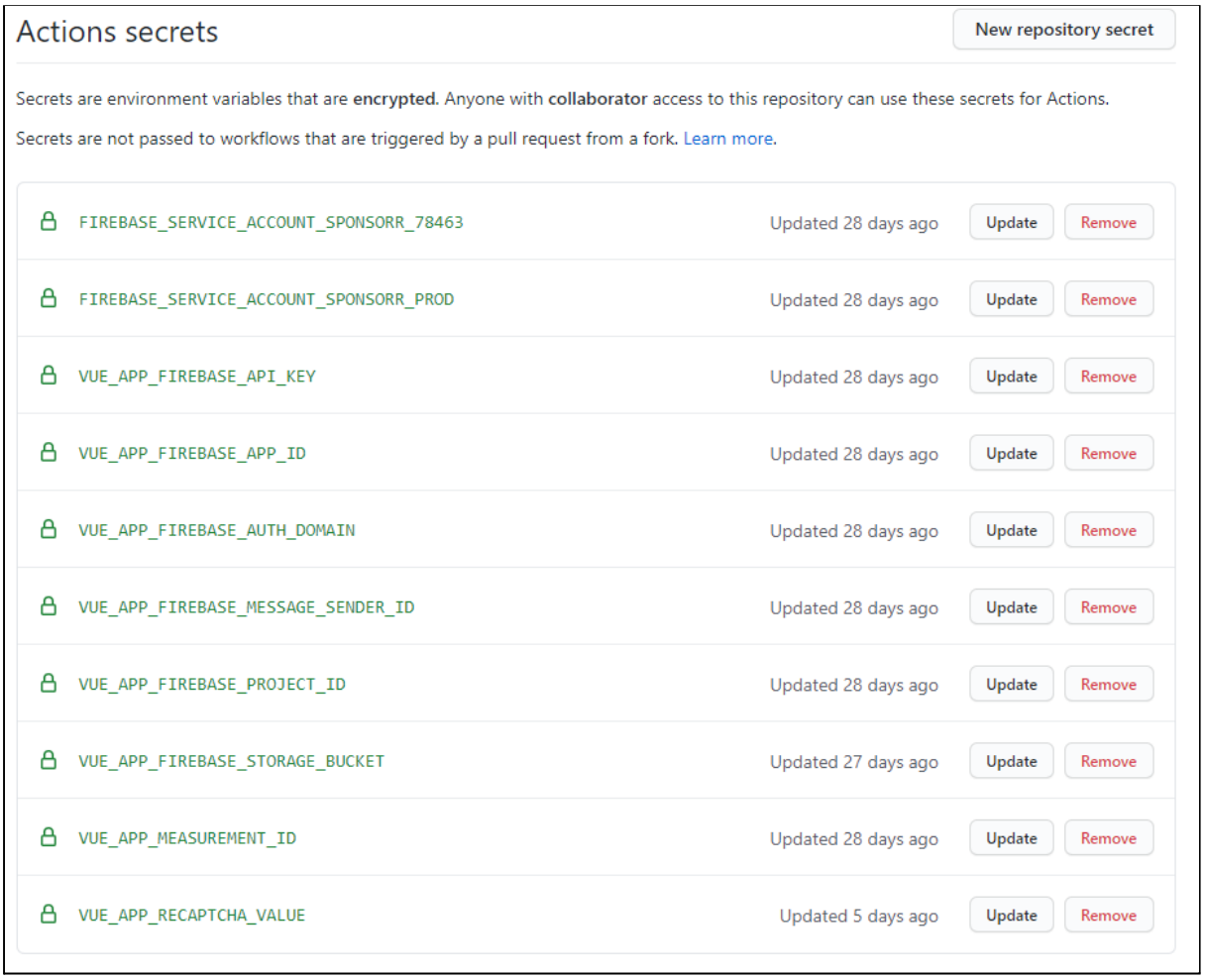
Building on from milestone 2, the team has added testing as part of the CI pipeline to ensure code quality. The diagram below shows how we made sure the test cases pass before building and deploying our backend services.



Automated Testing when push and merging

7.4.7 Environment Variable

The team values the security of our application as our topmost priority. As such no tokens or security keys are ever exposed to the public. One way that we manage our secret keys is through the use of GitHub actions secrets to provide access for our CI/CD pipeline



GitHub Action Secrets for our CI/CD pipeline

7.5 Security Vulnerabilities

7.5.1 Webhook

As we utilised webhook to design for our email confirmation system, there are some security issues that we have to address.

1. Forged requests
   A forged request is a request made to your webhook endpoint that acts as if it was sent from the original source.
2. Replay attack
   A replay attack is a form of network attack in which valid data transmission is maliciously or fraudulently repeated or delayed.

Preventive Measures

1. Secret Token for validation
   We validate the incoming body request (hash, email, user id) by checking if the incoming hash matches our signature generated in the backend.

   The email and user id is encrypted along with a secret token via a hashing algorithm (SHA-256). It is then validated with the incoming hash to guarantee that the payload has not been tempered.

