Progress Report 1

COSC 4P02 - Software Engineering II

February 23, 2025

Naser Ezzati-Jivan

Brock University

TA: Sarfaroz Yunusov

Project Title:

PRJ 4 – Event Management Platform (Festify)

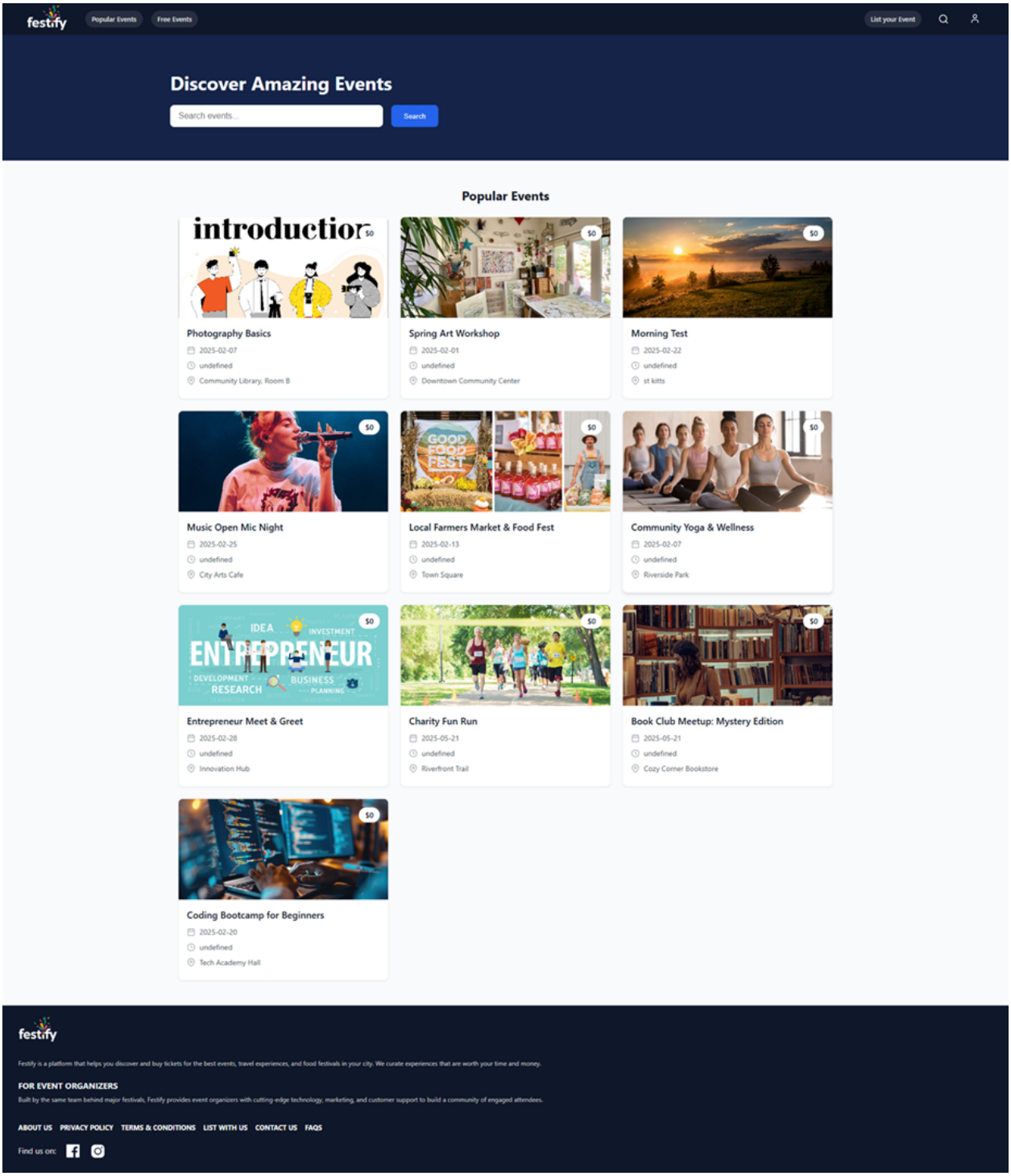
Team Members

Name	Student Number	Email
Gautam	7340433	gg21qv@brocku.ca
Harmanjot Malhi	7300973	hm21qv@brocku.ca
Kartikkumar Parekh	7107782	kp20zm@brocku.ca
Parampal Singh	7003114	pp20kv@brocku.ca
Rohal Kabir	7105836	rk20da@brocku.ca
Sumant Patel	6796841	sp19kp@brocku.ca
Vivek Salwan	6951826	vs19bf@brocku.ca

1 Screenshots of the Working Project

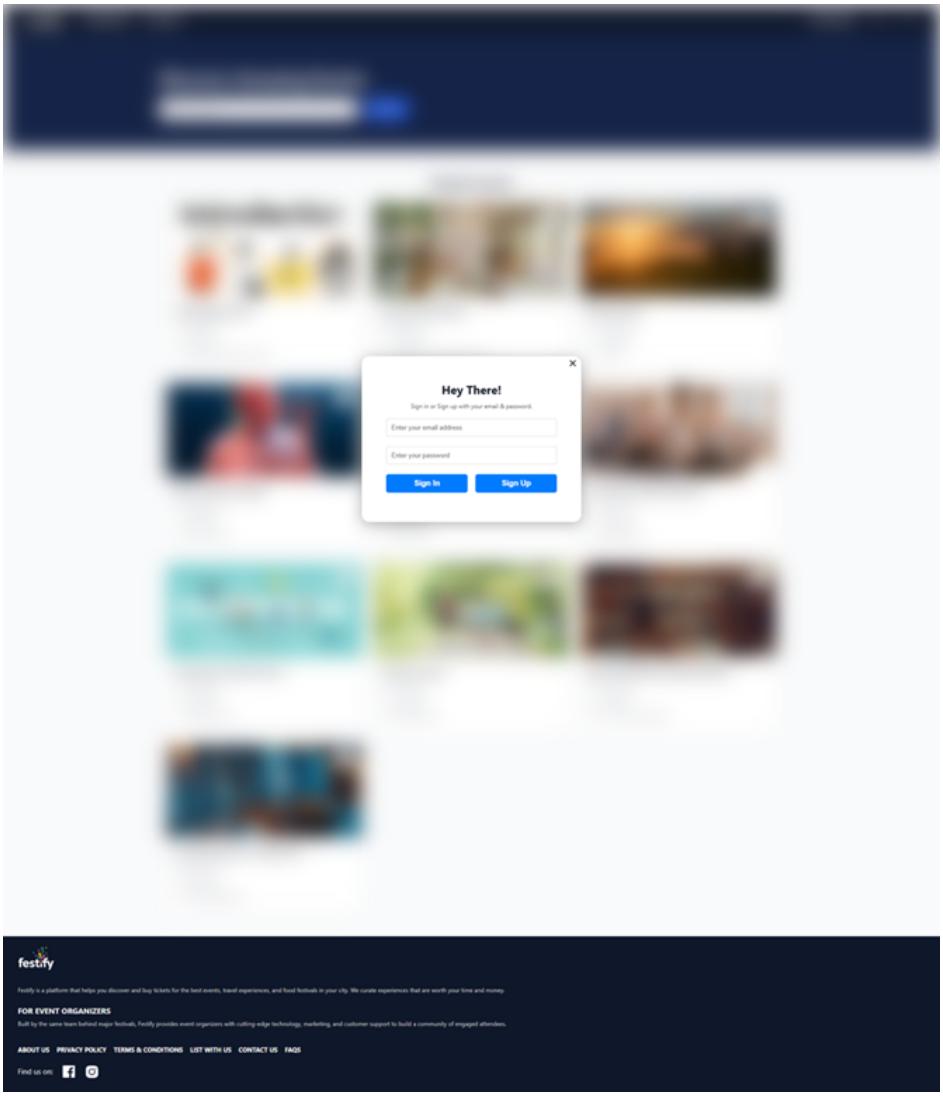
1.1 Home (Landing) Page

This is the landing page of our website. Once a user visits our domain, they will see the *Discover Event* page, where all the listed events are seen, and the user can view and scroll through them.

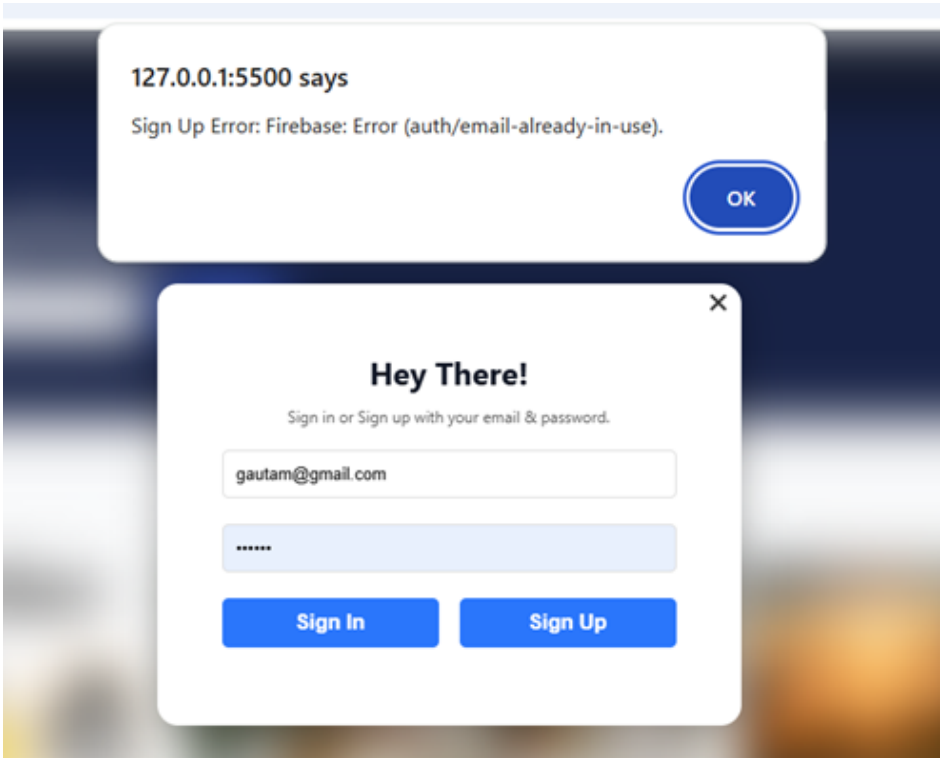


1.2 User Registration and Login

From the home page, the user can register or log in by clicking the profile icon.

