|  |
| --- |
| (20067691) Dean Sinnott  BSc in Applied Computing |

|  |
| --- |
| South East Technological University |
| Progressive Web Application for Event Management |
| Eventivity |

A logo for a university

AI-generated content may be incorrect.

Contents

[Abstract 3](#_Toc196946152)

[Introduction 4](#_Toc196946153)

[Project Background 5](#_Toc196946154)

[Objectives 6](#_Toc196946155)

[Problem Statement 7](#_Toc196946156)

[Choosing a Progressive Web App approach 8](#_Toc196946157)

[System Overview 8](#_Toc196946158)

[Technology Stack 10](#_Toc196946159)

[Frontend 10](#_Toc196946160)

[Reat 10](#_Toc196946161)

[Vite 10](#_Toc196946162)

[Tailwind CSS 10](#_Toc196946163)

[React Router 10](#_Toc196946164)

[Backend 11](#_Toc196946165)

[AWS Lambda 11](#_Toc196946166)

[AWS API Gateway 11](#_Toc196946167)

[AWS Cognito 11](#_Toc196946168)

[AWS S3 11](#_Toc196946169)

[AWS DynamoDB 11](#_Toc196946170)

[AWS IAM 12](#_Toc196946171)

[Google Maps API 12](#_Toc196946172)

[Hosting 12](#_Toc196946173)

[AWS Amplify 12](#_Toc196946174)

[Development Tools 12](#_Toc196946175)

[Vite Plugin PWA 12](#_Toc196946176)

[GitHub 13](#_Toc196946177)

[Postman 13](#_Toc196946178)

[Visual Studio Code 13](#_Toc196946179)

[Development Methodology 13](#_Toc196946180)

[App Components 15](#_Toc196946181)

[Event Map Component 15](#_Toc196946182)

[Forum Post Component 16](#_Toc196946183)

[Post list Component 17](#_Toc196946184)

[Comment Card and Comment List Component 18](#_Toc196946185)

[Event List Component 19](#_Toc196946186)

[Workshop Card and Modal Component 19](#_Toc196946187)

[Navigation Components 20](#_Toc196946188)

[Event Navigation Bar 21](#_Toc196946189)

[Landing Page Navigation Bar 21](#_Toc196946190)

[Map Icon Button 22](#_Toc196946191)

[Pages 22](#_Toc196946192)

[Landing Page 22](#_Toc196946193)

[Event Home Page 23](#_Toc196946194)

[Schedule Page 23](#_Toc196946195)

[Forum Page 24](#_Toc196946196)

[Workshops Page 24](#_Toc196946197)

[My Calendar Page 25](#_Toc196946198)

[Profile Page 26](#_Toc196946199)

## Abstract



Eventivity is a progressive web application (PWA) designed to serve as a dynamic, adaptive hub for large-scale local events. The idea originated from a real-world scenario: a close friend of mine is hosting the European Architecture Students Assembly (EASA) in Waterford in 2026, and I initially aimed to build a simple app tailored just for that event. However, through early discussions with my supervisor, the scope expanded. I was encouraged to introduce greater complexity and broader usefulness, ultimately shaping the project into a flexible platform that can adapt to multiple events while maintaining a consistent and intuitive user experience.

The core design of the app focuses on minimalism and clarity. It allows users to quickly find key event information, engage with content through forum posts, and benefit from themed navigation based on the event context. From a technical perspective, this project gave me the opportunity to apply my knowledge from the Cloud Computing stream of my course. I adopted a modern development approach that balanced custom control with cloud-managed services. For areas like frontend styling and routing, I retained full control using React and Vite. For identity management, media handling, serverless functions, and data storage, I leveraged AWS services, specifically Cognito, S3, Lambda, DynamoDB, and Amplify.

The result is a functional and scalable app skeleton that can support themed content, media uploads, user interaction, and offline capabilities. While the project is already usable in its current form, I’ve also identified clear paths for continued development such as dynamically fetching events and assets from the database, improving mobile layout, and introducing user chat and notifications. This will ultimately cause the application to be truly dynamic and rely less on static assets.

## Introduction

The inspiration for Eventivity came from a conversation with a close friend who had recently returned from an EASA (European Architecture Students Assembly) event in Spain. He mentioned that he had successfully bid to bring the event to Waterford in 2026. I saw this as a unique opportunity to align a real-world event with my final year project. My initial intention was to build a single-purpose web application tailored specifically to that event. However, after discussing the idea with my project supervisor, the concept began to grow in scope. The project evolved into a more ambitious goal of creating a scalable platform capable of supporting multiple large local events through a single adaptive interface.

Eventivity is designed to be more than just an information source. It acts as a digital companion for event attendees, offering themed navigation, workshop schedules, a user forum, and media interaction within a clean, app-like experience. I focused on delivering a minimalist and intuitive design to help users quickly find key information while also encouraging deeper engagement. One of the app’s strengths is its flexibility. It can present a unique visual theme and content for each event while retaining a consistent structure and set of core features.

As a student in the cloud computing stream of Applied Computing, I was eager to apply my cloud knowledge in a real development setting. This led me to take a hybrid approach. I chose to retain full control over areas such as the user interface and navigation while relying on cloud services to handle identity, storage, scalability, and deployment. By integrating AWS services such as Cognito, Lambda, DynamoDB, and Amplify, I was able to deliver a modern, maintainable, and scalable solution that reflects both the practical and theoretical skills I developed throughout my course.

The main learning objective of this project was to build a full-stack web application using a modern development approach that integrates both custom-built and cloud-managed components. I wanted to gain a deeper understanding of the trade-offs involved in software architecture decisions, while producing a real, working system that could be deployed, tested, and used in a real-world context.

## Project Background

I chose this project because I wanted to create something that would last beyond the final year deadline. From the beginning, my goal was to develop a working foundation that could be built on in a real-world context. The opportunity to work closely with a good friend made it even more appealing. We had previously collaborated on his architecture final year project and discovered that we work well together. We both take a careful and structured approach to problem-solving, breaking ideas into manageable parts and discussing each decision thoroughly before progressing. That dynamic helped shape the project into something both functional and carefully considered.

In addition to the personal motivation, I wanted this project to reflect a real-world development scenario. I took on the role of developer while collaborating with someone who understood the needs and goals of the event. This created a realistic workflow where decisions were shaped by actual user expectations and feedback. I also considered the growing trend of digital platforms in modern services. Whether ordering coffee or attending a music festival, users are increasingly supported by mobile applications that deliver key information and streamline engagement. I saw a clear opportunity to offer the same experience to local and academic events through a single, accessible platform.

