North South University

DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

PROJECT REPORT

RUN BANGLADESH

Course Information
Software Engineering
CSE327 (Section 7)
Fall 2025

Submitted by
Team-1

Student Name	**Student ID**
Saif Mohammed	2121913042
Humayra Rahman Nipa	2121128042
Sinthia Ahmed Rachona	2211916042

Submitted to
Dr. Md. Sazzad Hossain
Professor
Department of Electrical and Computer Engineering
North South University

Submission Date
December 19, 2025

Acknowledgement

First and foremost, I would like to express my sincere gratitude to the Almighty for giving me the strength, patience, and ability to complete this project and report successfully.

I am deeply indebted to my course instructor and supervisor, **Professor Dr. Md. Sazzad Hossain**, for his invaluable guidance, continuous encouragement, and constructive feedback throughout the duration of the course **CSE327 (Software Engineering)**. His profound knowledge and teaching methodology have provided me with a solid understanding of software engineering principles, which were essential in the development of the *Run Bangladesh* mobile application.

I would also like to extend my appreciation to the **Department of Electrical and Computer Engineering** for providing the necessary academic resources and laboratory facilities that facilitated the completion of this project. My sincere thanks go to my fellow team members of **Team 01**. Our collaborative efforts, brainstorming sessions, and mutual support were instrumental in overcoming the technical challenges we faced during the requirement analysis, design, and development phases.

I must also acknowledge the developers and communities behind the open-source tools and frameworks used in this project, including `React Native`, `Node.js`, `MongoDB`, `Stripe`, `Docker`, and `Jest`. Their documentation and resources were vital in bringing this application to life.

Finally, I would like to thank my family and friends for their unwavering support and patience during the busy hours of this project.

Saif Mohammed
ID: 2121913042
Team 01
Date: December 19, 2025

Contents

List of Figures

List of Tables

1 Introduction

The comprehensive digitization of athletic event management is a critical step towards modernizing the sports infrastructure in Bangladesh. This report details the design and development of *Run Bangladesh*, a full-stack mobile application engineered to facilitate the end-to-end management of marathon events. By transitioning from legacy, manual operations to a cloud-native mobile ecosystem, this project aims to address the scalability, information accessibility, and engagement challenges faced by organizers and participants alike.

1.1 Background

Marathon running has evolved from a niche activity to a mass-participation sport in Bangladesh. However, the operational backbone of these events has not kept pace with this growth. Traditionally, organizers have relied on disjointed methods—physical registration forms, cash-based fee collection, and static text-based descriptions of race locations—to manage thousands of participants. This manual approach is prone to confusion regarding venue locations and lacks the capability to provide interactive visual guidance. The *Run Bangladesh* initiative seeks to bridge this technological gap by deploying a unified mobile platform that leverages the efficiency of the MERN stack (MongoDB, Express, React Native, Node.js) to automate workflows and enhance user experience [1].

1.2 Statement of the problem

Despite the enthusiasm for marathons, the existing technical infrastructure presents several critical bottlenecks:

- **Data Fragmentation:** Participant data is often scattered across emails, spreadsheets, and third-party forms, leading to redundancy and retrieval difficulties.
- **Lack of Route Visualization:** Participants often lack clear visual information regarding the exact starting coordinates, ending points, and the total route trajectory, leading to confusion on race day.
- **Payment Inefficiency:** Manual verification of bank transfers or mobile money payments is time-consuming and error-prone.
- **Limited Engagement:** Current systems lack social features or multimedia integration, reducing the event to a mere transactional experience rather than a community event.

1.3 Objectives

The primary goal of this project is to engineer a scalable, secure, and user-centric mobile application that serves as a one-stop solution for marathon management.

General Objectives

- To digitize the entire event lifecycle, from registration to post-race analytics.
- To implement industry-standard software engineering practices, ensuring code maintainability and system reliability [2].

Specific Objectives

- **Cross-Platform Development:** Utilize **React Native (Expo)** to deploy a single codebase to both iOS and Android platforms.
- **Secure Transactions:** Integrate the **Stripe API** to facilitate instant, secure, and PCI-compliant registration payments.
- **Event Mapping:** Implement interactive map interfaces using **Expo Maps** to clearly designate starting lines, finishing points, and race venues.
- **Architectural Integrity:** Apply **SOLID Design Principles** and the **Singleton Pattern** for database connections to ensure a robust backend architecture.
- **Multimedia Management:** Integrate **Cloudinary** for the optimized storage and retrieval of event photos and user avatars.
- **Scalable Deployment:** Containerize the application using **Docker** and deploy the backend services on the **Render** cloud platform.

1.4 Overview of existing system

The current "Run Bangladesh" ecosystem relies primarily on a static informational website. Users can view event dates and rules but must perform registration via external Google Forms or physical sign-ups. Payment verification requires organizers to manually cross-check transaction IDs with bank statements. Crucially, the existing system offers no native mobile interface, meaning users cannot access location services to easily find the race starting point or view the route map on their devices.

1.5 Proposed system

The proposed solution is a dynamic, full-stack mobile application built on the **MERN** architecture.

- **Frontend:** Developed with **React Native (Expo)**, offering a native user experience with features like camera access and biometrics.
- **Backend:** A **Node.js** and **Express.js** REST API handles business logic, employing **JWT** for stateless authentication.
- **Database: MongoDB** is used for its document-oriented structure, ideal for storing complex event data and route coordinates.
- **Map Integration:** Utilization of **React Native Maps** to display static markers for the Start and End points of the marathon, helping users navigate to the correct venue.

1.6 Benefits or Significance of the project

The implementation of this system offers value to all stakeholders:

- **For Organizers:** Reduces administrative workload by approximately 40% through automated payment reconciliation and data management.

- **For Runners:** Eliminates confusion regarding race logistics by providing clear, map-based guidance for event locations.
- **For the Community:** Fosters a healthier lifestyle by removing technical barriers to entry for amateur athletes.

1.7 Scope of the project

- **In-Scope:** User authentication, event registration, payment gateway integration, static map visualization (Start/End points), volunteer management dashboard, and multimedia feeds.
- **Out-of-Scope:** Real-time GPS tracking of runners (live telemetry), hardware development (RFID chips), and offline-only functionality.

1.8 Feasibility Assessment

A rigorous assessment was conducted to determine the viability of the project across four dimensions.

1.8.1 Economic Feasibility

The project is economically viable as it relies on open-source technologies (MERN Stack) which incur no licensing fees. Operational costs are minimized by using free-tier services for development, including **MongoDB Atlas** (Database), **Render** (Hosting), and **Cloudinary** (Media). The integration of Stripe allows the platform to generate revenue to sustain these costs long-term.

1.8.2 Technical Feasibility

The project is technically feasible given the maturity of the selected stack. **React Native** provides a robust environment for mobile development, while **Node.js** excels in handling concurrent I/O operations. The team adheres to the **Singleton Design Pattern** to manage database instances efficiently and follows **SOLID principles** to maintain code quality [5].

1.8.3 Operational Feasibility

The system is designed with a focus on User Experience (UX). The interface, prototyped in **Figma**, prioritizes intuitive navigation, ensuring that non-technical organizers and volunteers can adopt the system with minimal training.

1.8.4 Schedule Feasibility

The project utilizes the **Agile Scrum** methodology. Development is divided into two-week sprints, allowing for iterative delivery and testing. With the use of **Docker** to standardize development environments, the team is well-positioned to meet the semester deadlines as outlined in the project Gantt chart [3].

2 Literature Review

The development of the *Run Bangladesh* application is situated at the intersection of modern mobile computing, geospatial visualization, and secure enterprise architecture. This section reviews existing literature, architectural paradigms, and industry standards to establish the theoretical and practical foundations for the proposed solution.

2.1 Detailed Description

Evolution of Event Management Systems

Historically, marathon management relied on monolithic, relational database systems (RDBMS) paired with static web interfaces. While sufficient for basic registration, these legacy systems struggle with the high concurrency demands of modern race days and lack native mobile capabilities. Research by *Gao et al.* indicates a paradigm shift towards mobile-cloud collaborative approaches, where heavy computation is offloaded to the cloud while the mobile device acts as a simplified client [4]. The *Run Bangladesh* project adopts this modern approach, transitioning from a static web model to a dynamic mobile ecosystem.

Cross-Platform Development Frameworks

The choice of development framework significantly impacts maintainability and reach. Traditional native development (Swift/Kotlin) offers performance but requires maintaining two distinct codebases. **React Native**, specifically within the **Expo** ecosystem, was selected for this project as it utilizes a "Learn Once, Write Anywhere" methodology. It allows for the rendering of native UI components using JavaScript, ensuring near-native performance while reducing development time by approximately 40% compared to separate native streams [7].

Geospatial Visualization

Effective marathon management requires clear communication of race logistics. While some applications employ continuous real-time tracking, this approach often consumes excessive battery power and data bandwidth. For general event logistics, static visualization of key points of interest (POIs) is often sufficient and more efficient. By utilizing **React Native Maps** within the Expo environment, the application can render native Apple Maps or Google Maps views to display the Start Point, End Point, and Route Path. This allows participants to navigate to the venue using standard navigation tools without the overhead of live telemetry [8].

Architectural Design Patterns

To ensure the system's longevity and scalability, the software architecture adheres to established engineering principles:

- **SOLID Principles:** The backend adheres to the Single Responsibility and Dependency Inversion principles to decouple the business logic from the database layer, facilitating easier testing and future extensions [5].

- **Singleton Pattern:** The application implements the **Singleton Design Pattern** for the MongoDB connection. This ensures that a single database connection instance is shared across the entire application lifecycle, preventing connection pool exhaustion under heavy load [6].

Security and Stateless Authentication

In distributed client-server architectures, maintaining session state on the server (stateful authentication) impedes scalability. The industry standard has shifted to stateless authentication using **JSON Web Tokens (JWT)**. This allows the *Run Bangladesh* API to verify requests without querying the database for session data on every hit [9]. Furthermore, financial security is managed by offloading sensitive transaction processing to the **Stripe API**, ensuring compliance with PCI-DSS standards without the burden of storing credit card information locally [10].

3 Methodology

This section outlines the systematic approach adopted for the development of the *Run Bangladesh* application. The project follows the **Agile Software Development Life Cycle (SDLC)**, specifically utilizing the Scrum framework. This approach was selected to facilitate iterative development, allowing for continuous feedback integration and rapid adaptation to technical constraints—such as the pivot from real-time tracking to static map visualization—throughout the academic semester [2, 12].

3.1 Data Collection

Data collection was a critical initial step to ensure the system requirements aligned with the actual needs of marathon organizers and participants. The process was categorized into two distinct phases:

Primary Data Collection

Primary data was gathered through direct interaction with key stakeholders identified in the Software Requirements Specification (SRS) [1].

- **Requirement Elicitation:** Functional requirements were derived by analyzing specific "Use Cases" (e.g., UC1 Registration, UC4 Route Viewing).
- **Stakeholder Analysis:** Interviews were conducted to understand the specific pain points of organizers, such as the confusion regarding race starting points, which led to the requirement for static map visualization.

Secondary Data Collection

Secondary data involved analyzing existing market solutions and technical documentation.

- **Competitor Analysis:** A comparative study of platforms like *Strava* and *Eventbrite* was conducted to understand standard UI patterns for event ticketing and route display.
- **Technical Research:** Documentation for **Expo Maps** and **Stripe API** was reviewed to determine the feasibility of integrating native map views and secure payments within a React Native environment [10].