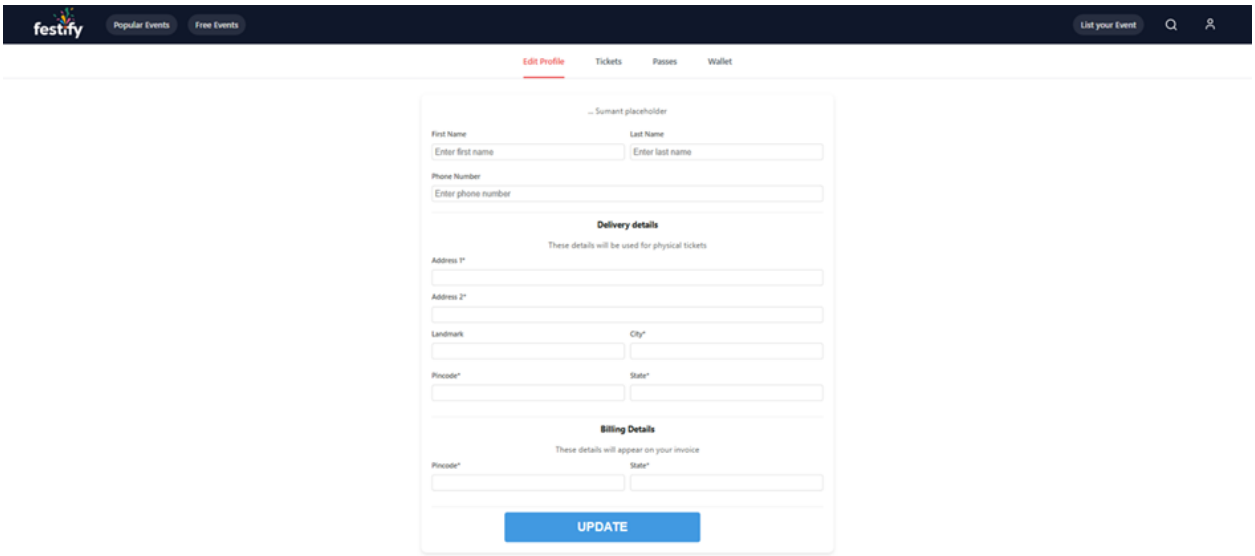


If the user already exists, a warning message is displayed:

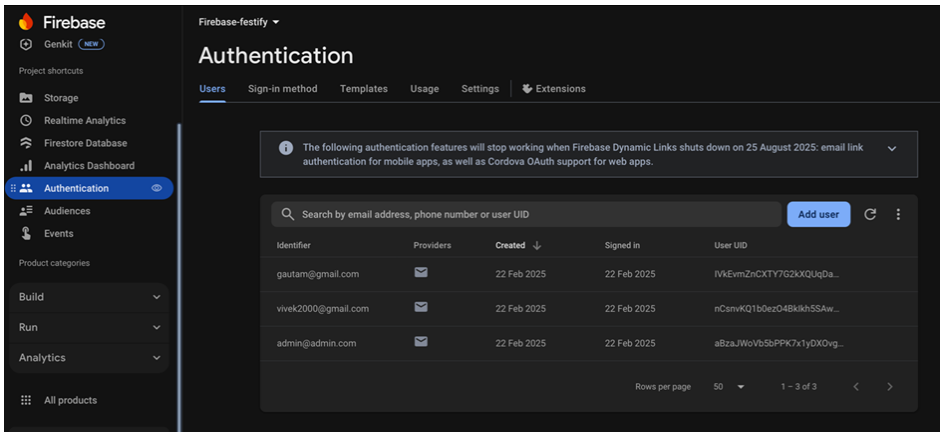


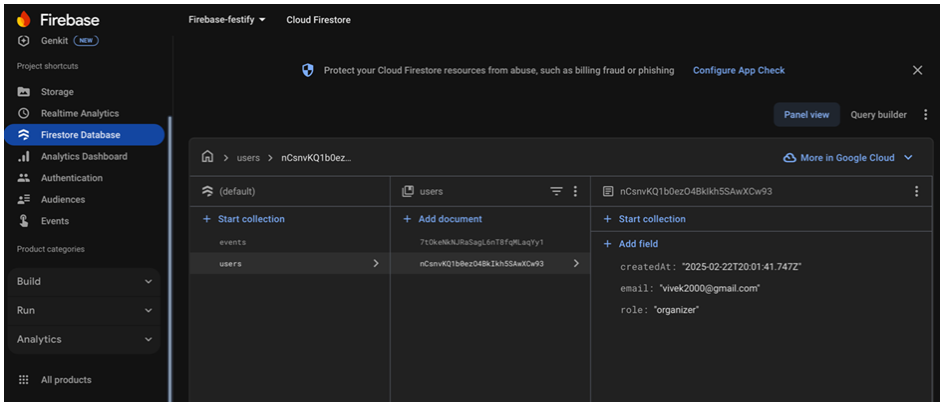
1.3 User Dashboard

After signing in, users are directed to their dashboard, where they can update or add their profile.



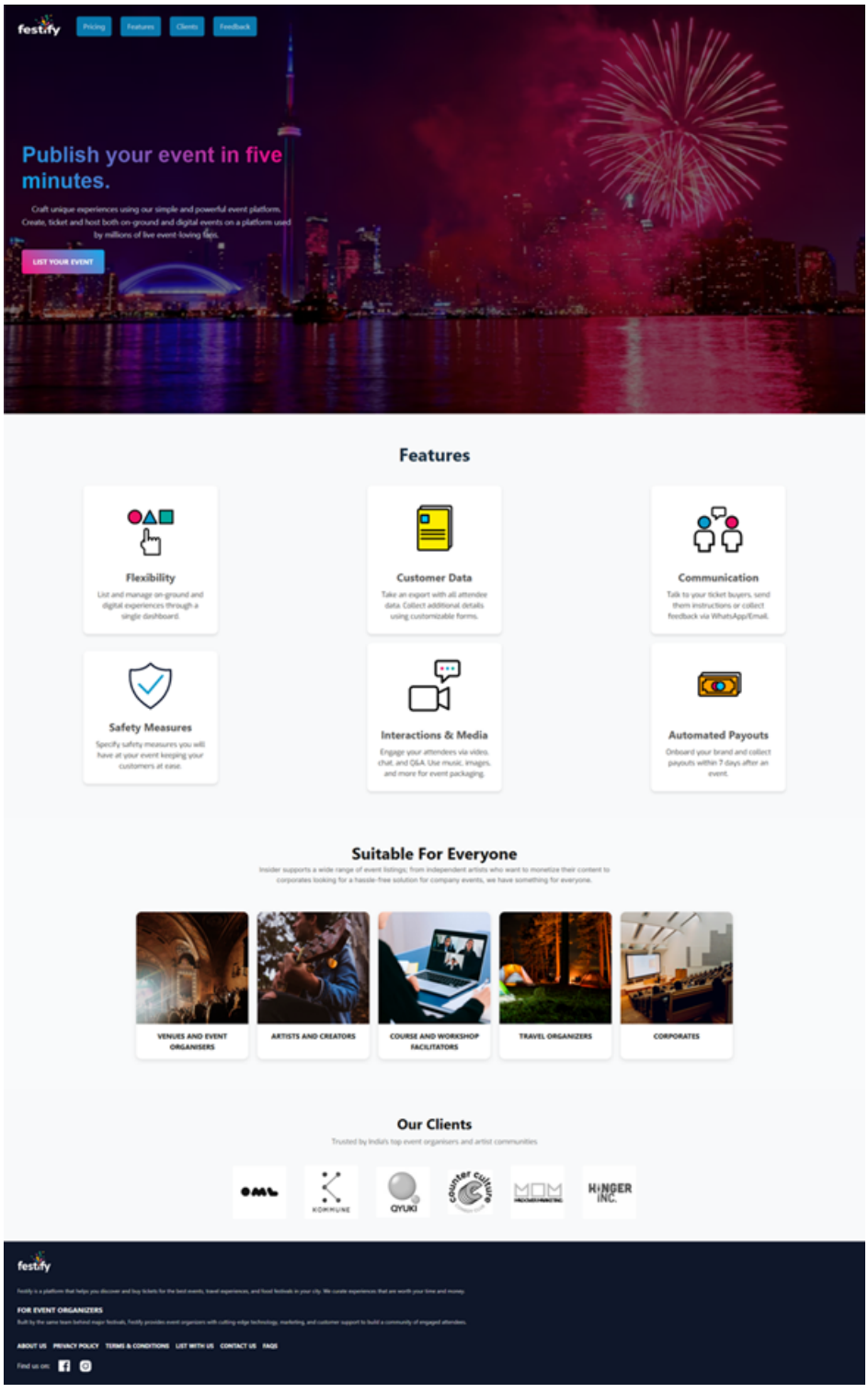
This profile is updated in the Database too:



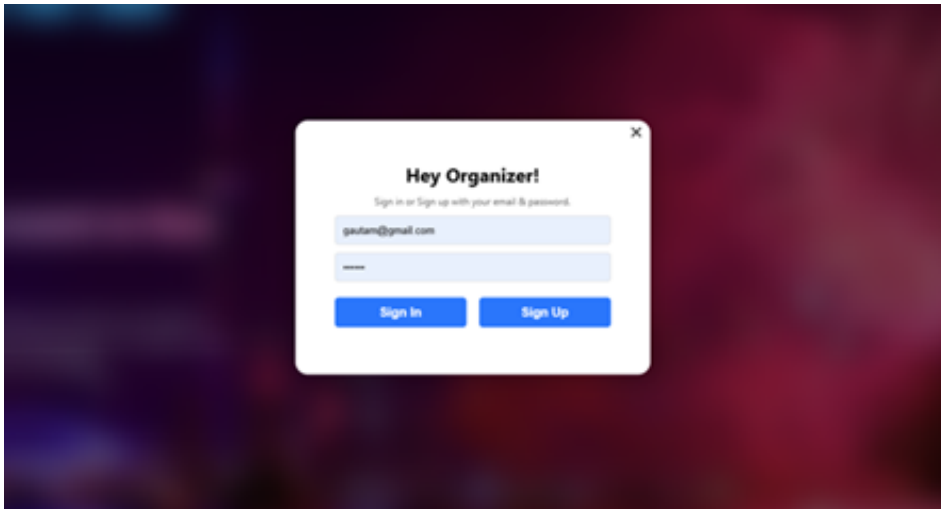


1.4 Organizer's Interface

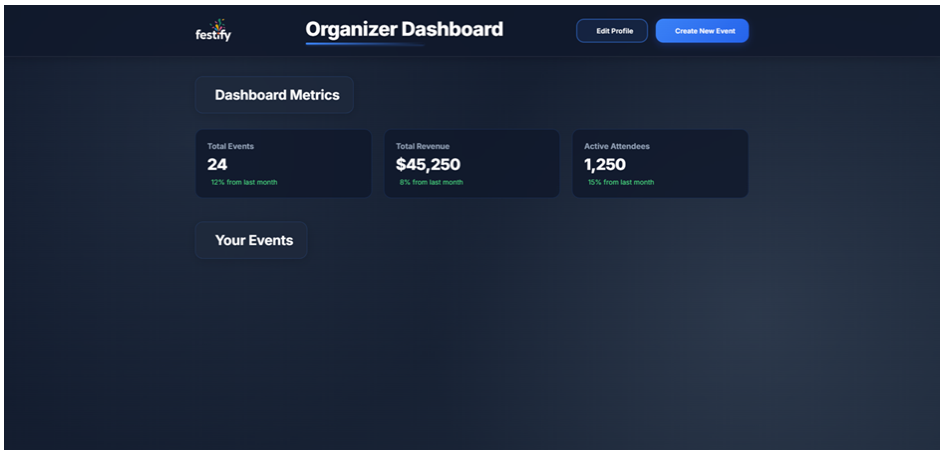
An organizer can navigate to list their event from the landing page by clicking on the *List Your Event* button, which takes them to another page showing all the details and features for the listings.



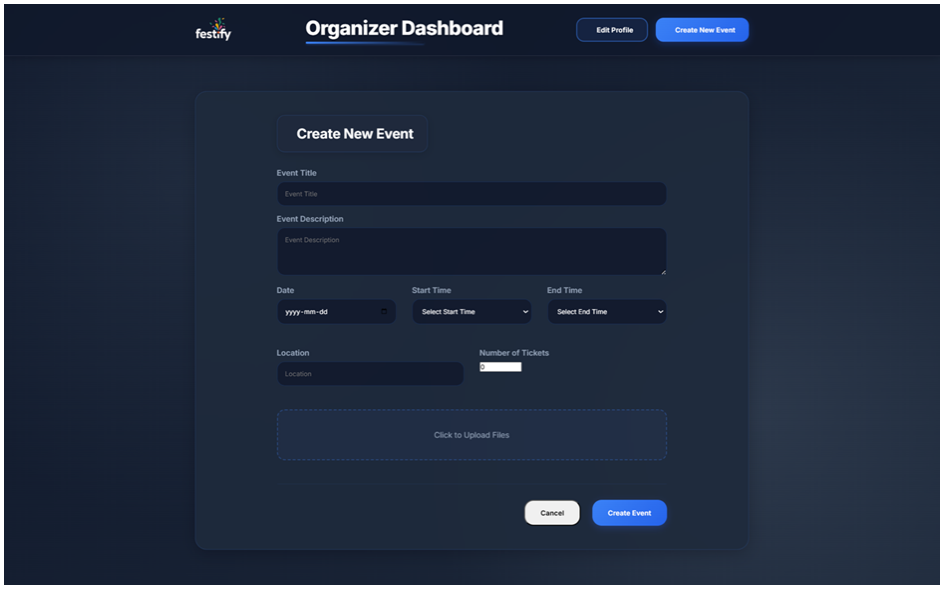
Clicking 'List Your Event' pops up the organizer registration, which on signing up takes the user to the organizer's dashboard with several options and stats.



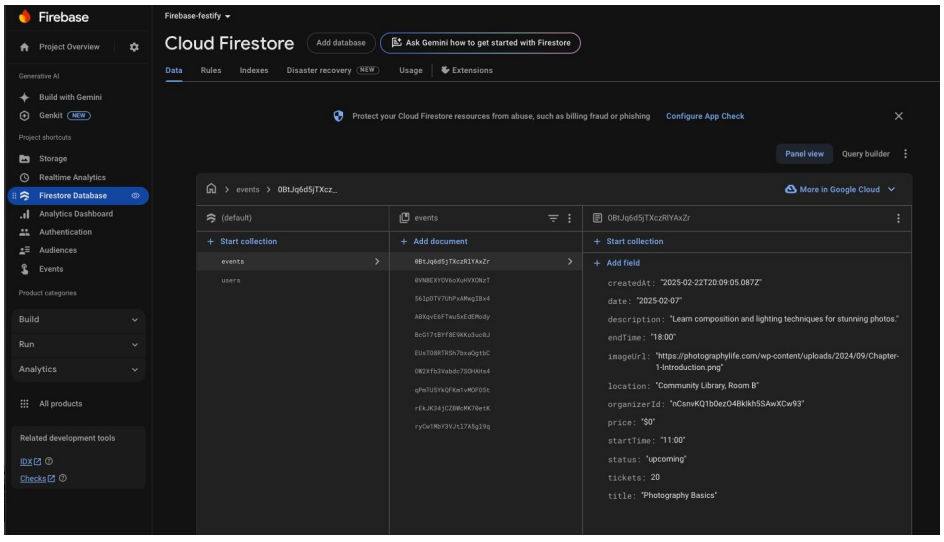
Upon successful registration, the organizer's dashboard appears:



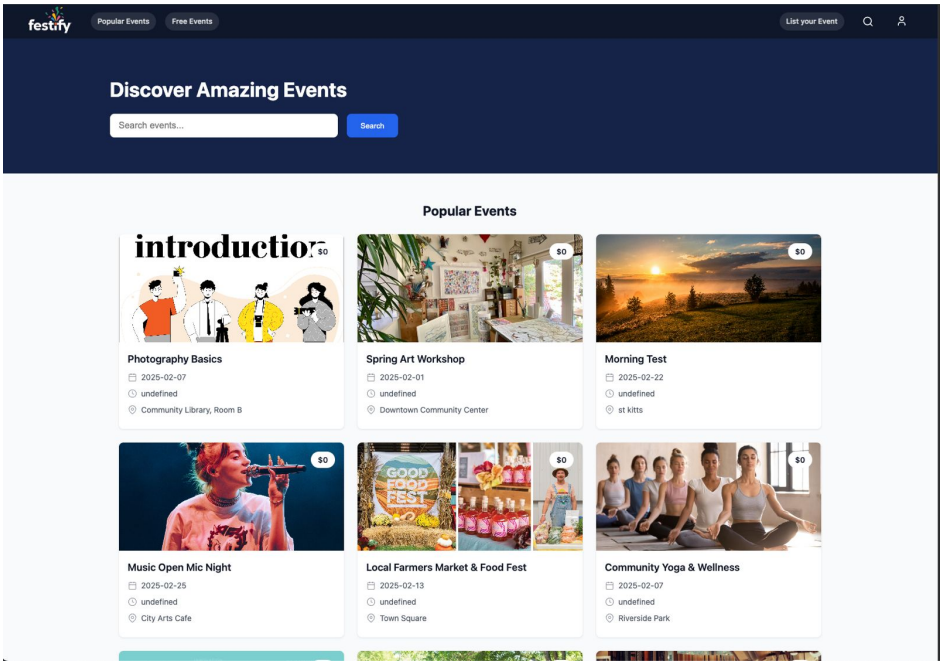
Organizers can choose the List Event Form to fill in the details for the event they are hosting and publish it. The corresponding event gets updated in the database and is visible on the landing page.