The idea of reusability became the central theme of my project. Many event apps are built to serve one specific purpose and are quickly abandoned afterward. I wanted to build a structure that could support multiple events, each with its own identity, while retaining a consistent and reliable user experience. The core features of the application, such as navigation, schedules, and user interaction, were designed to remain the same. This consistency helps users become familiar with the app quickly, even as the appearance and event content change. A reusable structure also makes the system easier to maintain and extend in the future. I have always believed that when simple elements are combined thoughtfully, they create a solution that is both powerful but easy to understand. That belief shaped every aspect of Eventivity's design and development.

## Objectives

The primary aim of this project was to develop a modern, reusable progressive web application capable of supporting multiple large-scale events through a single adaptive platform. To achieve this, the following objectives were identified at the outset of the project:

1. Develop a reusable progressive web application that can serve as a central hub for hosting diverse event types.
2. Enable dynamic theming and content adaptation to reflect the branding and identity of each event.
3. Ensure users can access essential event information quickly through a minimalist and intuitive interface.
4. Implement user authentication and basic account functionality using AWS Cognito.
5. Allow authenticated users to contribute forum posts with support for both text and media, including image and video uploads.
6. Provide offline access through service worker caching, ensuring the app remains usable in areas with limited connectivity.
7. Integrate modern cloud technologies to reduce infrastructure maintenance while retaining control over user experience and interface design.
8. Design the application to be fully responsive and accessible across a wide range of devices, including mobile and desktop platforms.
9. Create scalable foundation for future development, including features such as dynamic content fetching from DynamoDB, user notifications, and calendar integration.

These objectives were designed to guide the project through both the initial development phase and into long-term development.

## Problem Statement

Event-focused applications are typically designed with a single, specific purpose in mind. In many cases, these apps are built for large-scale commercial events and are discarded once the event concludes. This approach results in a lack of reusability and creates unnecessary overhead for developers and organisers when preparing for future events. Smaller events, especially local or academic ones, often do not receive the same level of digital support. As a result, organisers may rely on scattered channels such as email, social media, or printed materials to communicate with attendees. This can lead to confusion, missed updates, and a fragmented user experience.

A significant issue in current event app design is the lack of adaptability. Most existing solutions do not allow for simple repurposing or dynamic theming, which means that building an app for a new event often requires starting from scratch. This not only increases development time but also limits opportunities for design consistency and user familiarity across events.

In addition, many event venues experience poor network coverage, especially in rural or temporary settings. Without offline functionality, attendees may lose access to important information at the moments they need it most. For smaller event organisers who may not have the budget or technical resources to address these issues, there is a clear need for a lightweight, flexible, and cost-effective digital platform.

Eventivity addresses these challenges by offering a reusable and adaptive progressive web application that maintains a consistent structure while allowing each event to customise its content and appearance. It is designed to improve accessibility, reduce development effort, and enhance the attendee experience, particularly for events that would otherwise go unsupported by traditional apps.

## Choosing a Progressive Web App approach

As part of the early planning phase, I made the decision to build Eventivity as a Progressive Web Application (PWA). This architectural choice was influenced by the unique needs of the target audience and the challenges associated with delivering digital services at temporary or rural event locations. Most attendees expected at the 2026 architecture event will be travelling from outside Ireland. This raised concerns around mobile data usage and network coverage during their stay. I wanted to build an application that was lightweight, easily installable, and did not rely on constant internet connectivity. A PWA offered all these advantages.

Progressive Web Apps combine the best features of web and native applications. They can be installed directly onto a user’s device without going through an app store, avoiding vendor locking. PWAs also support offline functionality through caching, which allows users to access event information even in areas with limited or no internet access. This was a critical feature for the app, ensuring that users could always view schedules, maps, or other key content regardless of their connection status

Vite supports Progressive Web App development through a dedicated plugin called vite-plugin-pwa, which simplifies the integration of service workers and asset caching.

## System Overview

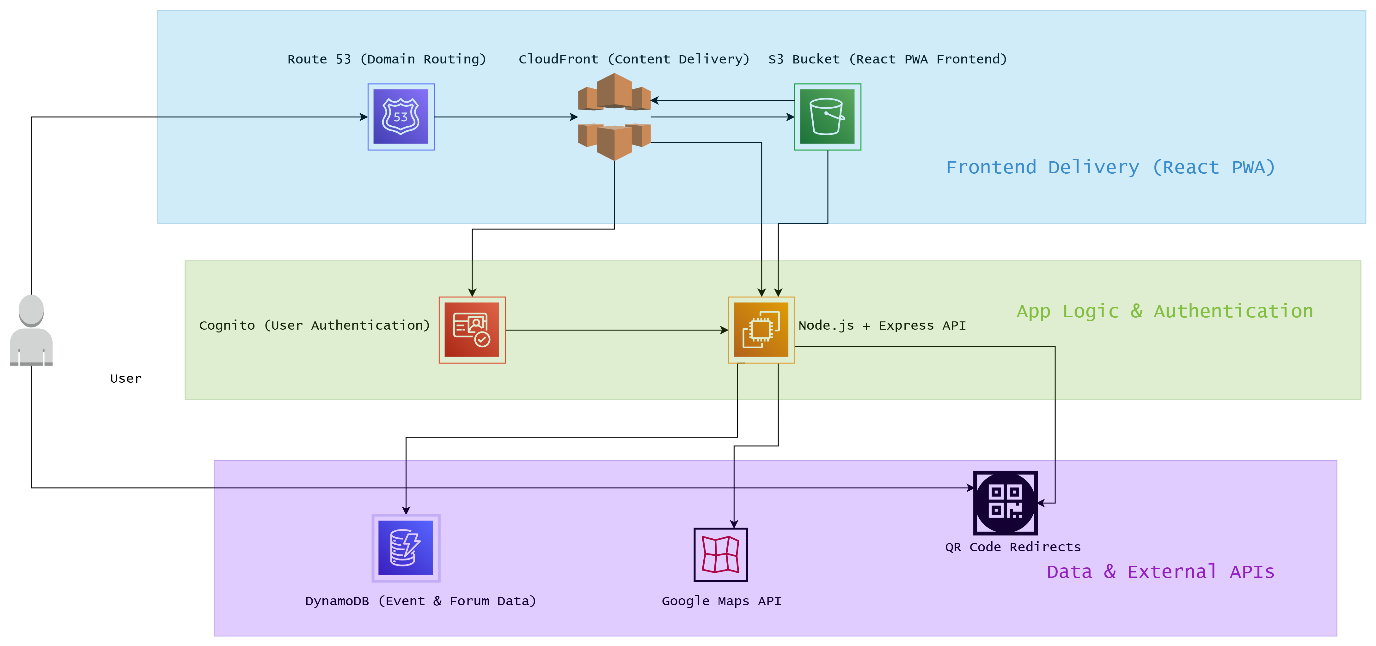


Figure 1. Proposed system diagram

Eventivity is a progressive web application designed to serve as a digital companion for any-scale local events. It provides a centralised platform where attendees can view schedules, access event-specific information, engage in forum discussions, and navigate to venues using integrated maps. The application prioritises user experience by offering a clean, intuitive interface that remains consistent across events while allowing each one to present a unique visual identity.