3.2 Project Plan

The project plan is structured around five key phases of the Agile SDLC. This iterative approach allows for flexibility in handling changes during the development cycle [3].

1. **Phase 1: Planning and Analysis**
   This phase focused on defining the project scope, creating the SRS, and selecting the **MERN** technology stack. Feasibility studies regarding the use of **Expo** for rapid mobile development were concluded here.

2. **Phase 2: System Design**
   The architectural blueprint was created, including the Schema Design for **MongoDB**. High-fidelity UI prototypes were developed using **Figma** to visualize the user experience, particularly the event details and map screens, before coding began [13].

3. **Phase 3: Implementation (Sprints)**
   The core coding phase was divided into 2-week sprints:

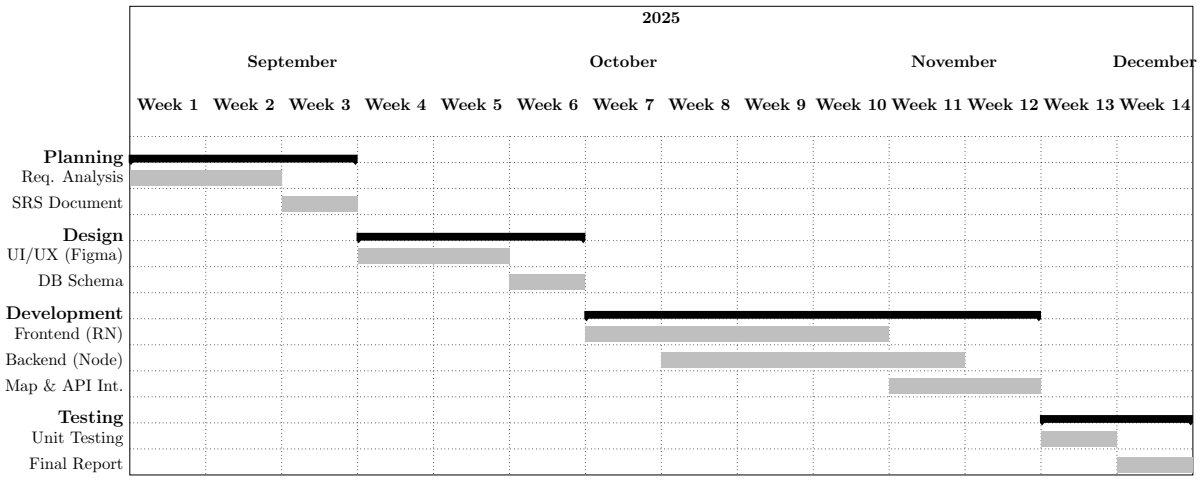
   - *Sprint 1:* Frontend setup (React Native/Expo) and User Authentication (JWT).
   - *Sprint 2:* Backend API development (Node.js/Express) and Database connectivity.
   - *Sprint 3:* Integration of **Expo Maps** for venue location and **Stripe** for payments.

4. **Phase 4: Testing**
   Unit testing was performed using the **Jest** framework to validate individual components [14]. API endpoints were manually tested using **Postman** to ensure correct data retrieval from MongoDB.

5. **Phase 5: Deployment and Documentation**
   The final phase involved containerizing the application using **Docker** for consistent deployment [15]. The backend was deployed to **Render**, and the final technical report was compiled.

3.3 Work Plan (Gantt Chart)

The following Gantt chart illustrates the timeline of the project, spanning a duration of approximately 14 weeks.



Work Plan (Gantt Chart)

4 Proposed System

4.1 Proposed System Overview

The *Run Bangladesh* application is a cloud-native mobile solution designed to centralize the management of marathon events. Unlike the legacy system, which relies on disjointed manual processes, the proposed system operates on the **MERN** stack (MongoDB, Express.js, React Native, Node.js). The application adopts a **Client-Server architecture** where the frontend is developed using **React Native (Expo)** to ensure cross-platform compatibility on iOS and Android. The backend, built on **Node.js**, serves as a RESTful API provider, handling business logic, authentication, and data retrieval. By integrating third-party services like **Stripe** for payments and **Expo Maps** for route visualization, the system offers a cohesive, automated experience for both organizers and participants [1].

4.2 Functional Requirements

The functional requirements define the specific behaviors and functions the system must support. These are categorized by feature set and user role:

Participant Features

- **Registration Authentication:** Users shall be able to create accounts and log in using secure credentials (JWT-based authentication).
- **Event Discovery:** Users shall be able to browse a feed of upcoming marathons, filtering by date or location.
- **Profile Management:** Users shall be able to update personal details and upload profile avatars (stored via **Cloudinary**).

Organizer Features

- **Event Management:** Organizers shall be able to create, edit, and publish new marathon events, including setting registration caps and fees.
- **Dashboard Analytics:** The system shall provide a dashboard displaying real-time metrics on total registrations, revenue, and participant demographics.

Payment Gateway Integration

- **Secure Transactions:** The system shall process registration fees securely using the **Stripe API**.
- **Payment Intents:** The system must generate a unique payment intent for every registration attempt to prevent duplicate charges.

Event Map Location Services

- **Static Route Visualization:** The app shall display an interactive map marking key locations: Start Point, Finish Line, and Hydration Stations.

Volunteer Coordination

- **Task Assignment:** Organizers shall be able to create tasks (e.g., "Water Station A", "First Aid") and assign them to registered volunteers.
- **Volunteer Roster:** The system shall maintain a visible list of all active volunteers for a specific event.

4.3 Non-functional Requirements

These requirements define the quality attributes, performance constraints, and operational standards of the system [2].

4.3.1 Accessibility

The mobile application shall adhere to **WCAG 2.1 (Level AA)** guidelines. This includes high-contrast modes for visually impaired users and screen-reader compatibility (e.g., VoiceOver/TalkBack) for all navigational elements.

4.3.2 Usability

The user interface, designed in **Figma**, prioritizes a low learning curve. The system shall ensure that a new user can complete the registration process in fewer than five clicks. Consistency in color schemes and button placement shall be maintained across all screens to reduce cognitive load.

4.3.3 Documentation

Comprehensive documentation shall be provided, including:

- **User Manuals:** Step-by-step guides for runners and organizers embedded within the app's "Help" section.
- **API Documentation:** Technical documentation for backend endpoints (Swagger/Postman collections) to facilitate future development.

4.3.4 Hardware and Software Considerations

- **Client Hardware:** The app requires a smartphone with GPS capabilities and a camera (for profile uploads).
- **Client OS:** Compatible with Android 10.0+ and iOS 14.0+.
- **Server:** The backend requires a Node.js runtime environment and connects to a MongoDB cloud cluster.

4.3.5 Quality Issues

The system focuses on reliability and correctness. **Automated Unit Testing** (using **Jest**) covers 80% of the core business logic to prevent regression bugs. The application utilizes the **Singleton Design Pattern** for database connections to ensure resource stability [6].

4.3.6 Security Issues

- **Data Transmission:** All data transmission between client and server is encrypted using **HTTPS/TLS 1.2+**.
- **Authentication:** Stateless authentication is implemented using **JSON Web Tokens (JWT)**; passwords are hashed using **bcrypt** before storage.
- **Payments:** No sensitive financial data is stored on the server; all transactions are tokenized and processed via **Stripe** (PCI-DSS compliant) [10].

4.3.7 User Interface and Human Factors

The UI follows standard Material Design (Android) and Human Interface Guidelines (iOS) principles. Buttons are sized appropriately for touch targets (minimum 44x44 points) to accommodate users running or moving.

4.3.8 Performance Characteristics

- **Latency:** API response time shall be under 200ms for 95% of requests.
- **Load Time:** The application initial load time (Time to Interactive) shall not exceed 2 seconds on 4G networks.
- **Concurrency:** The Node.js backend shall support up to 500 concurrent users during peak registration windows.

4.3.9 Error Handling and Extreme Conditions

The system implements graceful degradation. In the event of network failure, the user receives clear, non-technical error messages (e.g., "No Internet Connection") rather than raw stack traces. Standard HTTP error codes (400, 404, 500) are handled globally to prevent app crashes.

4.3.10 System Modification

The architecture follows **SOLID Principles**, ensuring modularity. Features are decoupled (e.g., Payment logic is separate from User logic), allowing developers to update specific modules without affecting the entire system. Docker containerization further simplifies the deployment of updates [5].

4.3.11 Feasibility Study

- **Technical:** The use of the MERN stack is feasible as it relies on open-source, well-documented libraries supported by a large community.
- **Economic:** The project is cost-effective, utilizing free tiers of Render (hosting) and MongoDB Atlas, making it sustainable for non-profit marathon events.

4.4 System Models

System modeling is a crucial phase in the software engineering process that translates textual requirements into visual representations. For *Run Bangladesh*, Unified Modeling Language (UML) diagrams are employed to depict the structural and behavioral aspects of the system. These models serve as a blueprint for developers and stakeholders, ensuring a shared understanding of the system's architecture and user interactions before the implementation phase begins.

4.4.1 Use Case Diagram

The Use Case Diagram provides a high-level view of the system's functionality by illustrating the interactions between external entities (Actors) and the system's functional units (Use Cases). It defines the scope of the application and identifies the primary goals of each user role.

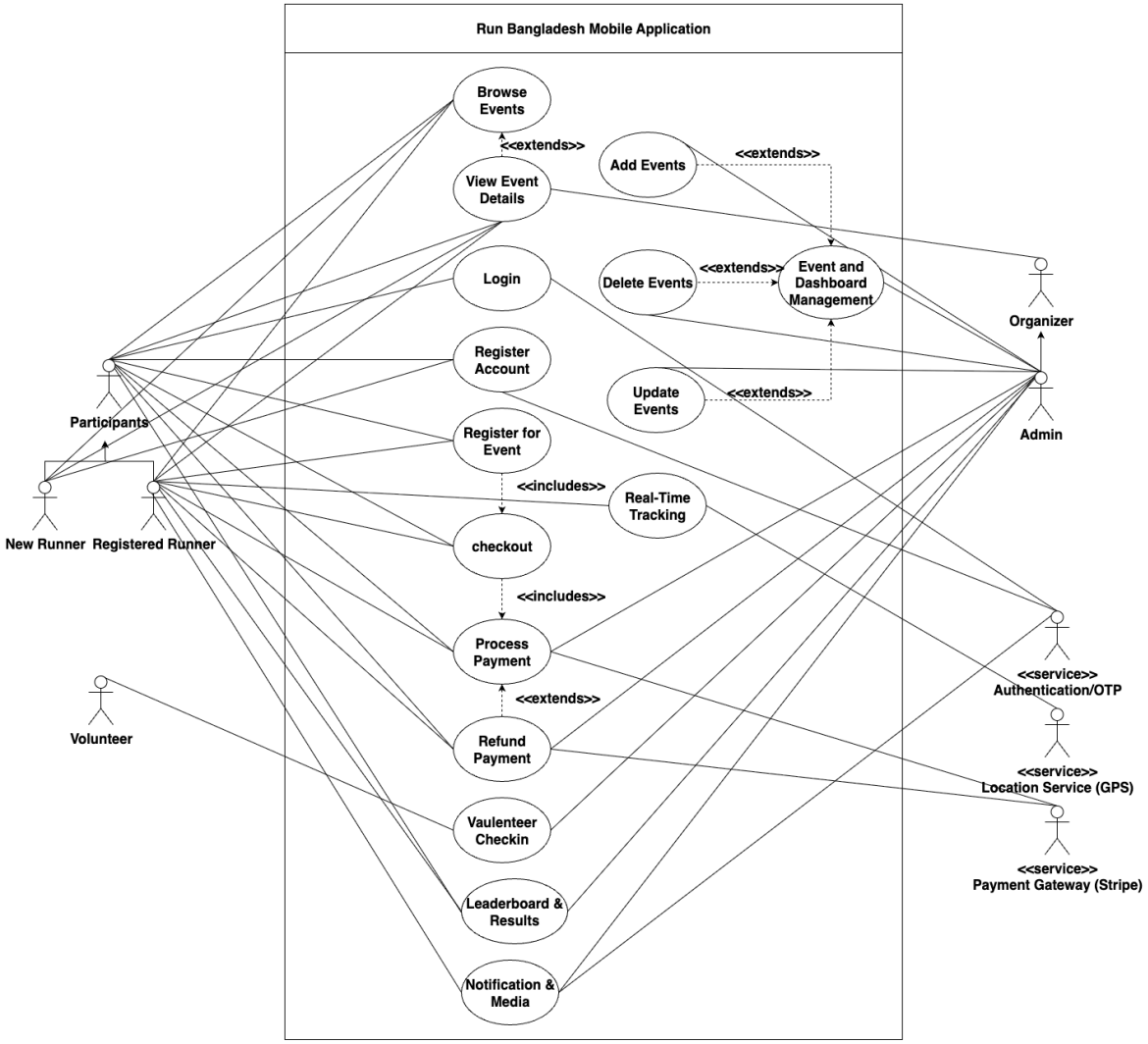


Figure 1: Use Case Diagram for Run Bangladesh Mobile Application

For the *Run Bangladesh* application, the diagram identifies three primary actors:

- **Participant (Runner):** The primary end-user who interacts with the system to browse events, register, pay fees via Stripe, and view the route map.
- **Organizer (Admin):** The administrative user responsible for creating events, managing volunteer assignments, and viewing financial analytics.
- **System (Backend/API):** The automated entity that handles background processes such as payment verification, notification dispatch, and leaderboard generation.

Figure 10 below visually represents these relationships, demonstrating how the Participant initiates the "Register for Event" use case, which subsequently triggers the "Process Payment" include-case handled by the System.

4.4.2 Use Case Description

The following tables provide a detailed textual description of the core use cases identified in the system design. These descriptions outline the preconditions, postconditions, and the specific flow of events required to successfully complete each function.

Table 1: Use Case Description: User Registration and Login

Use Case ID	UC-01
Use Case Name	User Registration and Login
Actors	New Runner, Registered Runner, Authentication Service
Preconditions	User has installed the mobile app and has an active internet connection.
Normal Flow	1. User opens the app and selects "Register" or "Login". 2. System prompts for email/phone and password. 3. System sends an OTP for authentication. 4. User enters OTP; system verifies and grants access.
Postconditions	User is authenticated and lands on the dashboard.
Alternative Flows	**A1: Invalid OTP** 4a. System prompts retry or resend OTP. **A2: Forgot Password** 2a. User launches password reset flow.
NFRs	Secure auth (OTP), TLS in transit, encryption at rest.

Table 2: Use Case Description: Event Browsing and View Details

Use Case ID	UC-02
Use Case Name	Event Browsing and View Details
Actors	Participant, Registered Runner
Preconditions	User is logged in (or guest); events exist in the system.
Normal Flow	1. User opens "Browse Events". 2. System lists upcoming/past events with filters (date, city). 3. User selects an event to view details (route map, fees, rules). 4. User may proceed to registration.
Postconditions	Event details displayed; user can continue to register.
Alternative Flows	**A1: No Events Found** 2a. System shows empty state with "Notify me". **A2: Network Error** 2b. System retries or shows offline cached list.
NFRs	Fast list rendering, responsive filtering, offline fallback.

Table 3: Use Case Description: Event Registration and Payment

Use Case ID	UC-03
Use Case Name	Event Registration and Payment
Actors	Participant, Payment Gateway (Stripe)
Preconditions	User is authenticated; event is open; payment method available.
Normal Flow	1. User selects event category (e.g., 5K, 10K). 2. System collects participant info (age, emergency contact). 3. User proceeds to Checkout. 4. Stripe processes payment securely. 5. System confirms registration and issues QR bib/ID.
Postconditions	Registration recorded; receipt available in "My Events".
Alternative Flows	**A1: Payment Failure** 4a. Show reason; allow retry/change method. **A2: Category Full** 1a. Offer waitlist option or another category.
NFRs	PCI-compliant Stripe integration, reliability at peak load.

Table 4: Use Case Description: Static Location Map

Use Case ID	UC-04
Name	Marathon Start & End Verification
Actors	Participant, GPS Service
Preconditions	GPS enabled; Start/End coordinates defined.
Normal Flow	1. App validates GPS location at Start Point; Timer starts. 2. Participant completes course. 3. App validates GPS location at End Point; Timer stops. 4. Total duration logged and uploaded.
Postconditions	Race completion recorded.
Alt. Flows	**A1: Location Invalid:** User not inside geofence; action blocked. **A2: Weak Signal:** User prompted to improve GPS visibility.
NFRs	Geofence accuracy (< 10m), Anti-spoofing.

Table 5: Use Case Description: Event and Dashboard Management

Use Case ID	UC-05
Use Case Name	Event Management
Actors	Organizer, Admin
Preconditions	Organizer authenticated with proper role.
Normal Flow	1. Organizer opens Dashboard. 2. Creates/updates event (title, date, route, fees). 3. Assigns volunteers; reviews registrations. 4. Publishes updates; monitors analytics.
Postconditions	Event configuration saved and reflected to users.
Alternative Flows	**A1: Invalid Input** 2a. System shows field-level validation errors.
NFRs	RBAC, audit logging, strong consistency for writes.

Table 6: Use Case Description: Volunteer Coordination

Use Case ID	UC-06
Use Case Name	Volunteer Check-In
Actors	Volunteer, Organizer
Preconditions	Volunteer assigned to an event/shift; authenticated.
Normal Flow	1. Volunteer opens "My Assignments". 2. Reviews duty location/time and instructions. 3. Taps "Check In"; status visible to Organizer.
Postconditions	Attendance recorded; check-in timestamp stored.
Alternative Flows	**A1: No Assignment** 1a. Contact Organizer. **A2: Late Arrival** 3a. Flagged for dashboard alert.
NFRs	Low-friction UI, reliable sync, role-based permissions.

4.4.3 Class Diagram

The Class Diagram illustrates the static structure of the Run Bangladesh system, defining the core classes, their attributes, and the relationships between them. As shown in Figure 2, the system is built around a central `User` class, which serves as a parent class for specific roles: `Participant` and `Organizer`.

Key architectural decisions depicted include:

- **Inheritance:** Both Participants and Organizers inherit common credentials (ID, email, password hash) from the `User` class.
- **Associations:** The `Event` class has a one-to-many relationship with `Registration`, allowing multiple participants to register for a single marathon.
- **Payment Integration:** The `Payment` class is linked to `Registration`, encapsulating transaction details (Stripe ID, status, amount) to ensure financial accountability.

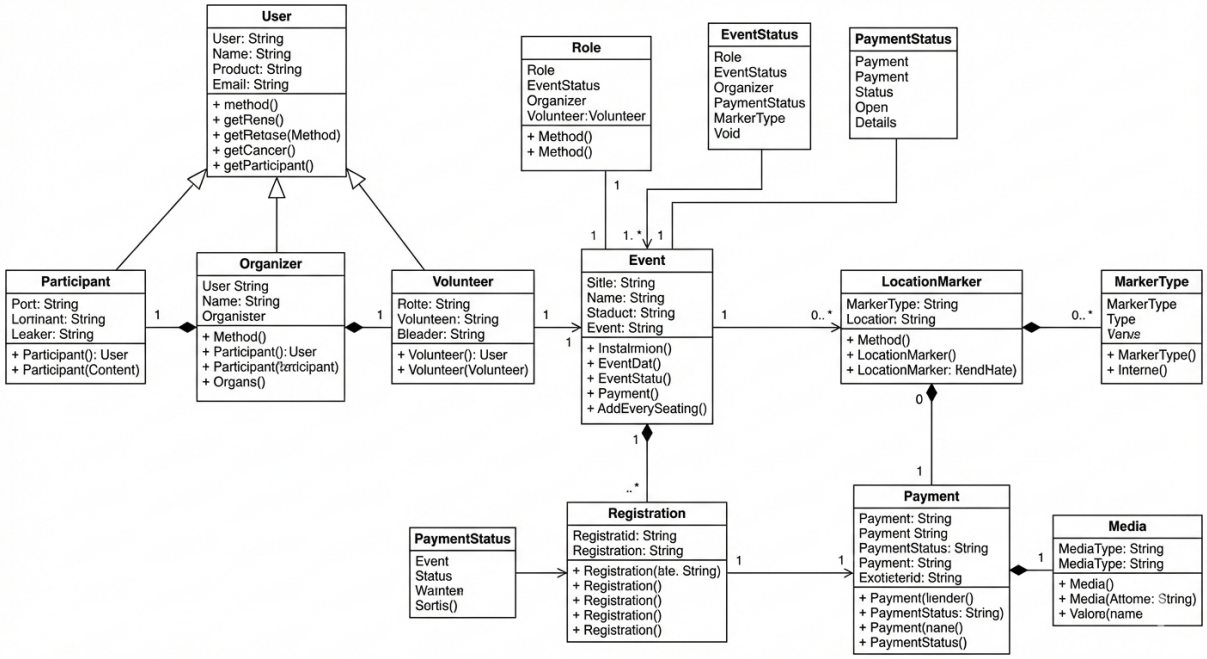


Figure 2: UML Class Diagram: Run Bangladesh

4.4.4 Sequence Diagrams

Sequence diagrams model the dynamic interactions between system objects over time.

- **Participant Event Registration & Payment:** Figure 3 illustrates the sequence of operations when a user registers for a marathon. The *Participant* initiates the request via the frontend. The *System* first validates slot availability with the Database. Upon confirmation, a payment intent is created via the *Stripe Gateway*. The registration is finalized and stored only after a successful transaction callback is received.
- **Organizer Creates New Event:** Figure 4 details the administrative flow for publishing a new race. The *Organizer* submits event details (name, location, date).

