CSC 431

Coral Gables Live Music Booking Application

System Architecture Specification (SAS)

Team 7

Alan Fiore	Software Developer
David Erulker	Software Developer
Tal Ram	Software Developer

Version History

Version	Date	Author(s)	Change Comments
1	04/07/2025	Alan Fiore, David Erulker, Tal Ram	Initial Draft - system overview
2	04/11/2025	Alan Fiore, David Erulker, Tal Ram	Structural design / sequence diagrams

Table of Contents

Table of Tables

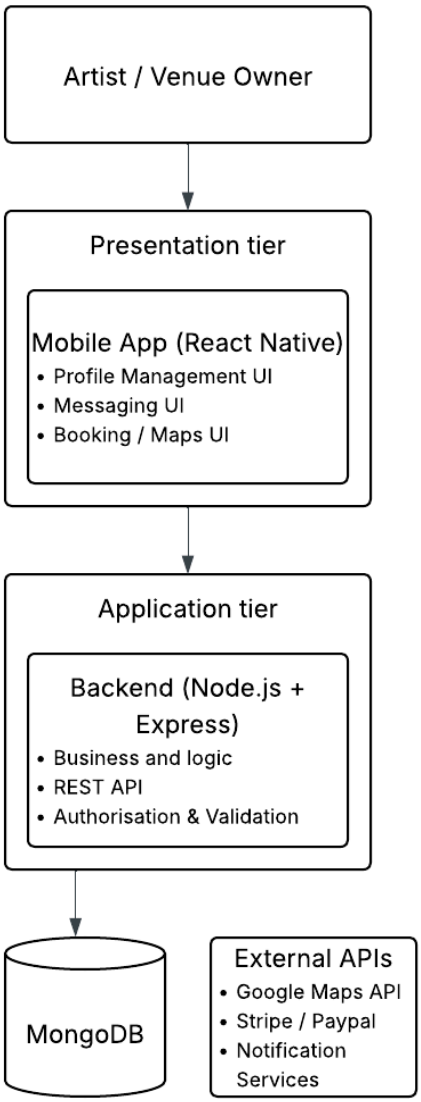
Table of Figures

1. System Analysis

1.1 System Overview

The Coral Gables Live Music Booking Application is a mobile platform that connects local artists with venues through profile management, geolocation discovery, and integrated booking workflows. Using a 3-tier layered architecture, the system separates presentation (React Native frontend), application logic (Node.js/Express backend), and data storage (MongoDB) to achieve scalability and maintainability.

1.2 System Diagram



1.3 Actor Identification

Actor Type	Actor	Roles and Responsibilities
Primary	Artists	Create profiles, search venues, negotiate bookings
Primary	Venue Owners	Manage venue profiles, handle booking requests
Secondary	Payment Processors (Stripe/PayPal)	Handle financial transactions
Secondary	Google Maps API	Provide geolocation services

Additional Context:

- Artists and venues interact via a unified interface, reducing friction in booking workflows.
- Third-party services (Stripe, Google Maps) are abstracted behind facades to simplify integration.

1.4 Design Rationale

1.4.1 Architectural Style

3-Tier Layered Architecture		
Architecture Tier	**Technology**	**Responsibilities**
Presentation Tier	React Native	Handles the user interface, providing a native-like experience on iOS and Android.
Application Tier	Node.js + Express.js	Manages business logic, API endpoints, and server-side operations.
Data Tier	MongoDB	Stores application data (user profiles, venue info, booking details).

Justification:

- Decoupling tiers allows independent scaling (e.g., backend servers vs. frontend updates).
- Mongoose adds validation (e.g., ensuring artist profiles include contact info) without sacrificing NoSQL agility.