The app supports both unauthenticated and authenticated users. Unauthenticated users can explore public information such as the event overview, schedules and workshop details, while authenticated users gain access to interactive features like the forum, where they can post stories or media about their experience. All forum content is stored in DynamoDB, and media uploads are handled through Amazon S3. Authentication and session management are powered by AWS Cognito, ensuring secure and scalable user access.

To support offline usage, service workers cache key content so that attendees can still browse the app without a stable internet connection. This is especially important for international visitors or events hosted in areas with poor mobile coverage.

While the original system design proposed a traditional Node.js and Express backend, an approach I had used in earlier web development projects, I chose to deviate from that model in favour of a more modern, scalable solution using AWS Lambda. This serverless approach allowed me to offload infrastructure management while still writing custom backend logic for tasks such as media deletion and post handling. AWS Lambda easily integrates with amazon’s API gateway service. It also gave me the freedom to focus more on frontend development, theming, and user experience, which were key goals of the project.

As part of the user experience design, I also explored the idea of using QR codes at workshop locations. Scanning a QR code would take users directly to the corresponding workshop detail page within the application. This would offer an interactive way for attendees to get more specialised information without needing to manually search, improving both engagement and accessibility during events.

## Technology Stack

Eventivity was developed using a modern web technology stack that combines a lightweight, responsive frontend with a cloud-native backend. Each component was selected to maximise flexibility, scalability, and ease of maintenance. Where possible, I used tools I was already familiar with from past coursework, while also adopting new technologies to learn more about modern cloud-native development practices.

## Frontend

### Reat

React is a JavaScript library for building interactive user interfaces. I used it because of its modular component structure, which made it easier to manage reusable parts of the app like cards, navigation bars, and modals. Its hook system, including ‘useState’ and ‘useEffect’, allowed me to manage local state and asynchronous data fetching cleanly within each component.

### Vite

Vite is a modern tool that helps run and build frontend web applications quickly. I chose it because it starts the app almost instantly during development and updates changes in real time without needing to reload the whole page. It also supports the latest JavaScript features by default. One of the main reasons I used Vite was because it has a plugin that makes it easy to turn the app into a Progressive Web App, allowing users to install it and use it offline.

### Tailwind CSS

Tailwind CSS is a CSS framework that lets developers’ style directly in the markup using pre-defined classes. I used Tailwind to speed up development and keep styles consistent across the app. It also allowed me to prototype designs quickly without writing large CSS files from scratch.

### React Router

React Router is a library for handling client-side routing in React applications. I used it to manage navigation between pages like the landing page, forum, and workshop detail views. It allowed me to maintain a smooth, single-page experience for users without full page reloads.

## Backend

### AWS Lambda

AWS Lambda is a serverless compute service that runs code in response to HTTP requests or events. I chose Lambda instead of setting up a traditional Node.js backend, which I had used in previous projects. This shift to serverless helped reduce infrastructure complexity and let me focus more on frontend development and integration.

### AWS API Gateway

API Gateway is a service that enables developers to create and manage secure APIs. I used it to expose HTTP endpoints that trigger Lambda functions. It simplified the process of connecting the frontend to backend logic without needing to manage my own web server.

### AWS Cognito

Cognito is a service from Amazon that helps manage user sign-up, login, and session tracking. I used it to handle authentication securely without having to build everything from scratch. Cognito provides predefined login and logout paths, which meant I could connect these to my own custom pages. This allowed me to design the user experience the way I wanted, while still using a secure and reliable system in the background.

### AWS S3

Amazon S3 is a scalable object storage service. I used it to store media files such as images and videos uploaded through the forum. Its integration with Lambda made it easy to trigger media uploads and deletions, and it provided reliable storage with minimal configuration.

### AWS DynamoDB

DynamoDB is a fully managed NoSQL database service that offers fast performance and high scalability. I used it to store dynamic content such as forum posts and workshop data. Its flexible schema made it a good fit for handling user-generated content with varying formats.

### AWS IAM

AWS Identity and Access Management (IAM) is used to control access to AWS services. I created custom IAM roles and policies to ensure that Lambda functions only had permission to access the resources they needed, such as reading from DynamoDB or writing to S3. This helped keep the project secure and aligned with security best practices, not using root user credentials.

### Google Maps API

The Google Maps API allows developers to embed interactive maps in web applications. I used it on the Eventivity landing page to show events as clickable map markers. This gave users a more visual way to explore events and served as a gateway to the event-specific experience.

## Hosting

### AWS Amplify

AWS Amplify is a cloud platform for deploying full-stack web apps. I used Amplify to host the Eventivity frontend and manage continuous deployment from GitHub. It automated the build and deployment process.

## Development Tools

### Vite Plugin PWA

The vite-plugin-pwa plugin allows Vite projects to register service workers and define caching strategies. I used it to implement offline functionality, allowing users to access key pages and resources even when disconnected from the internet. The Eventivity application can be installed from a browser.

### GitHub

GitHub is a platform for version control and collaborative development. I used it to manage the project’s source code, track changes, and log bugs through issues. It was also connected directly to AWS Amplify for continuous deployment. During this process, I used GitHub Guardian to help protect my repository by scanning for exposed AWS credentials and secrets. This helped me avoid accidentally committing sensitive information and added an extra layer of security to the development workflow.

### Postman

Postman is an API testing tool used to send requests and view responses from backend endpoints. I used it during Lambda development to test API Gateway routes, debug logic, and verify that permissions were correctly configured before integrating with the frontend.