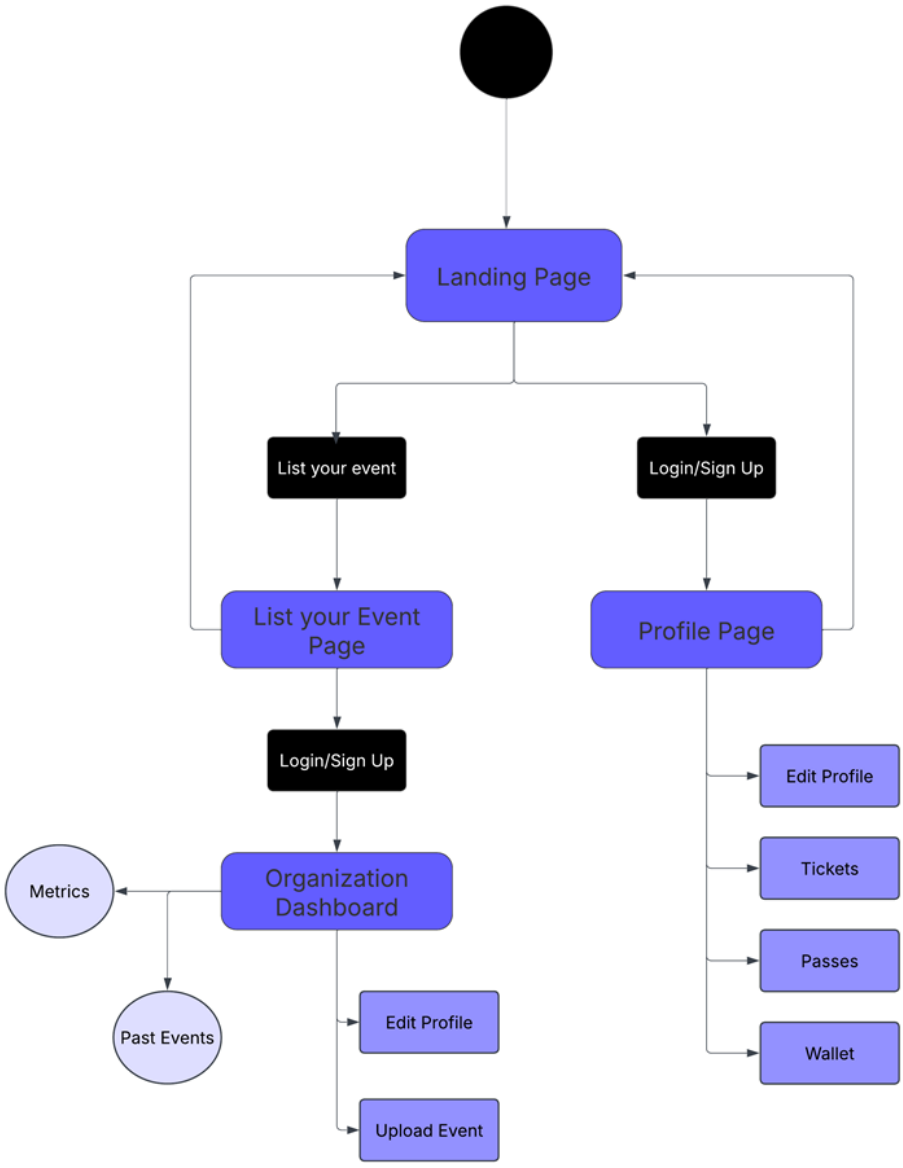
The event is recorded in the database:



Once the event form has been filled out properly, the event will become publicly visible on the main Festify page:



2 System Design and Implementation



This project is an event management website that allows organizations to create and manage events and users can use the platform to purchase tickets to those events. The website's frontend is designed using HTML/CSS and the backend is developed in JavaScript and for database we used Firebase.js. The project uses various services provided by the Firebase such as user authentication and data storage through Firestore.

The code of the project is organized in modular structure where each file is responsible for specific functionalities. This modular structure allows for easy maintainability and debugging. The Event driven architecture of the platform makes up for a responsive UI because it

dynamically updates the UI based on UI actions. Our website is structured in a way where there is a specific function tied to each user action which contributes to the responsive design. Although the codebase does not follow a specific design pattern, the code is structured in a modular way which allows for scalability.

2.1 Summary for each file

1. **list-your-event.html**: This file serves as the main interface for users to publish their events. It includes sections for event details, a form for submitting new events, and links to other parts of the application. The layout is designed to be user-friendly, encouraging quick event publishing.

2. **firebase.js**: This file contains the configuration and functions for interacting with Firebase services. It handles user authentication, event data storage, and retrieval from Firestore. This centralizes all Firebase-related logic, making it easier to manage backend interactions.

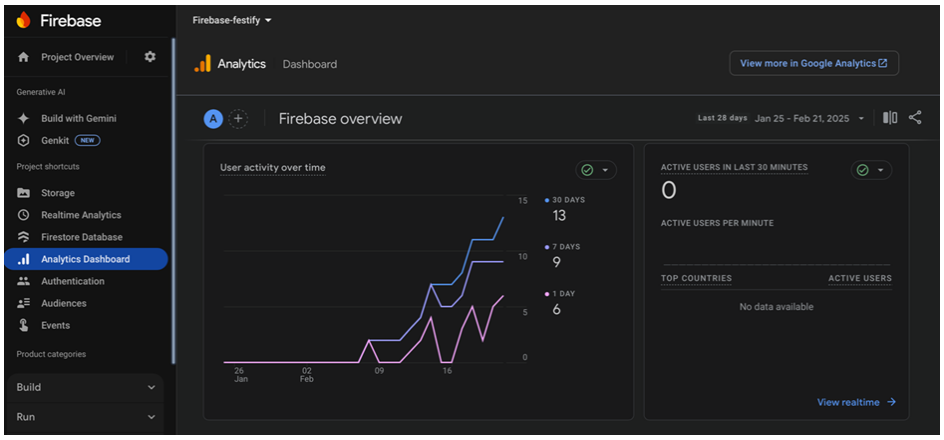
3. **organization-dashboard.html**: This file provides the dashboard interface for event organizers. It displays key metrics such as total events and revenue and includes sections for managing events and editing the organization's profile. The layout is structured to give organizers quick access to important information.

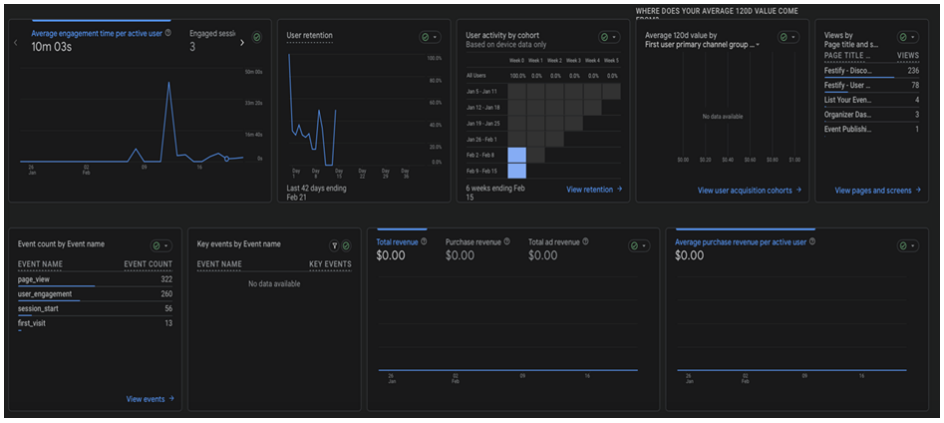
4. **script.js**: This file contains the main JavaScript logic for handling user interactions on the frontend. It manages event listeners, form submissions, and dynamic updates to the UI. This file is crucial for ensuring a responsive user experience.

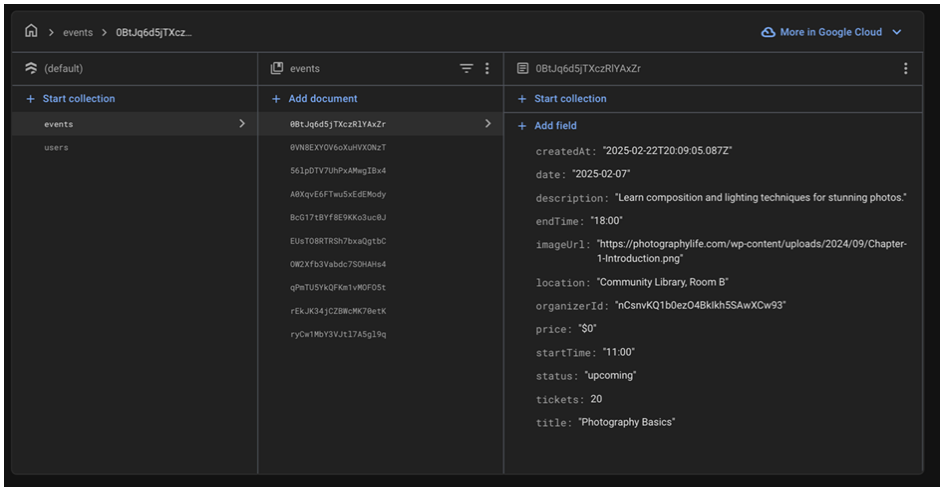
5. **organizer.js**: This file is responsible for managing functionalities related to the event organizer, such as user sign-up, sign-in, and profile management. It interacts with Firebase to handle user authentication and state changes.