1.4.2 Design Pattern(s)

Pattern	Application	Justification
Observer	Notify artists/venues about booking status changes.	Keeps status changes separate, so updates happen right away.
Facade	Simplify payment processing and geolocation APIs behind unified interfaces.	Reduces complexity for artists/venues interacting with Stripe/PayPal or Google Maps
Iterator	Traverse search results (venues/artists) consistently, regardless of data source.	Standardizes access to collections like venue/artist lists.
Decorator	Dynamically add features like promotional pricing and verified users.	Extends functionality without modifying core classes

Impact:

- Observer/Facade enhance reliability in critical paths (payments/notifications).
- Iterator/Decorator improve maintainability for evolving business rules.

1.4.3 Framework

Framework	Purpose	Selection Rationale
React Native	Enables cross-platform mobile development with reusable components and native performance.	Selected to efficiently support both iOS and Android platforms under project constraints.
Express.js	Provides a lightweight web framework for building robust APIs and handling HTTP requests.	Chosen for its flexibility and seamless integration with Node.js backend requirements.
Mongoose	Simplifies MongoDB interactions in schema validation and data modeling.	Supports data integrity and streamlines database operations for the application.

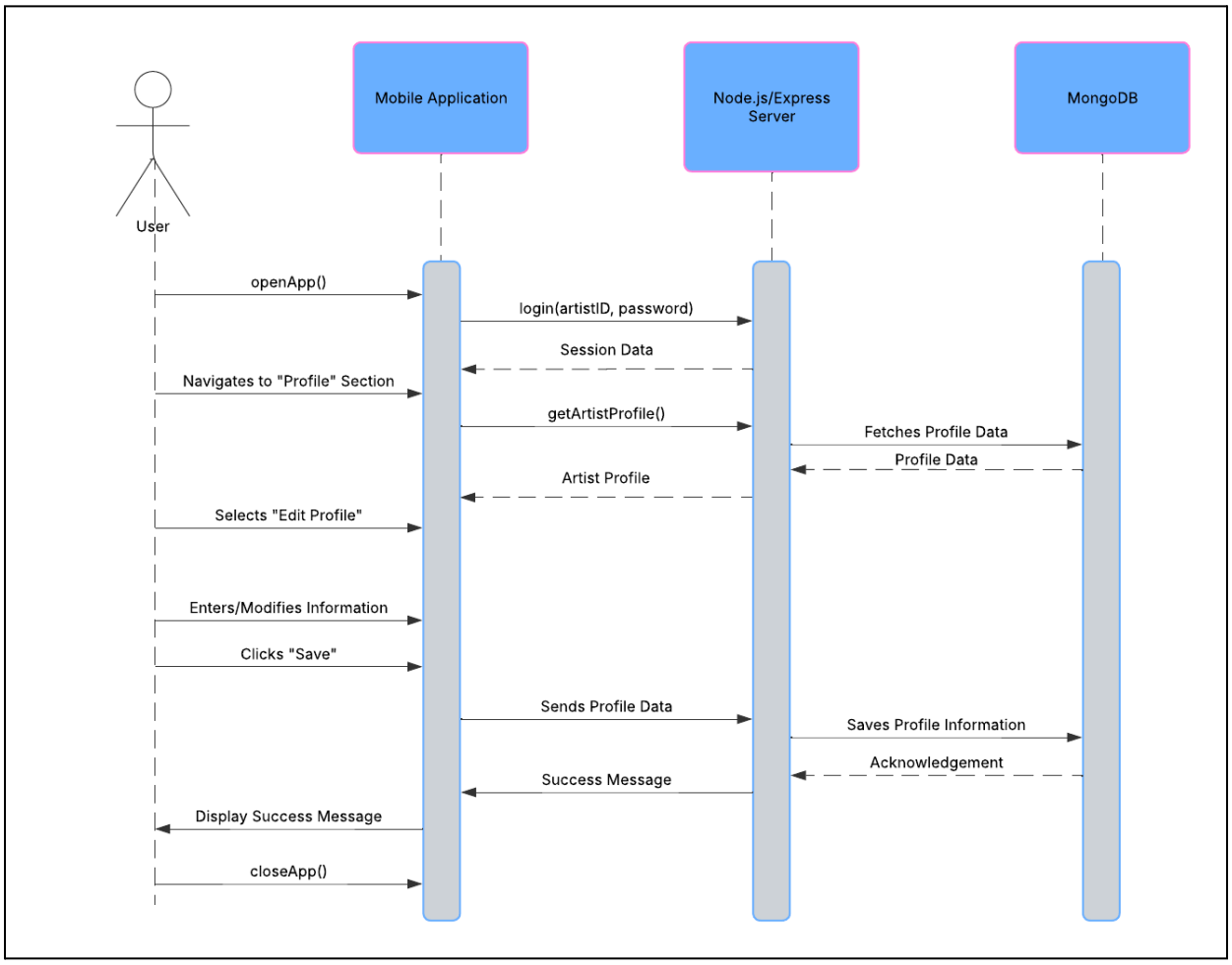
Tradeoffs Considered:

- React Native vs. Flutter: Chose RN for larger community support in JavaScript ecosystem.
- Express.js vs. NestJS: Opted for Express due to simpler learning curve and project scope.

2 Functional Design

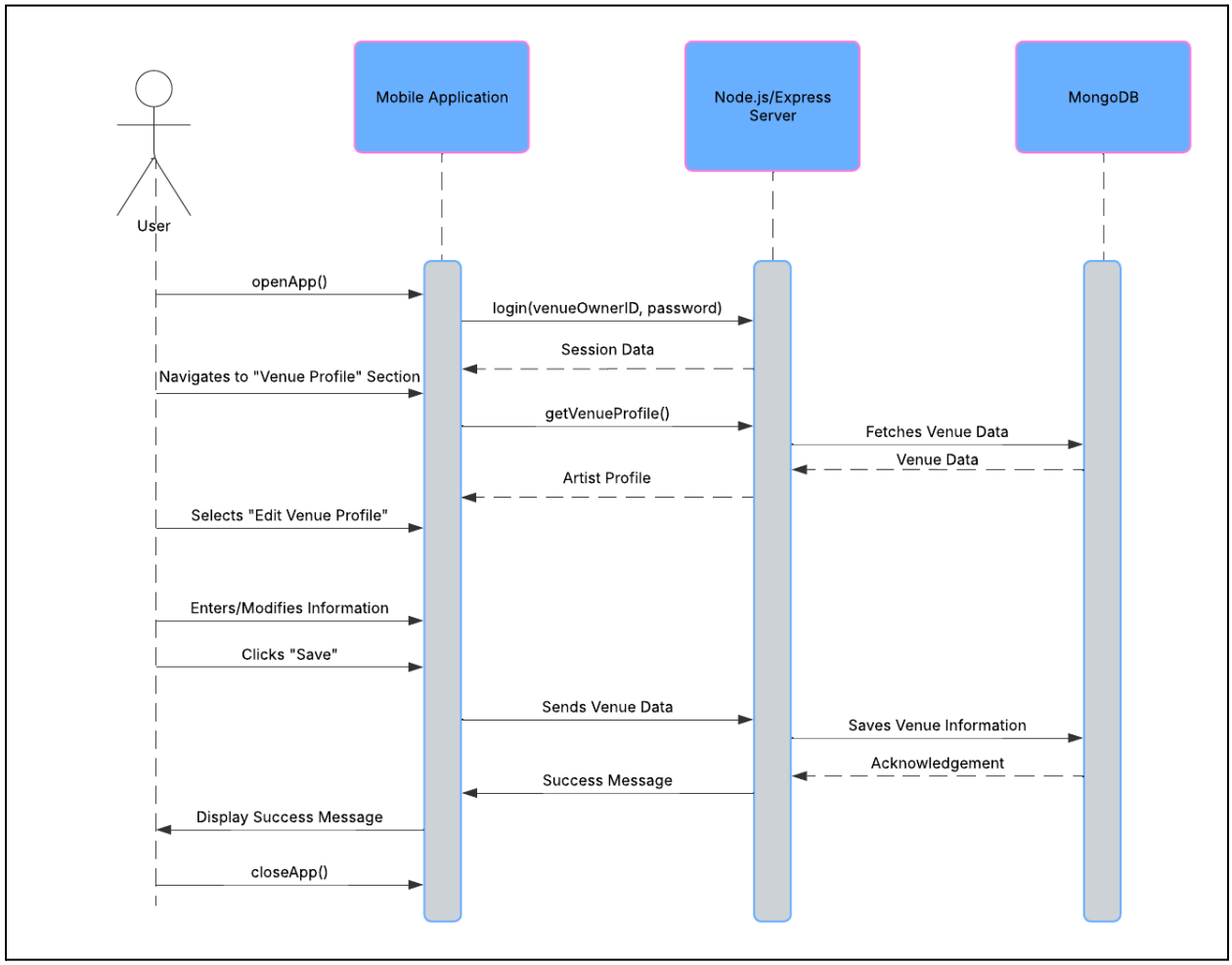
2.1 Artist Profile Management

The system allows artists to create and manage detailed profiles, including bios, music genres, performance samples (audio/video links), and availability calendars. Profiles are searchable by venues and support verification badges for authenticity. Artists can edit their information, set booking preferences (e.g., minimum fees, travel radius), and link social media accounts for credibility.



