Software Requirements Specification

for

<EventHub: A Campus Event Management System>

Version 1.0 approved

Prepared by <Wong Hon Jun, Low Wei Hong , Lakshay Arora , Lakshay Sachdeva , Jatanjeet Singh , Dinesh Amarakone >

<Team Group-P03-08>

<6 August 2025>

Table of Contents

Revision History

Name	Date	Reason For Changes	Version

All Team Members	6/8/2025	Initial Draft	1.0
All Team Members	24/8/2025	Reviewing for milestone 1	1.1

1. Introduction

1.1 Purpose

This document outlines the Software Requirements Specification (SRS) for the development of EventHub, a web-based campus event management system for RMIT University. The purpose of this document is to define a shared understanding between the development team and the client (Product Owner) about the system's intended behavior, features, and design.
EventHub facilitates the creation, discovery, and management of events by student clubs and organizations. It enables students to RSVP, receive event updates, and provide feedback, while allowing organizers to manage attendees and monitor engagement.

This SRS includes the functional and non-functional requirements, architectural decisions, user interface overview, system constraints, and external interface details that will guide the development process through subsequent sprints.

1.2 Document Conventions

The following conventions are used in this document:

- **Requirement IDs**: Functional Requirements are prefixed with FR-# (e.g., FR-1), and Non-Functional Requirements with NFR-# (e.g., NFR-2).
- **Bold text** is used for section titles and subheadings to improve readability.
- **Italics** are used for emphasis or placeholders.
- Each requirement is listed with its own unique identifier; priorities are not inherited and must be explicitly stated if used.
- Acceptance Criteria (for user stories) are written using Gherkin-style syntax: Given / When / Then.

1.3 Intended Audience and Reading Suggestions

Developers will use this to implement features based on requirements. Testers will use this to design test cases from the acceptance criteria. Product Owner (Tutor) will review if requirements align with project goals. Lastly, the Project Manager (Scrum Master) will plan sprints and releases.
Readers unfamiliar with the project should begin with Sections 1 and 2. Developers and testers should pay special attention to Sections 3, 4, and 6. The appendices provide reference material for definitions and diagrams.

1.4 Product Scope

EventHub is a centralized platform that enables the creation, discovery, and management of campus events. It empowers student clubs and organizations to promote events, manage attendance, and collect feedback. Students can RSVP, receive reminders, and view personalized event recommendations.

The platform supports real-time updates, mobile-friendly access, and administrative tools for content moderation and user management. EventHub aligns with RMIT's goal of increasing student engagement and enhancing campus life.

1.5 References

List any other documents or Web addresses to which this SRS refers. These may include user interface style guides, contracts, standards, system requirements specifications, use case documents, or a vision and scope document. Provide enough information so that the reader could access a copy of each reference, including title, author, version number, date, and source or location.

2. Overall Description

2.1 Product Perspective

EventHub is a freshly made standalone web application that is developed by RMIT University. It does not substitute any existing system-it just co-exists with them. The desired three-tier scheme of the project includes a front-end interface, a backend in the form of a RESTful API, which is implemented using Spring Boot, and a MySQL relational database to store persistent data.

EventHub utilizes third party services like an SMTP server to send email reminders and may also optionally load previews of static maps using the Google Maps API to keep things running smoothly. The stack will run on docker containers and cloud infrastructure will be used.

The product meets the internal student-engagement strategies at RMIT, and it is likely to be installed in an internal network of the university, or publicly via an appropriate authentication.

2.2 Product Functions

The core functions of EventHub are summarized as follows:

- Event Browsing
    - View a list of upcoming events on the homepage.
    - Filter events by date, category, and keywords.
- Event Participation
    - Students can RSVP to events.
    - Students can view and manage their RSVP history.
    - RSVP confirmation and reminder emails are sent to users.
- Event Management (Organizers)
    - Create, edit, and delete events.
    - View RSVP list for each event.
    - Upload post-event photos to create a gallery.
- Feedback Collection
    - Students can rate and leave comments on past events.
    - Feedback is viewable by event organizers.
- Event Recommendations
    - Personalized event suggestions based on tags, categories, or past RSVPs.
- Admin Capabilities

    - View and manage all events.
    - Delete inappropriate events.
    - Deactivate or ban user accounts.
- Extended features
    - Calendar view of events.
    - QR code check-in system for events.
    - Export RSVP list to CSV (organizers only).
    - Static Google Map preview for event locations.
    - Badge system to reward active users (e.g., “Frequent Attendee”).
    - Public REST API to allow external systems to query events.

