Campus Event Manager

Version 1.0

Project Manager

Ailyn Ibbay

Programmer

Kim Harold Bacus

kenneth Damaolao

Quality Assurance & Documentation Specialist

Ledelyn Galupo

May 27, 2025

TABLE OF CONTENTS

1. Document Overview

1.1 Scope

This documentation provides a comprehensive guide to the development and deployment of the Campus Event Manager, an Android-based mobile application designed to streamline event registration and check-in processes within the university setting.

The following aspects are covered in this document:

1. Functional Specifications: Description of features including event creation, registration, QR code generation, and check-in scanning.
2. Technical Specifications: System architecture, database schemas, API integrations, and third-party services.
3. UI/UX Design: Wireframes, user flow diagrams, and accessibility considerations.
4. Security and Privacy: Implementation of authentication, data encryption, and privacy compliance.
5. Testing Strategies: Functional testing, user acceptance testing (UAT), and performance validation.
6. Deployment & Maintenance: Procedures for app release, update cycles, and long-term support.

The following aspects are out of scope:

1. Cross-platform support (the app is Android-only for this version)
2. Web-based admin panel (all admin functions are mobile-based)
3. In-app payments or e-commerce features
4. Event analytics beyond check-in and registration logs

1.2 Audience

This document is intended for the following stakeholders:

1. Developers: To understand the system architecture, code organization, APIs, and deployment process.
2. UI/UX Designers: To align user experience goals with technical constraints.
3. Testers/QA Engineers: To execute test cases and validate the system's performance and reliability.
4. Project Stakeholders & Instructors: To track development progress and ensure the application aligns with academic and functional goals.
5. Future Maintainers: To aid in updating, debugging, or expanding the application post-deployment.

2. Project Overview

2.1 Executive Summary

The Campus Event Manager is a mobile application built using Android Studio that enables students and organizers within an academic setting to manage event registrations and streamline the check-in process through the use of QR codes. The application provides role-based access for both users (attendees) and administrators (event organizers), integrating modern APIs for authentication, data management, and real-time communication. This project aligns with the university's goals of embracing digital transformation and improving operational efficiency in managing campus events.

2.1.1 Objectives

2.1.1.1 Business Goals

1. Simplify event registration and attendance tracking on campus.
2. Reduce manual workloads and paper-based processes for organizers.
3. Improve event attendance monitoring and record-keeping.

2.1.1.2 Technical Goals

1. Implement secure authentication using Firebase Auth.
2. Enable real-time registration and data storage using Firestore.
3. Generate and scan QR codes for check-in functionality.
4. Send confirmation emails via an integrated email API.
5. Back up registration data to Google Sheets for audit and reporting purposes.

2.1.1.3 High-Level Features

1. Role-Based Access: Separate functionalities for Admin and User roles.
2. Event Creation: Admins can add and manage events through the app.
3. User Registration: Students can register for events with a single tap.

4. QR Code Generation: Registered users receive unique QR codes for entry.

5. Check-In via Scanning: Admins scan QR codes to mark attendance.
6. Email Notifications: Confirmation emails sent upon registration.
7. Registration Logs: Sync event data to Google Sheets for tracking.

2.2 Problem Statement

Event registration and attendance tracking on campus are traditionally handled through manual sign-up sheets or basic Google Forms, which are inefficient, prone to errors, and lack real-time monitoring. These outdated methods often result in long lines, duplicated records, and no proper way to verify attendance during events.

2.2.1 User Needs

1. Students need a quick and easy way to register for events and verify their participation.
2. Organizers require a centralized system for creating events, tracking attendees, and managing check-ins efficiently.
3. Administrators seek access to accurate and up-to-date reports on attendance and participation for evaluation and reporting.

2.2.2 Market Analysis Summary

Universities and schools increasingly rely on digital platforms to manage their activities. The demand for mobile-based event management systems has grown as institutions aim for paperless, streamlined solutions. While there are commercial event management apps, most are too complex or expensive for small-scale academic use. The Campus Event Manager addresses this gap by offering a free, focused, and user-friendly system tailored for academic institutions like Bukidnon State University. This gives the system a unique edge in targeting internal school events, seminars, orientations, and more.

3. Functional Specifications

3.1 Feature List

1. User Authentication
2. Role-Based Dashboard
3. Event Creation (Admin)
4. Event Registration (User)
5. QR Code Generation
6. QR Code Scanning & Check-In (Admin)
7. Email Confirmation Notification
8. Google Sheets Logging

3.1.1 Detailed Descriptions

Feature Name: User Authentication

Allows users to securely sign up, log in, and manage sessions using Firebase Authentication.