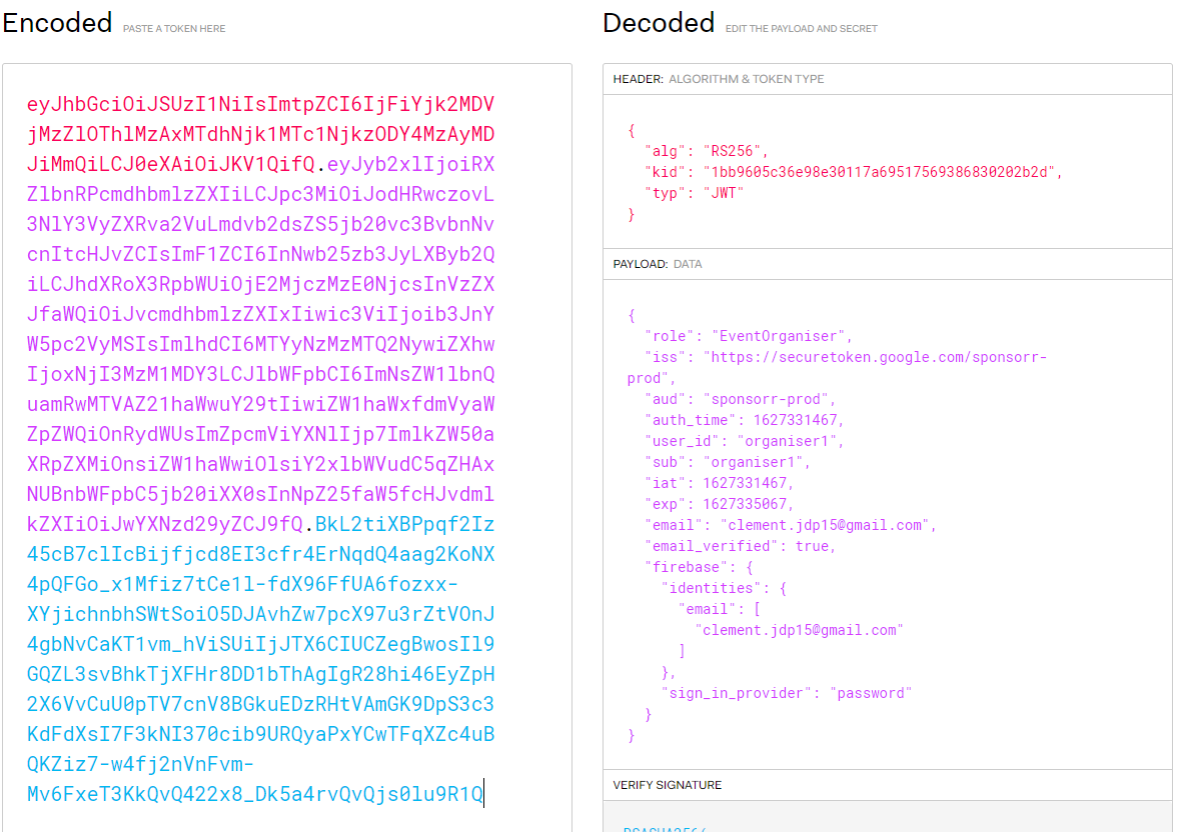
2. Idempotency
   Idempotence is the property of certain operations whereby they can be applied multiple times without changing the result beyond the initial application. Since our email verification will give out the same result, i.e. email verified, we could ensure repeated queries would not cause any security concerns.

7.5.2 JWT Token & Custom Claims

To achieve role-based authorisation to the database, we added the user's role (EventOrganiser/Sponsor) as part of the custom claim in JSON Web Token. The team is well aware that firebase in fact uses JWT to authenticate users. Hence we dig deeper to understand the working mechanism

What is JWT

The JWT is stored on the client-side usually in localStorage and it is sent as a unique key of that user when the user requests any data from the server or is performing any activity for that website. It has the following structure when decoded



Structure of Encoded and Decoded JWT

As you can see we have our custom claim role in the payload and the recipient could not possibly tamper with the role as it would alter the hash value (public signature). Eventually, when verified with the private signature, it would be invalid and reject the tampered token. This ensures the authenticity of the data.

Furthermore, the JWT will be used to access our database and custom claims play an important role in ensuring the correct role has the correct access to the database. Read on to see how custom claims are paired with database security rules.

7.5.3 Database Security Rules

The team follows the least permission principle when designing the database access permission. This ensures strict and secure security as users can never execute CRUD operations beyond their set permission.

Role-based authentication is achieved when we do *"request.auth.token.role"* to get the payload in the token to verify it is indeed the correct role to access the data.

```
rules_version = '2';
service cloud.firestore {
  match /databases/{database}/documents {
    match /users/{userId} {
      allow read: if request.auth != null

      // Only allow user to read, write and delete their own data
      allow write, delete: if request.auth != null && request.auth.uid == userId;
      allow create: if request.auth != null;
    }

    match /events/{eventId} {
      // Only allow user to read event if they are authenticated
      allow read: if request.auth != null

      // Only allow user to create event if they are event organiser
      allow create: if request.auth != null && request.auth.token.role == "EventOrganiser" && request.auth

      // Only allow user to write and delete their own event
      allow write, update, delete: if request.auth != null && request.auth.uid == resource.data.userId;
    }

    match /matches/{matchId} {
      // Only allow user to read matches if they are authenticated and they are the matched sponsor
      // or the matched event belongs to the user
      allow read, write: if request.auth != null && (request.auth.uid == resource.data.userId ||
                    resource.data.organiserId == request.auth.uid);

      // Only allow Sponsor to create, write and delete a match
      allow create, delete: if request.auth != null &&
                    request.auth.token.role == 'Sponsor';

    }
  }
}
```

Database Security Rule used in Firestore for CRUD permission

Side Note: Security Rule is generally disabled in development environments for debugging purposes.

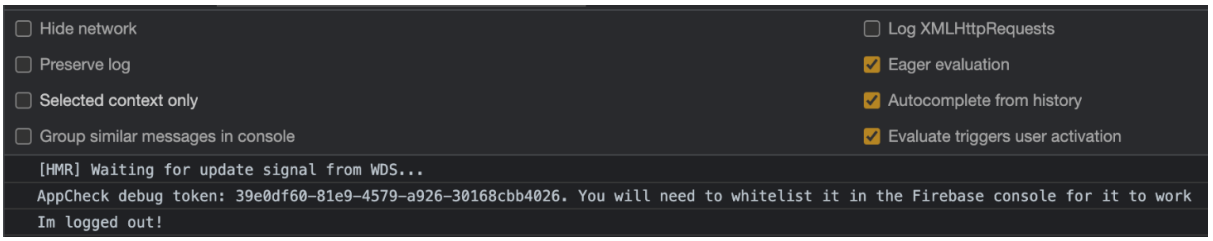
7.5.4 Recaptcha Token

The team utilises reCAPTCHA v3 as it will never interrupt our users to solve challenges, so they can run it whenever you like without affecting conversion. As our application relies heavily on user interaction with the application. It makes sense for us to use reCAPTCHA to decide legitimate and abusive behaviour to avoid abuse of our storage system.