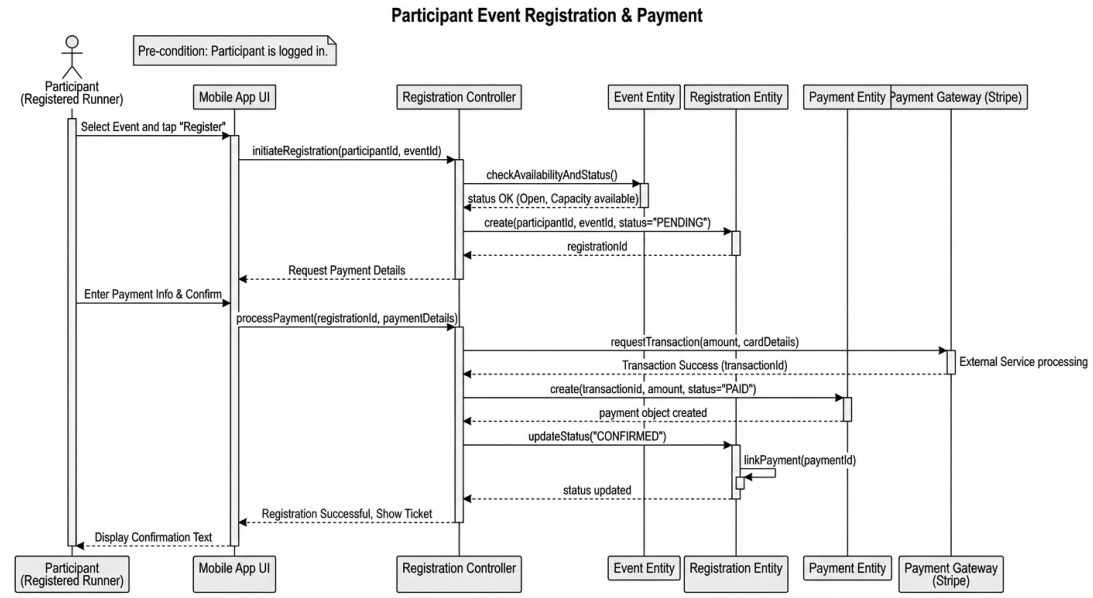


Figure 3: Sequence Diagram: Participant Event Registration & Payment

The *Backend Controller* validates these inputs (e.g., ensuring the date is in the future) and saves the event to the *MongoDB Database*. Finally, a confirmation response is sent back to the client.

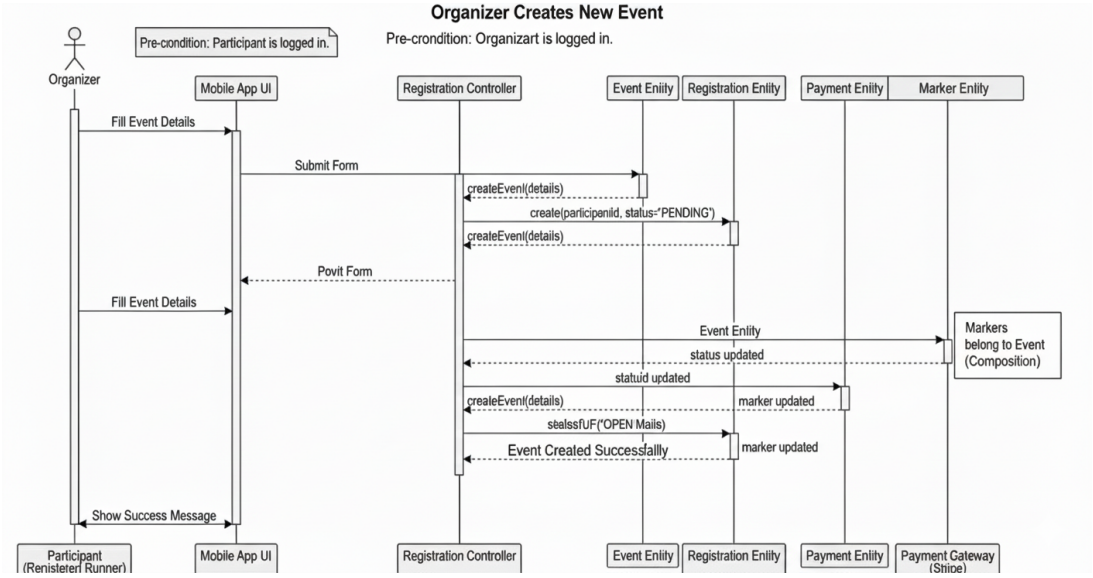


Figure 4: Sequence Diagram: Organizer Creates New Event

4.4.5 State Chart Diagrams

State chart diagrams describe the lifecycle of specific objects within the system as they transition between states in response to events.

- **Registration Lifecycle:** Figure 5 depicts the states of a registration object. It begins as *Pending* during the payment phase. If payment fails or times out, it transitions to *Cancelled*. A successful transaction moves it to *Confirmed*. Once the race is verified as run, the state updates to *Completed*.

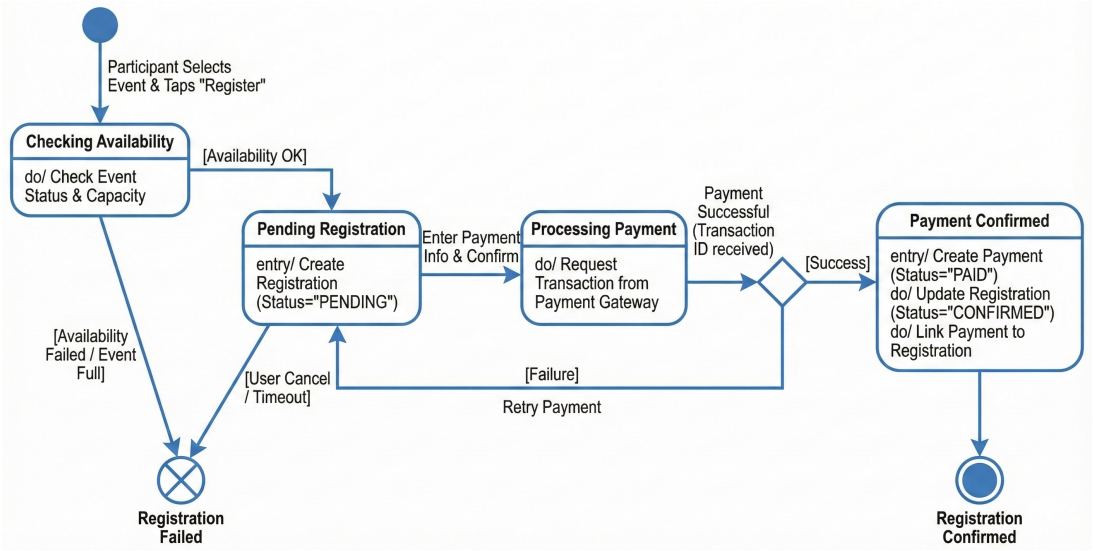


Figure 5: State Chart Diagram: Participant Event Registration & Payment

- **Event Lifecycle:** Figure 6 tracks the status of a marathon event. Initially created as a *Draft*, it transitions to *Open* when published by the organizer. It becomes *Closed* when the max capacity is reached or the registration deadline passes. Finally, after the event concludes, it is *Archived*.

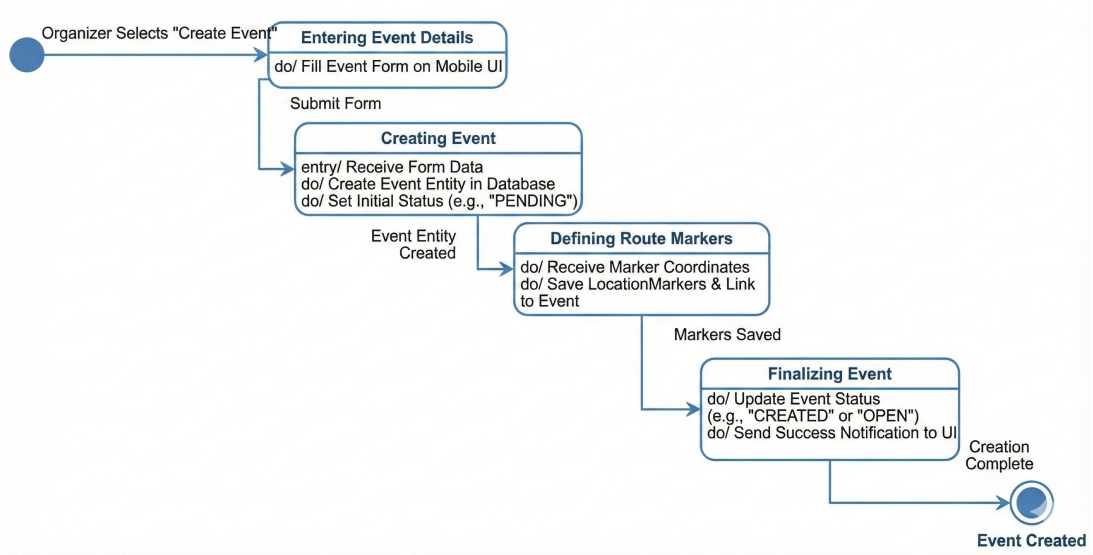


Figure 6: State Chart Diagram: Organizer Creates New Event

4.4.6 Activity Diagrams

Activity diagrams map the workflow of business processes, highlighting decision nodes and parallel activities.

- **Organizer Workflow: Creating an Event:** Figure 7 shows the decision logic for creating an event. The system checks if all required fields are valid and if the selected location is recognized. If validation fails, the flow loops back

to the input form. If successful, the system simultaneously saves the event data and establishes the geofencing parameters.

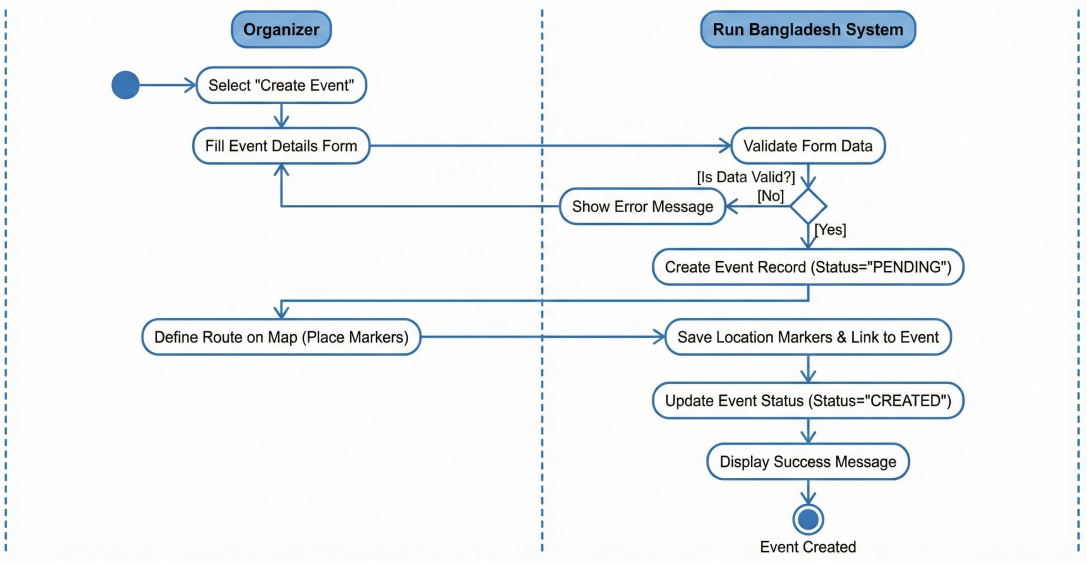


Figure 7: Activity Diagram: Organizer Creates New Event

– **Participant Workflow: Registration:** Figure 8 outlines the user journey. The flow begins with selecting an event. A decision node checks for vacancy ("Is Full?"). If slots are available, the user proceeds to payment. A subsequent check verifies the transaction status; success triggers the generation of a QR code, while failure prompts a retry.