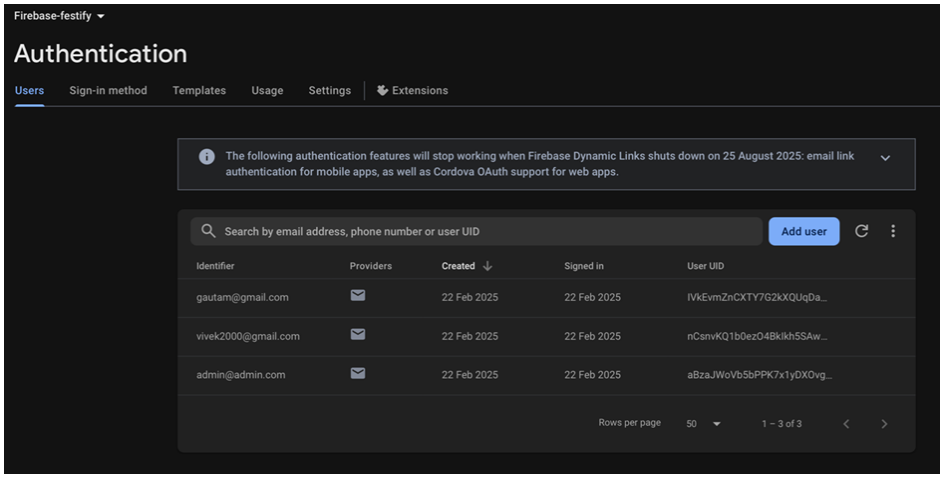
6. **styles.css**: This stylesheet defines the visual styles for the application, including layout, colors, fonts, and responsive design elements. It ensures a consistent look and feel across different components of the application.

3 Database Design









3.1 Why did we Choose Firebase?

Firebase was an excellent choice for Festify because it offers a comprehensive, scalable, and secure platform. Here's how it meets our needs:

- **Real-Time & Scalable:** Cloud Firestore provides real-time updates, ensuring that whenever events are created or modified, all users see these changes instantly. It also scales effortlessly as the number of users grows.
- **Robust Authentication:** Firebase Authentication simplifies the process of creating and managing user accounts with email/password options. It also allows for the integration of Google or multi-factor authentication in the future.
- **Serverless & Cost-Effective:** Firebase's serverless architecture eliminates the need for server management, reducing both operational complexity and costs. You only pay for the resources you use.

Application Flow:

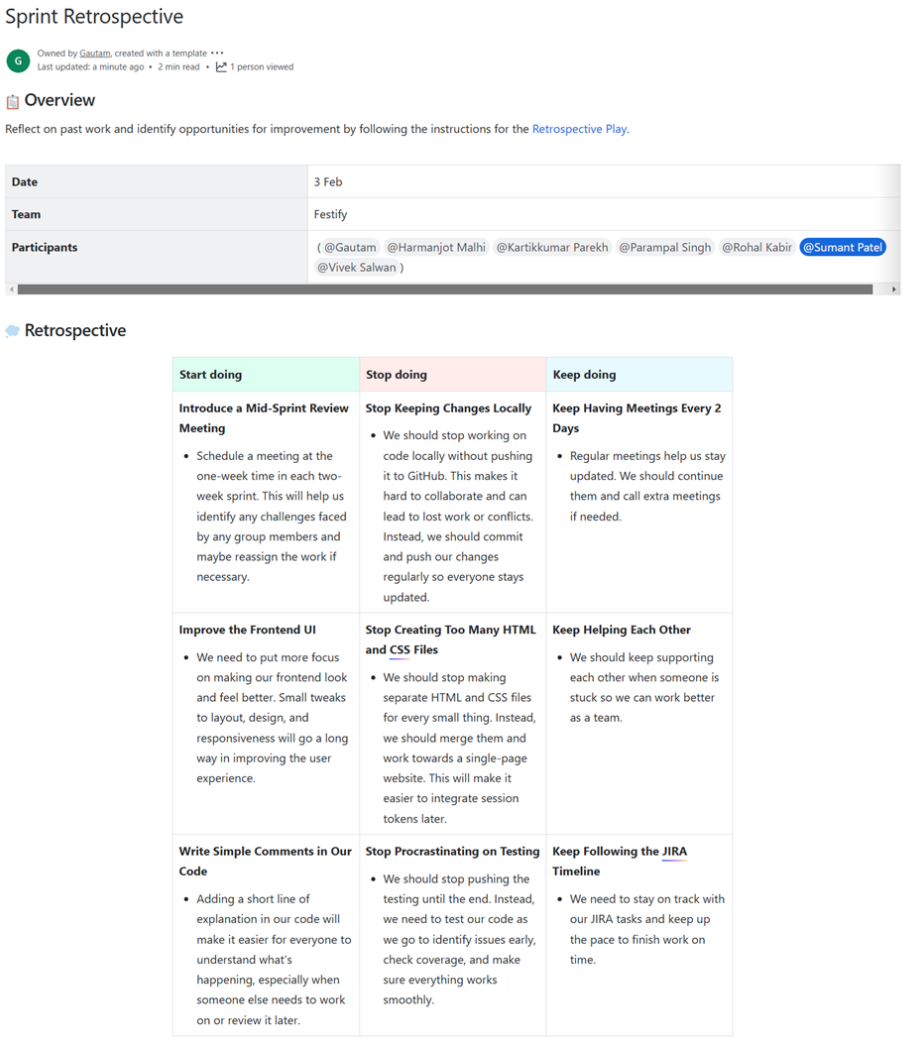
- **User Creation & Authentication:** Users are created and authenticated using Firebase Authentication. When a user signs up, a new account is generated and stored. During sign-in, Firebase verifies the credentials and grants access to personalized content.
- **Event Creation & Storage:** When an organizer creates an event, details such as the title, description, dates, and image URL are stored in Cloud Firestore. Images can be uploaded via Firebase Storage, and their download URLs are stored in the event record.

In summary, Firebase simplified the development process, secured user accounts, and effectively manages data and file storage.

4 Software Development Process and Sprints

At Festify, we used agile principles to build our event management website. Agile helped us stay flexible, adapt quickly, and focus on user needs. By working in small, manageable sprints, our team collaborated effectively and delivered new features faster.

This approach allowed us to continuously improve Festify based on user feedback. Regular updates ensured we found any issues early, making our website a reliable and user-friendly. Agile is also good for organizing tasks and allows us to easily reassign tasks in our weekly meeting if they aren't done right, making sure everything gets finished properly and on time.



Sprint Retrospective

Owned by Gautam, created with a template
Last updated: a minute ago • 2 min read • 1 person viewed

Overview

Reflect on past work and identify opportunities for improvement by following the instructions for the Retrospective Play.

Date	3 Feb
Team	Festify
Participants	(@Gautam @Harmanjot Malhi @Kartikkumar Parekh @Parampal Singh @Rohal Kabir @Sumant Patel @Vivek Salwan)

Retrospective

Start doing	Stop doing	Keep doing
Introduce a Mid-Sprint Review Meeting • Schedule a meeting at the one-week time in each two-week sprint. This will help us identify any challenges faced by any group members and maybe reassign the work if necessary.	**Stop Keeping Changes Locally** • We should stop working on code locally without pushing it to GitHub. This makes it hard to collaborate and can lead to lost work or conflicts. Instead, we should commit and push our changes regularly so everyone stays updated.	**Keep Having Meetings Every 2 Days** • Regular meetings help us stay updated. We should continue them and call extra meetings if needed.
Improve the Frontend UI • We need to put more focus on making our frontend look and feel better. Small tweaks to layout, design, and responsiveness will go a long way in improving the user experience.	**Stop Creating Too Many HTML and CSS Files** • We should stop making separate HTML and CSS files for every small thing. Instead, we should merge them and work towards a single-page website. This will make it easier to integrate session tokens later.	**Keep Helping Each Other** • We should keep supporting each other when someone is stuck so we can work better as a team.
Write Simple Comments in Our Code • Adding a short line of explanation in our code will make it easier for everyone to understand what's happening, especially when someone else needs to work on or review it later.	**Stop Procrastinating on Testing** • We should stop pushing the testing until the end. Instead, we need to test our code as we go to identify issues early, check coverage, and make sure everything works smoothly.	**Keep Following the JIRA Timeline** • We need to stay on track with our JIRA tasks and keep up the pace to finish work on time.

4.1 1st Sprint Results

In our first sprint, we worked on setting up the main features of our platform. Our focus was to make sure everything worked smoothly, and that user data was stored safely. These early steps helped us build a solid base for future updates.

1. Discover Events Page (Landing Page) We created a simple landing page that currently shows placeholder events. This gives us a starting point for adding real events in the future. Right now, it helps us test how the page looks and works.

Next Steps:

- Show real event listings instead of placeholder events.
- Add a search and filter option so users can find events easily.
- Improve the design and layout for a better user experience.

2. User Authentication (Sign-In/Sign-Up) We built a Sign-In and Sign-Up system so users can create an account and log in securely. After logging in, they are taken to their dashboard, where they can edit their profile details.

Next Steps:

- Add Google login option to make signing in easier.
- Introduce a password reset option.