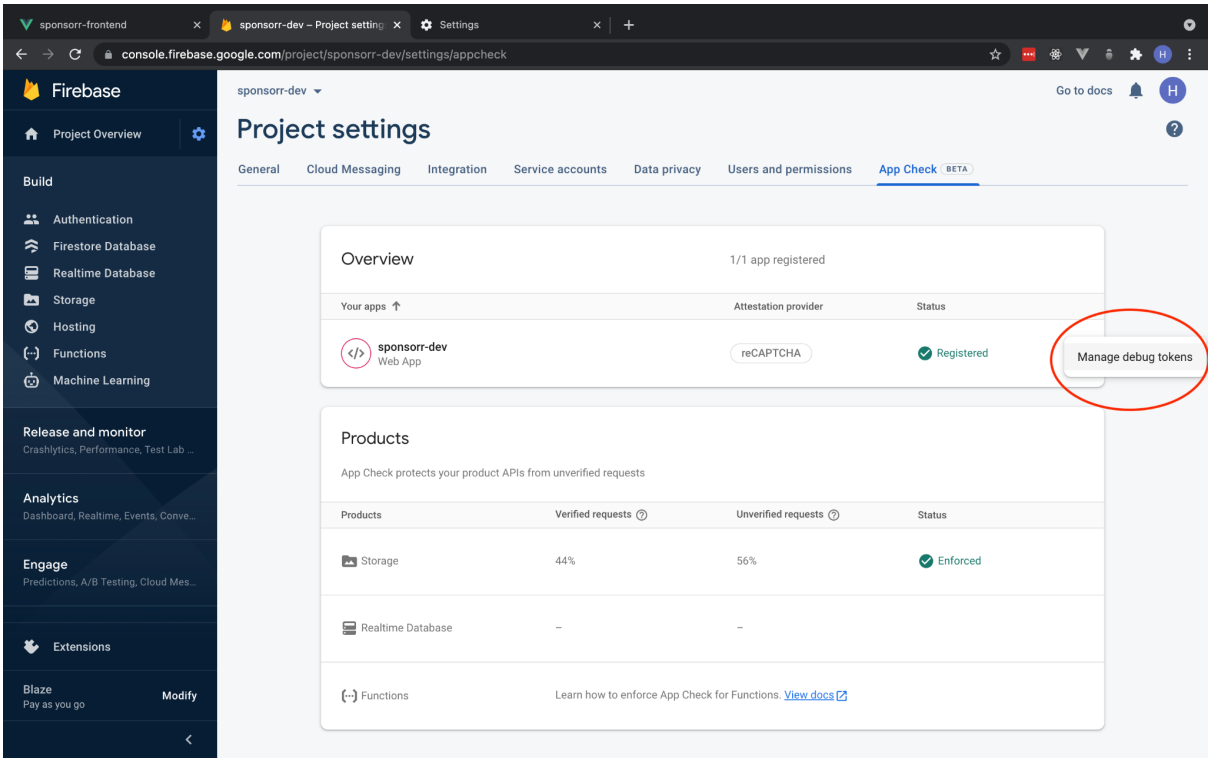
For this reason, we recommend including reCAPTCHA verification on forms or actions as well as in the background of pages for analytics.

For developing purposes, we bypassed Appcheck in debug mode via the following steps:

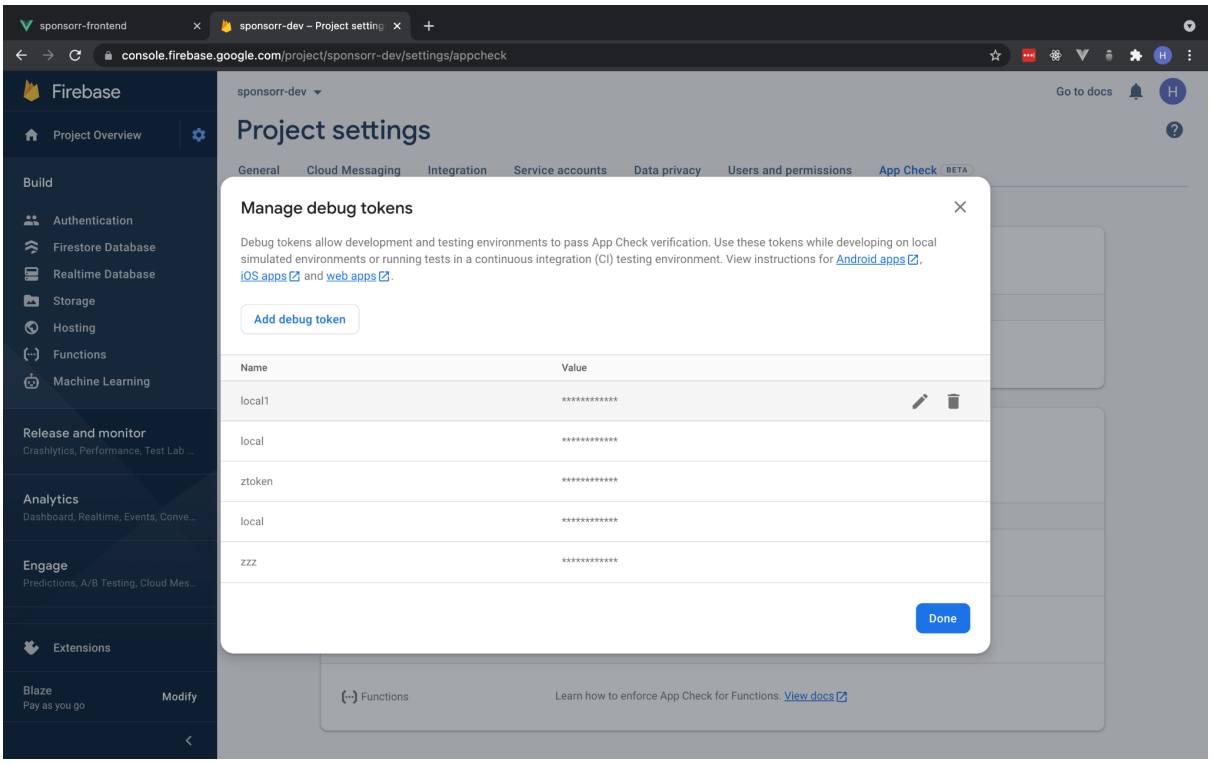
Debug Mode for reCAPTCHA token:



Open Firebase console App Check:



The app check token is retrieved from the DevTools console and whitelisted in the Firebase console:



7.6 Quality Assurance

7.6.1 Unit Testing

The team uses mocha and chai to execute unit tests for our application. Here are some of the few examples for our test cases.

1. The first test suite will be testing all the utility functions that we have in our application.
2. The second test suite is a mock firestore API test.

```
describe('Utility Function', () => {
  it('should return the date in DD/MM/YYYY format', () => {
    return assert.equal(generateDate(1624895383,'DD/MM/YYYY'),'28/06/2021')
  })
})

describe('DB Call', () => {
  const defaultApp = admin.initializeApp({
    credential: admin.credential.applicationDefault(),
});
  let defaultDatabase = defaultApp.firestore();

  it('should return user data', async () => {
    const user = await defaultDatabase.collection('users').doc('test')

    expect(user.id).to.equal('test');
  })
})
```

Test suites example

```
  Utility Function
    ✓ should return the date in DD/MM/YYYY format

  DB Call
    ✓ should return user data


  2 passing (20ms)

 MOCHA  Tests completed successfully
```

Test result upon running our test script

7.6.2 Security Rule Testing

As mentioned in section 7.4.6, the security rule testing is run when pushed and merged. The test cases are mainly to check for the user interactions with the database. No test coverage report is generated as no function is executed, thus it is not relevant here.