User Interaction Flow:

1. User opens the app → taps "Login" or "Sign Up"
2. Inputs email and password
3. Authenticated via Firebase and redirected based on role

Use Cases:

- Primary: User logs in to access event functions.
- Alternate: Invalid login displays error message and retry option.

Feature Name: Role-Based Dashboard

Displays a customized dashboard depending on the authenticated user's role (admin or regular user).

User Interaction Flow:

1. After login, system checks user role in Firestore
2. Admin → Event management tools
3. User → List of upcoming events and registration options

Use Cases:

- Primary: Admin accesses the event creation panel.
- Alternate: A user accesses available events for registration.

Feature Name: Event Creation (Admin)

Admins can create, update, and manage events including name, date, location, and capacity.

User Interaction Flow:

1. Admin selects “Create Event”
2. Fills in event form
3. Submits event → saved to Firestore

Use Cases:

- Primary: Admin creates a new seminar.
- Alternate: Admin edits an existing event’s date.

Feature Name: Event Registration (User)

Users can register for available events and automatically generate a QR confirmation.

User Interaction Flow:

1. User views event list
2. Taps “Register” on desired event
3. Receives success message and QR code

Use Cases:

- Primary: Student registers for a campus orientation.
- Alternate: User attempts to register for a full event → shown "Fully Booked" message.

Feature Name: QR Code Generation

Upon successful event registration, a unique QR code is generated and associated with the user’s registration ID.

User Interaction Flow:

1. Event registered → Generate QR code
2. Save QR to local device or screen display
3. QR is stored for scanning later

Use Cases:

- Primary: Student shows QR code at the event entrance.
- Alternate: QR code is regenerated if deleted/lost.

Feature Name: QR Code Scanning & Check-In

Admins scan attendee QR codes using a built-in scanner. The system verifies registration and updates attendance status in Firestore.

User Interaction Flow:

1. Admin taps “Scan” button
2. Camera opens → scans QR
3. If match found → mark as "Checked-in"
4. If already checked-in → show appropriate message

Use Cases:

- Primary: Valid QR scanned and attendance confirmed.
- Alternate: Duplicate check-in attempt → notify admin.

Feature Name: Email Confirmation Notification

Sends a registration confirmation email to the user with event details and QR code using an email API.

User Interaction Flow:

1. Successful registration triggers email API
2. Confirmation email is sent

Use Cases:

- Primary: User receives email immediately after registration.
- Alternate: Retry if email sending fails.

Feature Name: Google Sheets Logging

Backs up registration and check-in logs to a linked Google Sheet for administrative use.

User Interaction Flow:

1. After check-in → send data to Google Sheets via API
2. Admin can access logs online

Use Cases:

- Primary: Log used for event summary reports.
- Alternate: Sync retry in case of network issues.

3.2 User Stories & Requirements

3.2.1 User Stories

1. As a student, I want to register for campus events easily so that I can attend without hassle.
2. As an event organizer, I want to scan attendees' QR codes so I can track who checked in.
3. As a student, I want to receive a QR code and confirmation email after registering for an event.
4. As an event organizer, I want to create and manage multiple events so I can organize seminars and activities.

3.2.2 Acceptance Criteria

1. User must be able to log in or register using Firebase Auth.
2. Registered users receive a unique QR code and email notification
3. Admins/Organizers can create and delete events.
4. System must prevent duplicate check-ins.
5. Logs must be saved in Firestore and synced with Google Sheets.

