Final Project Report

Part A: Analytical Report

Project Title Student event Planner

Presented by Yes we can

66070501004	Kanyavan Tayaniphan
66070501005	Kanlaya Sutthapanya
66070501034	Pemika Chongkhwanyuen
66070501050	Veerisara Panviripakorn
66070501070	Jantarat Sanglao
66070501080	Monsicha Konteing
66070501082	Supisara Wongphatthanakit

CPE334 Software Engineering

Semester 1, Academic Year 2025

Department of Computer Engineering Faculty of Engineering

King Mongkut's University Of Technology Tho

Table of Contents

List of Figures

List of Tables

A0. Executive Summary

This project presents the design and development of Student Event Planner, a web-based application aimed at improving how university events are created, discovered, and managed. In many academic environments, event coordination is fragmented across multiple platforms such as social media posts, spreadsheets, or online forms, leading to inefficiencies, poor visibility, and low student participation.

The proposed system addresses these challenges by providing a centralized platform that supports event creation, discovery, registration, QR-based attendance tracking, and notifications. The system is designed for three primary user roles: Students, Organizers, and Administrators, each with clearly defined responsibilities and access privileges.

The project followed an Agile development methodology over five sprints. Core features such as authentication, event management, real-time seat availability, and QR code check-in were successfully implemented as part of the Minimum Viable Product (MVP). While some advanced analytics features were deferred to future work, the final system meets the primary functional goals defined during the planning phase.

Overall, the project achieved its MVP objectives and demonstrates the effective application of software engineering principles, including requirements analysis, iterative design, system architecture modeling, testing, and team-based development.

A1. Introduction & Overview

A1-1. Introduction

University events play an important role in enhancing student engagement, collaboration, and campus life. However, managing these events efficiently remains a challenge due to the lack of an integrated system that supports the entire event lifecycle—from creation and promotion to registration and attendance tracking.

Many institutions rely on a combination of manual processes and disconnected tools such as social media announcements, Google Forms, or messaging platforms. These approaches often result in inconsistent information, limited tracking capabilities, and additional workload for event organizers.

To address these issues, this project proposes Student Event Planner, a unified web application designed to streamline event management while improving accessibility and user experience for students and staff.

A1-2. Problem Statement & Objectives

Problem Statement

The existing event management processes in universities suffer from:

1. Fragmented event information across multiple platforms
2. Manual registration and attendance tracking
3. Limited visibility of upcoming events for students
4. High administrative overhead for organizers and administrators

These limitations reduce participation and make it difficult to evaluate event effectiveness.

Project Objectives

The primary objectives of this project are

1. To design and implement a centralized web-based event management system
2. To support role-based access for Students, Organizers, and Administrators

3. To provide essential features such as event creation, search, registration, and attendance tracking
4. To apply Agile software engineering practices throughout the development lifecycle
5. To deliver a functional MVP within a five-sprint development timeline

A1-3 Functional & Non-Functional Requirements

During development, some features were fully implemented, while others were intentionally. deferred or dropped. These decisions were made to manage project scope, development time, and technical complexity, ensuring that core functionalities were completed and delivered with acceptable quality within the project timeline.

Table 1: Feature Roadmap Status (MoSCoW Priority vs. Actual Implementation)

Feature / Requirement	Type	MoSCoW	Actual Status	Remarks
User Authentication	Functional	Must-have	Done	Firebase Authentication
User Profile Management	Functional	Must-have	Done	Users can edit profile
Event Creation & Management	Functional	Must-have	Done	Create, edit, delete events implemented
Event Search & View Details	Functional	Must-have	Done	detail pages available
Event Registration/ Cancellation	Functional	Must-have	Done	Registration and cancellation supported
Event Filtering	Functional	Should-have	Done	Filter event category.
Seat Availability	Functional	Should-have	Done	For Student, Organizer
QR Code Check-in	Functional	Could-have	Done	QR Code available
Notification & Event Reminder	Functional	Could-have	Done	In-web notifications
Feedback & Rating System	Functional	Could-have	Deferred	future improvement

Upcoming Events Display	Functional	Could-have	Done	Displayed on homepage
Admin Dashboard	Functional	Could-have	Done	Simple statistic
Admin Analytic report	Functional	Won't-have	Dropped	Out of MVP scope
Security (Authentication & Access Control)	Non-Functional	Should-have	Done	Auth and role-based access applied

A2. Project Management Analysis

This section analyzes the execution of the "Yes We Can" project by comparing the initial management plans against the actual development timeline and effort.

A2-1. Project Timeline (Plan vs Actual)

The project was originally scheduled to be completed over 5 Sprints (10 weeks), following the roadmap defined in the Master Project Plan (Ref: PM-01). While the Core MVP features were delivered, significant deviations occurred in Sprints 3 and 4 due to technical challenges and scope adjustments.

Timeline Comparison Analysis

1. **Sprint 1 & 2 (On Track)** The team successfully delivered the UI Prototypes, User Account Management, and basic Database setup as planned. The foundational authentication (Email/Google) was implemented on schedule.
2. **Sprint 3 (Delayed)** The development of "Event Discovery" and "Seat Availability" overlapped into Sprint 4. The delay was primarily caused by the "Cookie/Session Management" issue (Ref: PM-03), where the system failed to persist in user states, requiring a significant architectural refactor of the login flow.
3. **Sprint 4 (Deviated & Scope Change)**
   a. *Planned:* Focus on Organizer Tools and QR Check-in.
   b. *Actual:* While QR features were implemented (see Appendix A: Git Log, Commit 561df70), the team also introduced an Admin Dashboard and Admin Login (see Appendix A: Git Log, Commit 305017c on Nov 23), which was originally categorized as a "Won't-Have" feature in the Project Plan (Ref: PM-01, MoSCoW Table). This unplanned addition, along with the UI language consistency fixes, compressed the time available for testing.

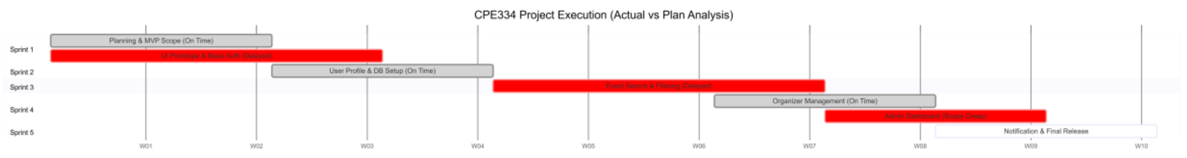


Figure 1 A2-1: Plan vs Actual Gantt Chart

The project timeline was originally structured as a 10-week engagement divided into five sprints. The Gantt chart (Figure1.A2-1) illustrates the actual execution of these sprints, highlighting critical paths, technical delays, and scope adjustments.

A2-2. Effort Estimation vs Actual

This section compares the estimated effort (in days/story points) from the Patch Planning (Ref: PM-02) against the actual time spent as recorded in the version control history.

Analysis of Key Deviations

1. **Authentication System** The initial estimation for the Login System was 9 days (Ref: PM-02). However, the Git logs indicate that work on authentication (Login, Register, Password Reset) spanned from Oct 14 to Nov 23. The "Password Reset" feature, initially facing email delivery configuration issues, required additional troubleshooting but was successfully completed and integrated before the final release.
2. **Admin Features:** No effort was allocated for Admin features in the initial plan. However, the implementation of AdminLogin and AdminDashboard components consumed approximately 3-4 days of unscheduled development time in late November (see Appendix A: Git Log, Commit 305017c).

Table 2: Effort Estimation Summary

Feature / Module	Estimated Effort (Ref: PM-01/PM-02)	Actual Effort (Approx)	Variance	Root Cause

Local User Login	5 Story Points / 9 Days	~15 Days	+6 Days	Critical bugs regarding Cookies and Session Timeout integration (Ref: PM-03).
Session Timeout	9 Days	5 Days	-4 Days	Implemented efficiently using standard JWT practices; less complex than anticipated.
Event Management	5 Story Points	7 Days	+2 Days	Complexity in handling image uploads and synchronization with the Homepage.
Admin Dashboard	0 Days (Out of Scope)	4 Days	+4 Days	**Scope Creep:** Added to support system management, impacting final testing time.
Password Reset	Included in Auth	2 Days	+3 Days	Required additional configuration for Firebase email triggers, now fully functional.