Moving forward for post-milestone 3, the team will come out with more edge cases to increase test coverage.

```
27
28  > functions@ test /home/runner/work/sponsorr-backend/sponsorr-backend
29  > mocha src/test/** -r ts-node/register --reporter src/test/reporter.ts --timeout 5000 --exit
30
31
32  Using emulators {
33    firestore: { host: 'localhost', port: 8080 },
34    hub: { host: 'localhost', port: 4400 }
35  }
36  Unit Testing Firebase rules
37  User authentication
38      ✓ can be created by user
39      ✓ can be read by other users
40  Events
41      ✓ can be created and updated by the user
42      ✓ can be read only by anyone
```

7.6.3 End-To-End Testing

Unfortunately, due to time constraints, the team is still working on integrating end-to-end testing into our application. Ideally, it would be done before splashdown. However, the team has already thought through the scenario and objective to generate the test cases. These include:

Feature to test	Test Cases/Scenarios	Test Result (To be completed)
Authentication	1. User sign up 2. User login as an event organiser with correct credentials 3. User login as a sponsor with correct credentials 4. User login as an event organiser with wrong credentials 5. User login as a sponsor with wrong credentials	

Event Creation	1. User uploads documents and images 2. User creates event	
Event Management	1. User unpublish an event 2. User delete an event 3. User publishes an event	
Marketplace	1. User search for events with the title 2. User inspect events	
Event Matches	1. User accepts a match 2. User rejecst a match 3. User views a match 4. User inspects sponsor	
Navigation	1. User navigates to all other pages 2. User navigates to routes that are not available to them	

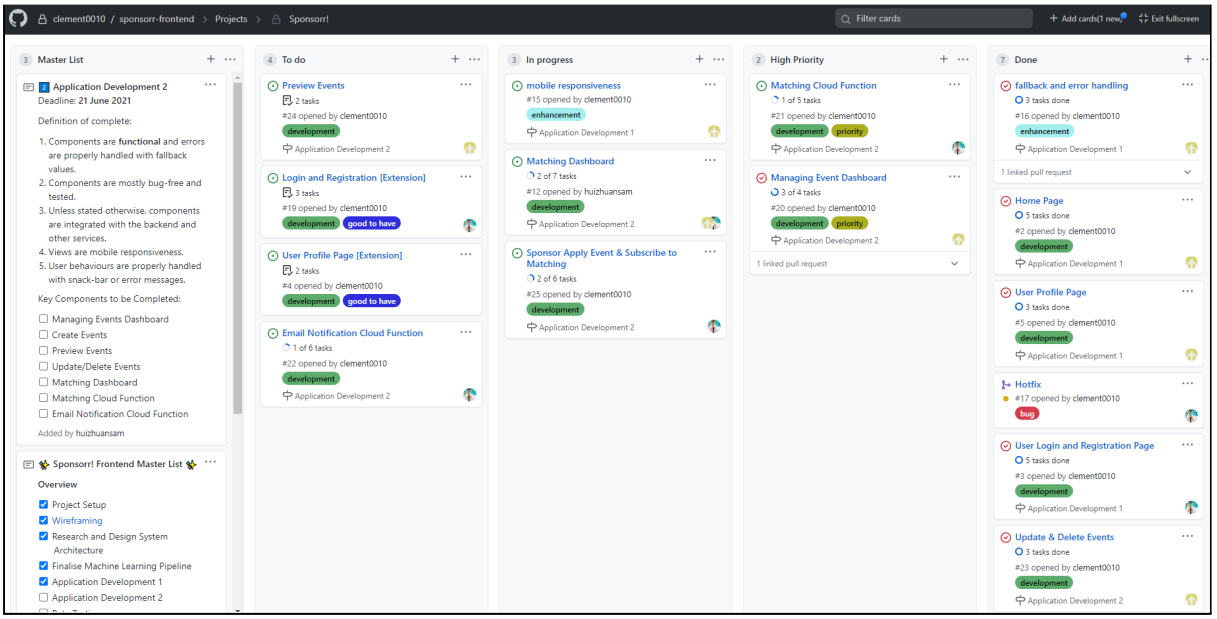
The test cases are not exhaustive and more will be added to increase test coverage.

8.0 Project Management

8.1 Github Project

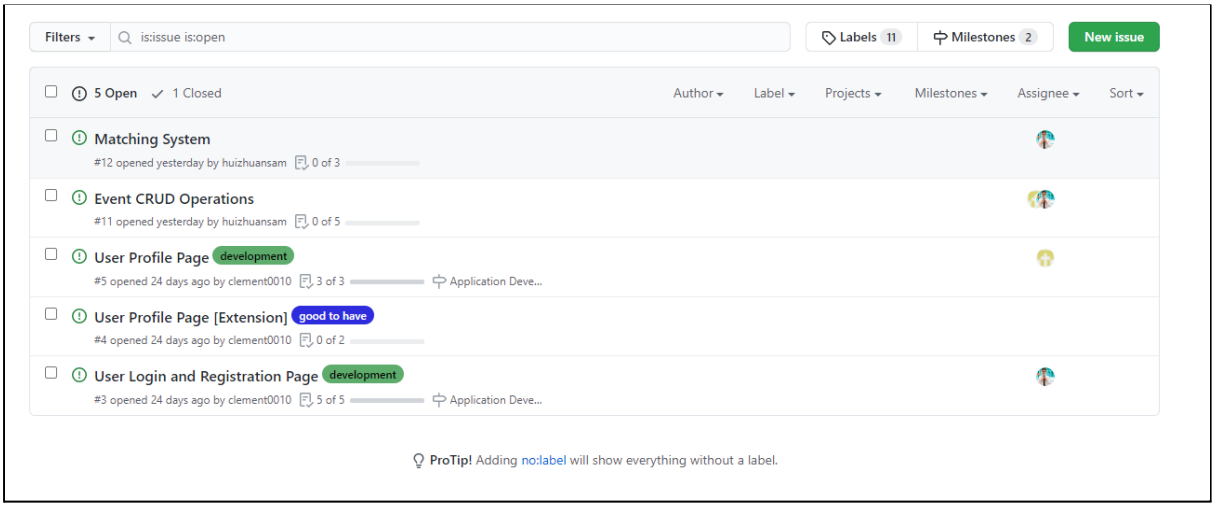
The team is using Github Projects as a tool to manage the project. We leveraged the kanban board to keep track of our application development tasks and progress.

During each development phase, the team will create new tasks and targets with the task breakdown write-up as shown below in the screenshots.