3.2.3 Non-functional Requirements

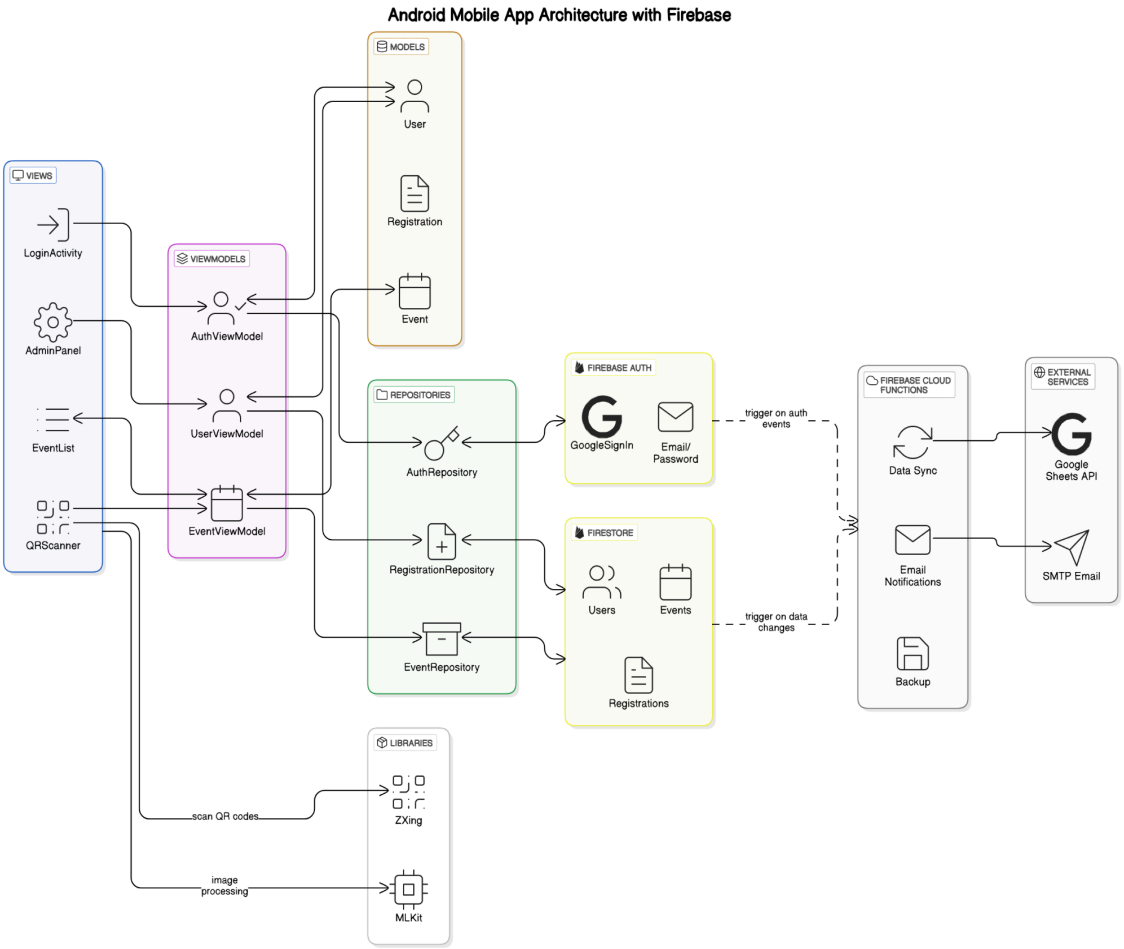
1. Performance: App must handle 100 concurrent users during peak hours.
2. Security: All user data must be stored securely; communication must use HTTPS.
3. Scalability: System should support future integration of a web admin panel.
4. Compatibility: Works on Android 8.0 and above.

4. Technical Specifications

4.1 Architecture

4.1.1 Overview Diagram

The Campus Event Manager leverages a modern MVVM (Model-View-ViewModel) architecture with repository pattern, backed by Firebase services.



4.1.2 Design Patterns

The app follows the MVVM architecture, which separates the presentation logic from business logic and data handling for better testability and maintainability.

Model: Represents the data layer, including Firebase data models and business logic.

Example: User.java, Event.java, Registration.java

Handles data from Firestore and local storage.

View: Android Activities and Fragments that display data and observe changes from the ViewModel.

Example: LandingPage.java, EventListActivity.java

ViewModel: Acts as the intermediary between View and Model. It holds and prepares the data to be displayed in the View, using LiveData or ObservableFields.

Example: EventViewModel.java, AuthViewModel.java

ViewModel fetches data from Firestore and updates LiveData objects, which the UI observes and reacts to automatically.

Why MVVM?

1. Encourages separation of concerns.
2. Makes it easier to write unit tests for ViewModels.
3. Reduces boilerplate code in Activities/Fragments.
4. Works well with Android Architecture Components like LiveData, ViewModel, and DataBinding (optional in Java).

4.2 Platform-Specific Considerations

4.2.1 Android Guidelines

The Campus Event Manager mobile application adheres to Material Design principles, which provide a consistent, responsive, and intuitive user experience across Android devices. The design follows Android UX standards, ensuring smooth navigation and clear user interactions. This includes:

Consistent Navigation: The app uses a structured navigation flow with clearly defined paths between Activities and Fragments.

Proper Use of Activities and Fragments: Activities are used for major screens, while Fragments allow flexible user interfaces within those screens.

Clear Button Hierarchy: The interface highlights primary actions using floating action buttons or prominent UI elements, with secondary actions placed accordingly.

User Feedback: Feedback mechanisms such as Snackbars and Toasts are utilized to notify users of completed actions or errors without disrupting the experience.