A3. Design & Architecture Analysis

A3-1. System Architecture Overview

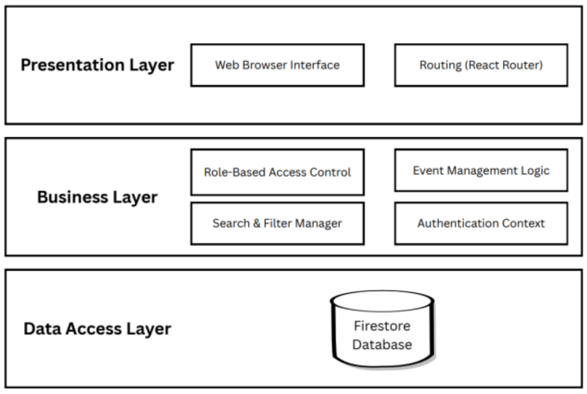


Figure 2 A3-1: Layered Architecture

Technology Stack & Architectural Pattern

Architecture Pattern

The project adopts a Layered Architecture, in which the system is structured into three main layers

1. Presentation Layer (user interface)
2. Business Layer (business logic)
3. Data Access Layer (data management)

This architectural approach enhances system organization, improves maintainability, and facilitates future development and scalability.

1. Frontend: The user interface is developed using React (Vite) as the main framework, together with Bootstrap and CSS for designing and arranging user interface components.
2. Backend: The backend leverages Firebase as a Backend-as-a-Service (BaaS) solution. Firebase Authentication is used to manage user authentication, supporting both email/password login and Google OAuth.
3. Database: Data is stored in Firebase Firestore, a highly flexible NoSQL database that supports real-time data handling and scalability.

DevOps & Deployment: GitHub is utilized for version control, and the system is

A3-2. Design Change Log & Rationale

A3-2.1 System Design Changes

Software Architecture Pattern

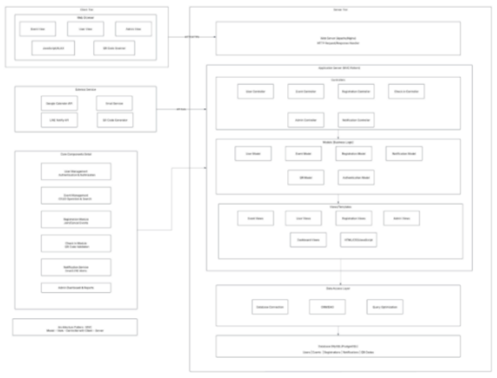


Figure 3 A3-2: Architecture Pattern: MVC

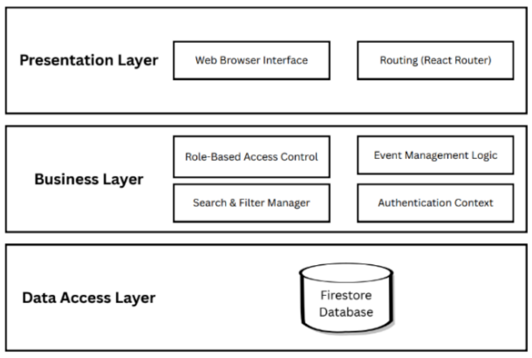


Figure 44 A3-1: Layered Architecture

Problem Statement

During the initial project planning phase (Lab 4), the development team adopted the Model–View–Controller (MVC) architectural pattern with the intention of clearly separating system responsibilities into presentation, control, and data management components. However, during the actual implementation using React for the frontend and Firebase as a Backend-as-a-Service (BaaS), several limitations were identified.

React follows a component-based architecture and primarily operates on the client side, while Firebase provides a ready-made backend service. As a result, there is no traditional server-side Controller component to handle application logic in accordance with the standard MVC paradigm. Consequently, applying the MVC architecture in this context was not fully compatible and led to increased complexity in managing application state and business logic.

Resolution

To better align the system architecture with the selected technology stack and actual development practices, the team decided to transition to a Layered Architecture, as deployed via Firebase Hosting for efficient and reliable application deployme presented in the final project presentation. The system is structured into three primary layers

1. Presentation Layer: Responsible for the web-based user interface and application routing, implemented using React components.
2. Business Layer: Responsible for handling business logic, including user authorization mechanisms such as Role-Based Access Control (RBAC) and internal state management (e.g., authentication context).
3. Data Access Layer: Responsible for data access and communication with cloud-based data storage services, specifically Firebase Firestore.

Outcome

The adoption of the Layered Architecture resulted in a clearer and more organized system structure by enforcing Separation of Concerns. Presentation logic, business logic, and data access operations are distinctly separated, making the codebase easier to understand, maintain, and modify (maintainability). Furthermore, this architectural approach enhances the system’s ability to support future feature expansion and growth (scalability).

Sequence Diagram: Event Registration Process

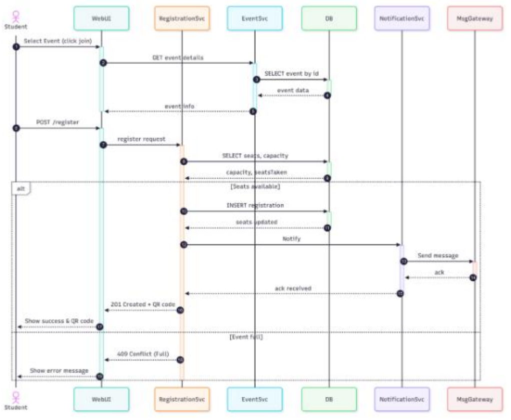


Figure 55 A3-2: Event Registration Flow

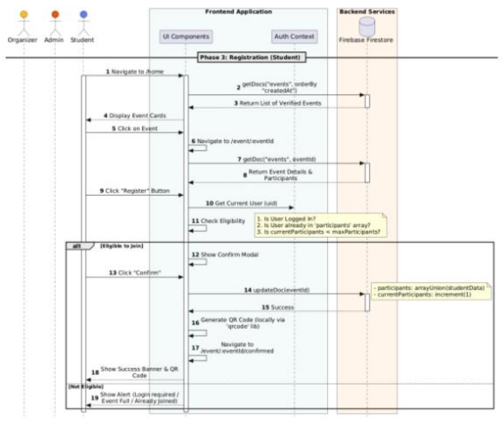

Organizer
Admin
Student
Frontend Application
UI Components
Auth Context
Backend Services
Firebase Firestore
Phase 3: Registration (Student)
1 Navigate to /home
2 getDocs("events", orderBy "createdAt")
3 Return List of Verified Events
4 Display Event Cards
5 Click on Event
6 Navigate to /event/:eventId
7 getDoc("events", eventId)
8 Return Event Details & Participants
9 Click "Register" Button
10 Get Current User (uid)
11 Check Eligibility
1. Is User Logged in?
2. Is User already in 'participants' array?
3. Is currentParticipants < maxParticipants?
alt
[Eligible to join]
12 Show Confirm Modal
13 Click "Confirm"
14 updateDoc(eventId)
- participants: arrayUnion(studentData)
- currentParticipants: increment(1)
15 Success
16 Generate QR Code (locally via 'qrcode' lib)
17 Navigate to /event/:eventId/confirmed
18 Show Success Banner & QR Code
[Not Eligible]
19 Show Alert (Login required / Event Full / Already Joined)

Figure 66 A3-2 Phase 3 Registration

Problem Statement

In the initial design phase (Lab 4), the team constructed the sequence diagram based on a microservices-oriented concept, where system responsibilities were explicitly separated into multiple services, such as Registration Service and Event Service. This design required multiple inter-service communications, resulting in a chain of service requests to complete the event registration process.

Upon further analysis, this approach was found to be overly complex for the scope and scale of the project. Moreover, it was not well aligned with the use of Firebase as a Backend-as-a-Service (BaaS), which is designed to support direct data access via client-side SDKs rather than multi-layered service orchestration.

Resolution

To better reflect the actual system architecture and implementation, the sequence diagram was revised in the final phase. The revised design removes unnecessary service layers and instead delegates control logic to the Frontend UI Components and Authentication Context.

