Title: **Karlein**, an event management platform for small businesses and communities

Written by:

Roshni Rana and Sarah Saiyed

# Contents


1. *Purpose*
    - 1.1. Scope
    - 1.2. Overview and Background

2. *Software Process Model - Prototype Model*
    - 2. 1. Introduction
    - 2.2 Requirement Gathering
        - 2.2.1. Feasibility Study
        - 2.2.2. Requirement Elicitation and Analysis
        - 2.2.3. Functional Requirements
        - 2.2.4 Non- Functional Requirements
        - 2.2.5. Prioritization and Negotiation
        - 2.2.6 Requirement Negotiation
        - 2.2.7 UML diagrams
    - 2.3 Requirement Validation
        - 2.3.1 Requirement Reviews
        - 2.3.2 Prototyping
        - 2.3.3 Test-Case Generation
    - 2. 4. Requirement Management
        - 2. 4.1 Requirement Change Management
        - 2. 4.2 Requirement Evolution
    - 2. 5. Quick Design
    - 2. 6. Building the prototype
    - 2. 7. User Evaluation and Feedback
    - 2. 8. Refinement and Reiteration
    - 2.9. Finalization and Deployment

3. *Advantages of Using the Prototype Model for this Project*
    - 3.1 Better Understanding of User Needs
    - 3.2 Increased User Involvement
    - 3.3 Early Detection of Issues
    - 3.4 Cost and Time Efficiency
    - 3.5 Risk Mitigation

4. *AI Application*

5. *Conclusion*

# 1. Purpose

The Event Management System (EMS) is designed to facilitate event creation, management, and ticketing for users, event organizers, and administrators. The system enables users to browse and book events, while event hosts can create and manage their events. The system also ensures a seamless user experience by incorporating automated notifications, real-time updates, and a user-friendly dashboard.

## 1.1 Scope

The Event Management System will provide a comprehensive platform for managing different types of events, such as *meet-ups, weddings, poetry-readings, and social gatherings.* The system will allow users to register, create events, browse existing events, purchase tickets, or RSVP, and handle event logistics seamlessly.

## 1.2 Overview and Background

The system will offer online event booking, event hosting and participant registration. It will also include a catalog of events to choose from, payment processing, and real-time updates. The application aims to provide a platform for small businesses, community organizations and even individual hosts, to find audiences for their events.

### 1. 2.1 Product Perspective

The system will serve as an online platform to manage various events, marketing to potential participants, coordinating with vendors, and handling logistics. It will support both public and private events.

### 1. 2.2 Product Functionality/Features

The system will include functionalities such as:

- Event creation and management
- Online booking and reservations
- Guest list and RSVP tracking
- Automated email and SMS notifications
- Payment processing and invoice generation
- Real-time event updates
- User-friendly navigation with a centralized dashboard
- Event catalog for browsing various events

### 1. 2.3 User Characteristics

The system will be accessible to users with basic internet knowledge and will require login credentials for secure access.

### 1. 2.4 Constraints

- Supported web browsers: Google Chrome, Mozilla Firefox, Microsoft Edge
- Supported operating systems: Windows 10 and above, macOS, Linux

In order to implement this, we have chosen the Prototype Model for software development.

# 2. Software Process Model

## 2. 1 Introduction

The **Prototype Model** is an iterative software development approach where a working prototype is built, refined, and enhanced based on user feedback before the final system is developed. This model is highly suitable for the **Event Management App for Small Businesses and Pop-ups** due to the following reasons:

- **Unfamiliarity of End Users with Online Event Platforms**: Small business owners and pop-up event organizers may not be well-versed in digital event management tools. A prototype will help them visualize and interact with the system, leading to better requirement discovery.
- **Dynamic Requirement Evolution**: Since users may not fully understand their needs initially, the prototype allows them to refine their requirements based on real-time usage and feedback.
- **Maximum User Interaction**: Given that the app is user-centric, direct input from users is essential for an optimal experience.
- **Risk Reduction**: By building and refining prototypes, development risks, such as usability issues or lack of essential features, can be identified early and resolved before full-scale implementation.