### Visual Studio Code

VS Code is my code editor of choice because it has support for many programming languages, an extensive library of helpful plugins, intelligent syntax highlighting and a familiar interface.

## Development Methodology

Although I was aware of various formal software development methodologies, I took a flexible and iterative approach to this project that aligned most closely with Agile principles. I focused on building the project in manageable stages, prioritising working functionality and adapting as I progressed. My main goal was to gradually piece together a fully functional system, one page and component at a time, while ensuring each part worked well before moving on to the next.

The first step in my process was to get the basic page navigation working across the application. Once routing was in place, I moved through each page individually, breaking it down into components and working on each until the entire page was complete or I encountered a major issue. This incremental style allowed me to focus deeply on each section and understand how components would need to interact within the wider system.

To keep my development organised, I used a branching strategy with GitHub. For each new feature or significant change, I created a separate branch and worked independently until that part was finished. Once complete and tested, I merged the changes back into the main branch. This helped prevent conflicts and kept the main build stable throughout development.

GitHub Issues were another helpful tool in my workflow. I used them to log bugs, unexpected behaviours, or design problems that did not immediately break the app but still needed to be addressed. Having these issues tracked in a dedicated space allowed me to return and fix them at appropriate points in the development cycle, especially when they overlapped or conflicted with other features.

My deployment workflow was supported by a continuous integration and deployment (CI/CD) setup using GitHub and AWS Amplify. Each time I pushed changes to the main branch, Amplify automatically rebuilt and deployed the latest version of the app. This made it easy to test updates in a live environment and reduced the time spent on manual deployment.

Developing the application feature by feature also allowed the app to grow at a manageable rate. This kept the project from becoming overwhelming and helped ensure that each part of the system was well integrated and tested before moving forward.

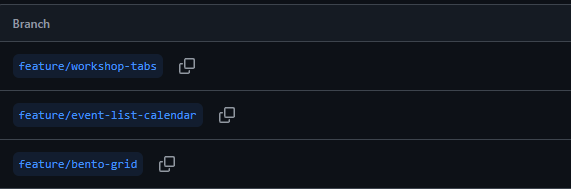


Figure 2. GitHub branches

## App Components

This section outlines the key features of Eventivity by breaking them down into functional components. Each component was developed individually and designed to work as part of a larger, reusable system.

### Event Map Component

The event map is the first interactive element users see when visiting the landing page. It uses the Google Maps API to display a zoomable map with custom markers representing each available event. I chose this approach because it provides a visually intuitive way for users to explore what’s happening in different locations.

Each marker is linked to a specific event. When a user clicks on one, they are brought to a themed version of the app that contains details, workshops, and forums related to that event. This direct connection between the map and the rest of the app helps keep navigation simple and consistent.

I had not used the Google Maps API before this project, so integrating it required a bit of research and trial-and-error. I learned how to place markers dynamically and control what happens when they’re clicked, by following guides in the Google Maps documentation. This interactive element with multiple options helped reinforce the idea that Eventivity is a multi-event platform.

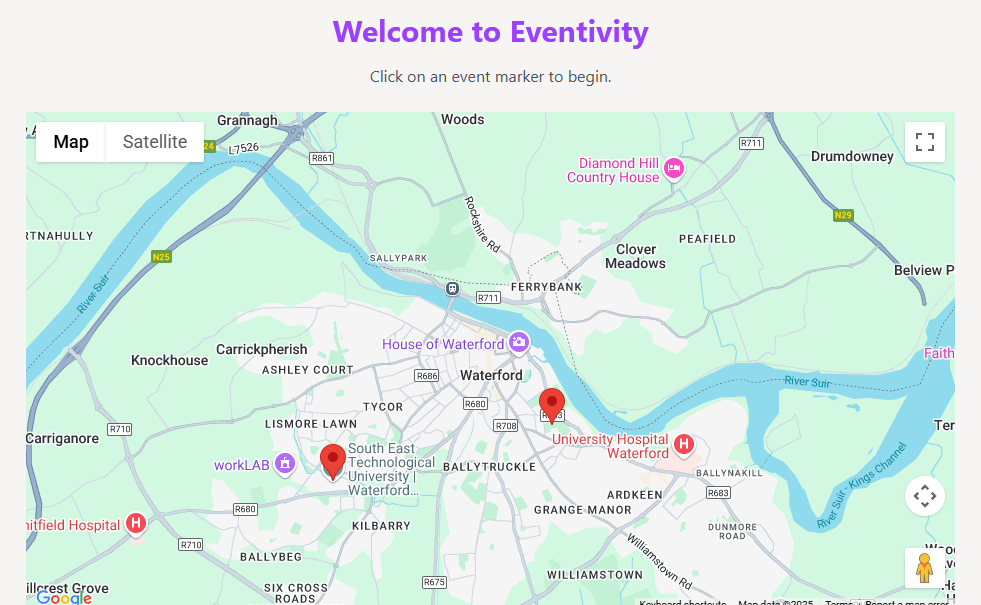


Figure 3. Google Maps component

### Forum Post Component

The forum post component allows a user to create a post that will then be shown in the post list component. Each post includes a title, message, and optional media like an image or video, and an event tag (e.g. Public, Worshop1...). It uses a basic forum layout for readability. When a post is submitted, the text is sent to a DynamoDB table and any optional media is stored in a S3 bucket.

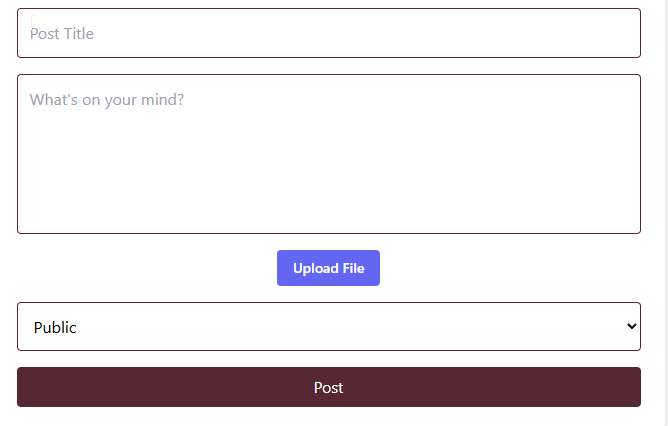


Figure 4. Post Component

### Post list Component

The post list component displays all the posts for a selected event. It loops through the list of posts and shows each one using the forum post component. Media is loaded conditionally, only shows if present, this allows the user to quickly read and scroll through posts. Comments on a post can be viewed by clicking the view comments link or the post and all attached comments can be deleted by clicking the trash can icon.