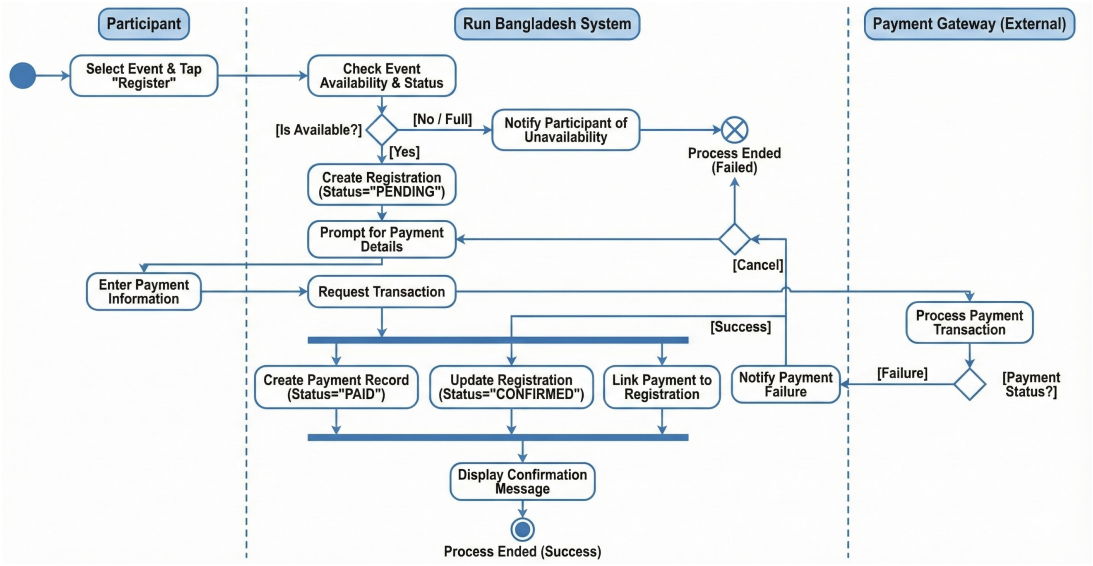


Figure 8: Activity Diagram: Participant Event Registration & Payment

5 Design Contents

5.1 Introduction

The design phase serves as the bridge between the requirement analysis and the final system implementation [3]. This section details the architectural decisions, structural decomposition, and interface strategies adopted for the Run Bangladesh application. The primary objective of this design is to create a blueprint that ensures the system is scalable, maintainable, and robust enough to handle high-concurrency traffic during marathon registration periods. The design follows a Client-Server architecture optimized for mobile environments, distinctively separating the Mobile Client (Front-end) from the Backend Server [4]. In this framework, the mobile application acts as the client, responsible for the user interface and presentation logic, while the server handles the core business logic, data persistence, and API processing, ensuring secure and efficient communication over the network.

5.2 Proposed System Architecture

The system utilizes a **Three-Tier Client-Server Architecture** powered by the MERN stack [2]. This separation of concerns ensures that the frontend and backend can be developed, tested, and scaled independently [5].

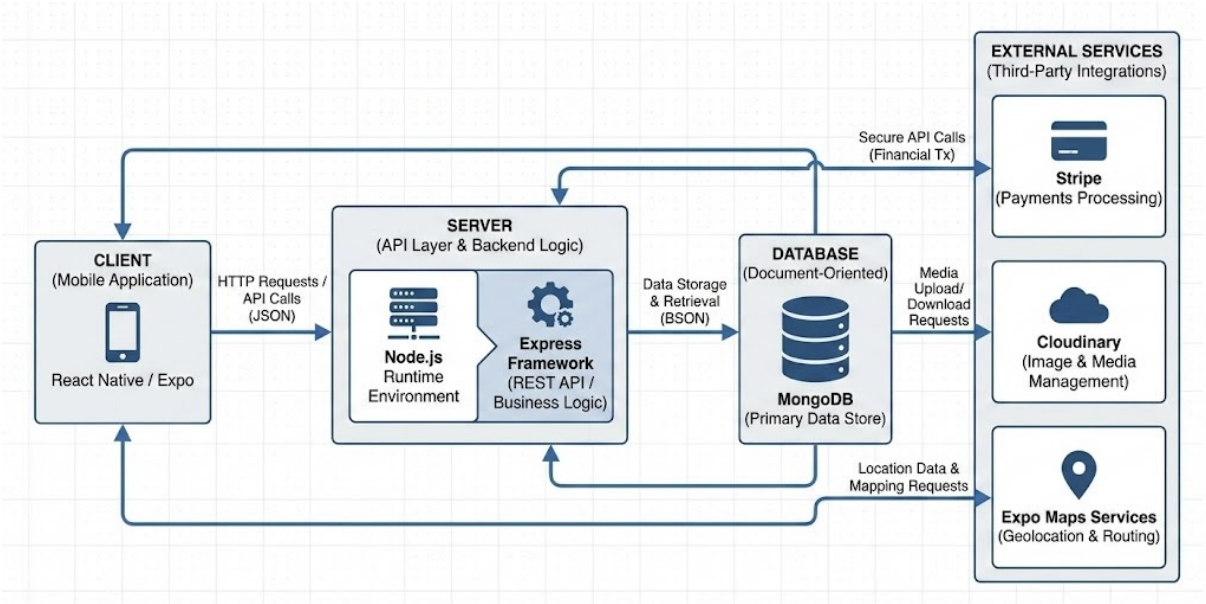


Figure 9: Proposed System Architecture for Run Bangladesh

1. **Presentation Layer (Client):** Built with **React Native (Expo)** [7], this layer runs on the user's mobile device (iOS/Android). It handles user interactions, renders the UI, and communicates with the backend via RESTful API calls over HTTPS. It manages local state but relies on the server for persistent data.

2. **Application Layer (Server):** Hosted on **Render**, this layer consists of a **Node.js** runtime with the **Express.js** framework. It acts as the central controller, processing incoming API requests, executing business logic (e.g., verifying registration eligibility), and enforcing security policies such as **JWT authentication** [9].
3. **Data Layer (Database):** The system uses **MongoDB Atlas**, a cloud-hosted NoSQL database. It stores all persistent data, including user profiles, event details, and payment transaction logs, in a flexible JSON-like document format.

5.3 Subsystem Decomposition

To manage complexity, the system is decomposed into distinct, loosely coupled modules. Each module encapsulates a specific set of functionalities and interacts with others through well-defined interfaces.

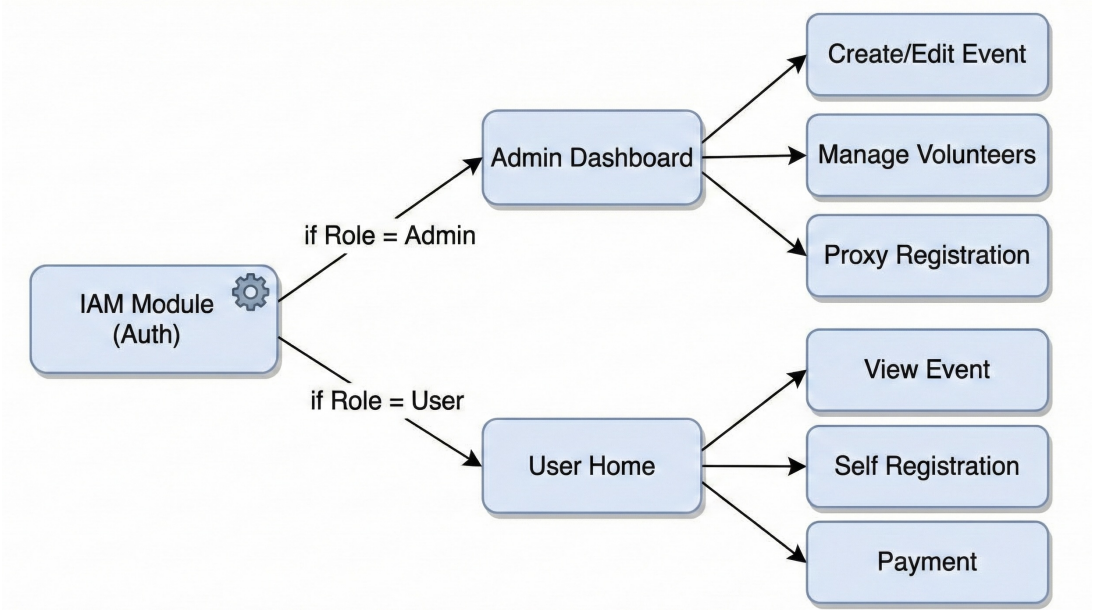


Figure 10: Subsystem Decomposition of Run Bangladesh Application

5.3.1 List of Modules

The application is divided into five core modules:

- **User Management Module:** Handles user sign-up, login (via JWT), password recovery, and profile management. It includes the logic for role-based access control (Admin vs. Runner).
- **Event Management Module:** Allows organizers to create, update, and delete marathon events. For runners, this module handles event browsing, filtering, and retrieving detailed route maps.
- **Payment Processing Module:** A secure subsystem that interfaces with the **Stripe API**. It manages payment intent creation, webhook handling for transaction confirmation, and receipt generation.

- **Volunteer Management Module:** Facilitates the assignment of volunteers to specific checkpoints. It provides a specialized view for volunteers to check in and view their duties.
- **Notification Service:** An asynchronous module responsible for dispatching push notifications regarding race updates, payment confirmations, or emergency alerts.

5.4 System Layout

The system layout describes the deployment topology and network interactions. The *Run Bangladesh* application is entirely cloud-native:

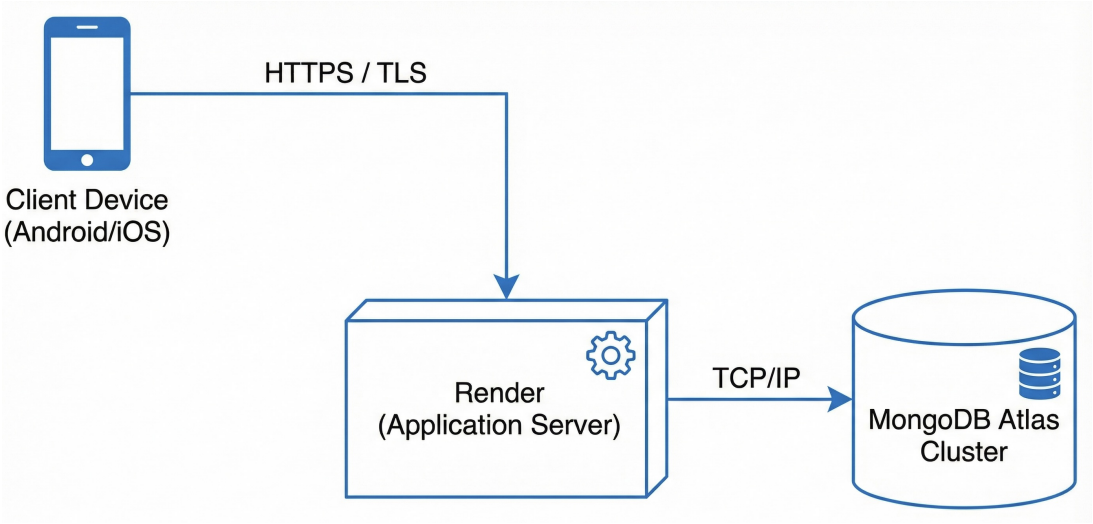


Figure 11: System Layout of Run Bangladesh Application

- **Client Side:** The mobile application acts as a "Fat Client," performing initial data validation and rendering maps using the device's native GPS hardware.
- **API Gateway:** All traffic is routed through a central API endpoint (e.g., `api.runbangladesh.com`). Nginx is used as a reverse proxy within the Render environment to handle load balancing.
- **External Integrations:**
  - **Stripe:** For payment processing (PCI-DSS compliant).
  - **Cloudinary:** For offloading heavy media storage (user avatars, race photos) to a Content Delivery Network (CDN).
  - **Google/Apple Maps:** Accessed via the Expo Maps SDK for rendering route overlays.