## 2.2 Requirement Gathering and Engineering

## 2.2.1 Feasibility Study

### 2.2.1.1 Technical Feasibility

The Event Management System will be developed as a web application with a user-friendly interface, allowing both event organizers and attendees to interact seamlessly. The chosen technology stack includes:

- **Frontend:** HTML, CSS (Tailwind, Bootstrap), JavaScript **(Alpine.js / Vanilla JS), Jinja2 Templating (Flask)**
- **Backend:** Flask (*Flask-Login*, *Flask-WTF*, *Flask-SQLAlchemy*, *Flask-Migrate)*
- **Database:** MySQL for structured event and user data storage
- **Authentication:** JWT-based secure login system
- **Hosting & Scalability:** Cloud-based deployment using AWS/GCP

The system will support CRUD functionalities for events, user authentication, booking processes, and administrative controls.

### 2.2.1.2 Economic Feasibility

A cost-benefit analysis suggests that this system will be financially viable given its target audience. The primary costs include:

- **Development Costs:** Initial programming, database setup, and UI/UX design
- **Operational Costs:** Server hosting, database maintenance, and customer support
- **Revenue Model:**
    - Commission-based earnings from ticket sales
    - Subscription-based plans for event organizers
    - Advertisement-based revenue for premium event placements

### 2.2.1.3 Operational Feasibility

The system is designed for ease of use with intuitive navigation and event management functionalities. The UI follows minimalistic principles with clear call-to-action buttons, ensuring smooth onboarding for both small business owners and event attendees.

### 2.2.1.4 Legal & Ethical Feasibility

The system will comply with data privacy laws such as GDPR and CCPA, ensuring:

- Secure user data handling (password encryption, two-factor authentication)
- Transparent event cancellation and refund policies

- Accessibility compliance (WCAG guidelines) for an inclusive experience

## 2.2.2. Requirement Elicitation and Analysis

### 2.2.2.1. Requirements Discovery

To ensure that the system meets user expectations, surveys and interviews were conducted with event organizers and attendees. The following survey questions can be used:

**Sample Survey Questions**

1. What challenges do you face when organizing events?
2. What event categories do you frequently attend or host?
3. How do you currently promote and manage ticket sales?
4. What payment methods do you prefer for booking events?
5. Do you require ticket refunds or transfer options?
6. What are the most common issues you face while attending events?
7. Would you like a feature for automated event reminders?
8. Do you prefer event recommendations based on your interests?
9. How important is social media integration for event promotion?
10. Would you be interested in a rating and review system for events?

### 2.2.2.2. Requirement Classification and Organization

The gathered requirements are categorized into functional and non-functional requirements.