Figure 5. Post Card Component

### Comment Card and Comment List Component

When “View Comments” is clicked a pop up consisting of the text box, post comment button and exit button appears. The simple text box allows a user to post a comment on a post, that is then displayed above the comment submission text box. Comments are hidden on the post list. The user also has the option to delete a single comment. In the future I need to add the logic that only the user of the post can delete their post.

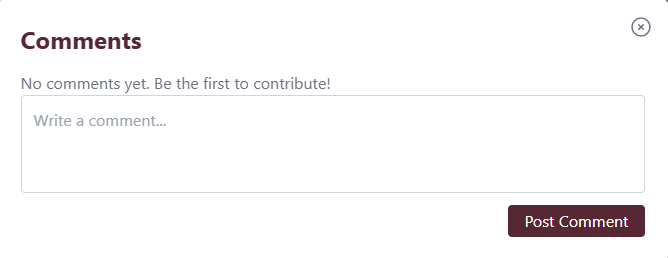


Figure 6. Comment List Component

### Event List Component

The event list component appears below the map on the landing page. It shows a short list of all events available in the app. Each event includes a title, date, and a short description. Clicking on a list item brings the user to that event’s themed version of the app. This feature is useful for users who prefer a list view over the map.

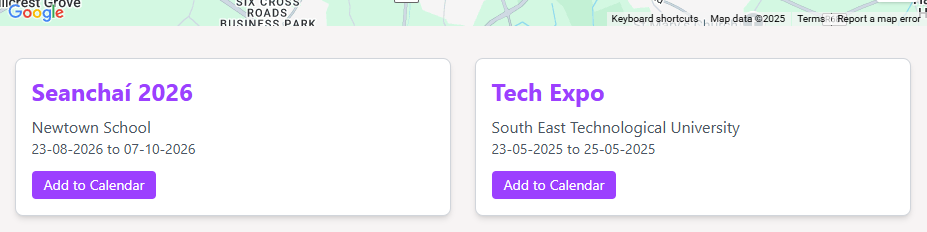
A user can also add the event to the MyCalendar page but clicking “Add to Calendar” button.

Figure 7. Event List Component

### Workshop Card and Modal Component

The workshop card is used on the Workshops page to show basic details about each workshop. It includes the workshop title, a short description. The cards are clickable and have a hover motion effect. When a card is clicked a modal appears in the middle of the screen. The workshop card has a title, summary description and an image placeholder.

The modal displays the workshop title and an event description. Ther is a interactive button to send the user to the specific workshop page to learn more.

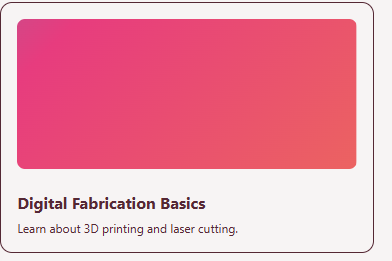


Figure 8. Workshop Card

A screenshot of a web page

AI-generated content may be incorrect.

Figure 9. Workshop modal

## Navigation Components

Navigation is a key part of any web app, and I wanted Eventivity to feel easy to move through without confusing the user. I created two main types of navigation: one for the landing page and one for the event pages. I also added small, reusable components like a user dropdown and map icon to keep the interface consistent and clean.

Avatar Dropdown Menu

This is a simple dropdown that appears when a logged-in user clicks their profile picture. It includes links to:

* Eventivity home => Eventivity landing page
* View Profile => User profile page
* My Calendar => User calendar page
* Logout => signs out user.

I used it to keep the navigation area tidy while still giving access to important user options.

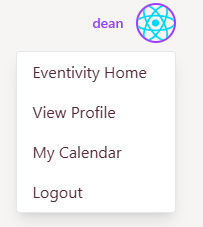


Figure 10. Avatar dropdown menu

### Event Navigation Bar

This navigation bar appears when a user selects an event. It shows links to pages like Home, Schedule, Forum and Workshops. The look of the navbar changes depending on the event theme, using different colours and styles to match. I separated this from the landing navigation to give event pages their own identity. It also helped reinforce the idea that users had entered a specific event space.



Figure 11. Event navbar

### Landing Page Navigation Bar

This version is simpler and more neutral in design since it’s not tied to any event. It helped create a clear entry point for new users and gave a good first impression. The landing page navbar has conditional rendering. If a user is not signed in it shows links to sign in and sign up, a signed-in user sees the avatar menu icon.



Figure 12. Landing navbar - user signed-in



Figure 13. Landing navbar - user not signed-in

### Map Icon Button

This icon appears on the left-hand side of the screen. It lets users return to the landing page map in a single click. It’s a small detail, but it added a nice usability improvement and helped make navigation feel smoother. I added it to every page to always allow the user to go back to the map if not signed-in.



Figure 14. Map Icon

## Pages

Each page in Eventivity is made up of reusable components that work together to deliver a specific purpose. This section breaks down the main pages of the app, their purpose and the components that make them work.

### Landing Page

The landing page is the first page users see. It includes the event map at the top and the event list below it. This page is designed to help users explore upcoming events quickly.

Components used:

* Event Map
* Event List
* Landing Navbar

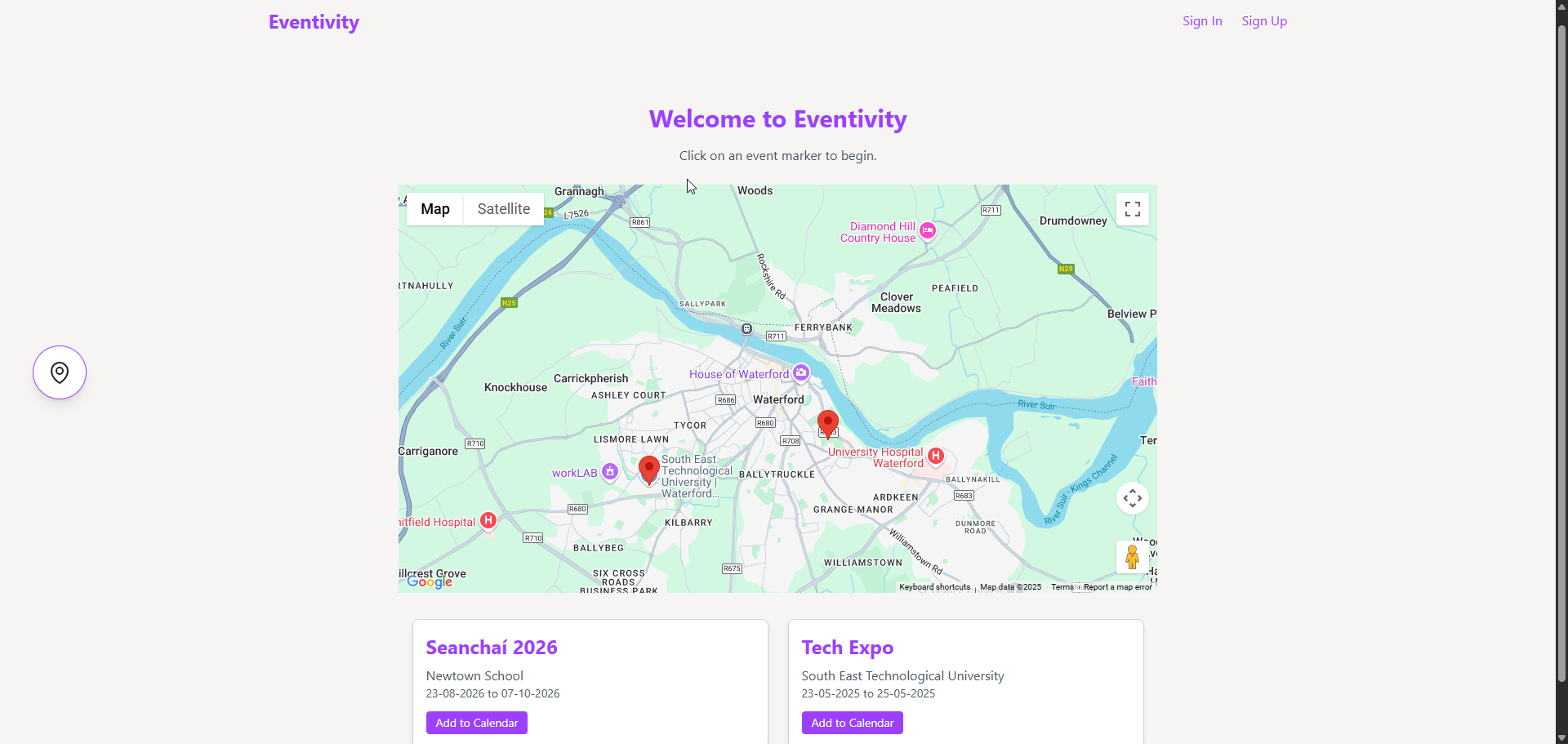


Figure 15. Landing page

### Event Home Page

After selecting an event, the user is taken to the event-specific home page. The theme and static content change depending on the selected event, giving it a unique identity. The purpose of this page is to inform the user about the event. A plan for future improvements to this page is to add more event specific content, saved in a DynamoDB table.

Components used:

* Theming Wrapper
* Event Navbar
* Static content

### Schedule Page

This page lists all the times for the workshops of an event. Functionality for this has not been implemented yet.

Components used:

* Theming Wrapper
* Event Navbar
* Static content

### Forum Page

This page is a protected route. Only users who are signed in can navigate to this page, otherwise they are sent to the Sign In page. This page displays all forum posts related to the event. Users can browse posts, view media, and comment.

Components used:

* Forum Post List
* Forum Pose Card
* Comment List
* Comment Card
* Event navbar

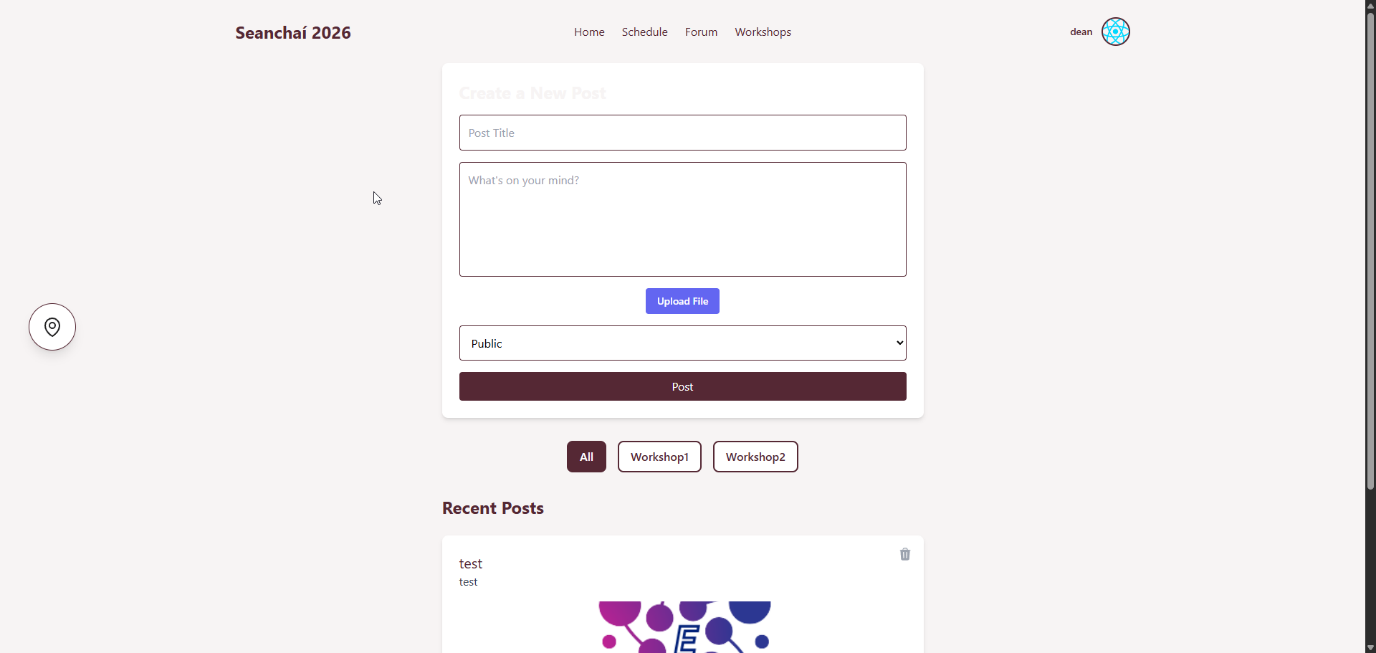


Figure 16. Forum page

### Workshops Page

The workshop page displays animated and interactive workshop cards in a modern bento grid format. The cards allow further navigation to workshop specific detail pages.