2.3 User Classes and Characteristics

- Student
    - A general user who wants to explore, RSVP, and engage with events.
    - Capabilities:
        - View and filter upcoming events.
        - RSVP to events.
        - Receive email notifications.
        - Submit event ratings and feedback.
        - View personal RSVP history.
- Organizer
    - A member of a student club or group who creates and manages events.
    - Capabilities:
        - All Student capabilities.
        - Create, edit, and delete their own events.
        - View and export RSVP lists.
        - Upload event photos to a gallery.
        - View feedback on their events.
- Admin
    - A system-level user responsible for moderation and system integrity.
    - Capabilities:
        - All Organizer capabilities.
        - View and manage all events.
        - Delete inappropriate content.
        - Deactivate or ban user accounts.

2.4 Operating Environment

The EventHub application will operate in the following environment:

- Client Side:
    - Runs on modern web browsers: Google Chrome, Safari, Mozilla Firefox, and others.
    - Optimized for both desktop and mobile platforms using responsive UI.
- Server Side:
    - Backend developed using Java with the SpringBoot framework.
    - RESTful APIs serve as the primary interface between the frontend and backend.
- Database:

  - MySQL will be used as the primary relational database for storing events, users, RSVPs, and feedback data.
- Deployment:
  - Containerized using Docker.
  - CI/CD pipeline manages using GitHub.
  - Hosted on a Linux-based cloud server.
- External Services:
  - SMTP server or third-party email service for RSVP notifications.
  - Integrates with Google Maps API for static event location previews.

2.5 Design and Implementation Constraints

When it comes to working on EventHub, it is necessary to balance a series of technical and temporal limitations dictated by the course itself and the project as a while. On the backend, we will need to construct all of it with Java 17+ and SpringBoot. The frontend will be a SpringBoot-friendly stack such as HTML, CSS, and JavaScript.

MySQL will also be used as the primary relational database. In terms of structure, we will adhere to the Model-View-Controller paradigm to make it organized and well manageable. Each component is supposed to be isolated and loosely coupled such that these additional features could easily be fit into it and that team members should be able to work independently on each component simultaneously.

The version control is strictly managed via GitHub, where all source code must be stored. We will do standard branching, commit regularly and use GitHub Projects to manage tasks and user stories. We will also automate with GitHub Actions Continuous Integration and Deployment (CI/CD) pipelines to maintain a smooth deployment. Docker containers will package the app when it is ready to push out and we will be able to move it around without any problems.

Security-wise, the application needs to authenticate users, as well as apply a role-based access control, ensuring that students, organizers and administrators obtain the functionality that each of them needs. On top of it all, the entire construction must remain on a 12-week semester schedule, operating in Agile Scrum sprints to release features in iterations.

2.6 User Documentation

EventHub system shall come with an in-depth user documentation to aid both the end users and administrators. This entails an online user guide publicly available that contains the major features and walk-through of the most common procedures, including browsing of events, RSVP process, feedback giving and event managing.

To improve usability, the application will also have in-app tooltips and contextual help messages that will direct the user on how to take certain actions such as creation of events, RSVP and submission of forms. These will assist in reducing the confusion of the users and facilitate an effective learning curve.

There will also be a long-term presence of a Frequently Asked Questions (FAQ) section, where such questions as the visibility of the event, access to accounts, confirmation of ESVP and notifications in the system can be answered. In addition to the above, internal set up configuration guide and deployment instructions will also be kept on aiding with future maintenance and deployment operations by system administrators and developers.

All documents are going to be version-controlled and stored in the project repository in the docs/ folder, which will allow them to be accessible, trackable, and updated as more features are added.

2.7 User Stories

The following user story represents the overarching vision and primary value proposition of the EventHub platform. This epic-level user story encompasses all major system functionalities and serves as the foundation for detailed feature requirements throughout this document.

As a campus community member, I want to discover, participate in, and manage campus events through a centralized platform so that I can stay connected with university life, find events that match my interests, and easily organize or attend activities that enhance my campus experience.