4.2.2 Hardware & OS Compatibility

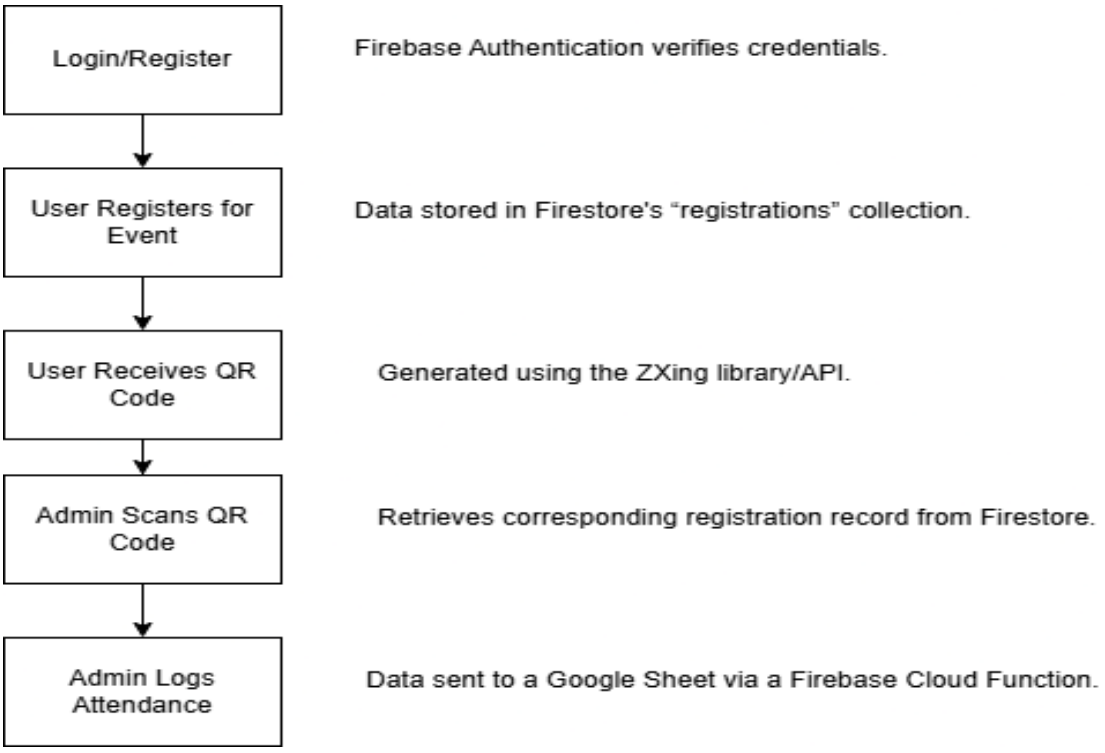
The application is optimized for a wide range of Android devices:

1. **Minimum SDK**: API Level 26
2. **Target SDK**: Latest stable release (e.g., API 24 – Android 14)
3. **Hardware Requirements**: Devices must include:
    - A functioning **camera** (for QR code scanning)
    - An **active internet connection** (for real-time synchronization and authentication).

4.3 Data Management

4.3.1 Data Flow Diagram

The data flow between the mobile app and various services is linear and straightforward, as shown in the following sequence:



4.3.2 Database Schemas

The application uses **Firebase Firestore** as its NoSQL cloud database, with the following collections:

1. **Users Collection**:
    1. `uid (string): Unique user ID from Firebase Authentication`
    2. `fullName (string): Full name of the user`
    3. `email (string): User's email address`
    4. `username (string): User's chosen username`
    5. `role (string): Role designation, either "user" or "organizer"`
    6. `createdAt (timestamp): When the user account was created`
2. **Events Collection**:
    1. `eventId (string): Unique event identifier`
    2. `name (string): Name of the event`
    3. `description (string): Detailed description of the event`
    4. `location (string): Where the event will take place`
    5. `date (timestamp): When the event will occur`
    6. `capacity (integer): Maximum number of attendees allowed`
    7. `organizerId (string): UID of the user who created the event`
    8. `organizerName (string): Name of the event organizer`
    9. `registeredCount (integer): Current number of registrations`
    10. `createdAt (timestamp): When the event was created`
3. **Registrations Collection**:
    1. `registrationId (string): Unique registration identifier`
    2. `eventId (string): Event associated with the registration`

3. `userId (string): User who registered for the event`
4. `userName (string): Name of the registered user`
5. `userEmail (string): Email of the registered user`
6. `registrationDate (timestamp): When the registration occurred`
7. `checkedIn (boolean): Whether user has checked in at the event`
8. `checkInTime (timestamp): When the user checked in (null if not checked in)`
9. `eventName (string): Cached event name for display purposes`
10. `eventLocation (string): Cached event location for display purposes`
11. `eventDate (timestamp): Cached event date for display purposes`