The updated process allows the frontend to communicate directly with Firebase Firestore to perform both eligibility verification (e.g., authentication and availability checks) and data persistence (document updates) within a single streamlined interaction, rather than through multiple service calls.

Outcome

1. Reduced Complexity (Simplicity): Eliminating redundant inter-service communication simplifies the registration flow, making it shorter and easier to understand.
2. Improved Performance: System latency is reduced by minimizing the number of API calls and replacing chained service requests with a single direct database operation.

3. Enhanced Data Integrity: Validation checks such as user authentication status and event capacity are consolidated into the same step as data updates, ensuring consistency and reducing the risk of partial or invalid transactions.

A3-2.2 UI/UX Changes (Prototype vs Implementation)

This section presents a comparison between the UI/UX prototype designed in Lab 6 using Figma and the actual application implemented in Lab 13. The objective is to analyze how the interface evolved from an ideal design concept to a functional system, and to explain the reasons behind these differences.

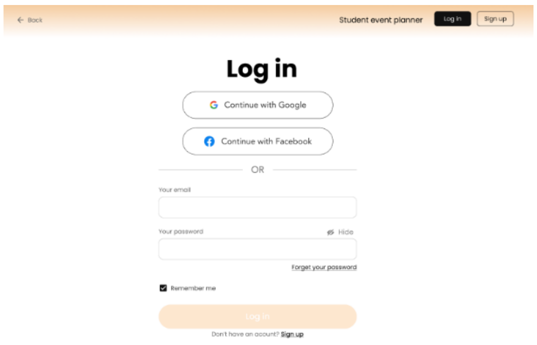


Figure 7 A3-2.2-1: Login Page - Prototype

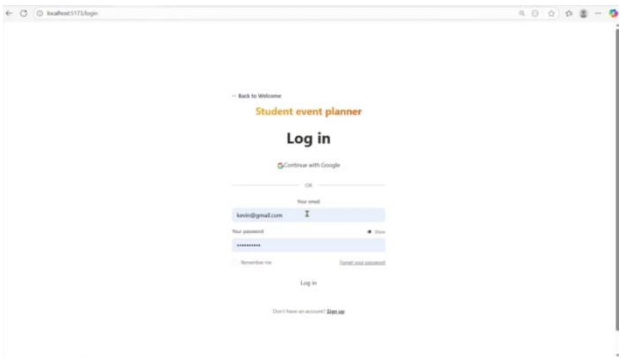


Figure 8 A3-2.2-2: Login Page – Application

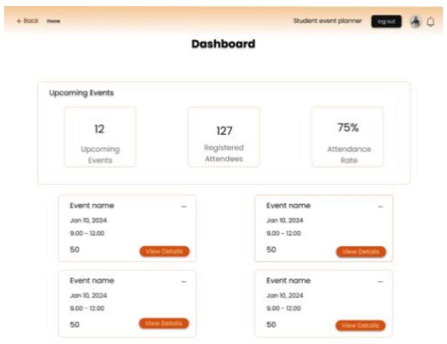


Figure 9 A3-2.2-3: Dashboard Page- Prototype

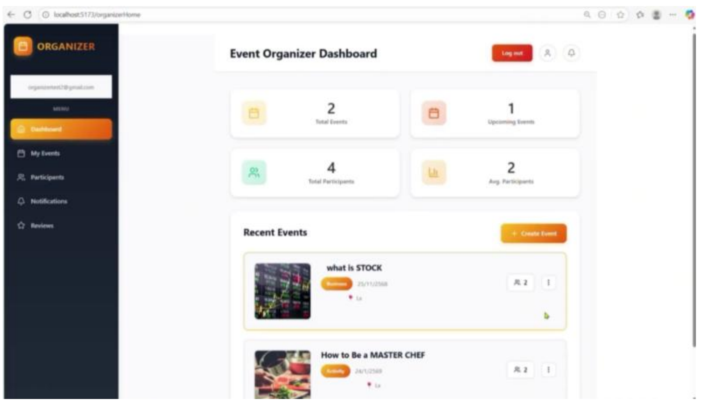


Figure 10 A3-2.2-4: Dashboard Page – Application

Table 3: *Comparison Between Prototype and Implementation*

UI Component	Figma Design (Lab 6 – Prototype)	Actual Application (Lab 13 – Implementation)
Overall Layout	Visually rich screens with detailed illustrations and promotional text	Cleaner and more minimal layout focusing on core functionality
Navigation Structure	Multiple conceptual navigation paths designed for demonstration	Simplified and consistent navigation aligned with system roles
Dashboard Design	Emphasis on visual presentation and sample statistics	Functional dashboard with real-time data and system-generated metrics
Button & Interaction Style	Custom-designed buttons and static interaction flow	Standardized components integrated with system logic and validation
User Feedback	Limited feedback, mainly visual placeholders	Real-time feedback such as notifications, status updates, and confirmations
Authentication UI	Conceptual login and role selection screens	Enhanced login UI with session timeout, login policy, and security constraints

Rationale for Differences Between Prototype and Implementation

The differences between the prototype and the actual application are intentional and reflect practical development considerations

1. **Technical Constraints**

   Some visual elements and interactions in the Figma prototype were not feasible to implement due to framework limitations, system performance concerns, or integration complexity with backend services such as Firebase Authentication and Firestore.

2. **Security and System Policies**

   In Lab 13, additional UI elements were introduced to support login policies, session timeout handling, and role-based access control. These requirements were not fully represented in the initial prototype.

3. **User Feedback and Usability Improvements**

   Feedback during development indicated that a simpler interface improved usability. As a result, the implemented UI focuses more on clarity, readability, and ease of navigation rather than visual richness.

4. **Performance and Responsiveness**

   The actual application prioritizes fast loading times and responsive behavior across devices. This led to the removal or simplification of certain animations and decorative elements present in the prototype.

5. **Development Timeline and Scope Adjustment**

   Due to sprint constraints, the team prioritized implementing core features such as event management, QR code check-in, notifications, and calendar integration, resulting in UI adjustments from the original design.

A4 Implementation & Quality

A4-0 Context & Scope

The project is a Student Event Planner web application. The implementation was built around a React 18 (Vite) frontend, React Router v6, and Firebase (Firestore + Firebase Authentication) as the backend/database layer. State was handled mainly with React Context API (e.g., UserAuthContext), and UI assets included Lucide React + custom CSS.

This section focuses on how quality was ensured during implementation: testing strategy (automated + manual), CI/CD maturity, and deployment approach—plus the hardest technical issues and how the team solved them.

A4-1 Implementation Summary (Testing, CI/CD, Deployment)

1. Testing Strategy & Results (Unit/E2E + Manual Testing)

1.1 Why we used Playwright

We selected **Playwright** because we needed **end-to-end confidence** over real user flows (not just isolated functions). E2E testing is especially useful when the system relies on:

1.1.1 Auth flows (register/login)

1.1.2 Routing/navigation (SPA behavior via React Router)

1.1.3 Real UI states and page transitions

1.1.4 Integration with Firebase Auth / Firestore behaviors

Playwright also supports running across multiple browsers (Chromium/Firefox/WebKit), giving a growth path to cross-browser coverage when time allows.

1.2 E2E test plan

From Lab 10, the E2E plan targeted the core “front door” features (the highest business value + highest risk area):

1.2.1 Welcome Page

1.2.2 Register

1.2.3 Register Failure (email already exists)

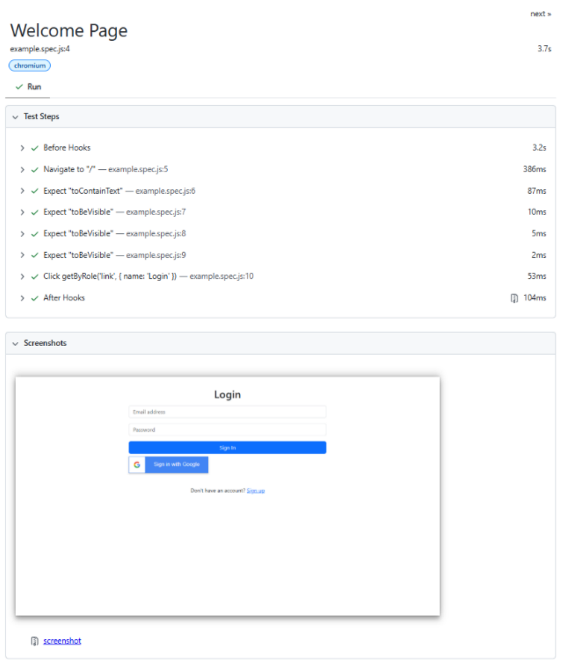
1.2.4 Login

1.2.5 Homepage after login (dashboard presence, search, event details, logout)

The tests were executed against a local Vite dev server (e.g., http://localhost:5173/) using realistic test data (test account credentials).