Acceptance Criteria:

- Given I visit EventHub, when I browse the platform, then I can see all upcoming campus events with filtering and search capabilities
- Given I find an interesting event, when I view its details, then I can see comprehensive information including location, timing, and organizer details
- Given I want to attend an event, when I RSVP, then I'm registered and receive confirmation and reminders
- Given I'm an event organizer, when I create and manage events, then I can track attendance, communicate with attendees, and gather feedback
- Given I attend events, when they conclude, then I can provide feedback and view event galleries to stay engaged with the community

Test Scenarios: Complete user journey from event discovery to attendance and feedback works seamlessly. Organizers can create, manage, and track events while students can easily find and participate in campus activities.

Definition of Done: Full EventHub platform deployed with all core modules functional. End-to-end user journeys tested, security implemented, performance benchmarks met, accepted by stakeholder review, ready for campus launch.

****For further User Stories on other parts of the website please refer to Appendix B****

2.8 Assumptions and Dependencies

The effective evolution of EventHub and its functioning depends on a number of assumptions and external reliance. The assumption here is that all end-users, the student, the organizers, and the administrators will be availed with modern browsers with stable internet connections. Moreover, the user authentication will be managed through RMIT credentials or simulated login system particularly at early stages of development.

This solution requires access to a configured SMTP server or third-party email provider (e.g. SendGrid) to service RSVP confirmation and reminder emails. A relational database service is also required in the system, namely MySQL, as a means of persistence and data integrity. The Containerization and deployment based on the Docker, and continuous integration or continuous delivery processes need GitHub Actions to define automated builds and tests.

Other dependencies can be the external APIs, e.g. to preview static location of an event (Google Maps API), and perhaps libraries to generate QR-codes or export a CSV document. Depending on these requirements, a change or alternative approaches might be necessary in case any of them is out of operation or limited.

3. External Interface Requirements

3.1 User Interfaces

EventHub will feature a modern, user-friendly, and responsive web interface designed for both desktop and mobile use. The major user interface components are:

- **Homepage**: Lists all upcoming events with filter/search functionality by category, keyword, or date.
- **Event Detail Page**: Displays full event information, RSVP button, and post-event feedback form.
- **RSVP History Page**: Allows users to view their own RSVP list and statuses.
- **Organizer Dashboard**:
  - Create/edit/delete events.
  - View RSVP statistics.
  - Upload event photos for gallery.
- **Admin Panel**:
  - View/Manage all events.
  - Deactivate user accounts.
  - Moderate reported events.
- **Responsive Design**: Layout adapts to various screen sizes (mobile-first design using CSS flex/grid).

UI design will follow accessibility best practices (e.g., keyboard navigation, sufficient contrast, alt text for images).

3.2 Hardware Interfaces

EventHub does not directly interface with any physical hardware. It is designed to run in a web browser and relies on standard HTTP communication. However, it assumes:

- User access via internet-enabled devices (PCs, tablets, smartphones).
- Hosting on a Linux-based server (for Docker containerization).

3.3 Software Interfaces

The system will interact with several internal and external software components:

- **MySQL Database:** Stores users, events, RSVPs, feedback, and galleries.
- **SpringBoot Framework:** Provides RESTful API layer and server-side logic.
- **SMTP or Email Service:** Sends RSVP confirmations and reminders.
- **Google Maps API:** Provides static location previews on event detail pages.
- **File Storage System:** Stores uploaded images for event galleries (can be local or cloud-based).
- **Browser Compatibility:** Chrome, Firefox, Safari, Edge.

3.4 Communications Interfaces

All communication within EventHub will be based on standard web protocols:

- **RESTful API over HTTPS:**
    - Secure communication between client and server.
    - JSON used as data interchange format.
- **SMTP:**
    - Outgoing emails for RSVP confirmations, reminders, and system notifications.
- **CI/CD Pipeline:**
    - GitHub Actions used for code integration, testing, and development.

Security protocols (e.g., HTTPS, authentication tokens) will be implemented to protect user data and maintain system integrity.

4. Nonfunctional Requirements

4.1 Performance Requirements

EventHub should be able to support at least 500 users accessing the system at the same time without delays. Common actions such as browsing events, RSVP submissions and viewing event details are expected to respond within 2 seconds under normal conditions and the response time should not be longer than 5 seconds during the peak usage periods such as the start of the semester events. The system will be deployed using scalable cloud infrastructure to ensure future growth can be accommodated without major redesigns.

4.2 Safety Requirements

Measures must be in place to prevent accidental data loss or corruption, although the system does not handle life-critical operations. Daily automated database backups will be maintained and restore procedures must be tested to ensure recoverability. Input validation will be applied at both client and server levels to prevent unintended loss of event or RSVP records. Only authorized users can make changes to event data so that this can reduce the risk of accidental or malicious actions.