3. Firebase Database Integration

- We set up a Firebase database and linked it to our authentication system.
- Any changes a user makes—such as creating an account, updating their profile, or modifying their information—are automatically updated in the database in real time.

4.2 2nd Sprint Results

In our second sprint, we focused on the organizers aspect of our website, ensuring that anyone could quickly list a new event on Festify.

1. List Your Event Page

- When a user clicks "List Your Event" on our home page, they are taken to `list-your-event.html`. This page provides an overview of our platform, which gives them an overview of Festify's features, existing clients, and benefits of listing events on our platform.
- On the same page the organizer can log in or sign up if they've not done it yet.

2. Event Creation Process

- Once the user is logged in, they are redirected to the event listing form, where they can enter all necessary event details.

3. Database Integration & Live Display

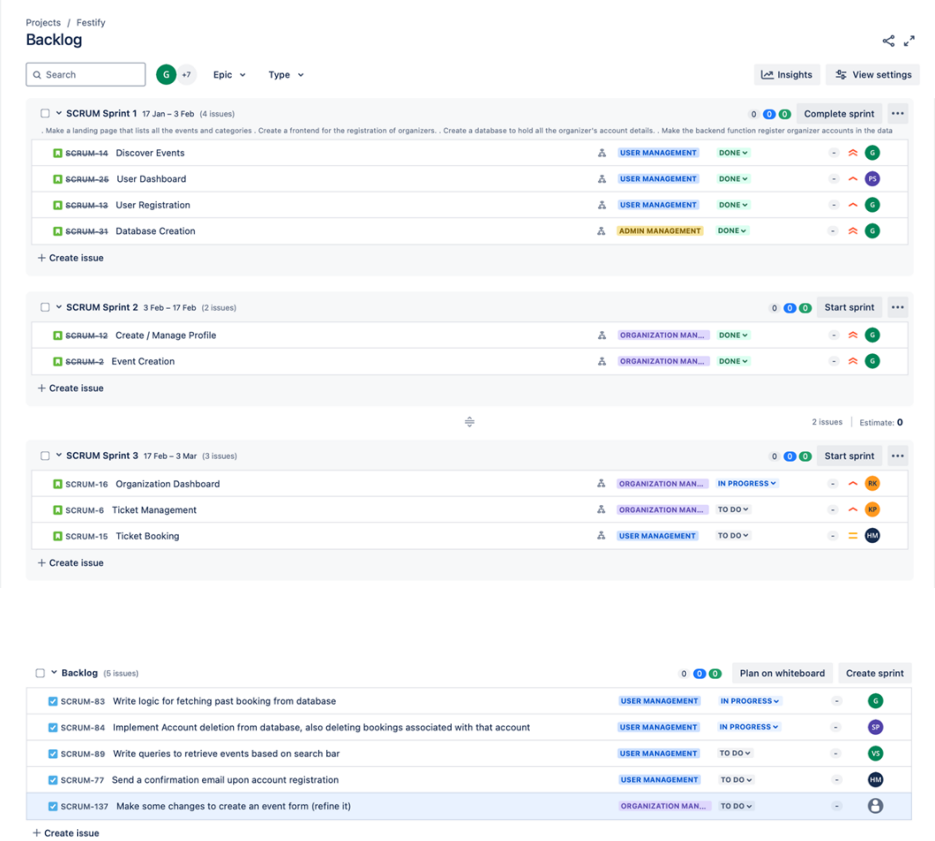
- As soon as the user submits the form, the event is immediately registered in the Firebase database.
- The newly created event is automatically displayed on the landing page, allowing users to browse through the latest events.

4.3 3rd Sprint Results (In Progress)

In Sprint 3, we are taking things further by building a fully functional Organizer Dashboard, which is currently in development.

- Now, after logging in, organizers are taken to the Organizer Dashboard instead of just the event listing form. The dashboard allows them to:
  - Edit their personal information,
  - View all past events they have created,
  - Generate new events using an event creation form.
- After this we are going to work on a ticket booking system wherein a user can actually go ahead and buy tickets, and work on a ticket management system.

4.4 Current Backlog



5 Challenges and Improvements

During our sprint, we faced several challenges that impacted our efficiency. Here's a breakdown of the issues and how we resolved them:

1. Poor Work Division & Task Overlap

- Multiple team members were working on the same features simultaneously without proper coordination.
- This led to wasted time, duplicated efforts, and conflicts when merging work.
- **Solution:** We made smaller teams with better task assignments to ensure each feature was handled efficiently.

2. Ineffective Team Structure

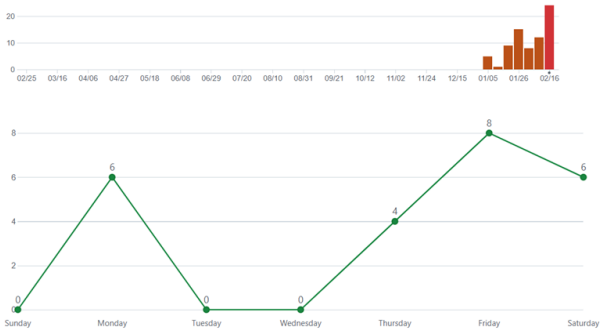
- Initially, when we divided teams based on Frontend, Backend, and Database roles.
- This created dependency, as frontend developers had to wait for backend, and database changes were often delayed.
- **Solution:** We redivided our teams to work on complete features, with both frontend and backend developers on a team working on a specific feature. We introduced mid-sprint review meetings (once a week) to help us identify any challenges faced by group members and reassign the work if necessary.