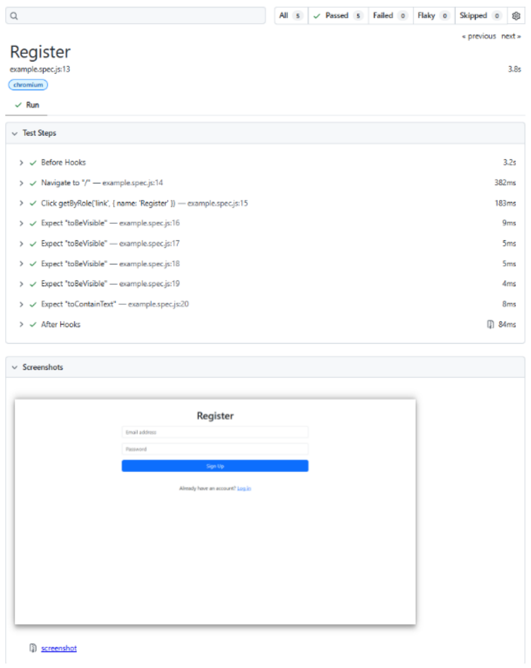
1.3 Execution evidence

Lab 10 includes evidence screenshots/output for each scenario (“Welcome Page / Register / Register Fail / Login / Homepage”), which demonstrates the tests were actually run and not just described on paper.



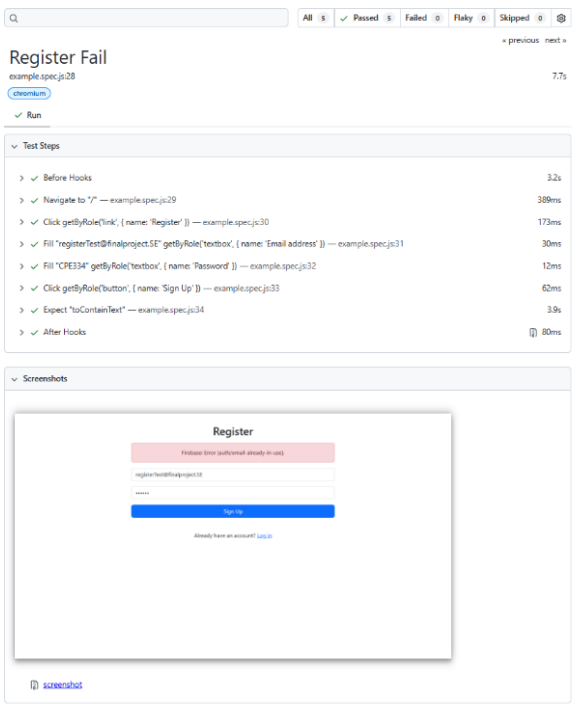
```
                                                              next »
Welcome Page
example.spec.js:4                                               3.7s
chromium
✓ Run

Test Steps
> ✓ Before Hooks                                                3.2s
> ✓ Navigate to "/" — example.spec.js:5                        386ms
> ✓ Expect "toContainText" — example.spec.js:6                  87ms
> ✓ Expect "toBeVisible" — example.spec.js:7                    10ms
> ✓ Expect "toBeVisible" — example.spec.js:8                     5ms
> ✓ Expect "toBeVisible" — example.spec.js:9                     2ms
> ✓ Click getByRole('link', { name: 'Login' }) — example.spec.js:10   53ms
> ✓ After Hooks                                                104ms

Screenshots
screenshot
```

Figure 11 A4-1: Welcome Page



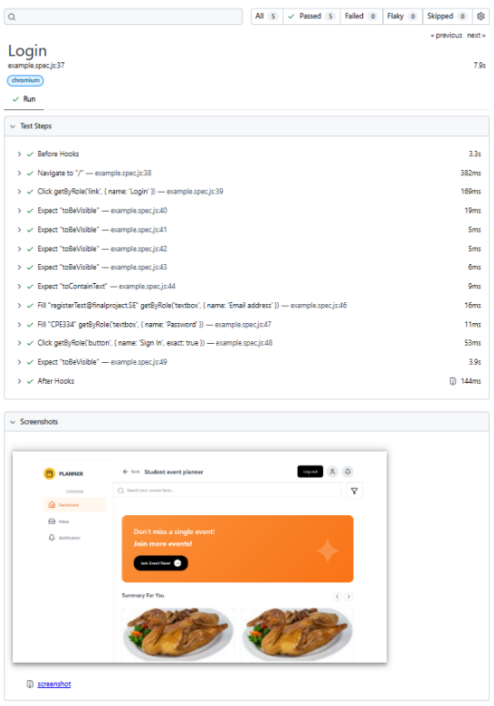
```
All 5 | ✓ Passed 5 | Failed 0 | Flaky 0 | Skipped 0
                                                    « previous  next »
Register
example.spec.js:13                                              3.8s
chromium
✓ Run

Test Steps
> ✓ Before Hooks                                                3.2s
> ✓ Navigate to "/" — example.spec.js:14                       382ms
> ✓ Click getByRole('link', { name: 'Register' }) — example.spec.js:15   183ms
> ✓ Expect "toBeVisible" — example.spec.js:16                    9ms
> ✓ Expect "toBeVisible" — example.spec.js:17                    5ms
> ✓ Expect "toBeVisible" — example.spec.js:18                    5ms
> ✓ Expect "toBeVisible" — example.spec.js:19                    4ms
> ✓ Expect "toContainText" — example.spec.js:20                  8ms
> ✓ After Hooks                                                 84ms

Screenshots
screenshot
```

Figure 12 A4-1: Register



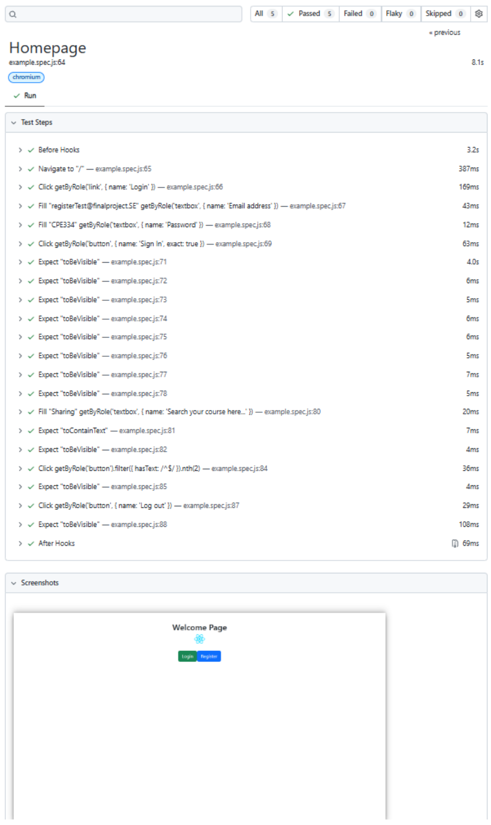
```
All 5   ✓ Passed 5   Failed 0   Flaky 0   Skipped 0
                                              « previous  next »
Register Fail
example.spec.js:28                                                7.7s
chromium
✓ Run

Test Steps
> ✓ Before Hooks                                                   3.2s
> ✓ Navigate to "/" — example.spec.js:29                          389ms
> ✓ Click getByRole('link', { name: 'Register' }) — example.spec.js:30     173ms
> ✓ Fill "registerTest@finalproject.SE" getByRole('textbox', { name: 'Email address' }) — example.spec.js:31     30ms
> ✓ Fill "CPE334" getByRole('textbox', { name: 'Password' }) — example.spec.js:32     12ms
> ✓ Click getByRole('button', { name: 'Sign Up' }) — example.spec.js:33     62ms
> ✓ Expect "toContainText" — example.spec.js:34                     3.9s
> ✓ After Hooks                                                    80ms

Screenshots
Register
Firebase: Error (auth/email-already-in-use).
screenshot
```