4.3 Security Requirements

User accounts will be authenticated using RMIT credentials. Role-based access control will be enforced so that students, organizers and administrators can only access features relevant to their roles. All communication between the client and server will be secured with HTTPS and sensitive information such as passwords will be hashed and stored securely.

Important activities such as event creation, deletion or user account changes must be logged to provide traceability

4.4 Software Quality Attributes

- **Usability:** The interface should be intuitive and follow accessibility standards to support all students, including those with disabilities.
- **Reliability:** The system should achieve at least 99.9% uptime during the semester.
- **Maintainability:** Code will follow modular design practices and be documented to make future updates easier. Automated testing will be used to reduce errors and maintain quality.
- **Portability:** EventHub should run on any Docker-compatible environment without major configuration.
- **Scalability:** The design should allow for the addition of new features, such as extended analytics or payment gateways, with minimal changes to the existing code.

4.5 Business Rules

- Students can RSVP to many events but not duplicate RSVP to the same event
- Organizers are only able to edit or delete the events they created.
- Feedback can only be submitted once the event has finished.
- Attendee lists can only be exported by event organizers.
- Administrators have the authority to remove any inappropriate events and manage user accounts.

5. Other Requirements

EventHub must comply with relevant data protection guidelines, such as the Australian Privacy Principles to ensure personal information is handled responsibly. The system should also be designed with future localization in mind, enabling support for other languages if required.

The database must maintain ACID properties for consistency and reliability, with indexing in frequently queried fields such as event date and category. Components like authentication, RSVP management and notification handling should be built in a modular way so they can be reused in future projects.

Finally, EventHub should provide an extensible REST API that allows integration with other RMIT systems, such as the student portal, ensuring long-term relevance and usability beyond the initial deployment.

6. System Architecture

EventHub employs a **modular monolithic architecture** implemented with Spring Boot (backend) and React.js (frontend). This architectural approach was strategically chosen to balance development efficiency, maintainability, and scalability requirements for a campus-scale event management platform.

The system is organized into five core business modules—User, Event, RSVP, Notifications, and Content—alongside a dedicated Admin module for system governance. All modules share common infrastructure components including a PostgreSQL database for persistence and Redis for caching and session management.

This design enables a 5–6-person development team to work efficiently on distinct functional areas while maintaining a single, cohesive deployable unit. The architecture prioritizes:

- **Rapid Development**: Leveraging proven frameworks (Spring Boot, React) for faster time-to-market
- **Performance**: Redis caching, asynchronous processing, and optimized database queries
- **Security**: Spring Security with JWT authentication, comprehensive input validation, and role-based access control
- **Future Extensibility**: Clear module boundaries provide a migration path toward microservices if scaling demands require it

The modular monolith strategy aligns with academic project constraints by minimizing operational complexity while demonstrating industry-standard architectural principles.

6.1 Architecture Overview

The system follows a **layered modular architecture** with clear separation of concerns:

6.1.1 Frontend Layer (React.js)

- Responsive user interfaces for students, organizers, and administrators
- Event browsing, search functionality, RSVP management, and analytics dashboards
- Communicates exclusively with backend through REST APIs

6.1.2 API Gateway & Security Layer (Spring Boot + Spring Security)

- RESTful endpoint exposure with comprehensive request validation
- JWT-based authentication and role-based authorization
- Request logging, rate limiting, and security headers enforcement

6.1.3 Business Logic Modules (Spring Boot)

- **User Module**: Registration, authentication, profile management, achievement system
- **Event Module**: Event lifecycle management, search, recommendations, categorization
- **RSVP Module**: Attendance tracking, QR code check-ins, capacity management
- **Notifications Module**: Multi-channel messaging via email and in-app notifications
- **Content Module**: File management, image galleries, data export functionality
- **Admin Module**: System configuration, user moderation, analytics, system monitoring

6.1.4 Infrastructure Services Layer

- **PostgreSQL Database**: Centralized data persistence for all business entities
- **Redis Cache**: Session management, query result caching, and performance optimization
- **File Storage**: Local filesystem (development) with AWS S3 integration (production)
- **Background Processing**: Asynchronous task handling for notifications and maintenance

6.1.5 External Integrations

- **Google Maps API**: Location visualization and mapping services

- **Email Services**: SMTP (development) and SendGrid (production) for reliable message delivery
- **Cloud Services**: Optional AWS integration (S3, SES) for production scalability

This layered approach ensures **loose coupling** between components, enabling independent development, testing, and maintenance of system modules.