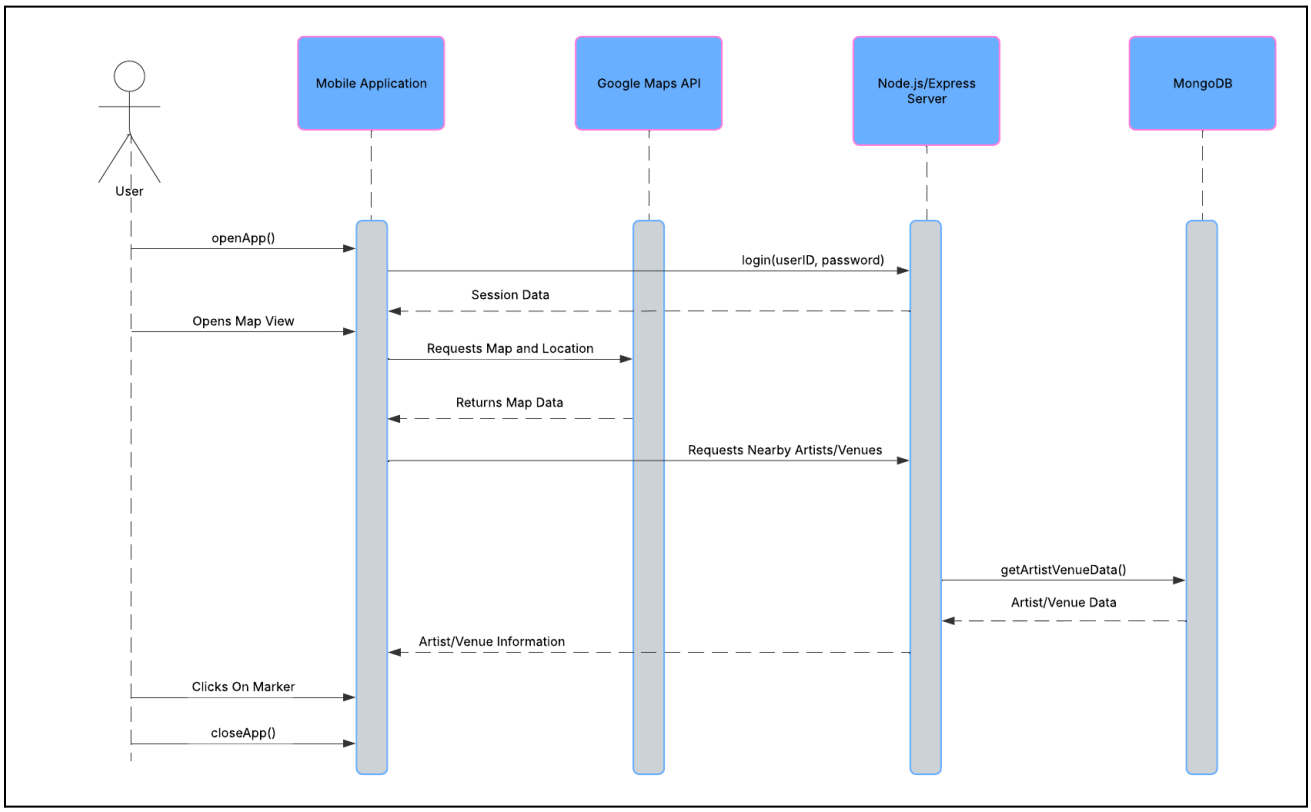
2.2 Venue Profile Management

Venues can register and maintain profiles featuring photos, capacity, amenities (e.g., stage size, sound equipment), and booking policies. Owners can define pricing tiers, blackout