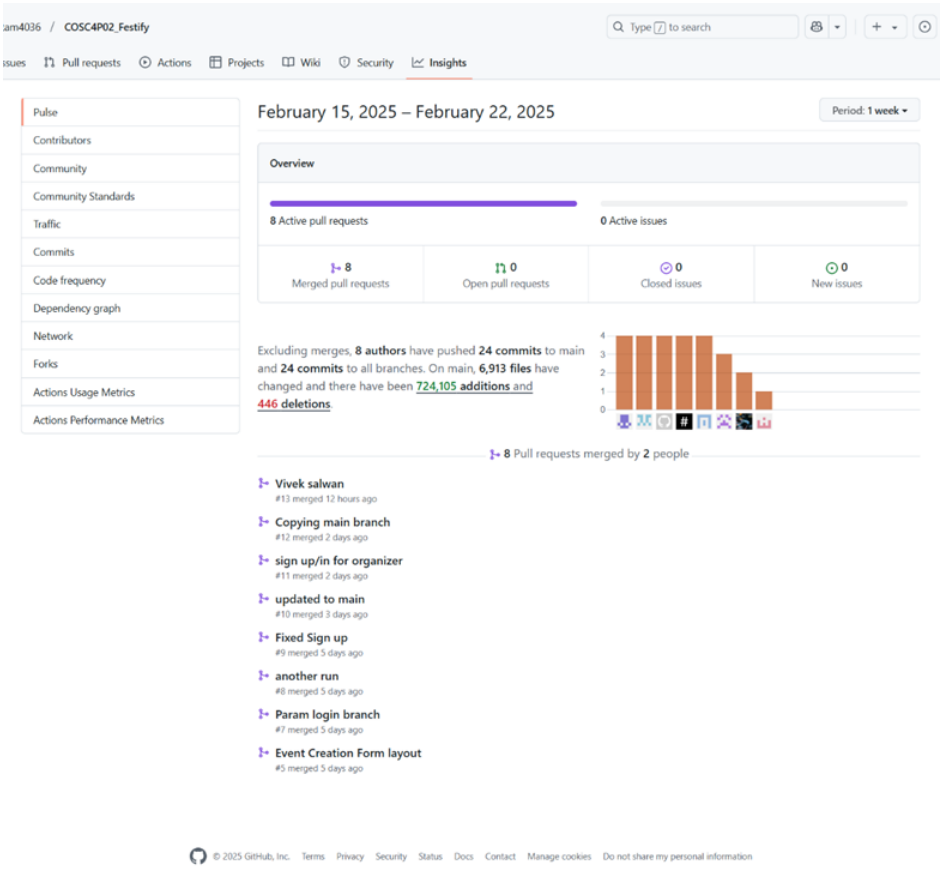
3. Technical Challenges

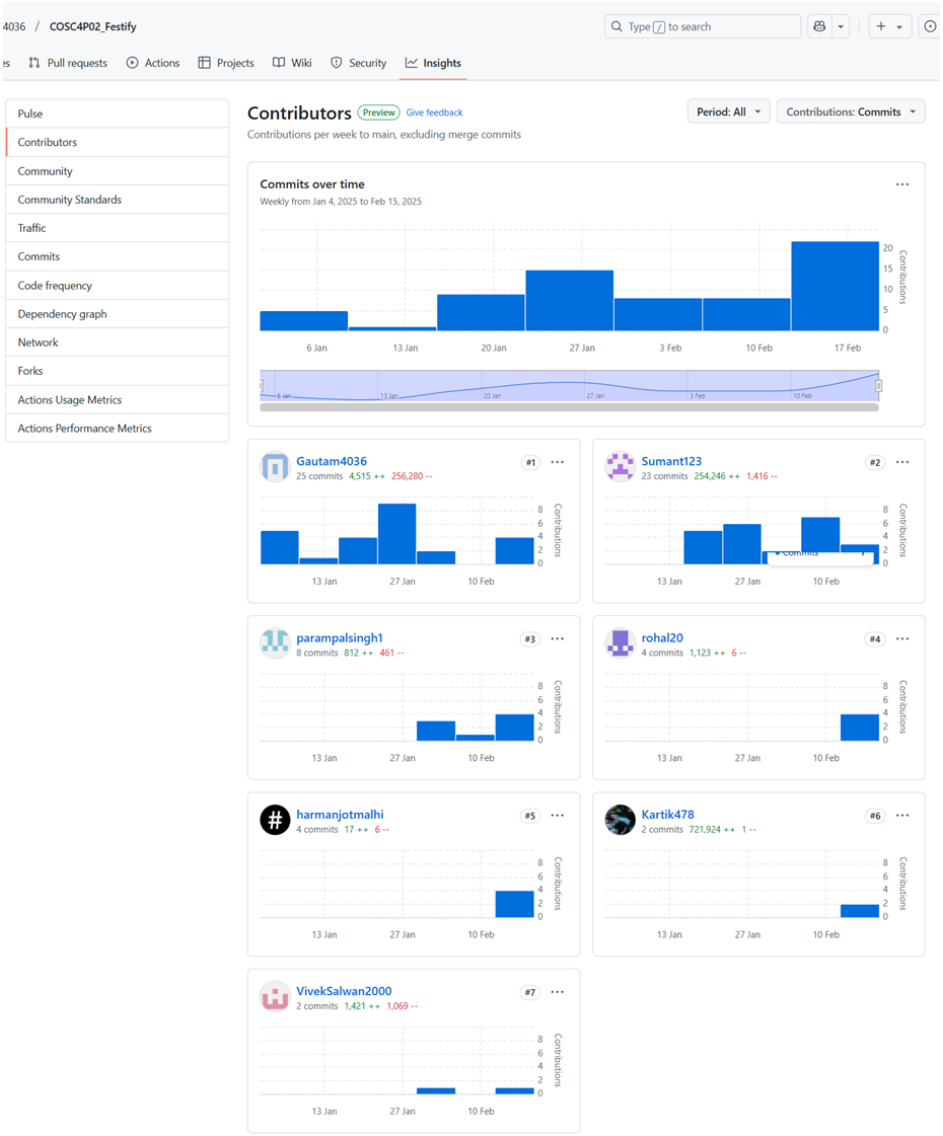
- Not committing changes to GitHub regularly: Led to lost progress and version conflicts.
- Lack of comments in the code: Made it harder for others to understand and debug.
- Delaying testing until the end: Resulted in longer debugging times and last-minute fixes.
- **Solution:** We set up regular code reviews, improved documentation, and assigned the testing part to Kartik to regularly test our code and resolve other issues.

6 GitHub Logs

This is the graph showing our commit history. The bar graph above it is for all time commits from the group and the line graph below shows the commits for the past week (Feb 15 – 22). This shows that as a group there has been consistent commits and work being done to achieve the sprint goals.







This screenshot shows each member's commitment; it should be noted that overall contributions to the project should not be fully judged here as during the beginning of the project team members were not pushing or committing properly and some members would work together on the same computer. Also, group members such as Vivek worked on the database (Firebase) and other features included in that UI such as authentication which cannot be visibly seen in the GitHub logs as there is no way to show any commit action for that.

Link to Commits: https://github.com/Gautam4036/COSC4P02_Festify/commits/main

7 Testing in Jest for JavaScript

Explanation of Jest tests

Below is a detailed explanation of each of the Jest test files for our Festify webcode. The goal is to clarify how these tests work, what they test, and the rationale behind them.

1. `app.test.js`

Overview: This test file verifies logic contained in our `app.js` file. Specifically, it tests:

- `createEventCard(event)`: Generates an HTML string for an event card.
- `renderEvents()`: Renders multiple events into a container on the DOM.
- `loadFooter()`: Dynamically fetches and injects a footer HTML snippet into the DOM.

Structure: At the top, we have:

```
/**
 * @jest-environment jsdom
 */
```

This tells Jest to use the jsdom environment (simulating a browser-like DOM) for these tests.

(a) `createEventCard()` Test

Purpose: Ensures that the function returns an HTML string containing all the relevant event data (title, date, location, time, imageUrl, price).
Mechanism:

- A mock event object is created with specific values.
- `createEventCard` is called with that object.
- The returned HTML string is checked (via `expect(...).toContain(...)`) to see if it has the event's properties.

(b) `renderEvents()` Test

Purpose: Checks that the events are correctly rendered to a DOM container with the ID `eventsGrid`.
Mechanism:

- JSDOM is used to simulate the presence of a `<div id="eventsGrid">`.
- `window.events` is overridden with test data (so that `renderEvents()` can pick them up).
- `renderEvents()` is called, and the rendered HTML is checked to see if it contains the titles of the mock events.

(c) `loadFooter()` Test

Purpose: Verifies that the footer is fetched and appended to the DOM or, if fetch fails, logs an error.
Mechanism:

- **Success Case**: `fetch` is mocked to resolve with some test footer HTML. Then `loadFooter()` is called and is expected to populate the body with the test footer content.
- **Failure Case**: `fetch` is mocked to reject with an error, triggering the error path in `loadFooter()`. A spy on `console.error` verifies the error message is logged.

2. `firebase.test.js`

Overview: This file tests the Firebase-related functions in `firebase.js`. Because these functions rely on Firebase Auth and Firestore, the test file uses Jest mocks to simulate the Firebase modules.
Key Point: These mocks replace real Firebase calls with Jest mock functions. This allows testing logic *without* making actual network requests to Firebase.

(a) `signUpUser()` Test

Purpose: Ensures that `signUpUser` properly calls `createUserWithEmailAndPassword` with the correct arguments.
Mechanism:

- A mock resolved value `{ user: { uid: '123' } }` is set.
- Then `signUpUser` is called.
- The test checks if the mock function was called with the correct parameters and if the returned value matches the mock object.

(b) `signInUser()` Test

Purpose: Same pattern as `signUpUser` but for `signInWithEmailAndPassword`.

(c) `signOutUser()` Test

Purpose: Confirms that `signOutUser` calls the `signOut` function.

(d) `onUserStateChanged()` Test

Purpose: Ensures the callback is registered correctly with Firebase Auth.

(e) `saveUserProfile()` Test

Purpose: Ensures that user profile data is saved (merged) in Firestore.

(f) `getUserProfile()` Tests

Purpose: Tests retrieving user data from Firestore.
Mechanism:

- Mocks `getDoc` to simulate "exists" or "does not exist" in Firestore.

3. `inline.test.js`

Overview: This file tests inline JavaScript functions presumably added in `index.html` (or another HTML file). They are not in a separate module, so you define them on `window` within the test. Because these are DOM-related features (like toggling classes, form submission, etc.), jsdom is again crucial.

```
/**
 * @jest-environment jsdom
 */

// Add the dummy implementation for the unimplemented API.(was throwing an error...)
HTMLFormElement.prototype.requestSubmit = function() {};
```

Explanation: `requestSubmit` is not implemented by jsdom by default, so you provide a no-op to prevent errors.

(a) `toggle()` and `closePopup()` Tests

Purpose: Tests the toggling and closing functionality for our popups.
Mechanism:

- The DOM is set up with `content` and `popup` elements having initial classes (`active`, `hidden`).
- The inline functions are defined on `window`.
- The test then asserts that toggling adds/removes classes as expected.

(b) Login Form Click Handlers Test

Purpose: Verifies that clicking the sign-in or sign-up buttons calls the right functions with the right arguments.
Mechanism:

- Creates a `signInUser` and `signUpUser` mock function.
- Simulates button clicks via `dispatchEvent`.
- Checks that the mock functions have been called with the correct credentials.

4. `script.test.js`

Overview: This file tests various utility and UI functions in `script.js`:

- **Utility**: `formatDate`, `formatCurrency`
- **Event Rendering**: `renderEvents`
- **Form Handling**: `showEventForm`, `hideEventForm`, `showProfileForm`, `hideProfileForm`, `resetDashboard`
- **File Input Trigger**: `triggerFileInput`

Structure:

(a) `formatDate()` Test

Purpose: Confirms that the date is formatted in a readable style (though the exact format can vary by locale).
Mechanism: A known date is passed, and the output is checked with regexes or partial string checks.

(b) `formatCurrency()` Test

Purpose: Ensures currency values are properly formatted with a dollar sign and comma separators.

(c) `renderEvents()` Test

Purpose: Confirms that calling `renderEvents()` populates the DOM with event data.
Mechanism: The function presumably has a default set of events or references `window.events`. After calling, the test checks the resulting HTML to confirm known events appear.

(d) Form Display Functions

Purpose: These tests verify that calling show/hide methods sets the correct `display` styles for the relevant sections.
Key: They also ensure that when the forms are shown, the metrics or events sections are hidden (and vice versa).

(e) `triggerFileInput()` Test

Purpose: Checks that calling `triggerFileInput()` programmatically clicks a hidden `<input type="file">` element, which is a common pattern to open a file picker dialog from a custom button.
Mechanism: The test replaces the `.click()` method with a Jest spy, calls `triggerFileInput()`, and expects the spy to have been triggered.