6.2 Architectural Decisions

6.2.1 ADR-001: Modular Monolith Selection

Decision: Adopt modular monolithic architecture over microservices
Rationale: Balances modularity benefits with operational simplicity suitable for academic team constraints. Provides clear migration path to microservices while avoiding distributed system complexity during development phase.

6.2.2 ADR-002: PostgreSQL Database Choice

Decision: PostgreSQL as primary database management system
Rationale: Superior JSON support for flexible data models, excellent Spring Boot integration, ACID compliance for transactional integrity, and advanced query capabilities for analytics features.

6.2.3 ADR-003: Redis Caching Integration

Decision: Redis for caching and session management
Rationale: Significantly improves response times for frequently accessed data, reduces database load, supports horizontal scaling, and enables secure JWT token management with blacklisting capabilities.

6.2.4 ADR-004: JWT Authentication Strategy

Decision: Stateless authentication using JWT tokens with Spring Security
Rationale: Eliminates server-side session storage, scales horizontally, integrates seamlessly with React SPAs, and provides secure, standards-compliant authentication mechanism.

6.2.5 ADR-005: Asynchronous Notification Processing

Decision: Background job processing for email and notification delivery
Rationale: Prevents blocking of main application threads, ensures reliable message

delivery with retry mechanisms, and supports future extension to additional notification channels.

6.2.6 ADR-006: Technology Stack Selection

Decision: React.js frontend with Spring Boot backend
Rationale: Industry-standard technologies with extensive community support, comprehensive documentation, mature ecosystems, and strong integration capabilities suitable for demonstrating professional software development practices.

7. User Interface Design

7.1 Overview:
The goal of EventHub's User Interface (UI) is to give administrators, organizers, and students a responsive, easy-to-use, and transparent platform for interacting with campus events. The user interface (UI) enables administrators to manage users and events, organizers to manage their events and view attendance, and students to browse, search, and RSVP to events. The design places a strong emphasis on usability, accessibility, and smooth page navigation to make sure users can locate and carry out their intended tasks without difficulty.

7.2 Principles:
EventHub's user interface adheres to these fundamental design guidelines:

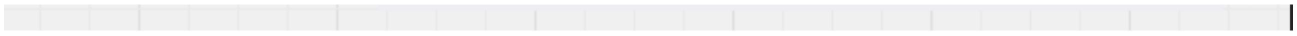
- **Consistency:** Every page has the same layout, buttons, colors, and typeface.
- **Simplicity:** Users learning curve is reduced by the interface's lack of clutter.
- **Responsiveness:** The user interface adjusts to various screen sizes on mobile, tablet, and desktop computers.
- **Accessibility:** All users may use keyboard navigation, legible text sizes, and high contrast typefaces.
- **Feedback:** When users perform tasks like creating an event, confirming an RSVP, or submitting a form incorrectly, they receive unambiguous feedback.

7.3 Wireframes:

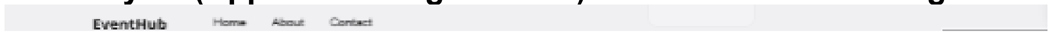
7.3.1 Home Screen (landing page):
At the top of the landing page is a search box that allows users to filter events by area, category, or keywords. A grid structure with links to each event's detailed page underneath highlights or future events. The top and footer offer branding and extra site connections, but the layout prioritises simple browsing, fast access to event details, and intuitive navigation.

7.3.2 Organiser Dashboard:

This dashboard provides organisers with tools to manage their events, including quick event creation, an events table with edit/delete/RSVP actions, RSVP management with CSV export and QR check-in, event gallery upload, and analytics insights. It covers **User Stories 5,10,11,15,17,19** and related organiser tasks.



Admin Panel:
This wireframe illustrates the administrator's view for managing events and users. The top section provides an interface to filter, sort, search, and manage pending or active events, with options to view, approve, or delete. The lower section enables user management, including search, filtering by role, and actions to deactivate or ban multiple users. It covers **User Story 21 (Approve/Manage Events)** and **Admin user management tasks**.