dates, and preferred artist genres. Real-time availability sync ensures no double bookings, while ratings/reviews from past events enhance credibility.



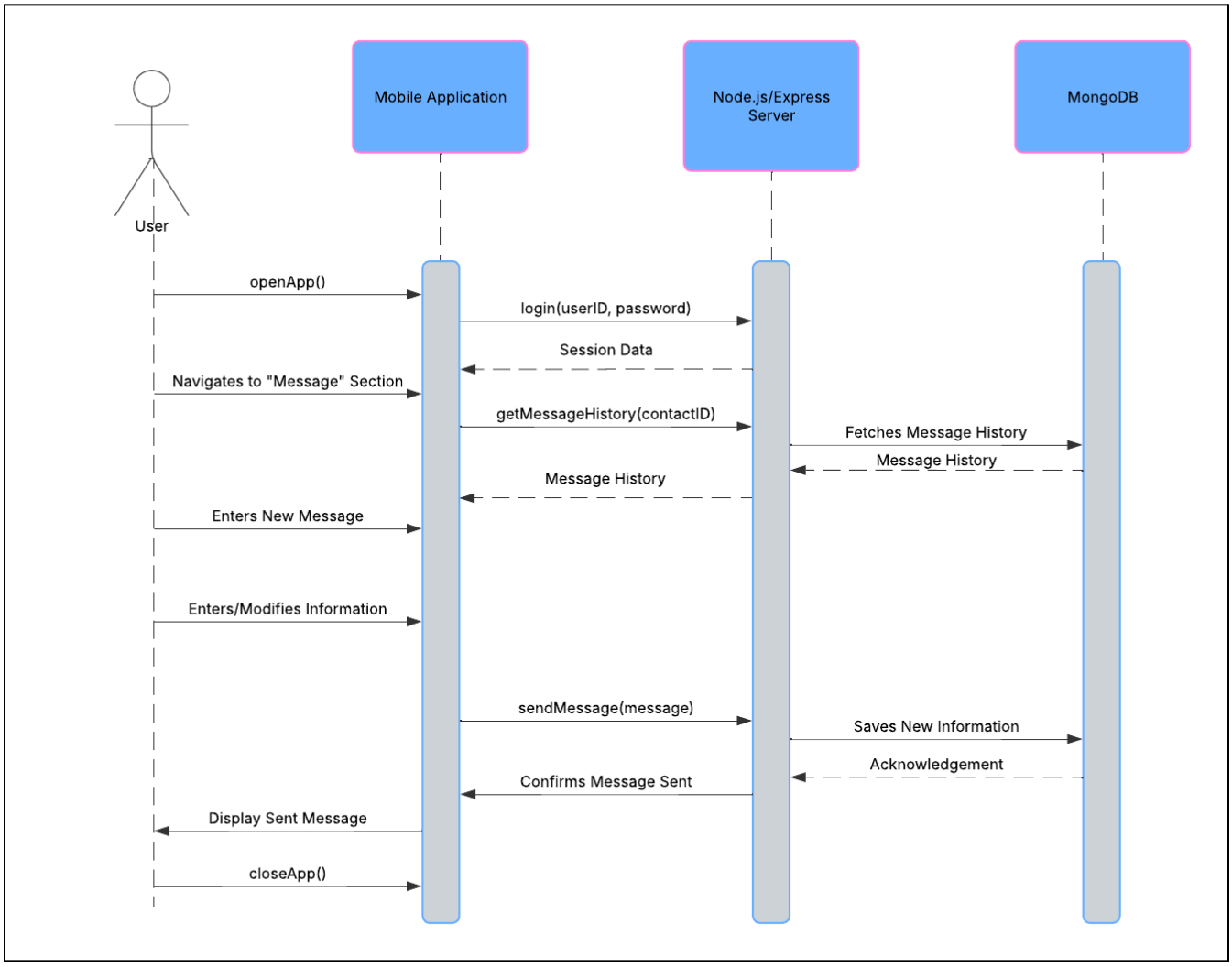
2.3 Geolocation Services for Venue and Artist Discovery