Figure 13 A4-1: Register Fail



```
All 5   ✓ Passed 5   Failed 0   Flaky 0   Skipped 0
                                              « previous  next »
Login
example.spec.js:37                                                7.9s
chromium
✓ Run

Test Steps
> ✓ Before Hooks                                                   3.3s
> ✓ Navigate to "/" — example.spec.js:38                          382ms
> ✓ Click getByRole('link', { name: 'Login' }) — example.spec.js:39     169ms
> ✓ Expect "toBeVisible" — example.spec.js:40                      19ms
> ✓ Expect "toBeVisible" — example.spec.js:41                       5ms
> ✓ Expect "toBeVisible" — example.spec.js:42                       5ms
> ✓ Expect "toBeVisible" — example.spec.js:43                       6ms
> ✓ Expect "toContainText" — example.spec.js:44                     9ms
> ✓ Fill "registerTest@finalproject.SE" getByRole('textbox', { name: 'Email address' }) — example.spec.js:46     16ms
> ✓ Fill "CPE334" getByRole('textbox', { name: 'Password' }) — example.spec.js:47     11ms
> ✓ Click getByRole('button', { name: 'Sign In', exact: true }) — example.spec.js:48     53ms
> ✓ Expect "toBeVisible" — example.spec.js:49                      3.9s
> ✓ After Hooks                                                   144ms

Screenshots
PLANNER   Student event planner
Don't miss a single event!
Join more events!
Summary For You
screenshot
```

Figure 14 A4-1: Login



```
All 5   ✓ Passed 5   Failed 0   Flaky 0   Skipped 0
                                                   « previous
Homepage
example.spec.js:64                                                8.1s
chromium
✓ Run

Test Steps
> ✓ Before Hooks                                                   3.2s
> ✓ Navigate to "/" — example.spec.js:65                          387ms
> ✓ Click getByRole('link', { name: 'Login' }) — example.spec.js:66     169ms
> ✓ Fill "registerTest@finalproject.SE" getByRole('textbox', { name: 'Email address' }) — example.spec.js:67     43ms
> ✓ Fill "CPE334" getByRole('textbox', { name: 'Password' }) — example.spec.js:68     12ms
> ✓ Click getByRole('button', { name: 'Sign In', exact: true }) — example.spec.js:69     63ms
> ✓ Expect "toBeVisible" — example.spec.js:71                      4.0s
> ✓ Expect "toBeVisible" — example.spec.js:72                       6ms
> ✓ Expect "toBeVisible" — example.spec.js:73                       5ms
> ✓ Expect "toBeVisible" — example.spec.js:74                       6ms
> ✓ Expect "toBeVisible" — example.spec.js:75                       6ms
> ✓ Expect "toBeVisible" — example.spec.js:76                       5ms
> ✓ Expect "toBeVisible" — example.spec.js:77                       7ms
> ✓ Expect "toBeVisible" — example.spec.js:78                       5ms
> ✓ Fill "Sharing" getByRole('textbox', { name: 'Search your course here...' }) — example.spec.js:80     20ms
> ✓ Expect "toContainText" — example.spec.js:81                     7ms
> ✓ Expect "toBeVisible" — example.spec.js:82                       4ms
> ✓ Click getByRole('button').filter({ hasText: /^$/ }).nth(2) — example.spec.js:84     36ms
> ✓ Expect "toBeVisible" — example.spec.js:85                       4ms
> ✓ Click getByRole('button', { name: 'Log out' }) — example.spec.js:87     29ms
> ✓ Expect "toBeVisible" — example.spec.js:88                     108ms
> ✓ After Hooks                                                    69ms

Screenshots
Welcome Page
Login  Register
```

Figure 15 A4-1: Homepage

1.4 Manual testing + external user testing

Even with E2E coverage, the team still did manual testing continuously throughout implementation (feature-by-feature as development progressed). In the final phase, the team also had people outside the team test the product to catch usability and real-world workflow issues (a good move—fresh eyes break things instantly, as nature intended).

Comparison (E2E vs Manual)

E2E (Playwright) Best for repeatable “core flow” regression checks (login/register/home routing) and preventing accidental breakage.

Manual testing Better for UX issues, edge cases, visual alignment, and exploratory discovery—especially when requirements shift sprint-to-sprint.

In Sprint 5, the project explicitly emphasized MVP testing, functional testing, user testing, and bug fixing, which aligns with this blended strategy.

2. CI/CD Strategy & Team Workflow

2.1 CI/CD Approach & Maturity Level

In this project, CI/CD automation was implemented through GitHub Actions, but it wasn't the primary quality enforcement mechanism. Instead, the team relied on a combination of:

Branching + merging discipline with clear ownership of different features/files

Human-controlled quality gates: running tests locally, manual QA reviews, then merging

Automated GitHub Actions pipeline as a safety net rather than the main gatekeeper

2.1.1 Pipeline Maturity Assessment:

2.1.1.1 Stable version control workflow with minimal conflicts

2.1.1.2 Local E2E testing + manual testing consistently performed

2.1.1.3 Automated "tests-on-every-push" existed but wasn't the primary quality gate

Deployment process not tightly coupled with automated testing

2.2 GitHub Actions Pipeline (Playwright Tests)

The actual CI pipeline was implemented in Lab 10 through .github/workflows/playwright.yml:

2.2.1 Triggers

2.2.1.1 Push events to main/master branch

2.2.1.2 Pull requests targeting main/master

2.2.2 Key Features

2.2.2.1 Automatically runs Playwright E2E tests

2.2.2.2 Uploads playwright-report as artifacts for historical review

2.2.3 Rationale

2.2.3.1 Provides an automated "checkpoint" for every code integration

2.2.3.2 Reduces risk of breaking critical flows without detection

2.2.3.3 Creates verifiable testing evidence (not just verbal confirmation)

2.3 Branching & Merge Workflow

The team started with multiple branches early in development, then gradually consolidated work on main as the project matured. Merge conflicts were minimal because:

2.3.1 Clear feature/file ownership among team members

2.3.2 Limited overlap in code modifications

2.3.3 Good communication about who's working on what

Result**:** Merges were smooth, and the CI pipeline effectively validated stability after integrating everyone's work—especially for critical user flows covered by E2E tests.

3. Quality in Implementation

Examples of How the Team "Thought About Quality" During Development

Beyond testing and CI, many implementation aspects were designed to prevent long-term quality issues, such as

3.1 Data Structure & Validation (Firestore)

The core data structure consists of users/ and events/ collections. The events collection was designed to support participant lists and a review/rating system, with denormalized fields like totalReviews and averageRating to enable fast display on organizer pages and reduce redundant calculations.

3.2 Multi-layer Security & RBAC

The system employs a multi-layer approach:

3.2.1 Frontend: UX-level validation (e.g., must register first / event must be completed / comment length requirements)

3.2.2 Firestore Rules: Field-level validation + role checks

3.2.3 (Future) Cloud Functions for more complex business logic

3.3 Data Integrity / Race Conditions

The team encountered state sync issues and stale data problems during review updates. They adopted a read-then-write approach and carefully controlled the calculation sequence for aggregate values (totalReviews, averageRating).

Future improvement consideration: Implementing transactions to fully prevent race conditions in critical operations.

4. Deployment (Firebase Hosting)

4.1 Deployment approach

The team deployed using Firebase Hosting via Firebase CLI. This fits the architecture because the system already uses Firebase services (Firestore/Auth), so Hosting is a practical “same ecosystem” choice.