User Profile Page:

The User Profile wireframe presents a clean, card-based layout with three main sections: profile summary, navigation tabs, and content display. The profile summary highlights the user's avatar, name, handle, and quick stats (total RSVPs, badges, and upcoming events). Tabs in the right middle allow users to switch between My RSVPs, Badges, and Notifications within the content area, RSVP cards show event details (title, date, venue, status) with interactive actions such as Remind Me, View Details, or Cancel RSVP. The badges section uses circular icons with progress indicators, while notifications are displayed as a list with toggle options. The design ensures clarity, responsiveness, and easy navigation across devices.

Event Detail Page:

The Event Detail page showcases the complete user journey from event discovery to post-event engagement, integrating essential features including event search functionality, detailed event information with Google Maps preview, an interactive photo gallery, and a feedback system for ratings and comments. The design emphasizes user engagement through clear information hierarchy and seamless navigation between event browsing and detailed viewing experiences.

Appendix 1: Further User Stories

- ### User Story 1: List upcoming events (home page)

As a student, I want to view a list of upcoming events on the homepage so that I can see what events are available to attend.

Acceptance Criteria: The events are shown chronologically. Title, date/time, venue, and category are displayed for each event. When you click on an event, the event detail page loads.

- ### User Story 2: Search/filter events (by date, category, keyword)

As a student, I want to filter events by date, category, or keyword so that I can easily find events that match my interests.

Acceptance Criteria:
The homepage features filters (category dropdown, date picker, and keyword search).
Results that have been filtered are updated dynamically.
If no events meet the filter parameters, a message is presented.

- ### User Story 3: RSVP to events

  As a student, I want to RSVP to an event so that the organizer knows I plan to attend.

  Acceptance Criteria:
  On the event detail page, an RSVP button is offered.
  RSVP is kept in a database.
  After RSVP, a confirmation message is displayed.
  Avoid sending out identical RSVPs

- ## User Story 4: View list of RSVP’d events

As a student, I want to view a list of events I have RSVP’d to so that I can track the events I plan to attend.

Acceptance Criteria:
All RSVPed events are displayed on the "My Events" tab.
Each event's title, time, place, and RSVP status are displayed.
When you click on an event, the event detail page loads.

User Story 5: Create Event
As an event organiser, I want to create a new event with a title, description, date/time, location, category, and keywords so that students can discover and attend it.

Acceptance Criteria:

- Given I am logged in as an organiser, when I fill in all required fields and click “Create,” then the event is saved and appears in the upcoming events list.
- Given a required field is missing or invalid, when I click “Create,” then I see an inline error message and the event is not saved.
- Given the event is created successfully, when I’m redirected, I see a confirmation message and a link to the event’s detail page.

User Story 6: Edit Event
As an event organiser, I want to edit my own events so that I can update the details if plans change.

Acceptance Criteria:

- Given I am the organiser of the event, when I change event fields and save, then the updated details replace the old ones in both the event detail page and upcoming events list.
- Given I am not the organiser, when I attempt to access the edit page, then I receive a permission error and cannot make changes.
- Given I enter invalid data, when I save then I see a validation error and the update is not applied.

User Story 7: Delete Event

As an event organiser, I want to delete my own events so that I can cancel events if needed.

Acceptance Criteria:

- Given I am the organiser, when I click "Delete" and confirm, then the event is removed from the platform and no longer visible in searches or lists.
- Given I cancel the delete confirmation, then no changes are made.
- Given I am not the organiser, when I attempt to delete, then I receive a permission error.

User Story 8: View Event Details

As a student, I want to view full details of an event so that I can decide whether to attend.

Acceptance Criteria:

- Given the event exists, when I open its detail page, then I see the title, description, date/time, location, category, keywords, and organiser's name, along with the RSVP status if I'm logged in.
- Given the event does not exist or is deleted, when I open its detail page, then I see a "not found" message or page.
- Given the page is loading, then a loading indicator is displayed.

• User Story 9: View RSVPs for My Events

As an event organizer, I want to view a list of RSVPs for my events so that I can track attendance and prepare accordingly.

Acceptance Criteria:

Given I am logged in as the event organizer, when I navigate to my event dashboard, then I can see a list of attendees who RSVP'd to each of my events.

The RSVP list should display attendee name, contact information (if shared), and RSVP date/time.

If I have no RSVPs yet, the list should display a "No RSVPs yet" message.

• User Story 10: Collect Event Feedback (Rating/Comments)

As an event organizer, I want to collect ratings and comments from attendees so that I can improve future events.