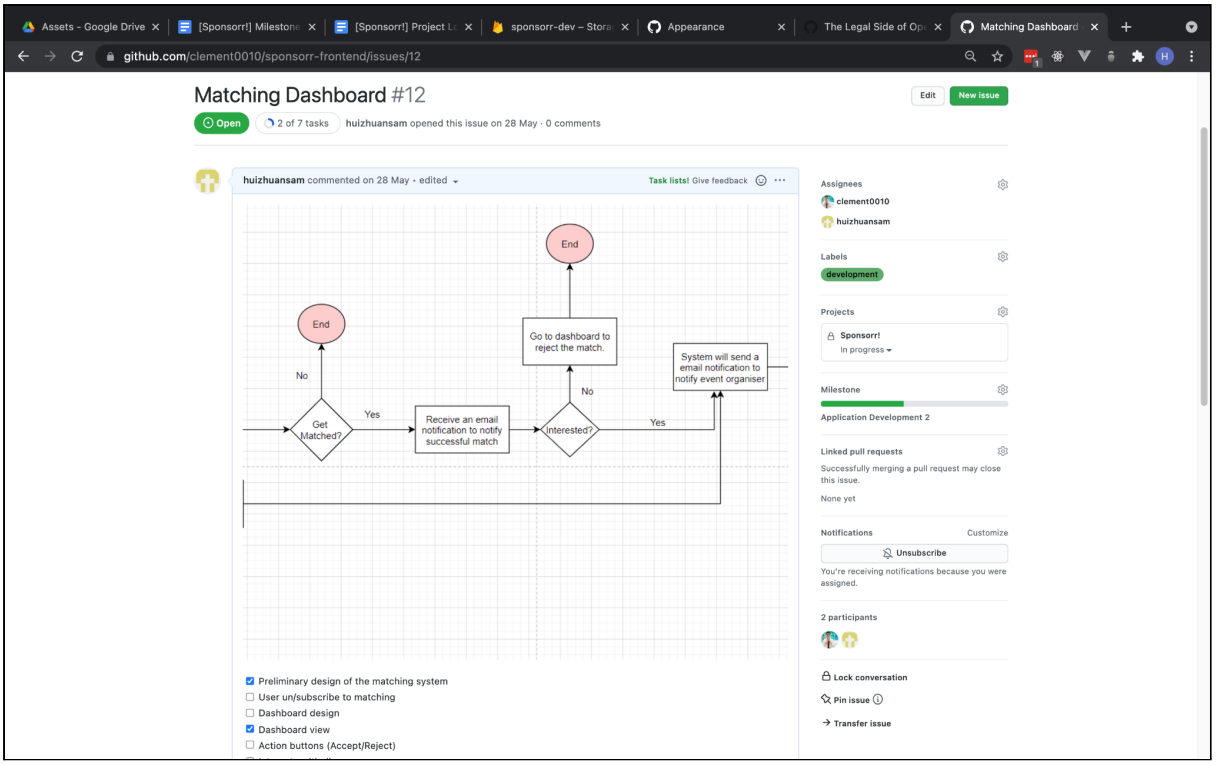


A screenshot view of our Kanban Board

Apart from that, we use Github Issue to assign tasks and keep track of who is working on the specific feature.



A screenshot view of our current Github issues

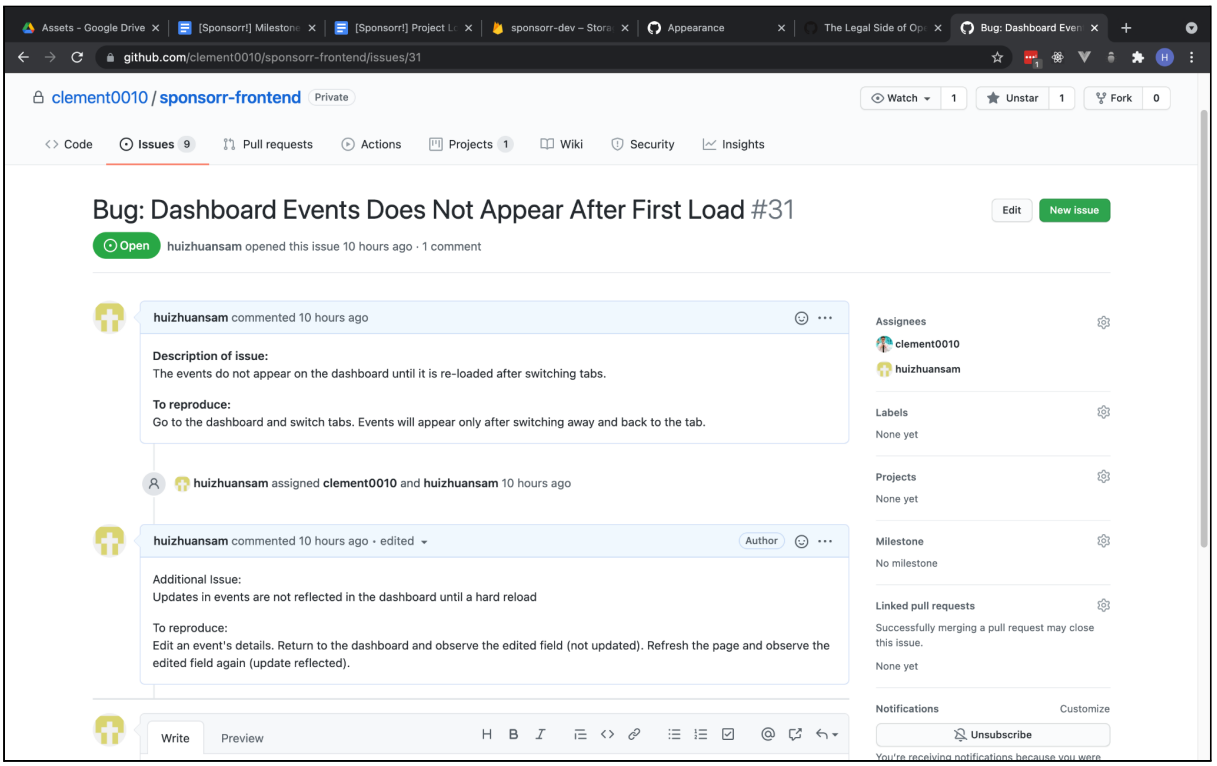


A screenshot view of a task write up.