Key Takeaways

- **jsdom Environment:** All tests that manipulate the DOM (via `document`) use `@jest-environment jsdom` at the top. This simulates a browser-like environment in Node.js.
- **Mocking:**
  - **Firebase:** The `firebase.test.js` file demonstrates how to mock external modules (Firebase) using Jest. This avoids real network requests and verifies that our code calls Firebase APIs correctly.
  - **`fetch`:** In the `loadFooter()` tests, `global.fetch` is mocked to simulate both success and failure.
- **`beforeEach` vs `beforeAll`:** `beforeEach` is used to reset or re-initialize test conditions (e.g., the `document.body.innerHTML`) for each test, ensuring tests are isolated and do not affect each other.
- **Testing DOM Manipulation:** When testing UI-related functions (`toggle`, `showEventForm`, `renderEvents`, etc.), set up necessary elements in `document.body.innerHTML`, call the function, and check classes or `innerHTML` for correct behavior.
- **Spies and Mocks:**
  - `jest.fn()` creates a mock function that can track calls.
  - `jest.spyOn` allows you to spy on existing object methods (e.g., `console.error`) without fully replacing them unless needed.
  - For example, in the `loadFooter()` test, `console.error` is spied on to confirm it logs the correct message if `fetch` fails.

- **Coverage Report:** The test suite automatically generates a coverage report with an `index.html` file. This file is created during the test run and provides a visual summary of code coverage.

7.1 Reasons for Using Babel

Why Used Babel?

Our source files (like app.js and firebase.js) use ES module syntax. If you change our tests to use CommonJS (require), our source files must also be compatible with CommonJS. Jest now supports ES modules—but only when configured properly using Babel or Jest's ESM support.

7.2 Reasons for Using Jest-JUnit

Why Choose Jest-JUnit?

A popular choice is to use the jest-junit reporter, which outputs test results in a JUnit XML format. This makes it easy to integrate with CI systems, while also ensuring that a detailed coverage report (with a generated `index.html`) is automatically created.

7.3 Mocking Remote Firebase Modules

Handling Remote Firebase Modules

Since Jest cannot load modules from remote URLs, we had to configure Jest's moduleNameMapper to use local mock implementations for Firebase modules. The following mappings were added to `jest.config.js`:

```
"moduleNameMapper": {
  "^https://www\.gstatic\.com/firebasejs/11\.2\.0/firebase-app\.js$": "<
     rootDir>/__mocks__/firebase-app.js",
  "^https://www\.gstatic\.com/firebasejs/11\.2\.0/firebase-auth\.js$":
     "<rootDir>/__mocks__/firebase-auth.js",
  "^https://www\.gstatic\.com/firebasejs/11\.2\.0/firebase-firestore\.
     js$": "<rootDir>/__mocks__/firebase-firestore.js",
  "^https://www\.gstatic\.com/firebasejs/11\.2\.0/firebase-analytics\.
     js$": "<rootDir>/__mocks__/firebase-analytics.js"
}
```

This ensures that Jest loads local mock implementations instead of attempting to fetch Firebase modules from the web.

7.4 Some Test Failures and Solutions

7.4.1 Failures Encountered

The following errors were observed:

1. **Unimplemented Browser APIs:** jsdom does not implement some methods like `HTMLFormElement.prototype.requestSubmit`, causing errors in inline tests.
2. **Remote Module Imports:** Jest cannot load Firebase modules imported using remote URLs (e.g., `https://www.gstatic.com/firebasejs/11.2.0/firebase-auth.js`).
3. **ES Module Syntax Errors:** Jest runs in a CommonJS environment, whereas the test files use ES module syntax.

7.4.2 Solutions Implemented

Fixes for Test Failures

- Added a Babel configuration file `babel.config.js` to enable ES module support in Jest.
- Installed necessary Babel dependencies using:

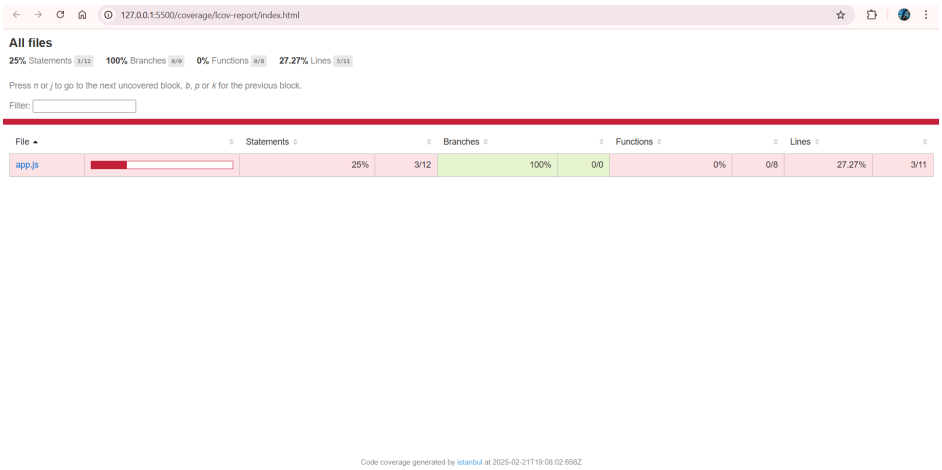
```
npm install --save-dev @babel/core @babel/preset-env babel-jest
```

- Mocked missing browser APIs in jsdom to prevent errors.
- Changed test imports to be compatible with CommonJS where needed.

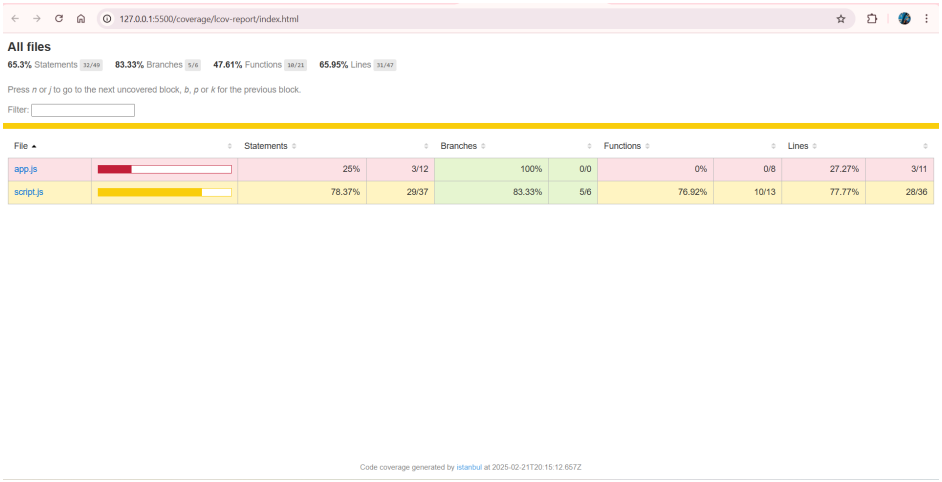
7.5 Coverage Results

The Jest test coverage results from multiple runs are provided below:

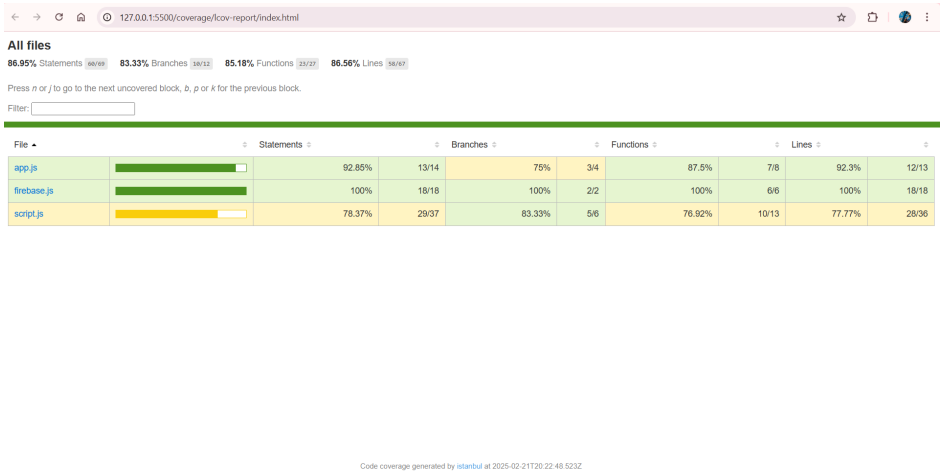
Test Coverage Results 1



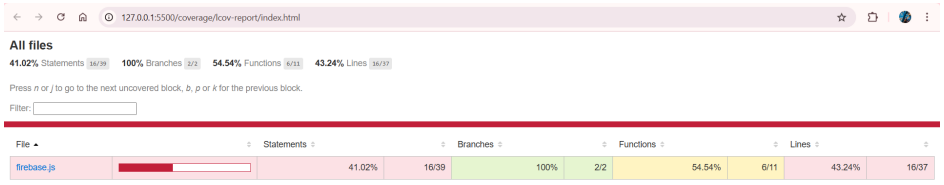
Test Coverage Results 2



Test Coverage Results 3



Test Coverage Results 4



Final Notes from Testing

These tests collectively ensure our major functionalities (DOM rendering, Firebase interaction, user interface toggles, etc.) behave as intended. While some tests verify the presence of text in the DOM (`expect(...).toContain(...)`), others verify method calls and parameters (`toHaveBeenCalledWith`). This comprehensive approach covers both "happy path" and error/edge cases, giving confidence that the key parts of our application are functioning correctly.

Additionally, a detailed coverage report in the form of an `index.html` file is automatically generated during test runs. This file provides a clear, visual breakdown of our code coverage, further assisting in identifying areas for improvement.

Conclusion and Future Planning

This progress report details the initial development and deployment of the **Event Management Platform (Festify)**. Future sprints will focus on enhancing functionality, improving user interface designs, and integrating additional features for robustness and scalability.