Acceptance Criteria:

Given the event has ended, when attendees visit the event page, they are prompted to submit a rating (1–5 stars) and optional comment.

Given feedback is submitted, when I view my event dashboard, then I can see all feedback entries sorted by date.

Feedback should only be visible to the event organizer and system administrators.

- ### User Story 11: Upload Event Photos to Gallery

As an event organizer, I want to upload photos after the event so that attendees can view and share event memories.

Acceptance Criteria:

Given I am logged in as the event organizer, when I open the event gallery, then I can upload one or more photos (JPEG, PNG) with a maximum size limit.
Photos should be displayed in a gallery layout on the event detail page.
Only the event organizer and admins can upload/delete photos for that event.

- ### User Story 12: Show Recommended Events Based on Past RSVPs/Tags

As an event organizer, I want to see recommended events based on my past RSVPs and event tags so that I can discover relevant upcoming events.

Acceptance Criteria:
Given I am logged in, when I visit the recommendations section, then I am shown a list of upcoming events that match categories/tags from my past RSVPs or events I created.
Recommendations should include event title, date/time, location, and category.
If there are no matching events, a message should display: “No recommendations at this time.”

- ### User Story 17: View all events

As an admin, I want to see a complete list of all events so that I can keep track of what’s happening on the platform and spot any issues early.

Acceptance Criteria:
When I log in as an admin and open the “All Events” page, I can view every event regardless of its status.
Event details shown should include title, organizer, date/time, and whether it’s active, cancelled, or awaiting approval.
I can search by event name or organizer, and sort results by date.

• User Story 18: Delete Inappropriate Events

As an admin, I want to remove any event that violates our community rules so that the platform stays safe and appropriate for all users.

Acceptance Criteria:
If I'm logged in as an admin, I can click a "Delete" button on any event.
A confirmation popup will appear so I don't delete things by mistake.
The event disappears immediately from all public views once deleted.
The deletion is recorded in an audit log with date, time, and my admin username.

• User Story 19: Manage Users accounts

As an admin, I want to deactivate or ban certain users so that we can stop spam or harmful behavior on the platform.

Acceptance Criteria:
I can open a user's profile and see options to deactivate which is temporary or ban which is permanent.
Deactivated users can't log in until reactivated.
Banned users are blocked from making a new account with the same email address.
All actions I take are stored in the admin activity log for record-keeping.

• User Story 20: Role Enforcement

As an admin, I want the system to enforce role-based permissions so that only authorized people can do sensitive things like deleting events or banning users.

Acceptance Criteria:
Normal users should never see admin-only buttons or controls.
If I'm an admin, I should have full access to all admin tools.
If my admin role is removed, I lose that access straight away without logging out or logging in.

• User Story 21: Approve Pending Events

As an admin, I want to review and approve events before they go live so that we can make sure they follow community rules and meet content standards.

Acceptance Criteria

There is a "Pending Events" section accessible only to admins.

each pending event. I can Approve an event to make it public immediately.

I can Reject an event with an optional reason that is sent to the company while approved events appear instantly in the public event list (homepage and search) and rejected events MUST NOT be appear to public and show a reason for rejection to the company.

• User Story 22: Share Events

As a student, I want to share an event link so that other users can easily access the event details.

Acceptance Criteria

Clipboard available - I am on an event detail page, when I click “Share”, the event link is copied to my clipboard, and I see a confirmation message

Clipboard not available - Given clipboard access is not available, when I click “Share”, the event link is displayed on the screen so I can copy it manually.

• User Story 23: Administrative Management and Analytics (Admin Tools)

As a university administrator, I want to oversee all campus events, manage user accounts, and access comprehensive analytics so that I can ensure platform quality, maintain community standards, and make informed decisions about campus programming and resource allocation.

Acceptance Criteria:

Given I need platform oversight, when I access the admin dashboard, then I can view, edit, and remove any events or user accounts to maintain community standards and platform quality

Given I require operational insights, when I review analytics, then I can access comprehensive reports on event popularity, attendance trends, user engagement patterns, and platform usage metrics

Given inappropriate content or behavior occurs, when I use moderation tools, then I can quickly address violations by removing content, warning users, or deactivating accounts to maintain a safe environment

Given I need strategic planning information, when I generate reports, then I can export detailed data on campus event trends, student interests, and participation patterns to inform programming decisions

Given system issues arise, when I monitor platform health, then I can access logs, performance metrics, and user feedback to ensure optimal system operation

Appendix 2: Milestone 2 Architecture Change (Thymeleaf → React SPA)

Background:
In Milestone 1, our EventHub prototype used Spring Boot with Thymeleaf for server-side rendering of HTML templates. While this approach was suitable for quickly proving the architecture and implementing a thin vertical slice (create, list, view events), it posed limitations as we planned more interactive features for Milestone 2 and beyond.