Powered by Google Maps API, the system enables location-based searches with filters (distance, venue type, artist genre). Artists see nearby venues on an interactive map, while venues discover local talent. Proximity-based notifications alert users to new opportunities within their radius.



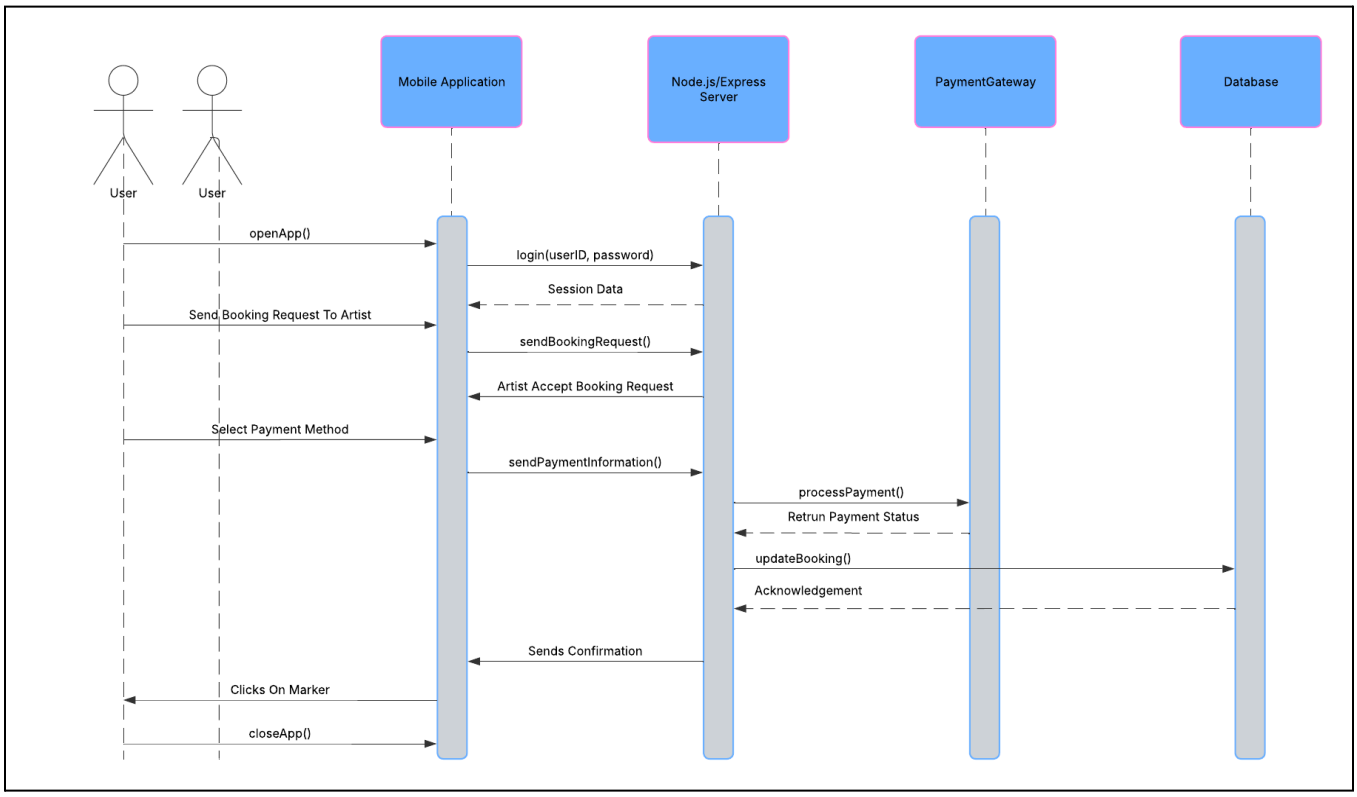
2.4 In-App Messaging System

A real-time chat interface facilitates direct negotiation between artists and venues. Messages support attachments (contracts, rider documents) and include read receipts. Automated reminders prompt users to respond, and conversation history is preserved for future reference.