8.2 Github Issue

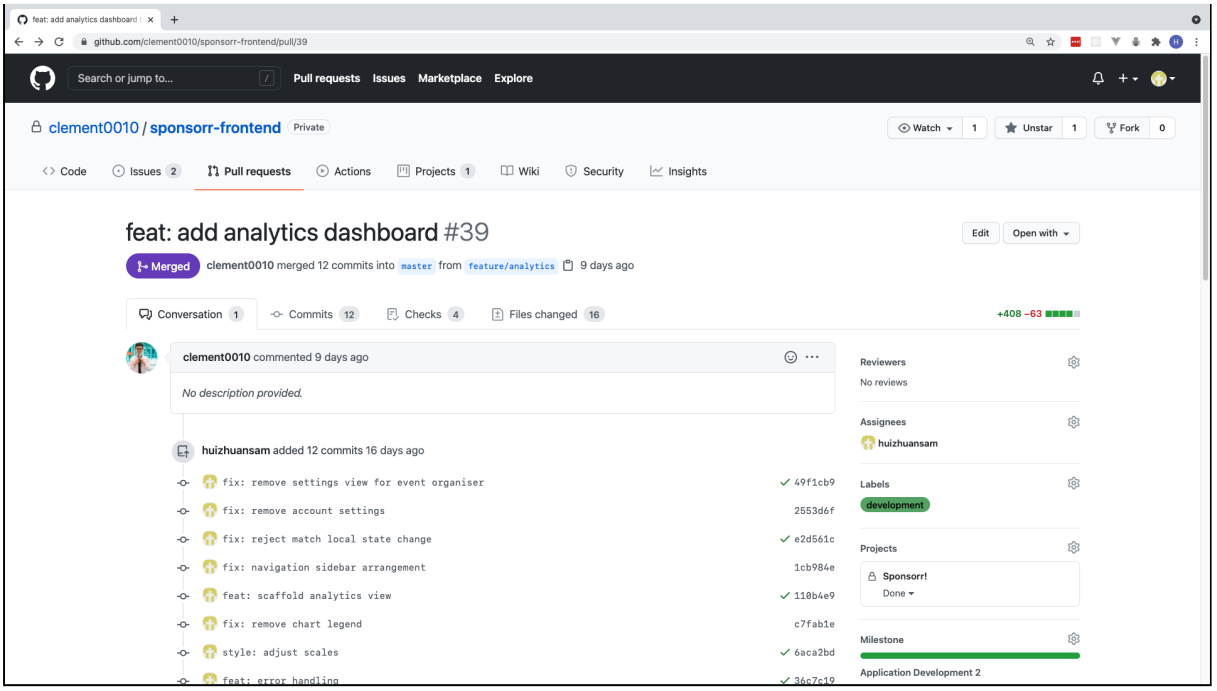
Furthermore, if there is a bug after a pull request is closed. A hotfix branch will be created and tagged with an issue. We also enforce a strict rule in writing issue, it should contain information but not limited to the following:

1. Bug Description
2. Steps to reproduce
3. Screenshots/Video (if applicable)
4. Code snippet



A screenshot view of an issue write up

8.3 Pull Request



A screenshot view of an issue pull request

9.0 Response to Milestone Evaluation

The evaluation we received from our peers for Milestone 2 was generally positive. The critical feedback we received are:

- Buggy navigation sidebar - experienced by one team.
- Lack of testing

The lack of real critical feedback gave us some issues deciding on which area of the application we should improve on. Nevertheless, we worked on whatever feedback we had.

9.1 Actions Taken

- The navigation sidebar has been redesigned and reimplemented, removing the potential bugginess.
- We conducted user acceptance testing, which yielded much better results in terms of bug catching and spotting UI/UX flaws. See section 10.0, User Acceptance Report for more information.

10.0 User Acceptance Report

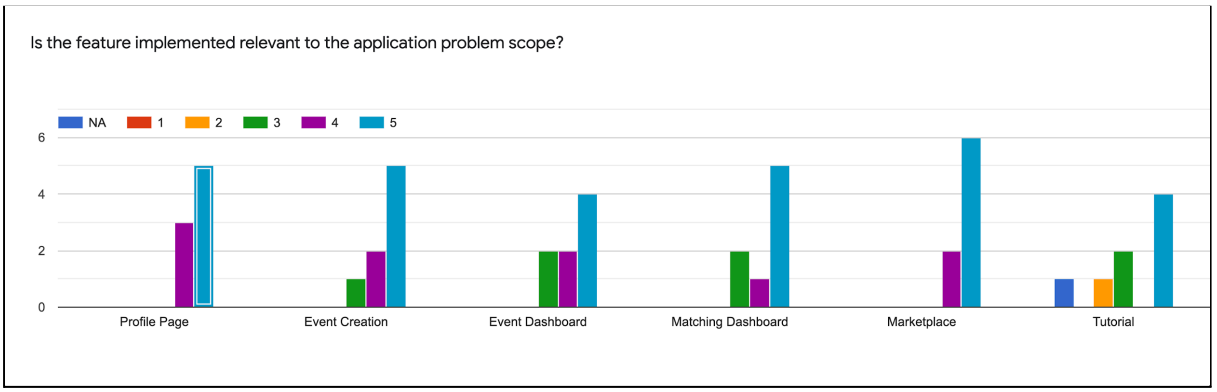
10.1 How it works

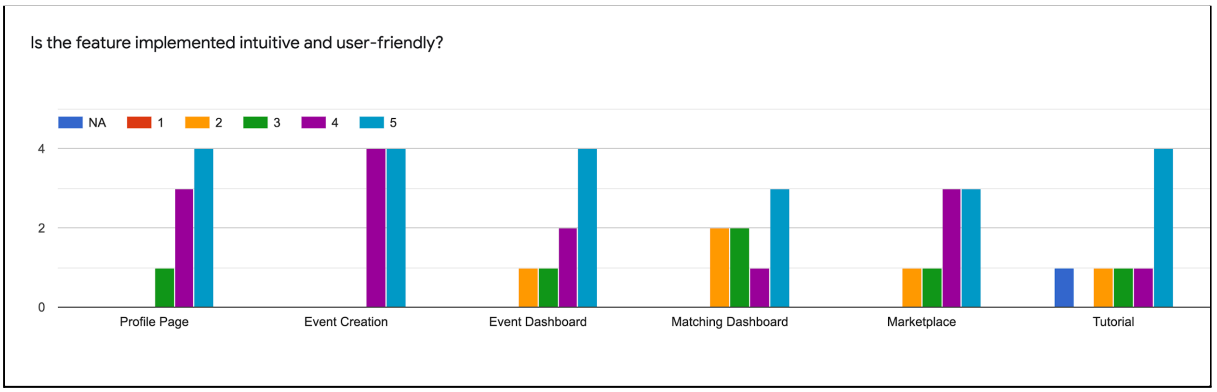
The team recruited 8 volunteers to help test out the application. The testing methodology used is user acceptance testing, where the volunteers are given the application to use as-is and without any instructions. Areas of the application where confusion or unexpected bugs occurred were noted down. At the end of the acceptance testing, the volunteers gave us anonymous feedback on the usability of the application and suggested improvements via Google Forms.