Decision:

For Milestone 2, the team agreed to switch from Thymeleaf templating to a modern React + Vite frontend while keeping the backend as a REST API in Spring Boot. The backend now only exposes JSON endpoints, while the React SPA consumes these APIs and handles all client-side rendering and routing.

This change reflects our sprint goal to support:

- Richer interactivity (calendar views, RSVP updates without full-page reloads).
- Cleaner separation of concerns (frontend vs backend responsibilities).
- Better scalability for future features like authentication, notifications, and photo uploads.

Rationale:

1. User Experience
    a. React enables a Facebook-style interface with dynamic event feeds, filters, and inline interactions.
    b. Single Page Application (SPA) reduces page reloads and improves responsiveness.
2. **Maintainability**
    a. **A clear separation:**
        i. **Backend (Spring Boot):** APIs, business logic, database.
        ii. **Frontend (React):** UI, state management, routing.
    b. Makes it easier for frontend and backend developers to w**ork ind**ependently.
3. **Tooling & Ecosystem**
    a. Vite provides fast hot-reload for development.
    b. React ecosystem (components for calendar views) accelerates feature delivery.
    c. Aligns with industry practice, preparing us for real-world projects.
4. **Database Persistence**

a. Backend now uses H2 in file mode instead of in-memory mode.

b. This ensures data (events, RSVPs, users) persists across restarts, which was a limitation in Milestone 1.

Implications:

- Refactor of Views: All Thymeleaf templates (events.html, create-event.html, etc.) were retired in favor of React pages (EventsPage.jsx, CreateEventPage.jsx, etc.).
- API-first Design: Controllers (e.g., EventApi.java, RsvpApi.java) return JSON via @RestController, rather than rendering server-side HTML.
- Static Build Integration: The React app is bundled via Vite (npm run build) and served by Spring Boot from /static.
- **Testing:**
- Backend: JUnit + MockMvc for REST endpoints.
- Frontend: React Testing Library (planned).

Risks & Mitigations:

- Learning Curve: Some members had less React experience → mitigated with pair programming and tutorials.
- Integration Overhead: Initial time spent wiring React build into Maven lifecycle → documented build scripts for consistency.
- Dependency Conflicts: Resolved React 19 vs testing-library mismatch by pinning React 18.2.0 for stability.

Outcome:

This architectural pivot strengthens EventHub as a scalable, modern web app. It better positions the team to deliver interactive features in later sprints (e.g., live RSVPs, admin dashboard, analytics).

Appendix 3: Milestone 3 Database Evolution (H2 → MySQL)

Background:
Initially, EventHub used an **H2 in-memory database** to support fast development and quick feature testing during Milestone 1. This lightweight setup simplified local testing but did not retain

data between restarts. To address this, the team later switched to **H2 file mode** in Milestone 2, allowing persistent local storage for realistic user and event data testing.
Finally, for deployment readiness, the system migrated to **MySQL**, enabling reliable data persistence and scalability through Docker-based configuration.

Decision:

- Transition from **H2 (in-memory)** to **H2 (file mode)** for persistence in early development.
- Migrate to **MySQL** for production deployment with Docker Compose integration.
- Maintain H2 for lightweight testing and MySQL for staging and production.

Rationale:

Data Persistence: MySQL provides long-term reliability for user accounts, RSVPs, and events

Scalability: Supports concurrent users and higher data volumes.

Deployment Readiness: Seamlessly integrated into Docker Compose with environment variables for automated setup.

Industry Practice: Aligns with standard enterprise architectures using Spring Boot and MySQL.

Implications:

- Updated application.properties with MySQL configurations.
- Added MySQL service in docker-compose.yml.
- Adjusted entity mappings to ensure schema compatibility.
- Local testing continues H2 for faster feedback

Outcome:

The progressive shift from **H2 to MySQL** strengthened EventHub's persistence, scalability, and deployment stability. This evolution reflects the team's iterative engineering strategy which starting lightweight for rapid prototyping, then maturing toward a robust, production-ready architecture.