Deployment steps used (Firebase CLI / Hosting)

1. Install Firebase tools

   ```
   npm install -g firebase-tools
   ```

2. Login

   firebase login

   (linked to the deployer's own Firebase account)

3. Initialize Firebase project

   firebase init

   3.1 Select the project

   3.2 Set hosting public directory to: **dist**

   3.3 Configure as SPA rewrite to /index.html: **Yes**

4. Deploy

   firebase deploy

4.2 Quality note: deploy was not automated + tests were not gated

Deployment was performed near the end (final-stage deployment) and **was not connected** to an automated CI pipeline that runs tests before shipping. Instead, the process was closer to

4.2.1 Deploy ⟶ manual smoke test (check main flows still work) ⟶ fix if needed

Comparison (Manual deploy vs CI-gated deploy):

Manual deploy: fast to do, low setup overhead, but riskier (human can forget to test).

CI-gated deploy: more reliable and repeatable, but requires workflow setup + stable test environment.

A4-2 Technical Challenges & Resolutions (Top issues)

Challenge 1 — Cookies & Session did not work properly

Problem Cookies and session handling did not work reliably; session information was not stored/retrieved correctly.

Root Cause Session data and user structure were not aligned with the actual login/session workflow, so persistence logic didn't consistently reflect the authenticated state.

Resolution The team redesigned the user data structure and integrated cookies directly into the login and session-management workflow, enabling proper storage and retrieval of session information.

Outcome / Quality Impact More stable authentication experience; fewer "random logouts / missing session" behaviors.

Challenge 2 — Firestore Security Rules complexity (permissions + debugging pain)

Problem Firestore Security Rules frequently produced "missing or insufficient permissions" errors, and overly complex rules became difficult to debug and maintain.

Root Cause Trying to encode complex business logic inside rules (e.g., validating whether a user is registered via complex participant mapping) increases rule complexity and reduces debuggability.

Resolution The team adopted a simpler rules strategy

1. Use security rules mainly for field-level validation
2. Push business logic to frontend validation (or Cloud Functions in future)
3. Keep rules readable and narrow (example: allow only certain fields to change using affectedKeys().hasOnly([...]))

Outcome / Quality Impact

1. Clearer permission model
2. Faster debugging
3. Reduced risk of accidentally over-permissive rules (a common "oops" when rules get too clever)

Challenge 3 — Playwright E2E tests had repeated context issues

Problem Playwright E2E tests repeatedly ran into browser context instability/issues.

Root Cause Tests were not sufficiently isolated; state leaked between tests, and UI elements were sometimes not fully loaded when assertions ran.

Resolution

1. Added beforeEach() to reset browser context before every test
2. Used waitForSelector() to wait for UI readiness

   This made the E2E suite stable and reliable.

Outcome / Quality Impact Reduced flaky tests ⟶ less time wasted ⟶ more trust in automation.

A5. Conclusion & Lessons Learned

A5-1. Learning Curve & Key Lessons Learned

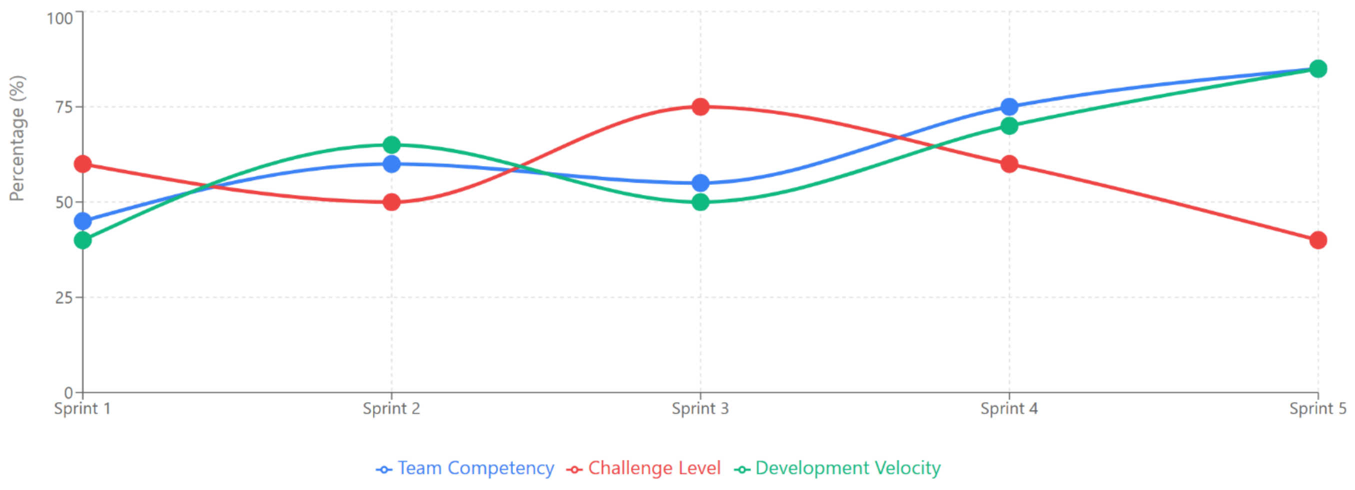


Figure 16 A5-1: Learning Curve

This graph illustrates the team's progress across five sprints using three key indicators: Team Competency, Challenge Level, and Development Velocity.

The percentages shown are relative evaluations based on the team's internal benchmarks for each sprint. These benchmarks consider factors such as familiarity with the technology stack, task complexity, and the amount of work successfully delivered. The values are not derived from a fixed mathematical formula but are used to represent trends and overall development throughout the project.

1. **Sprint 1 Getting Started**

   The team focused on setting up the system foundation and learning React and Firebase. Authentication and basic UI development were challenging due to limited familiarity with the technology, resulting in lower productivity (Competency 45%, Challenge 60%, Velocity 40%).

2. **Sprint 2 Building Momentum**

   Development became more structured as UI designs were completed in Figma and core user features were implemented. Improved understanding of workflows reduced challenges and increased delivery speed (Competency 60%, Challenge 50%, Velocity 65%).

3. **Sprint 3 Facing Technical Issues**

   Significant difficulties arose from cookie and session management, causing inconsistent authentication behavior. These issues required extensive debugging, increasing challenges and slowing progress, with some features postponed (Competency 55%, Challenge 75%, Velocity 50%).

4. **Sprint 4 Recovery and Improvement**

   After resolving authentication problems, development momentum recovered. The team refined user data structures and added an Admin Dashboard, improving system management despite increased scope (Competency 75%, Challenge 60%, Velocity 70%).

5. **Sprint 5 Finalization and Delivery**

   The final sprint emphasized refinement, testing, and stabilization. Advanced features and comprehensive testing enabled the delivery of a stable MVP, reflecting high efficiency and technical confidence (Competency 85%, Challenge 40%, Velocity 85%).

Key Lessons Learned

Throughout the development of the Student Event Planner web application, the team gained several important lessons related to software engineering practices, quality assurance, and team coordination. These lessons emerged from real implementation challenges and iterative problem-solving across multiple sprints.

1. **End-to-end testing provides the most practical confidence for user-facing systems.** Implementing Playwright E2E tests highlighted that testing real user flows (such as registration, login, and navigation) is more effective than relying solely on isolated unit tests, especially in systems that depend on authentication, routing, and external services like Firebase.

2. **Balancing automation with manual testing is essential for product quality.**
   While automated tests helped prevent regressions in critical flows, manual and external user testing were crucial for identifying usability issues, edge cases, and real-world workflow problems that automation alone could not detect.
3. **CI/CD maturity should match project scope and stability.**
   The team learned that a lightweight CI/CD setup can still be effective when combined with disciplined branching, local testing, and human quality checks. Full automation is valuable but should be introduced gradually as the system and test coverage mature.
4. **Keeping security rules simple improves maintainability and safety.**
   Attempts to encode complex business logic in Firestore Security Rules increased debugging difficulty and risk. Simplifying rules and moving logic to the application layer resulted in clearer permissions and more reliable access control.
5. **State management and data consistency require deliberate design.**
   Issues related to session handling, stale data, and race conditions emphasized the importance of aligning data structures with real user workflows and carefully controlling read–write sequences, especially when aggregate values are involved.