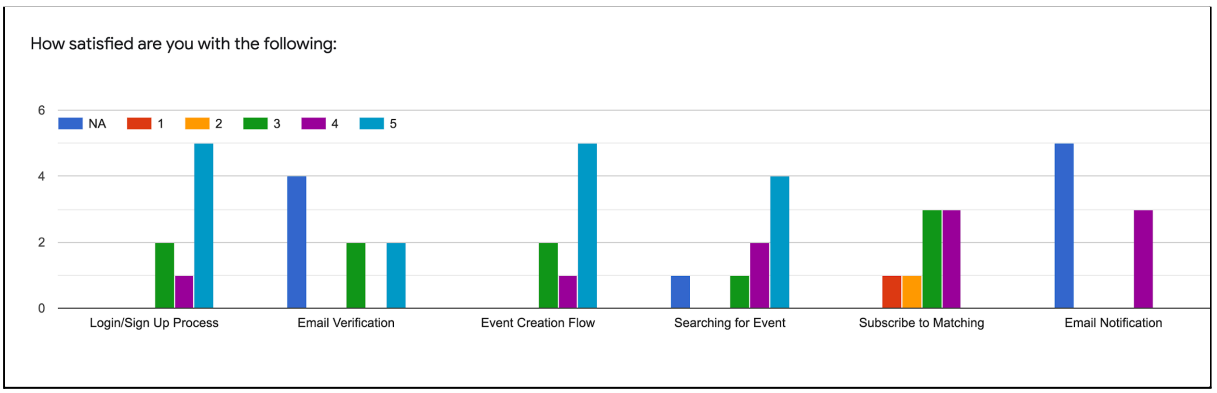
(N.B.: due to COVID-19 restrictions, testings was done over Discord video calls)

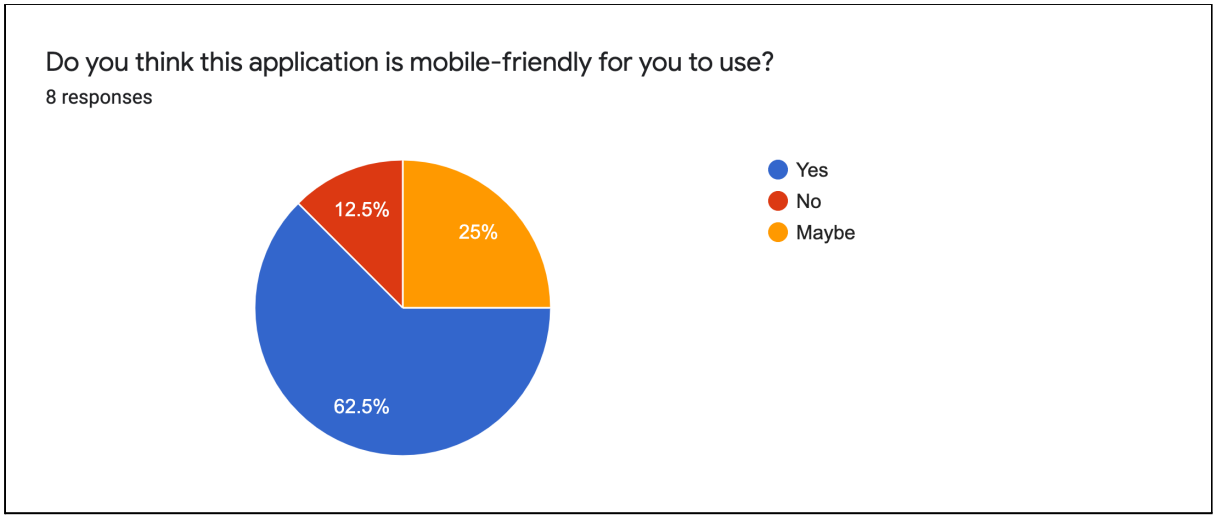
10.2 Results

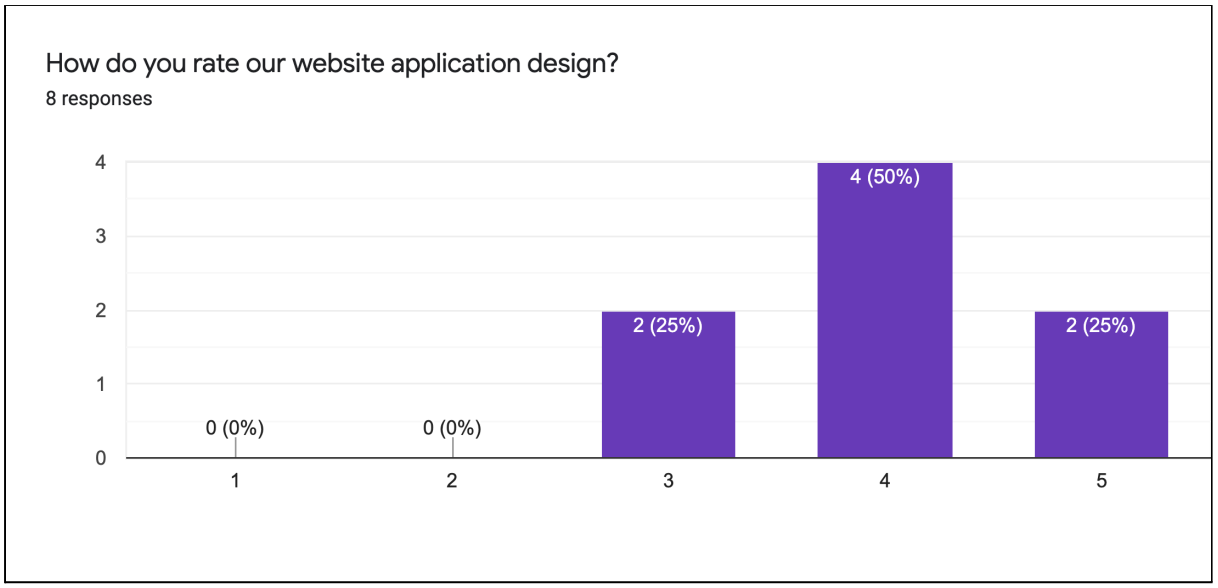
The results for closed-ended questions are summarised as below:

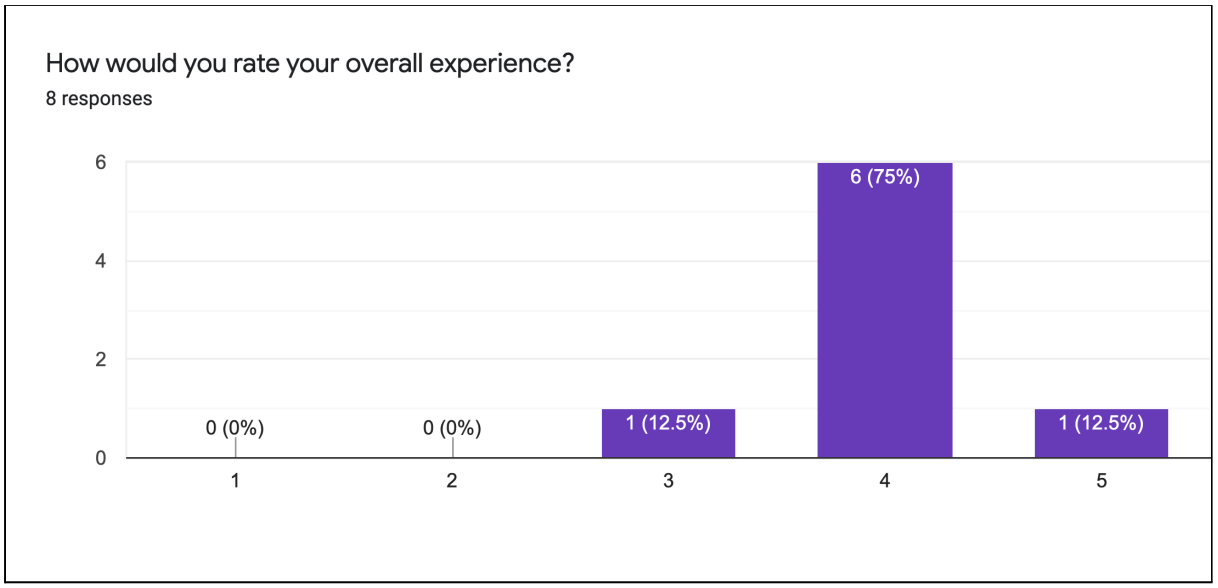












In summary, we learned that our application is **average** to **above-average** in terms of **usability**.

There are some concerns about the **lack of intuitiveness** using **complex pages** such as the dashboards and marketplace, hence we worked on making these pages contain more helpful information and better directives.

The decision to use **Vuetify** as our UI component library seems to have paid off, as users had generally **positive experience** interacting with the application's UI thanks to it based on **Google's Material Design language**.

10.3 Actions Taken

The feedback for open-ended questions are collated and summarised in the table below:

Feedback	Solution/Action Taken
The matching dashboard is not intuitive to use	Added additional UI elements such as **tooltips**, **modals**, and **colour-coding** to give the users clear usage directions.
Help popup is too brief	Expanded on the help popup with more detailed write-ups and directions.
Unable to see the full uploaded event image	Added a button that expands the image to its full size when clicked.
Difficult to determine whether an event has been accepted or rejected	Added **visual cues** (green, yellow, and red badges for accepted, pending, and rejected respectively), and **clear categorization** of events and matches in their dashboards.
Sponsor not too sure what message to send to event organisers	Will add clear directions for first-time users. Messages should be kept simple and conversational, with minimal formalities and straight to the point.
Add time-series charts to compare data over a period of time	Reserved for further extension. The current analytics page is kept simple as a functional proof of concept.
Add chatroom feature for further communication between sponsor and event organiser	Reserved for further extension. Consider using Firebase real-time storage to implement the feature.
Email verification for security	Added **email verification service** to the application. Users are now required to verify their email addresses before being allowed to use the application.

Add app notifications	Reserved for further extension. Good to have.
Support video pitch	Not enthusiastic about uploading video content to our database due to file size and legal concerns (copyrighted and R-rated content, to name a few). However, we might consider allowing organisers to attach links to videos uploaded on external video hosting platforms.
Mobile responsive fonts	Addressed this issue in the UI refresh. Fonts respond to viewport sizes and adjust accordingly.
Real-time updates to events' and matches' status	Not under consideration. However, we made a refresh button to re-fetch updated data from the database without needing to hard-refresh the webpage. The intention of the feedback is understood, and we took a cheaper strategy than approximately achieves the intended outcome. See section 12.2

11.0 Demonstration

11.1 Project Video

The project demo video can be found here.

11.2 Project Poster

12.0 Project Roadblocks

12.1 Problems with Proposed Solution

Legend

Critical - Blocking the progress of the entire project

Technical - Technical Debt/Technical Restriction

Minor - Not harming any progress

Problems	Potential Solutions/Actions Taken
Validating user input from forms	Use regex and VueJS' built in validation features
Frequent merge conflicts	Weekly merge conflict resolution meetings
Standardising code style	Use ESLint for code linting to ensure consistent and understandable code. CI/CD checks using Github actions prevent poorly written/non-compliant code to be deployed.
Implementing firebase unit test locally	Due to the nature of webpack not including firebase/app in the testing environment. The team resorted to creating a mock firebase to simulate similar behaviour.
Large code base and boilerplate code	To achieve better reusability the team might need to refactor redundant code to follow the best practices.
State Management	The team overcame this issue by leveraging VueJS' Composition API, managing state using a single source of truth.

12.2 Limitations

Limitations	Counter Measures
Email Service API requests	The previous email service provider was SendGrid. While they provide seamless integration to send SMTP emails with your Gmail account, the quota allowed was insufficient to accommodate our needs. Hence, we pivoted to Mailgun as our email service provider though it requires quite a bit of setting up and requires a domain. Our team managed to overcome this challenge by getting a free domain from namecheap and setting up the SMTP emailing service.
Real-Time Updates	As of now, the application does not support real-time updates. Thus the user has to re-fetch the data to view the latest matches. This is expected as our application does not have the capacity to set up WebSocket to accommodate such needs. No further actions required as this does not add as much value to our end-user compared to the costs incurred for such service.
Analytics Page View	Due to design issues, user actions timestamps such as accept/reject events are not stored in the database. As such, we are not able to display time series data in our analytics dashboard. Fortunately, we are using a NoSQL database so it's much easier to add additional fields in the future for such extension with slight modification of our database schema.

13.0 Project Logs

The project log can be found here.

14.0 Reference

1. Firebase Cron Job
2. Firebase Rules
3. Firebase Appcheck

15.0 Appendix

15.1 Past Milestone Submissions

Materials	Link
Milestone 1 Project Report	[Sponsorr!] Milestone 1 Submission
Milestone 1 Project Log	[Sponsorr!] Project Log - Milestone 1
Milestone 2 Project Report	[Sponsorr!] Milestone 2 Submission
Milestone 2 Project Log	[Sponsorr!] Project Log - Milestone 2