Components used:

* Workshop card
* Workshop modal
* Bento grid

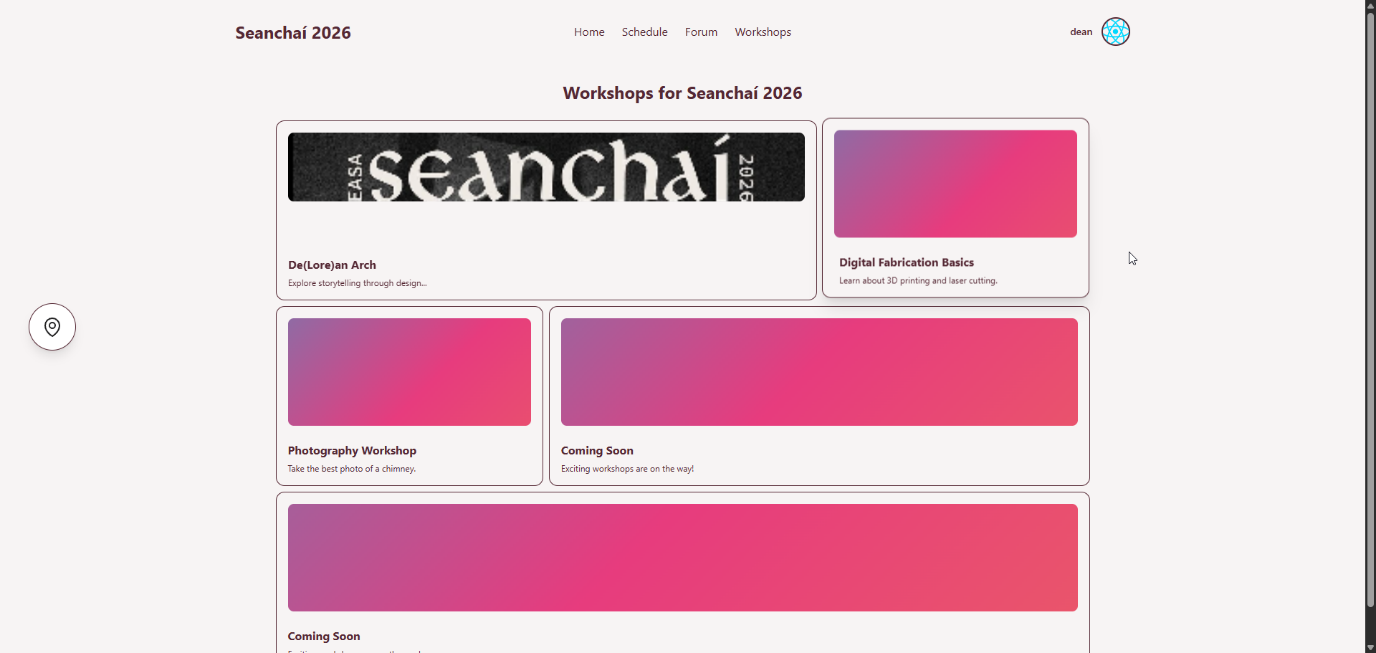


Figure 17. Workshops page

### My Calendar Page

This is a personal planning page. It shows a list of saved or selected events or workshops. Saved events can only be added once and can be deleted.

It is not yet connected to a real calendar but acts as a placeholder for future integration. I also plan to be able to add workshops to this list.

Components used:

* Calendar list view
* Event cards
* Landing navbar

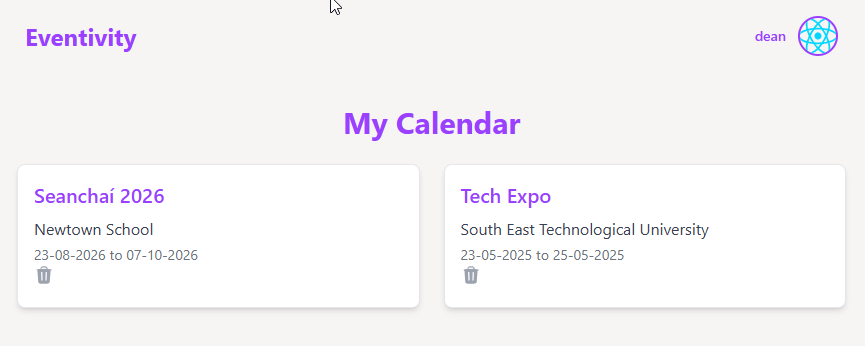


Figure 18. Calendar page

### Profile Page

This page displays basic information about the user. It shows username, avatar icon and an empty input text box. Fure development of this page involves user uploaded avatar photo and the functionality to write and update a brief description about themselves.

Components used:

* Landing navbar
* Dynamic username

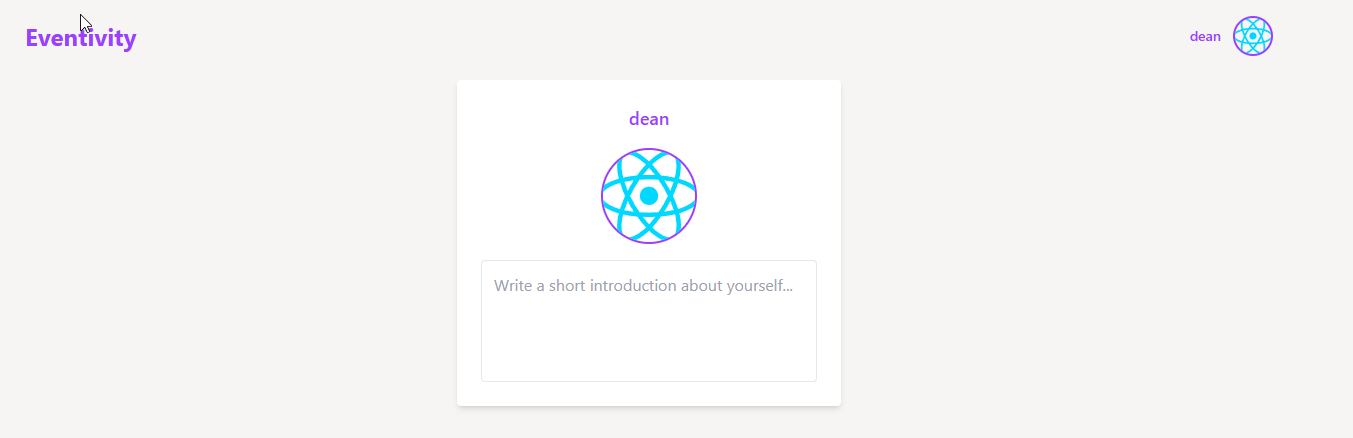


Figure 19. Profile page

## Dynamic Theming

A big part of this project’s concept was the ability to reuse the same app structure across multiple events, while allowing each event to present its own visual identity. To support this, I created a custom React hook called useTheme.