4.3.3 API Endpoints

The app leverages Firebase SDK methods through specialized repository classes rather than using raw RESTful APIs. Key operations and their implementations include:

Operation	Firebase SDK Implementation
Create Event	FirebaseFirestore.collection("events") .document().set(event.toMap())
Get Event by ID	FirebaseFirestore.collection("events") .document(eventId).get()
Get All Events	FirebaseFirestore.collection("events") .orderBy("date").get()
Get User Events	FirebaseFirestore.collection("events") .whereEqualTo("organizerId", userId).get()
Update Event	FirebaseFirestore.collection("events") .document(eventId).update(event.to

	Map())
Delete Event	FirebaseFirestore.collection("events").document(eventId).delete()
Update Registration Count	FirebaseFirestore Transactions on registeredCount field
Register User	FirebaseAuth.createUserWithEmailAndPassword(email, password)
Login User	FirebaseAuth.signInWithEmailAndPassword(email, password)
Google Sign-In	FirebaseAuth.signInWithCredential(GoogleAuthProvider.getCredential())
Logout	FirebaseAuth.signOut()
Register for Event	FirebaseFirestore.collection("registrations").document().set(registration.toMap())
Get Event Registrations	FirebaseFirestore.collection("registrations").whereEqualTo("eventId", eventId).get()
Get User Registrations	FirebaseFirestore.collection("registrations").whereEqualTo("userId", userId).get()
Check In User	FirebaseFirestore.collection("registrations").document(registrationId).update("checkedIn", true, "checkInTime", new Date())
Generate QR Code	Custom implementation using the ZXing library and registration data
Scan QR Code	Integration with camera API and ZXing for QR code scanning

4.4 Security & Privacy

Authentication & Authorization

Authentication is handled securely through **Firebase Authentication**, using an **email-password login** model. Firebase manages **session tokens** to maintain user sessions across app launches. **Role-based access control** is enforced by storing user roles within the Firestore users collection.

Data Encryption

1. **In-transit**: All communications are secured using HTTPS/TLS protocols via Firebase.

2. **At-rest**: Firebase handles encryption of stored data automatically.

3. **On-device**: Although SharedPreferences is used for caching, sensitive data is obfuscated or encrypted to prevent unauthorized access.

Compliance & Privacy Practices

Although the app is not formally governed by laws such as **GDPR** or **HIPAA**, it follows **privacy-conscious practices**. These include:

1. Requiring **user consent** during account registration

2. Providing the **option to delete user data**

3. Ensuring **minimal data collection** for only essential operations

A **privacy policy** is presented to users, clearly detailing:

1. What data is collected (e.g., name, email, registration info)

2. The purpose of data collection (event management, QR check-ins)

3. Who has access to the data (admin role)

4. Security protocols and opt-out mechanisms

4.5 Third-Party Integration

SDKs & Libraries

The following third-party tools are integrated to enhance application features:

Library / SDK	**Purpose**
Firebase SDK	Authentication, Firestore, Storage
ZXing	QR Code generation and scanning
Google Sheets API	Attendance logging via Cloud Function
SMTP	Sending email confirmations
InternetPermissions	Managing camera and internet permissions

Integration Details

1. QR Code Functionality: The ZXing library is used to generate QR codes upon event registration and to scan them during check-in. QR codes are displayed via standard `ImageView` widgets.

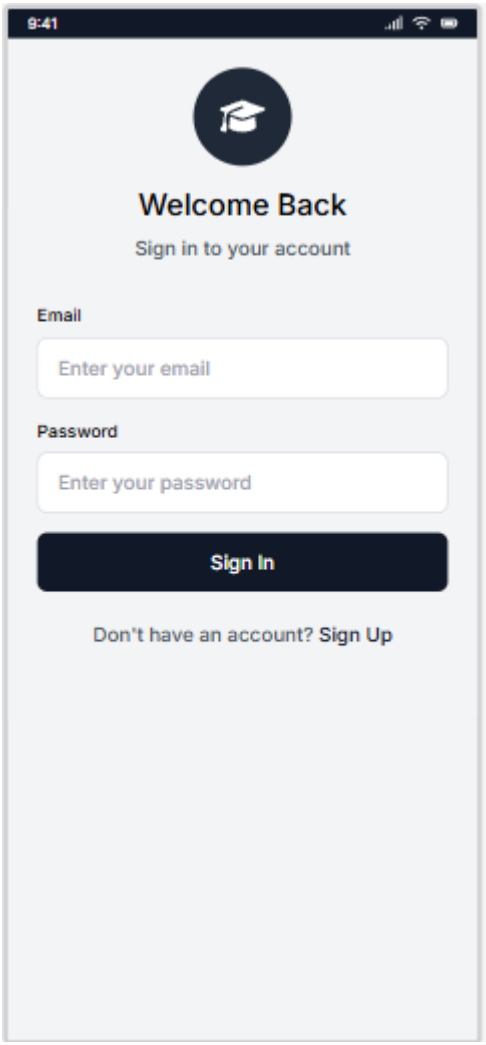
2. Attendance Logging: A Firebase Cloud Function processes scanned check-in data and writes it to a Google Sheet, serving as a backup and review tool for admins.