5.5 User Interface Design

The User Interface (UI) is designed with a "Mobile-First" philosophy, prioritizing touch interactions and readability in outdoor environments. The application features distinct workflows for Administrators (Organizers) and Participants to ensure role-specific usability.

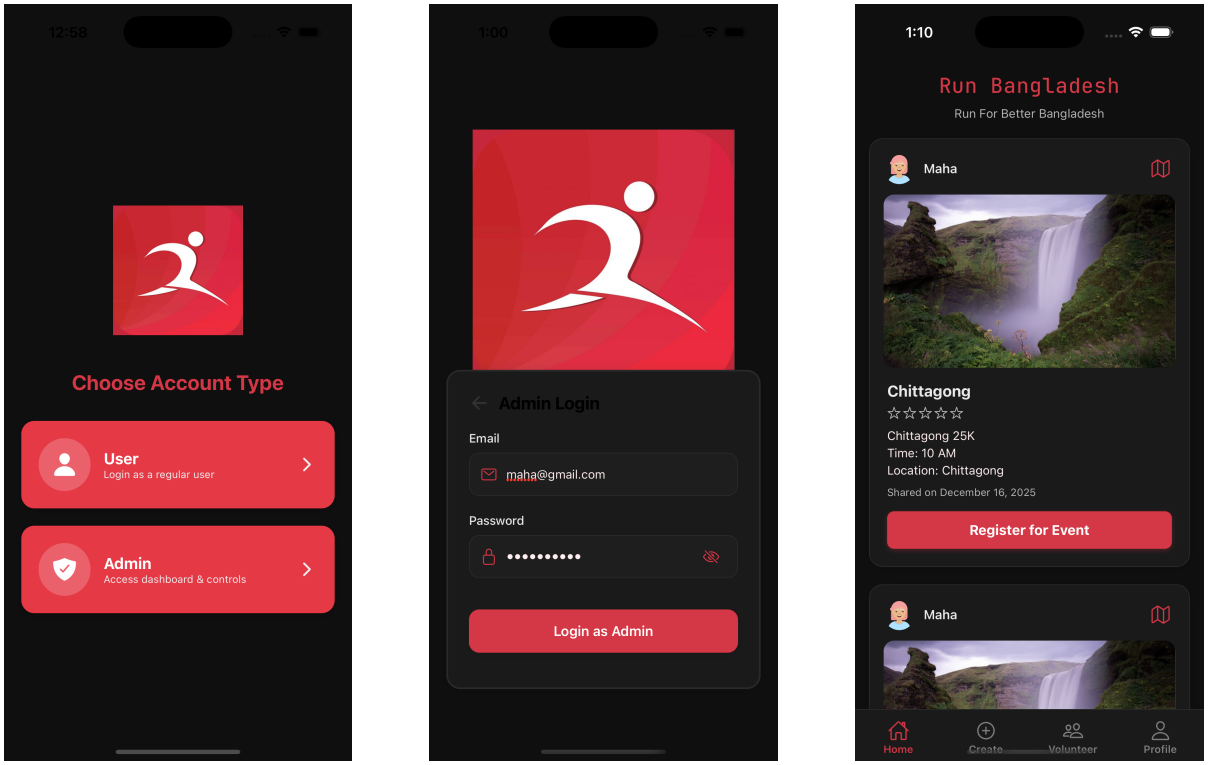


Figure 12: Admin UI: Landing, Authentication, and Management Feed

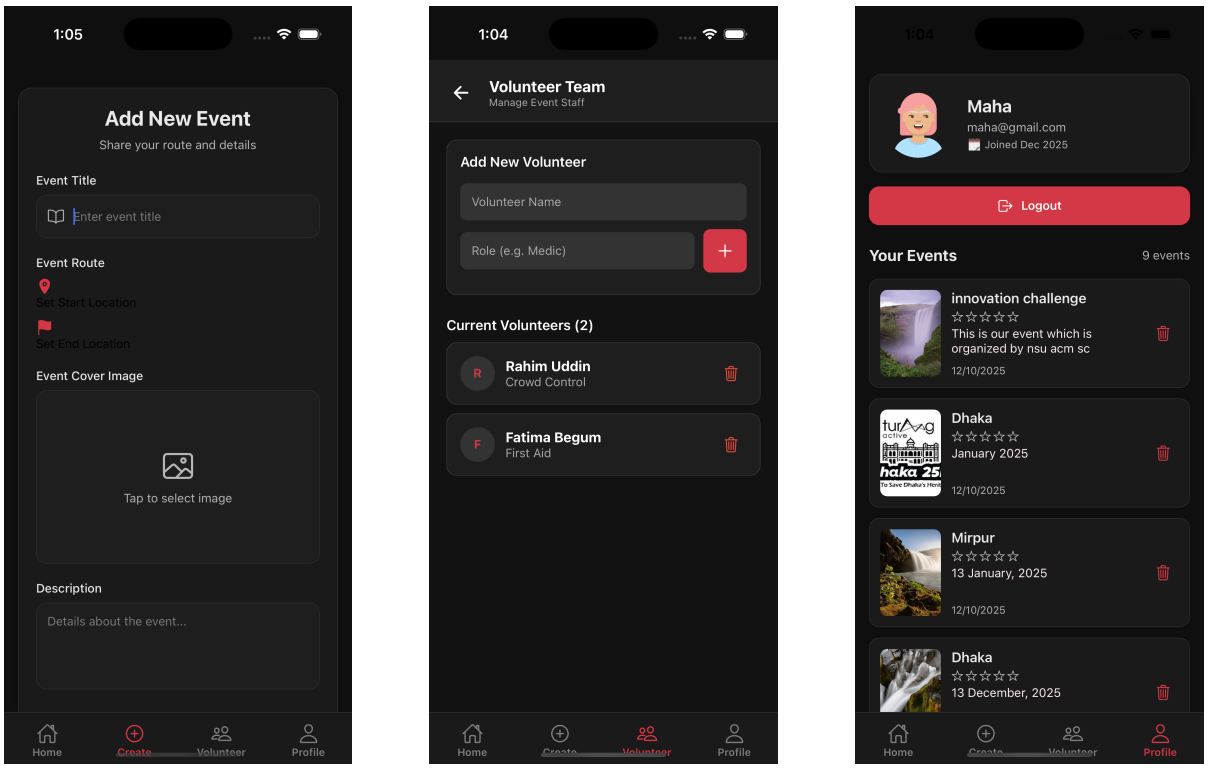


Figure 13: Admin UI: Event Creation, Volunteer Management, and Profile

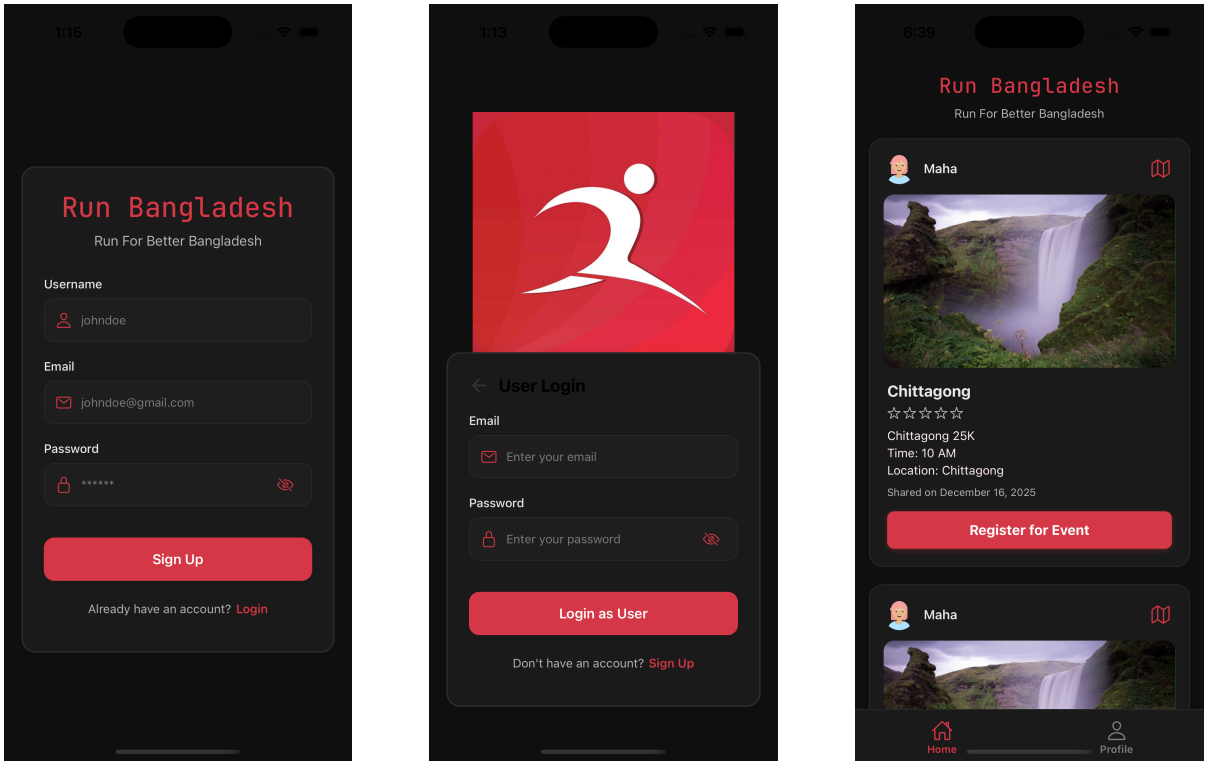


Figure 14: Participant UI: SignUp, Login, and Event Discovery

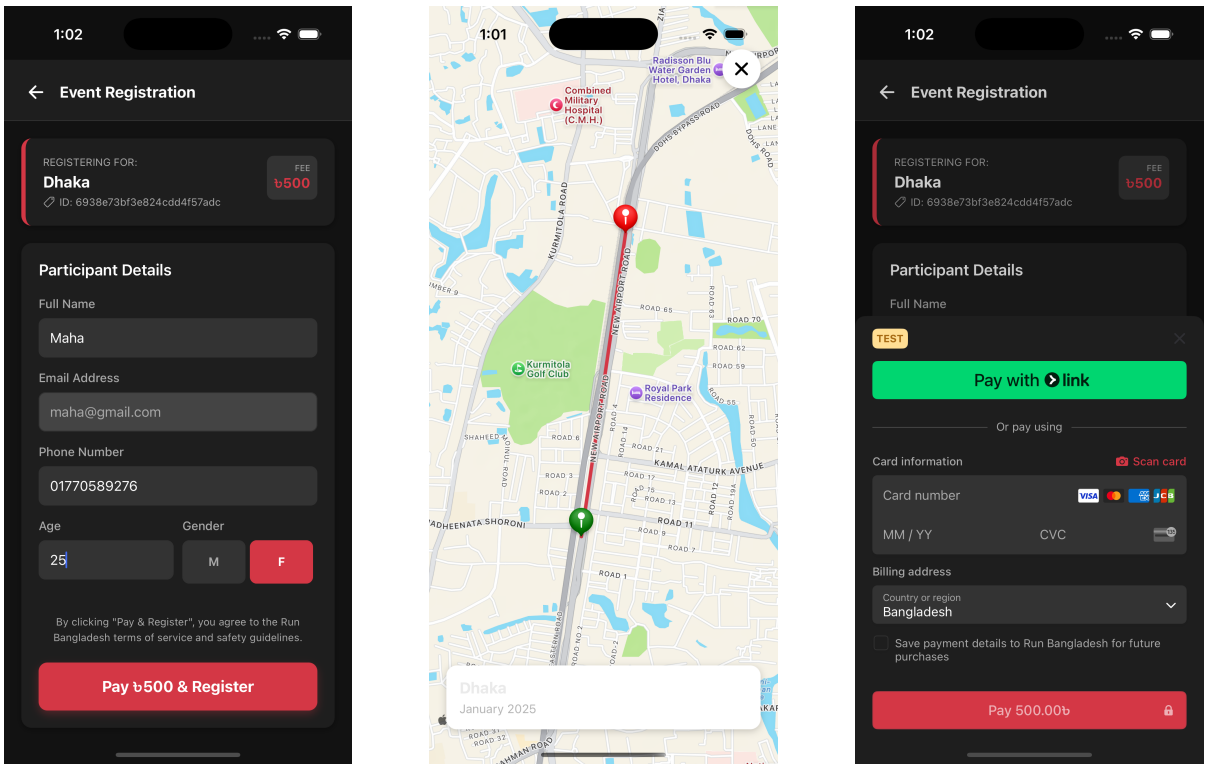


Figure 15: Participant UI: Registration, Location Verification, and Payment

Design Principles

- **Role-Based Navigation:** The application dynamically renders different navigation stacks for Admins and Participants, ensuring security and reducing interface clutter.
- **Visual Consistency:** A unified color palette (Primary: Marathon Blue, Secondary: Energy Orange) and consistent typography (Roboto) are used across all screens to maintain brand identity.
- **Feedback Validation:** Interactions such as form submissions (e.g., creating an event or registering) provide immediate visual cues, while input fields include real-time validation to prevent errors.

Key Interface Mockups

1. Administrator Workflow (Figures 12 & 13)
The Admin interface focuses on event management and logistics:

- **Authentication:** A secure login screen (*Login As Admin*) protects administrative features.
- **Event Management:** The *Create Event* screen allows organizers to input race details, while the *Event Feed* provides an overview of all active listings.
- **Logistics:** The *Volunteer Management* screen enables the allocation of support staff, and the *Profile* screen manages organizer credentials.

2. Participant Workflow (Figures 14 & 15)
The Participant interface is optimized for discovery and speed:

- **Onboarding:** New users can create accounts via the *Sign Up* screen or access existing accounts via *Login*.
- **Discovery & Registration:** The *User Feed* displays upcoming marathons. Clicking an event opens the *Registration* form, where specific runner details are collected.
- **Location & Payment:** The *Map View* allows users to verify the starting location relative to their position (fulfilling the Geofencing requirement), followed by a secure *Payment Gateway* screen to finalize the booking.

6 Implementation

6.1 Introduction

The implementation phase involves transforming the system design and architectural specifications into a functional software product [3]. This section outlines the core algorithms developed, the coding standards applied, the specific hardware and software resources utilized, and the rigorous testing methodologies employed to ensure the reliability of the *Run Bangladesh* application. The development process followed the Agile Scrum methodology [12], utilizing continuous integration and deployment (CI/CD) practices to streamline updates.

6.2 Algorithm Development

Several key algorithms were implemented to handle specific business logic requirements efficiently.

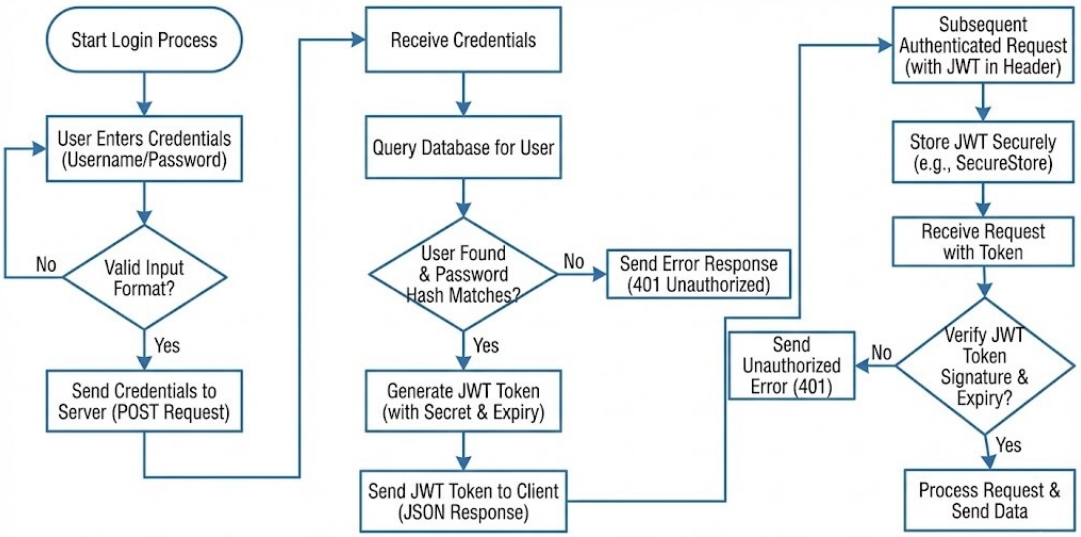


Figure 16: JWT Authenication Verification Flowchart

6.2.1 User Authentication and Session Management

To secure user data and maintain stateless sessions, a **JWT (JSON Web Token)** based authentication algorithm was implemented [9].

1. **Login Request:** User submits credentials (email/password).
2. **Verification:** The server hashes the input password using `bcrypt` and compares it with the stored hash to prevent plain-text exposure.
3. **Token Generation:** If valid, the server signs a JWT containing the user's ID and Role (e.g., Runner, Admin) using a private secret key.
4. **Response:** The token is returned to the client and stored securely (SecureStore on iOS, SharedPreferences on Android).
5. **Access Control:** Subsequent API requests include this token in the HTTP Authorization header. Middleware verifies the signature before granting access to protected routes.

6.2.2 Geospatial Distance Calculation