This hook pulls theme data from the currently selected event, which is stored in a shared context. When a user selects an event from the landing page, that event is saved globally in context. From that point on, any component that uses the useTheme hook can access colours like background, accent, and text. This makes it easy to dynamically restyle components without hardcoding values or creating separate versions of the app for each event.

If no theme is selected (such as when the user is still on the landing page), the app falls back to default event. This keeps the experience consistent and prevents errors from missing data.

Theming in this way helped me better understand how to separate logic from presentation. It also made the app feel more flexible and professional, as each event can now load its own colours and apply them automatically to the layout, buttons, and other styled components.

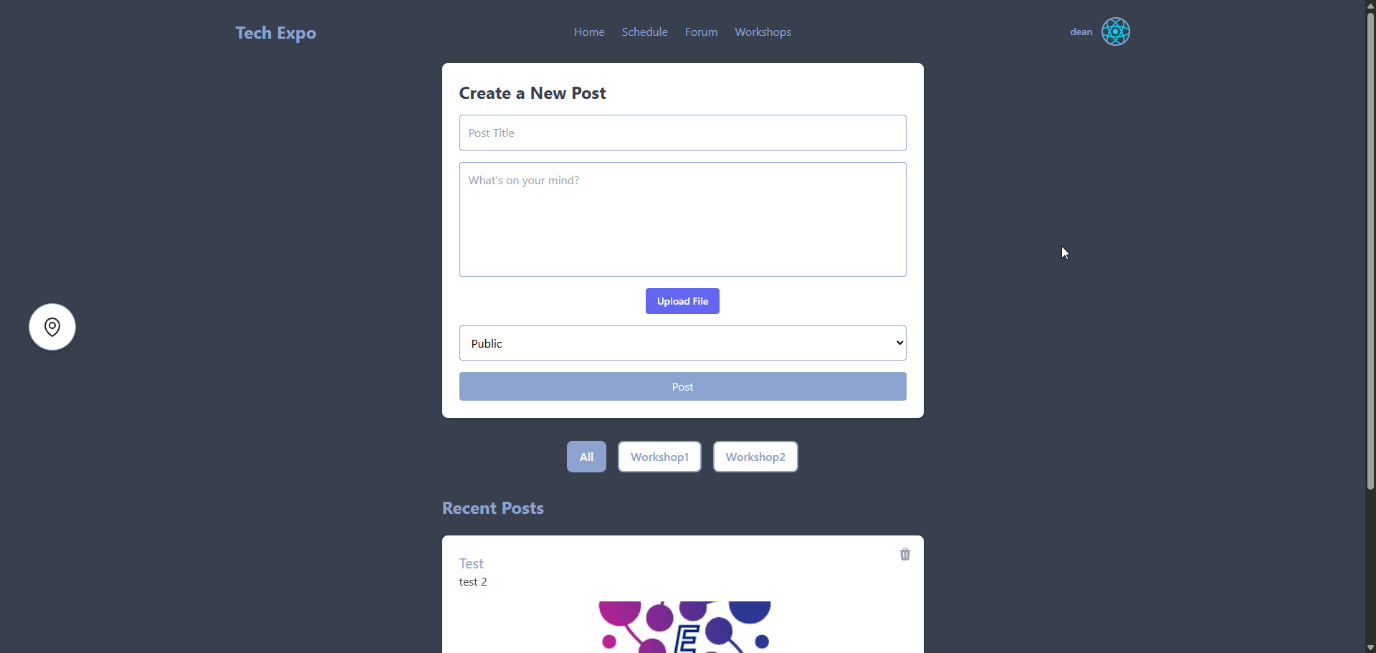


Figure 20. Post Forum - Event A

A screenshot of a computer

AI-generated content may be incorrect.

Figure 21. Post Forum - Event B

## Future Improvements

While the core functionality of Eventivity is complete and working, there are several features and enhancements I plan to develop in the future, some I have mentioned in previous sections. These additions would improve the user experience, make the system more dynamic, and expand its use for real-world events.

* Dynamic Event Loading from Database

Currently, events are manually defined in the frontend. In future versions, I would like to load event data directly from DynamoDB. This would make it easier to manage new events and scale the app without redeploying.

* User Chat or Messaging System

To increase interaction between users, I would like to add a private messaging or chat feature. This could be used for workshop coordination, group discussions, or general social engagement during events.

* Push Notifications

Adding support for push notifications would allow users to receive real-time updates about event changes, new posts in the forum, or reminders for saved workshops.

* Improved Mobile Layout

Although the app is responsive, some UI elements could be adjusted further for better mobile usability. This includes optimising spacing, text size, and touch targets for smaller screens.

## Conclusion

Developing Eventivity was a valuable learning experience that allowed me to bring together many of the skills I’ve developed throughout my degree. It challenged me to think not only about how to build an application, but how to design something that is usable, adaptable, and ready for real-world use. Although I had built web apps before as part of coursework, this was my first time planning, designing, and implementing something at this scale and with long-term goals in mind.

Throughout the project, I encountered a wide range of decisions — both technical and practical. Choosing the right tools, figuring out how different components should work together, and planning the flow of the app taught me how important structure and reusability are in modern development. I learned to balance custom development with managed cloud services, and to make decisions based on scalability, maintenance, and user experience.

One of the most important lessons I took from this project was how helpful it is to work incrementally. Building feature by feature allowed me to manage the complexity of the app without becoming overwhelmed, and it helped me keep the project focused. Along the way, I gained more confidence in using cloud services, version control tools, and frontend frameworks like React and Tailwind.

More than anything, this project gave me the chance to apply what I’ve learned in a practical setting, solving real problems and creating something I can continue to improve. I now have a much clearer understanding of how a full-stack web application comes together, from authentication to deployment — and I’m looking forward to building on that knowledge in future work.