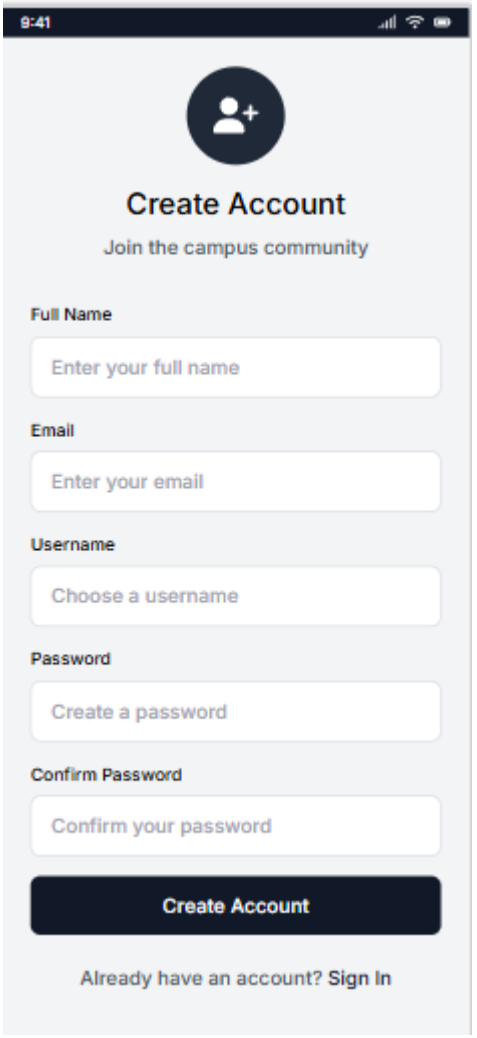
3. Email Service: Upon successful registration, Firebase Cloud Functions interact with SendGrid or SMTP to send users an email confirmation containing event details and the corresponding QR code.