Cross-check References between Part A and Part B

A comprehensive cross-check was conducted between Part A (Analytical Report) and Part B (Technical Documentation & Artifacts) to ensure alignment, traceability, and consistency across the project documentation. The analytical discussions presented in Part A are consistently supported by concrete and

1. **Requirements in A1-3** (Functional and Non-Functional Requirements) directly reference REQ-01 to REQ-07 in Part B (B2). Deferred and dropped features listed in Table A1-3 are consistently reflected in the Requirements Version History (Table B2-2).

2. **Project management analysis in** A2 is supported by PM-01 to PM-04 in Part B (B6). Timeline deviations and scope changes discussed in A2-1 and A2-2 correspond exactly to issues documented in PM-03 (Sprint 4 Report).
3. **Design evolution in A3** is traceable to DES-01 to DES-04 in Part B (B3). The architectural shift from MVC to Layered Architecture described in A3-2.1 aligns with the final architecture artifact (DES-04).
4. **UI/UX comparison in A3-2.2** directly references UI-01 to UI-13 in Part B (B4). Differences between prototype and implementation are consistent with the constraints and rationale explained in Part A.
5. **Testing, CI/CD, and quality practices in A4** are supported by QA-01 and TECH-01 in Part B (B5), including Playwright E2E testing evidence, CI pipeline configuration, and architecture decision records.

This confirms that Part A functions as the analytical narrative, while Part B provides verifiable evidence, without duplication or contradiction.

Formatting & Consistency Check

To ensure a professional, coherent, and academically consistent presentation, a final formatting and consistency review was conducted across Part A and Part B. This review focused on structural alignment, terminology usage, cross-referencing accuracy, and documentation–artifact coherence, as outlined below.

1. Section numbering, headings, and subsection labels were reviewed to ensure that Part A (A1–A5) and Part B (B1–B6) follow a consistent, clearly separated, and non-overlapping structure.
2. Terminology, including the system name, user roles, technology stack, and feature names, was applied uniformly across all sections to maintain clarity and consistency.
3. Figures and tables were checked to confirm that numbering follows a clear and sequential convention (e.g., A3-2, A4-1, B3-1), enabling accurate referencing throughout the document.

4. Cross-references between text, figures, tables, and artifacts consistently use standardized identifiers (REQ, DES, QA, PM), ensuring traceability between analysis and supporting evidence.
5. The repository structure and artifact organization in Part B were verified to align with the descriptions and references made in Part A, reinforcing coherence between documentation and implementation.

A6. Appendix (Part A)

Git Log

```
commit 305017cee5285d6550b32ca06e3a05a9ed08b526 (HEAD -> main, origin/main)

Author: Pemika Chongkwanyuen <paeng.pemika@gmail.com>

Date:   Sun Nov 23 18:44:42 2025 +0700


    feat: add admin login and dashboard components with styling


    - Implement AdminLogin component for admin authentication using Firebase.

    - Add routing for admin login and dashboard in main application.

    - Create AdminDashboard component and corresponding CSS for styling.

    - Develop AdminLogin CSS for improved UI/UX.


commit 8aa0c1c86eaa67823f66352c86f2a03c2802990d

Author: Pemika Chongkwanyuen <paeng.pemika@gmail.com>

Date:   Fri Nov 21 22:50:54 2025 +0700


    fix: show event image in homepage


    refactor: Clean up event fetching and registration loading logic


commit 9705043142433c2f5de04f25a3e7d5f60cedd413

Author: kkanyavann <kkanyavann@gmail.com>
```

```
Date:   Fri Nov 21 21:50:04 2025 +0700

    feat: Implement user profile fetching and registration handling in event management

commit f6f9bf4c835e314891553414c9152dece7889521
Author: kkanyavann <kkanyavann@gmail.com>
Date:   Tue Nov 18 13:32:25 2025 +0700

    feat: Enhance user profile management and event loading for organizers and students

commit 3287e2b9872aca6f48f381445e597726e31239cc
Author: Lamoose <asiasupisara@gmail.com>
Date:   Sun Nov 16 16:29:19 2025 +0700

    header_update (About, For paartner, study tips)

commit e71f7bf5a5b559e8f4924e4190522c37d36ff633
Author: Lamoose <asiasupisara@gmail.com>
Date:   Sat Nov 15 20:04:49 2025 +0700

    for partner, about

commit f97f017f32e005328f5d6d09ece7d95e875470a2
Author: Lamoose <asiasupisara@gmail.com>
```

```
Date:   Sat Nov 15 13:38:10 2025 +0700

    logo, register

commit a8727404ddf0069642227c685ea6438b5871158b
Author: Pemika Chongkwanyuen <paeng.pemika@gmail.com>
Date:   Sat Nov 15 12:12:36 2025 +0700

    feat: Implement user profile management and role-based navigation

    - Added Profile component for user profile management including editing personal information and changing passwords.
    - Integrated Firestore to fetch and update user data.
    - Enhanced Login component to navigate users based on their roles (organizer or student).
    - Updated OrganizerHome to include a profile navigation button.
    - Improved Register component to include user role during registration.
    - Added loading states and error handling for better user experience.
    - Created Profile.css for styling the new Profile component.

    Note: All features involved password changing and password resetting still don't work.
    - Unable to receive password reset email.
    - Unable to change password on profile page due to credential issue.
```

```
commit 125c7fbe9347a4fdb06474ecaccc0f6674cad270

Author: kkanyavann <kkanyavann@gmail.com>

Date:   Fri Nov 14 18:48:08 2025 +0700


    resolve store register DB


commit 9dd7de5d6bd02a6e740b9a530bba7e02e3d7b373

Author: Pemika Chongkwanyuen <paeng.pemika@gmail.com>

Date:   Wed Nov 12 17:33:29 2025 +0700


    add forgetPassword but cannot sent reset pass email


commit 60581e932041e7cecd473d217d6c8444e95b0b69

Merge: ece4c8f 561df70

Author: Pemika Chongkwanyuen <paeng.pemika@gmail.com>

Date:   Wed Nov 12 15:57:09 2025 +0700


    Merge branch 'main' of
https://github.com/PowderPePang/CPE334_FinalProject_StudentPlanner


commit ece4c8fe0eedfe955ef992c0e14f19e7b0fb792d

Author: Pemika Chongkwanyuen <paeng.pemika@gmail.com>

Date:   Wed Nov 12 15:55:11 2025 +0700
```

```
    add organizer management

commit 561df70cb9cc4976868aff9aab7ec3941ce08d5c
Author: Junior <juniormonsicha.kt@gmail.com>
Date:   Wed Nov 12 15:29:35 2025 +0700

    add-qrcode,notification,pop up registeration,link database in event detail

commit 74a362831da79def249f584159835d90f8bc139a
Merge: 7528644 9d07f83
Author: Junior <juniormonsicha.kt@gmail.com>
Date:   Tue Nov 11 18:47:23 2025 +0700

    Merge branch 'main' of
https://github.com/PowderPePang/CPE334_FinalProject_StudentPlanner

commit 752864446292d96e88462d94c106173b8e64beec
Author: Junior <juniormonsicha.kt@gmail.com>
Date:   Tue Nov 11 18:47:18 2025 +0700

    create event detail

commit 9d07f83305347e1e44c06d3b2e50c7f4b5ee0b7d
Merge: a3fe9ec a6a7184
```

```
Author: Lamoose <asiasupisara@gmail.com>
Date:   Tue Nov 11 17:17:37 2025 +0700

    Merge branch 'main' of
https://github.com/PowderPePang/CPE334_FinalProject_StudentPlanner

commit a3fe9ec3201d5480338ed674a2c436a549537ab8
Author: Lamoose <asiasupisara@gmail.com>
Date:   Tue Nov 11 17:17:30 2025 +0700

    app, reg

commit a6a718448fb1a2f5d56a8325521573546b32225b
Author: May <jantarat32443@gmail.com>
Date:   Tue Nov 11 16:57:13 2025 +0700

    add role in register

commit 504f5e53eaa5d02f1859aa1da6ce5f351f496f72
Author: Lamoose <asiasupisara@gmail.com>
Date:   Sat Nov 8 20:41:06 2025 +0700

    App.jsx (dec)
```

```
commit 8fd9c2c00369396bca79f98f2502154bd214c059

Author: Pemika Chongkwanyuen <paeng.pemika@gmail.com>

Date:   Thu Nov 6 13:19:23 2025 +0700


    decorate Login and Register UI


commit 9ce425c56ed653f968a21b0cf68a22c5e6f18169

Author: Pemika Chongkwanyuen <paeng.pemika@gmail.com>

Date:   Thu Nov 6 12:25:00 2025 +0700


    add entering user info in register page


commit 590e6cb3c14e0db54651338eb4bdc4519c78e3c7 (tag: v1.0.0)

Merge: 2005b2c c812f80

Author: Pemika Chongkwanyuen <paeng.pemika@gmail.com>

Date:   Wed Nov 5 22:24:27 2025 +0700


    Merge branch 'main' of
https://github.com/PowderPePang/CPE334_FinalProject_StudentPlanner


commit 2005b2caddbccfa611a703d0c5764157f298b5c3

Author: Pemika Chongkwanyuen <paeng.pemika@gmail.com>

Date:   Wed Nov 5 22:16:17 2025 +0700
```