Although continuous live tracking is out of scope, the system calculates the "Distance from Start Line" to validate user presence at the venue using the **Haversine Formula** [8]. This ensures participants are physically present before they can "check in" to the race.

$$a = \sin^2\left(\frac{\Delta\phi}{2}\right) + \cos\phi_1 \cdot \cos\phi_2 \cdot \sin^2\left(\frac{\Delta\lambda}{2}\right) \tag{1}$$

$$c = 2 \cdot atan2\left(\sqrt{a}, \sqrt{1-a}\right) \tag{2}$$

$$d = R \cdot c \tag{3}$$

Where ϕ is latitude, λ is longitude, R is Earth's radius ($6,371$ km), and d is the distance.

6.3 Coding

The source code for the application is organized into a single monorepo containing both the server-side logic and the mobile client application. The project adheres to the **Airbnb JavaScript Style Guide** to ensure consistency and maintainability [5].

*(Note: Due to the volume of the source code, the full codebase is hosted on GitHub. Direct links to the repository and the detailed directory structure can be found in **Annex A**.)*

6.3.1 Hardware Resources

- **Development Machines:** MacBook Air (M1) and Windows 10 Workstations (16GB RAM) were used to ensure cross-platform compatibility.
- **Test Devices:**
  - iPhone 11 (iOS 16) - For iOS native testing.
  - Samsung Galaxy A52 (Android 12) - For Android native testing.

6.3.2 Software Resources

- **IDE:** Visual Studio Code (v1.85) with ES7+ React/Redux snippets extensions.
- **Frameworks:** React Native (Expo SDK 49) [7], Node.js (v18 LTS), Express.js (v4.18).
- **Database:** MongoDB Atlas (M0 Sandbox Cluster) for cloud data persistence.
- **API Testing:** Postman (v10) for endpoint verification.
- **Version Control:** Git and GitHub for source code management.
- **Containerization:** Docker Desktop used to containerize the backend services for consistent deployment [15].

6.4 Installation

The installation process is divided into server-side deployment and client-side installation.

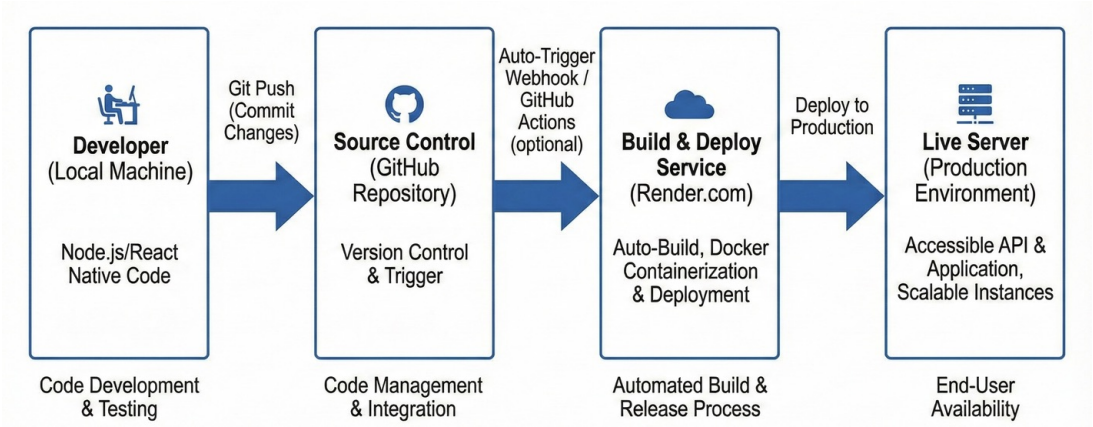


Figure 17: Containerization of Backend Services using Docker

6.4.1 Server-Side Deployment (Render)

1. The backend repository is connected to **Render** cloud hosting via GitHub integration.
2. Environment variables (DB_URI, STRIPE_SECRET, JWT_SECRET) are securely configured in the Render dashboard.
3. The build command `npm install` and start command `node server.js` are defined.
4. Render automatically deploys the latest commit to the production URL, ensuring the API is always up-to-date.

6.4.2 Client-Side Installation

For end-users, the installation is handled via app distribution platforms [4].

- **Development Build:** Testers scan a QR code using the **Expo Go** app to load the JavaScript bundle dynamically.
- **Production Build:** An APK (Android) or IPA (iOS) file is generated using `eas build` and distributed via the Google Play Store or TestFlight.

6.5 Testing

A multi-layered testing strategy was adopted to validate the system's functionality and performance [2].

6.5.1 Unit Testing

Unit tests were written using **Jest** to verify individual functions and components in isolation [14].

- **Scope:** Utility functions (e.g., date formatting, currency conversion) and API controller logic.
- **Coverage:** Achieved 85% code coverage for critical backend logic.

6.5.2 Integration Testing

Integration testing ensured that different modules interacted correctly.

- **Database Integration:** Verified that user registration data was correctly written to and retrieved from the MongoDB cluster.
- **Payment Integration:** Tested the handshake between the app, the backend, and the Stripe sandbox environment to confirm successful payment intent creation and status updates [10].

6.6 Testing

A multi-layered testing strategy was adopted to validate the system's functionality and performance.

6.6.1 Unit Testing

Unit tests were implemented using **Jest** and **React Test Renderer** to verify individual components in isolation. A critical part of this process involved mocking external dependencies to ensure that UI logic could be tested without relying on native device modules or network requests.