5. UI/UX Design Specifications

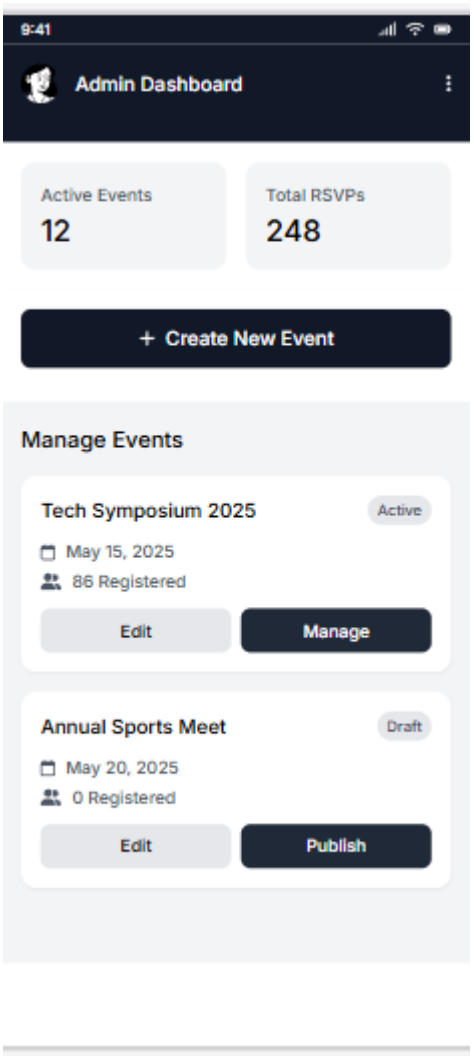
5.1 Wireframes & Mockups

Login/Signup



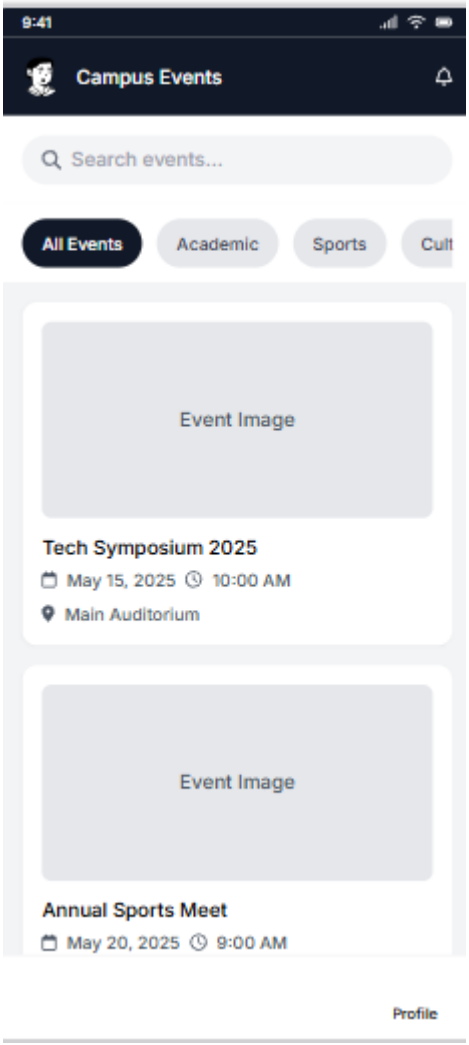


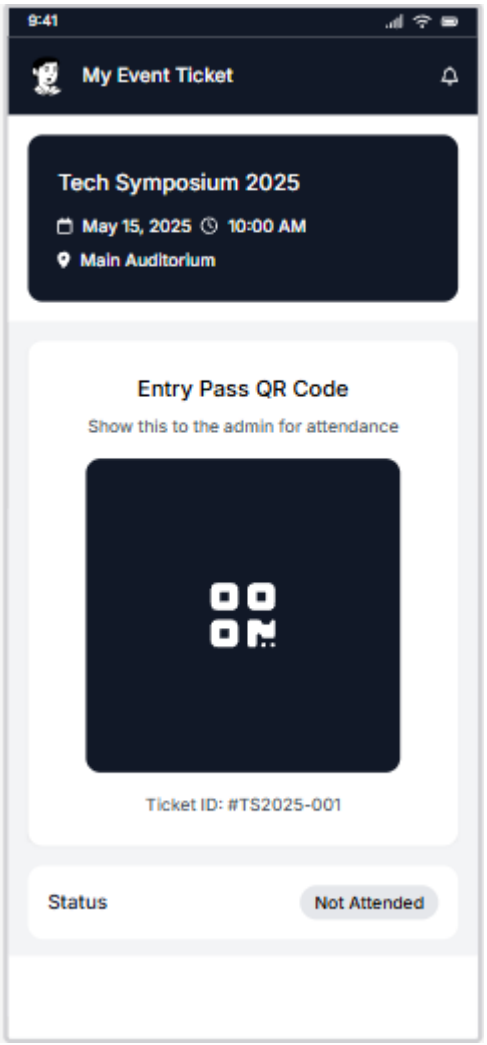
Admin Side


9:41
Admin Dashboard
Active Events
12
Total RSVPs
248
+ Create New Event
Manage Events
Tech Symposium 2025
Active
May 15, 2025
86 Registered
Edit
Manage
Annual Sports Meet
Draft
May 20, 2025
0 Registered
Edit
Publish


9:41
Scan QR Code
Tech Symposium 2025
Scanning attendees for verification
Position the QR code within the frame to scan
Recent Scans
John Smith
9:30 AM
Sarah Johnson
9:28 AM

User Side





5.2 Navigation & Flow

5.2.1 User Flow Diagrams

User flow diagrams map out possible journeys a user can take throughout the application. These include both regular users and admins. Primary flows are:

1. User Registration Flow:
   Launch App → Login Screen → [New User] Register (Enter Details) → Firebase Authentication → Landing Page
2. Standard User Event Registration Flow:

   Landing Page → [View Events Button] → Event List → Event Details → Register Button → Registration Confirmation → QR

Code Generation → Email Notification → View in My Registrations

3. User Event Attendance Flow:
   Landing Page → My Registrations → Saved Registration Details → Show QR Code at Event → Admin Scans Code → Check-In Confirmed
4. Admin/Organizer Event Creation Flow:
   Landing Page → [Create Events Button] → Create Event Form → Fill Event Details (Name, Date, Location, Capacity, Description) → Submit → Event Created in Firestore → Returns to Events List
5. Admin/Organizer Event Management Flow:

   Landing Page → [Manage Events Button] → Event List (Admin View) → → Edit Event → Update Details → Submit Changes → Delete Event → Confirm Deletion → View Attendees → Attendance List

6. Admin Check-In Process Flow:

   Landing Page → [Admin Panel Button] → Admin Panel → QR Scanner Option → Camera Access → Scan Attendee QR Code → Validate Registration → Update Check-In Status in Firestore → Show Success/Failure → Return to Scanning

7. Data Export & Synchronization Flow:

   Landing Page → [Google Sheets Sync Button] → Sync Settings → Configure API Connection → Export Options (All Events, Specific Event) → Initiate Sync → Authentication → Data Transfer to Google Sheets → Confirmation

Accessibility Guidelines

The application implements accessibility features through all user journeys:

1. Screen Reader Support: All navigation paths include content descriptions for buttons and images, making them accessible via screen readers
2. Keyboard Navigation: Users can navigate through screens using keyboard controls for those with motor limitations

3. Color Contrast: High contrast between foreground and background elements is maintained throughout all flows
4. Touch Target Size: All interactive elements in the navigation paths meet the minimum 48dp x 48dp size requirement
5. Error Recovery: Clear error messaging and simple recovery paths are provided when users encounter issues
6. Font Scaling: Text elements support Android's system font size settings for users with visual impairments

6. Deployment & Maintenance

6.1 Deployment Plan

6.1.1 Release Process

The deployment process for the mobile application will follow a structured and quality-assured pipeline to ensure stability, performance, and user satisfaction upon release. The key steps are:

1. Build Compilation: The app is built using Android Studio with the latest stable SDK version. ProGuard rules are applied for code obfuscation and size optimization.
2. Testing Phase:
    - Internal Testing: Performed by the development team and selected QA personnel.
    - Closed Alpha Testing: Released to a limited group of internal users via Google Play Console for real-world testing and bug tracking.
    - Open Beta Testing: Expanded testing group involving stakeholders and volunteers to gain broader feedback.
3. Publishing to Google Play Store:
    - App is signed with a secure release key.
    - Metadata, screenshots, and privacy policy are uploaded.
    - Compliance with Google Play Store guidelines is verified.
4. Approval and Launch:
    - After review and approval, the application is made publicly available.
    - Post-launch monitoring ensures any critical issues are addressed quickly.

7. Appendices

7.1 Glossary of Technical Terms and Acronyms

Term/Acronym	Definition
API (Application Programming Interface)	A set of rules that allows different software components to communicate.
SDK (Software Development Kit)	Tools and libraries provided for building applications on a specific platform.
MVVM (Model-View-ViewModel)	A software architectural pattern used to separate the development of the graphical user interface from the business logic.
UI/UX (User Interface / User Experience)	The overall experience of a user when interacting with an app, and the visual layout or interface design.
QR Code (Quick Response Code)	A type of matrix barcode that stores data readable by machines, often used for encoding event or user data.
ZXing (Zebra Crossing)	An open-source library used to generate and read QR codes and barcodes.
Firestore	A cloud-hosted NoSQL database from Firebase for storing and syncing data in real time.
Firebase Auth (Firebase Authentication)	A service that allows easy user authentication via email/password and other methods.
Cloud Functions	Serverless functions hosted on Firebase that execute backend code in response to events triggered by Firebase features or HTTPS requests.

SMTP (Simple Mail Transfer Protocol)	A protocol for sending emails across networks.
OAuth (Open Authorization)	A standard for token-based authorization used for granting third-party applications limited access to user data.
JWT (JSON Web Token)	A compact, URL-safe token format used for securely transmitting information between parties.

References

Firebase. (n.d.). *Firebase documentation*. Google. https://firebase.google.com/docs

Google. (n.d.). *Google Sheets API documentation*. https://developers.google.com/sheets/api

ZXing. (n.d.). *ZXing ("Zebra Crossing") barcode image processing library*. GitHub. https://github.com/zxing/zxing

Android Developers. (n.d.). *Android developer guide*. Google. https://developer.android.com/guide

SendGrid. (n.d.). *SendGrid API documentation*. Twilio. https://docs.sendgrid.com/

Oracle. (n.d.). *Java SE documentation*. https://docs.oracle.com/en/java/javase/

Material Design. (n.d.). *Material design guidelines*. Google. https://m3.material.io/

Android Developers. (n.d.). *Android UX design guidelines*. Google. https://developer.android.com/design

Android Developers. (n.d.). *Google Play Store launch checklist*. https://developer.android.com/distribute/best-practices/launch/launch-checklist