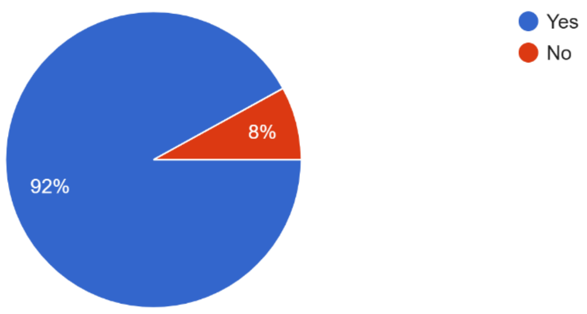
```
    add Unauthorized401 page but has not been connected with main page

commit c812f808806ceee186a073b14077ed8302383608
Author: kanlayaa <amkanlaya2@gmail.com>
Date:   Wed Oct 29 18:42:46 2025 +0700

    add picture duck in event

commit d5ef9f84edc38851a3b718d6c0f9fc8a6b23bdaa
Author: kanlayaa <amkanlaya2@gmail.com>
Date:   Wed Oct 29 17:29:05 2025 +0700

    add

commit 44c72c0bb56fa97dda3c22c91a2a97e68ff170f6
Author: Pemika Chongkwanyuen <paeng.pemika@gmail.com>
Date:   Wed Oct 29 13:49:27 2025 +0700

    delete github actions and add step in README.md

commit 38f45fc72c09b051b472ff031ec04fedc1e1350d
Author: Pemika Chongkwanyuen <paeng.pemika@gmail.com>
Date:   Wed Oct 15 21:02:25 2025 +0700
```

```
    Add github actions

commit 4462d008033f32a113817ffef342744e106aa01a
Author: Pemika Chongkwanyuen <paeng.pemika@gmail.com>
Date:   Tue Oct 14 02:41:41 2025 +0700

    Edit README.md

commit 0b940357901a4a774182902dc62e0af7cbe8240a
Author: Pemika Chongkwanyuen <paeng.pemika@gmail.com>
Date:   Tue Oct 14 02:24:08 2025 +0700

    first commit
```

User Testing

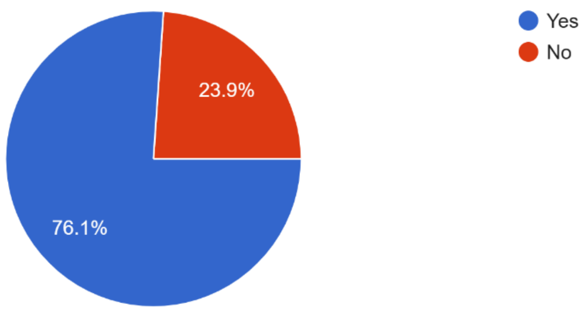
Have you ever use our webside 'Yes we can - planner'

50 responses



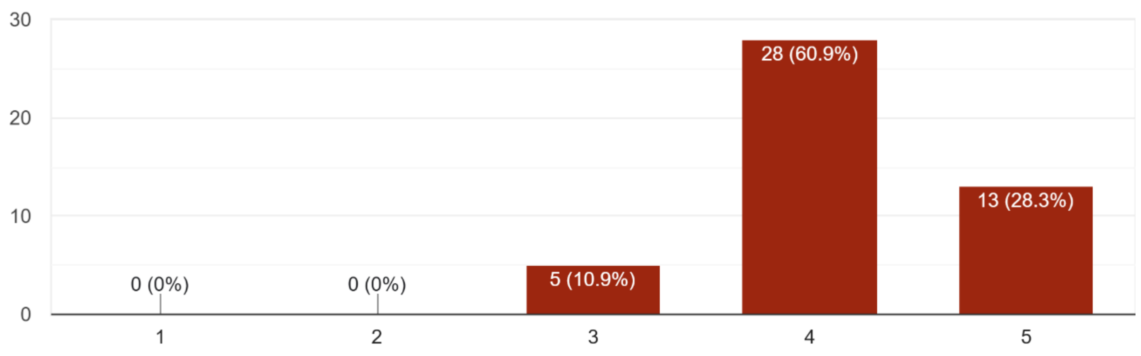
Do you attend CPE KMUTT?

46 responses

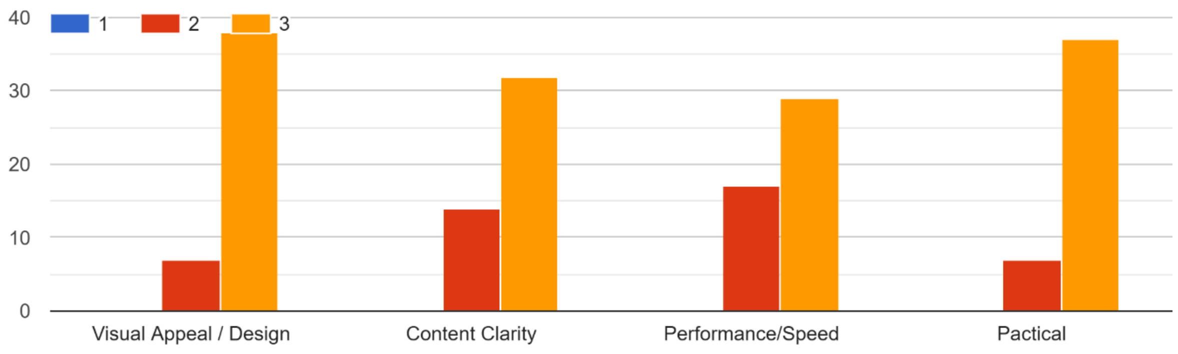


What is your impression on the planner website?

46 responses

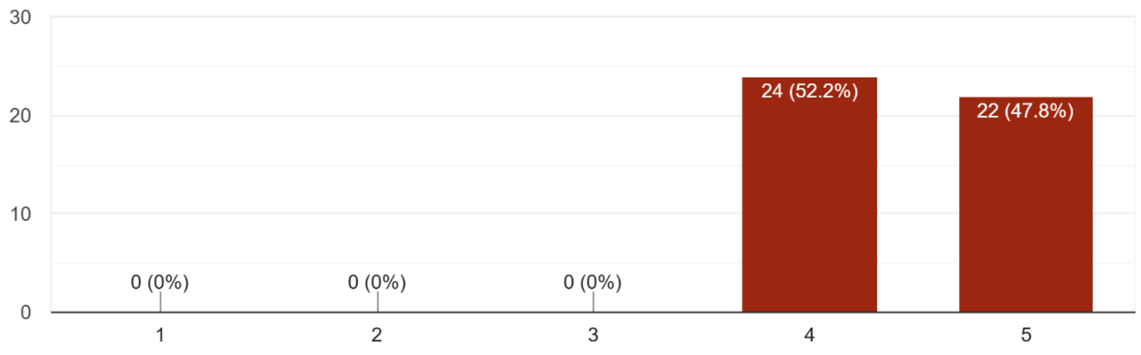


How you found our website...?



How likely are you to recommend "Yes We Can Planner" to a friend or colleague?

46 responses



30
20
10
0
0 (0%)
0 (0%)
0 (0%)
24 (52.2%)
22 (47.8%)
1
2
3
4
5