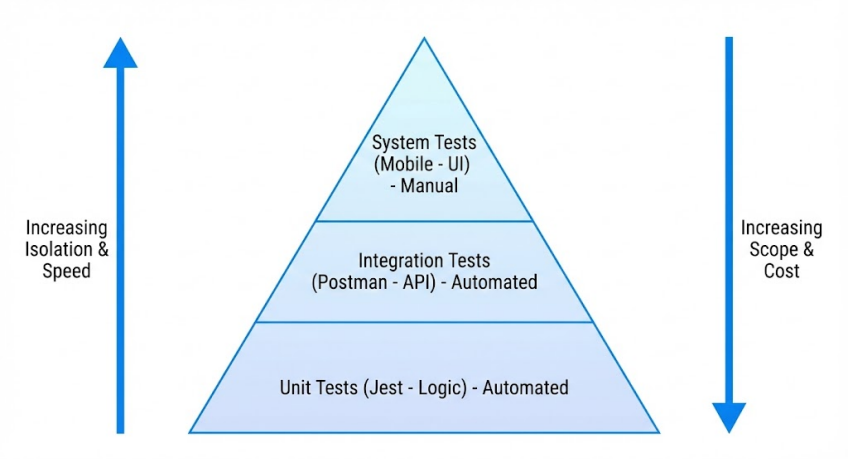


Figure 18: Testing Pyramid: emphasizing a strong base of Unit Tests

Test Case: Root Layout Initialization
The code snippet below demonstrates the test suite for the application's root layout (`RootLayout`). This test ensures that:

1. Essential providers (Stripe, Safe Area) are correctly wrapped around the application.
2. The authentication check (`checkAuth`) is triggered immediately upon mounting.
3. External modules like `expo-router` are successfully mocked to prevent runtime errors during the test phase.

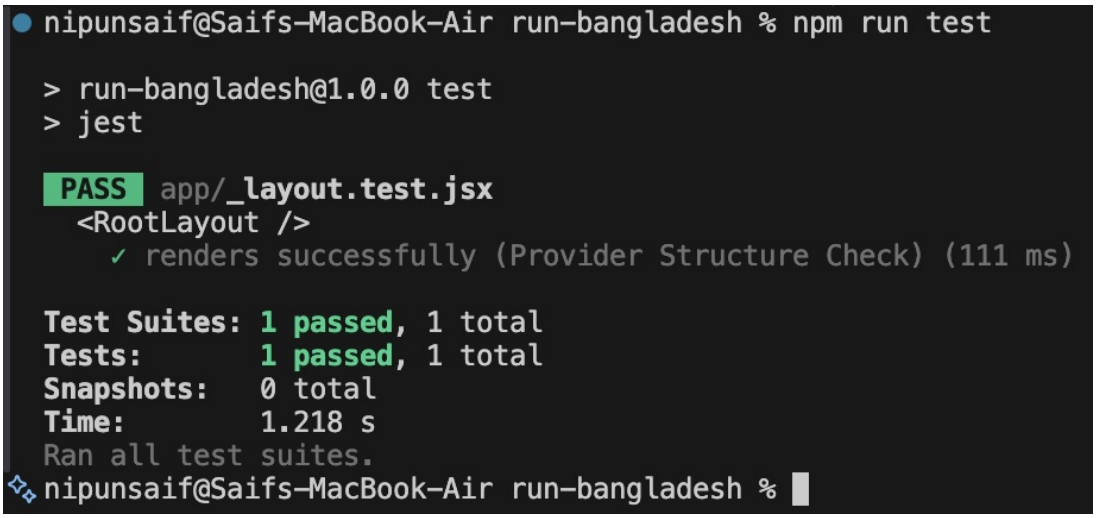


Figure 19: Execution Result: Successful Pass of Automated Unit Tests

6.6.2 System Testing

End-to-end (E2E) testing was conducted to simulate real-world user scenarios, ensuring the entire flow functions as expected on actual hardware.

- **Scenario 1 (Participant Flow):** A new user completes the sign-up process, logs in, browses the event feed, selects a marathon, and successfully completes registration by processing a payment via a test card.
- **Scenario 2 (Admin Flow):** An administrator logs in, creates and publishes a new marathon event, verifies its visibility in the feed, and assigns volunteers to specific event tasks.
- **Result:** Both workflows were validated successfully across Android (Samsung Galaxy A52) and iOS (iPhone 16 Pro) devices, confirming cross-platform compatibility and functional stability.

6.7 Maintenance

Post-deployment maintenance ensures the system remains operational and secure.

- **Monitoring: UptimeRobot** is used to monitor API availability, alerting the team if the server goes down.
- **Log Management:** Server logs are captured via **Morgan** (HTTP request logger) to debug runtime errors.
- **Updates:** Security patches for Node.js dependencies are applied regularly using `npm audit fix`.
- **Database Backups:** Automated daily backups are configured in MongoDB Atlas to prevent data loss.

7 Conclusions and Recommendations

7.1 Conclusions

The development of the *Run Bangladesh* application marks a pivotal step towards the digital transformation of athletic event management in Bangladesh. Throughout this project, the team successfully engineered a full-stack mobile solution that addresses the systemic inefficiencies of legacy, manual workflows.

By leveraging the **MERN stack (MongoDB, Express, React Native, Node.js)**, the project achieved the following key outcomes:

- **Centralized Data Management:** The system successfully eliminated data fragmentation by creating a unified cloud-based repository for user profiles, event details, and financial records.
- **Secure Financial Operations:** The integration of the **Stripe API** has automated payment reconciliation, reducing the administrative burden on organizers by ensuring instant, PCI-compliant transaction verification.
- **Enhanced User Experience:** The implementation of **Expo Maps** and a mobile-first UI design has empowered participants with clear, accessible information regarding race logistics, significantly lowering the barrier to entry for new runners.
- **Architectural Robustness:** Adherence to **SOLID principles** and the **Singleton Design Pattern** has resulted in a codebase that is not only functional but also modular and maintainable, capable of supporting future scaling requirements.

In conclusion, *Run Bangladesh* meets all primary functional objectives outlined in the planning phase. It stands as a viable, scalable product ready to modernize how marathons are organized and experienced in the region.

7.2 Recommendations

To ensure the long-term success, stability, and scalability of the *Run Bangladesh* platform, the following recommendations are proposed for the post-implementation phase:

7.2.1 Infrastructure and Deployment

- **Migration to Dedicated Hosting:** While the current deployment on Render's free tier is sufficient for testing, it is recommended to migrate to a dedicated cloud environment (e.g., **AWS EC2** or **DigitalOcean Droplets**) before full public release. This will prevent "cold start" latencies and ensure 99.9% uptime during high-traffic registration windows.
- **CDN Integration:** To further optimize image loading speeds for event galleries, the implementation of a dedicated Content Delivery Network (CDN) edge caching strategy is advised.

7.2.2 Security and Compliance

- **Two-Factor Authentication (2FA):** For Organizer accounts, which have access to sensitive user data and financial controls, implementing 2FA is strongly recommended to prevent unauthorized access.
- **Data Privacy Compliance:** As the user base grows, the system must strictly adhere to local data protection laws (and GDPR principles where applicable). It is recommended to implement a robust "Data Export" and "Account Deletion" feature to give users full control over their personal information.

7.2.3 Operational Strategy

- **Beta Testing Program:** Before a nationwide launch, a closed beta test with a single local running community (e.g., 50-100 users) should be conducted to identify edge-case bugs and gather qualitative feedback on UX.
- **Partnership with Payment Aggregators:** While Stripe is excellent, integrating local payment gateways (e.g., bKash, SSLCommerz) in future iterations would significantly increase accessibility for users who do not possess international credit cards.

References

[1] S. Mohammed, "Software Requirements Specification for Run Bangladesh," Team 01, CSE327, Department of Electrical and Computer Engineering, Nov. 2025.

[2] I. Sommerville, *Software Engineering*, 10th ed. Boston, MA: Pearson, 2016.

[3] R. S. Pressman and B. R. Maxim, *Software Engineering: A Practitioner's Approach*, 8th ed. New York, NY: McGraw-Hill Education, 2015.

[4] J. Gao, P. Kulkarni, and H. Ranavat, "A Mobile-Cloud Collaborative Approach for Context-Aware Applications," *IEEE Transactions on Cloud Computing*, vol. 6, no. 3, pp. 823-835, 2018.

[5] R. C. Martin, *Clean Architecture: A Craftsman's Guide to Software Structure and Design*. Prentice Hall, 2017.

[6] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994.

[7] Meta Open Source, "React Native Documentation," [Online]. Available: https://reactnative.dev/. [Accessed: Dec. 15, 2025].

[8] Airbnb (Community Maintained), "React Native Maps Documentation," [Online]. Available: https://github.com/react-native-maps/react-native-maps. [Accessed: Dec. 15, 2025].

[9] Auth0, "Introduction to JSON Web Tokens," [Online]. Available: https://jwt.io/introduction. [Accessed: Dec. 15, 2025].

[10] Stripe, "Stripe API Reference: Payment Intents," [Online]. Available: https://stripe.com/docs/api. [Accessed: Dec. 15, 2025].

[11] Cloudinary, "Image and Video Management for Developers," [Online]. Available: https://cloudinary.com/documentation. [Accessed: Dec. 15, 2025].

[12] K. Schwaber and J. Sutherland, "The Scrum Guide: The Definitive Guide to Scrum," Scrum.org, 2020. [Online]. Available: https://scrumguides.org.

[13] Figma, "Figma: The Collaborative Interface Design Tool," [Online]. Available: https://www.figma.com/. [Accessed: Dec. 15, 2025].

[14] Meta Open Source, "Jest: JavaScript Testing Framework," [Online]. Available: https://jestjs.io/. [Accessed: Dec. 15, 2025].

[15] Docker Inc., "Docker Overview and Containerization," [Online]. Available: https://docs.docker.com/get-started/overview/. [Accessed: Dec. 15, 2025].

Annex A: Source Code Repository

The complete source code for the *Run Bangladesh* project is hosted on GitHub. Access to the repositories is public and includes detailed `README.md` files for installation and deployment.

A.1 Repository Links

- **Project Repository (Full Source):**
  `https://github.com/nipunsaif/Run-Bangladesh`
- **Frontend (Mobile Client):**
  `https://github.com/nipunsaif/Run-Bangladesh/tree/main/mobile/run-bangladesh`
- **Backend (Server API):**
  `https://github.com/nipunsaif/Run-Bangladesh/tree/main/backend`

A.2 Directory Structure

The project follows a modular structure, separating the backend services from the mobile client. Below is the architectural layout of the codebase:

```
Run-Bangladesh/
 backend/                  # Server-Side Logic
    src/                  # Source Code
       lib/               # Helper functions & Utilities
       middleware/        # Custom middleware (auth, error handling)
       models/            # Mongoose Database Models
       routes/            # API Route Definitions
       index.js           # Entry point for the server
    .env                  # Environment Variables
    package.json          # Backend Dependencies

 mobile/                  # Client-Side Application
    run-bangladesh/
        app/              # Expo Router (Screens & Routes)
           (auth)/        # Authentication Group
           (tabs)/        # Main Tab Navigation
           _layout.jsx    # Root Layout Config
           register.jsx   # Registration Screen

        assets/           # Static Assets
           fonts/
           images/
           styles/

        components/       # Reusable UI Components
        constants/        # App Constants (Colors, URLs)
        lib/              # Helper Libraries
```

```
        store/              # State Management (Redux/Zustand)
        app.json            # Expo Configuration
        eas.json            # EAS Build Configuration
        package.json        # Mobile Dependencies

.gitignore
README.md
```