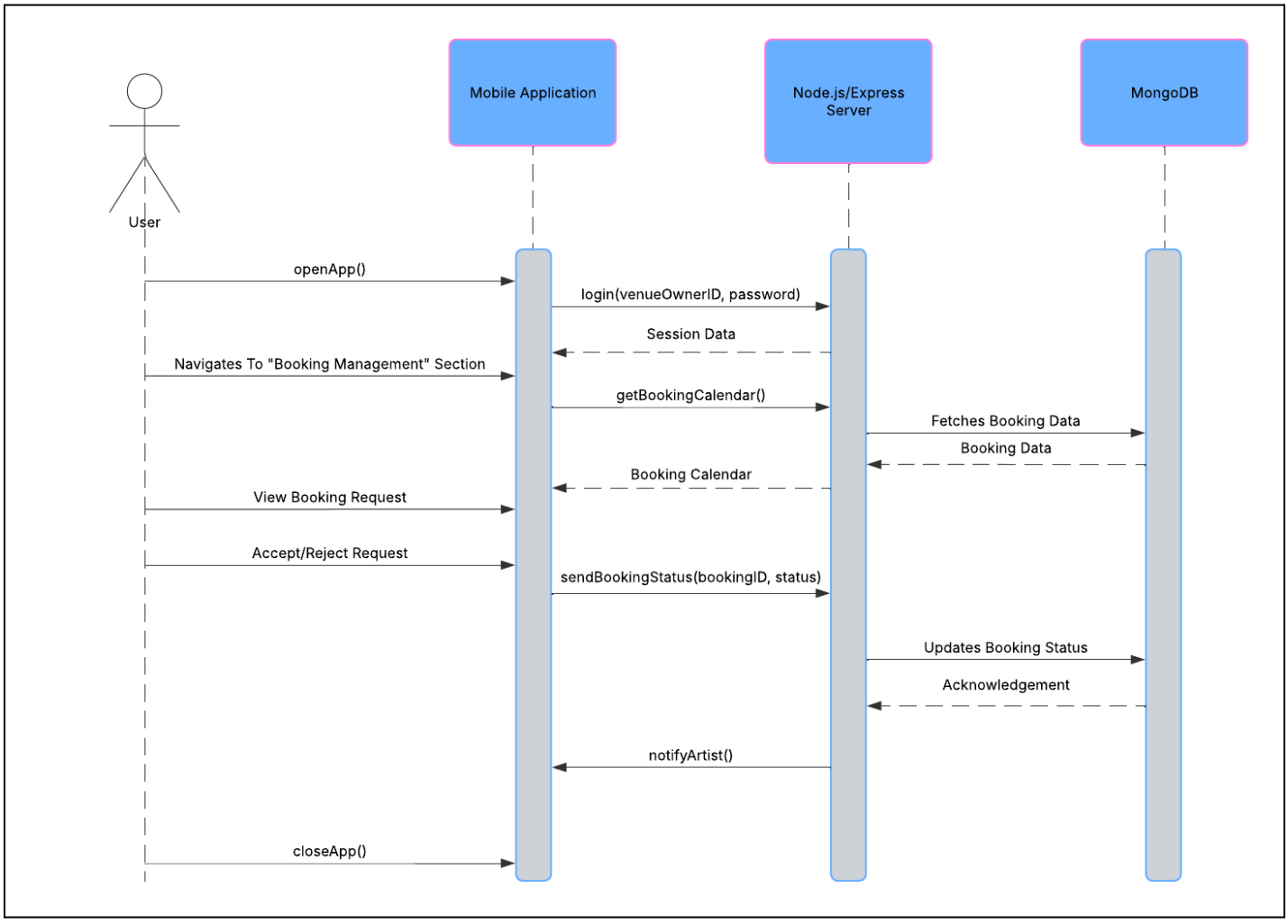
2.5 Integrated Payment Processing

Secure payments are handled via Stripe/PayPal, supporting deposits, splits (e.g., artist/venue/platform fees), and refunds. Escrow options hold funds until gig completion, with automated payouts triggered by confirmation from both parties. Receipts and tax documentation are generated automatically.



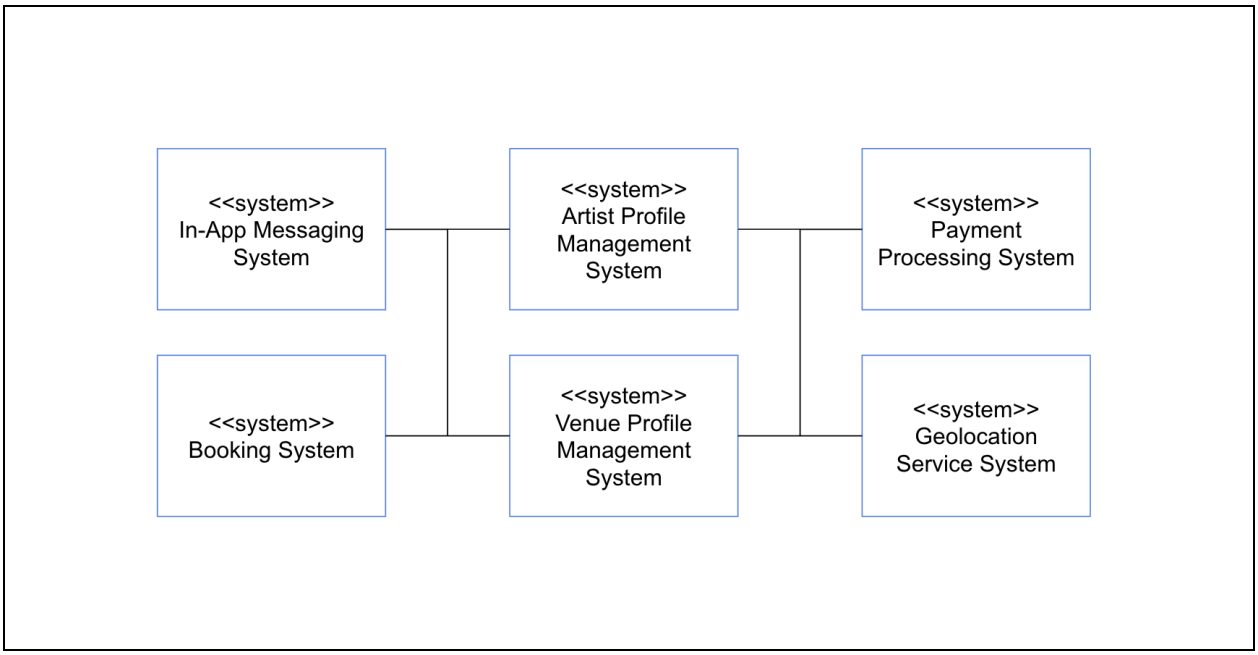
2.6 Booking Management System for Venues

Venues receive, approve, or decline booking requests through a centralized dashboard. Conflict detection prevents overlapping events, and calendar integrations (Google Calendar, Outlook) sync bookings. Post-event, venues can leave artist reviews and manage invoices/payouts.



3. Structural Design: Context Model and Class Diagram

Context Model



Class Diagram