## 2.2.3. Functional requirements

### 2.2.3.1 User Requirements

#### 2.2.3.1.1 Guests

2.2.3.1.1.1 User Management and Authentication:

2.2.3.1.1.1 Can register an account and log in using email and password.

2.2.3.1.1.2 Can recover their password via an email verification process.

2.2.3.1.1.2 Online Booking and Reservations:

2.2.3.1.2.1 Can browse available events using filters (date, category, location).

2.2.3.1.2.2 Can book tickets for events and receive confirmation emails.

2.2.3.1.2.3 Can cancel reservations if they are unable to attend.

2.2.3.1.1.3 Centralized Dashboard:

2.2.3.1.3.1 Access a personalized dashboard to view upcoming bookings and manage profile settings.

## 2.2.3.1.2 Hosts

2.2.3.1.2.1 User Management and Authentication:

2.2.3.2.1.1 Can register an account and log in to manage their events.

2.2.3.2.1.2 Can edit their profile details (contact information, event preferences).

2.2.3.1.2.2 Event Creation and Management:

2.2.3.2.2.1 Can create new events by providing details (name, date, time, location, description).

2.2.3.2.2.2 Can update or modify event details post-creation.

2.2.3.1.2.3 Online Booking and Reservations:

2.2.3.2.3.1 Can view detailed booking information for their events (guest details and ticket counts).

2.2.3.1.2.4 Guest List and RSVP Tracking:

2.2.3.2.4.1 Can track the guest list, including confirmed and pending RSVPs.

2.2.3.2.4.2 Can approve or reject RSVPs based on event capacity or specific criteria.

2.2.3.1.2.5 Centralized Dashboard:

2.2.3.2.5.1 Manage all created events, track ticket sales, and view attendee details.

2.2.3.1.2.6 Automated Notifications:

2.2.3.2.6.1 Receive email and SMS alerts for bookings, cancellations, and event reminders.

2.2.3.1.2.7 Real-time Event Updates:

2.2.3.2.7.1 Get live updates about changes in event schedules, ticket availability, or venue modifications.

### 2.2.3.1.3 Admins

2.2.3.1.3.1 User Management and Authentication:

2.2.3.1.3.1.1 Manage user accounts by activating, deactivating, or deleting accounts.

2.2.3.1.3.1.2 Implement role-based access control to secure user privileges.

2.2.3.1.3.2 Event Creation and Management:

2.2.3.1.3.2.1 Oversee all event activities to ensure compliance with platform guidelines.

2.2.3.1.3.2.2 Approve or reject events before they are published.

2.2.3.1.3.2.3 Delete events that are inappropriate or fraudulent.

2.2.3.1.3.3 Online Booking and Reservations:

2.2.3.1.3.3.1 Monitor booking activities and detect any anomalies in the reservation system.

2.2.3.1.3.4 Guest List and RSVP Tracking:

2.2.3.1.3.4.1 Monitor attendance records and RSVP statuses for all events.

2.2.3.1.3.4.2 Generate RSVP reports for analysis and future improvements.

2.2.3.1.3.5 Automated Notifications:

2.2.3.1.3.5.1 Configure notification settings, customize message templates, and adjust frequency.

2.2.3.1.3.6 Payment Processing and Invoice Generation:

2.2.3.1.3.6.1 Oversee secure payment processing (via integrated gateways like PayPal or Stripe).

2.2.3.1.3.6.2 Generate and send invoices for completed transactions.

2.2.3.1.3.6.3 Resolve any payment disputes.

2.2.3.1.3.7 Real-time Event Updates:

2.2.3.1.3.7.1 Monitor and verify event updates before they are broadcasted to users.

2.2.3.1.3.8 Centralized Dashboard:

3.1.3.8.1 Access a comprehensive dashboard to monitor all system activities, including user actions, bookings, and payments.

### 2.2.3.2 Performance Requirements

2.2.3.2.1 Support at least 1000 concurrent users without performance degradation.

2.2.3.2.2 Optimize application response times to provide a smooth user experience.

2.2.3.2.3 Utilize load balancing to distribute user traffic evenly across servers.

2.2.3.2.4 Implement caching mechanisms to reduce server load and improve data retrieval times.

2.2.3.2.5 Continuously monitor system performance and resource utilization to proactively address bottlenecks.

### 2.2.3.3 External Interface Requirements

2.2.3.3.1 Software Interfaces

2.2.3.3.1.1 Database Management System: SQL Server

2.2.3.3.1.2 Web Application Framework: Flask, Python

2.2.3.3.1.3 Payment Gateway: Stripe, PayPal

2.2.3.3.2 Communication Interfaces

2.2.3.3.2.1 The system will utilize HTTP/HTTPS protocols for data transmission.

2.2.3.3.2.2 Email and SMS integration for notifications.

## 2.2.4 Non-Functional Requirements

### 2.2.4.1 Security

4.1.1 Implement data encryption and secure authentication mechanisms to protect user information.
4.1.2 Use role-based access control (RBAC) to restrict permissions according to user roles.
4.1.3 Regularly conduct security audits and vulnerability assessments to identify and mitigate risks.
4.1.4 Implement secure storage solutions with redundancy and backup encryption.

### 2.2.4.2 Usability

4.2.1 Ensure an intuitive, responsive design that adapts across devices and screen sizes.

4.2.2 Adhere to accessibility standards (such as WCAG) to make the interface usable for individuals with disabilities.

4.2.3 Maintain consistency in design and navigation throughout the application for a seamless user experience.

4.2.4 Integrate user feedback mechanisms to continuously improve the interface and overall usability.

4.2.5 Ensure the system performs reliably across different web browsers and operating systems.

### 2.2.4.3 Reliability/Availability

4.3.1 Maintain 99.9% uptime to ensure continuous service availability.

4.3.2 Deploy redundant systems and servers to eliminate single points of failure.

4.3.3 Establish robust backup and disaster recovery plans to quickly restore services in case of failures.

4.3.4 Integrate automated failover mechanisms to switch to backup systems seamlessly.

4.3.5 Set up real-time monitoring and alerting systems to detect and address issues promptly.

## 2.2.5. Prioritization and Negotiation

Using the **MoSCoW prioritization technique**, the requirements were ranked as:

- **Must-have:** User authentication, event browsing, booking system
- **Should-have:** Social media sharing, personalized recommendations

- **Could-have:** Augmented reality event previews, blockchain ticketing
- **Won't-have (for now):** AI-powered chatbots for event assistance

## 2.2.6 Requirement Negotiation

- Stakeholders debated the necessity of advanced analytics for event performance.
- Organizers prioritized user engagement tools over complex ticketing options.
- Cost-benefit analysis led to postponing certain  premium features for future updates.

## 2.2.7 UML diagrams

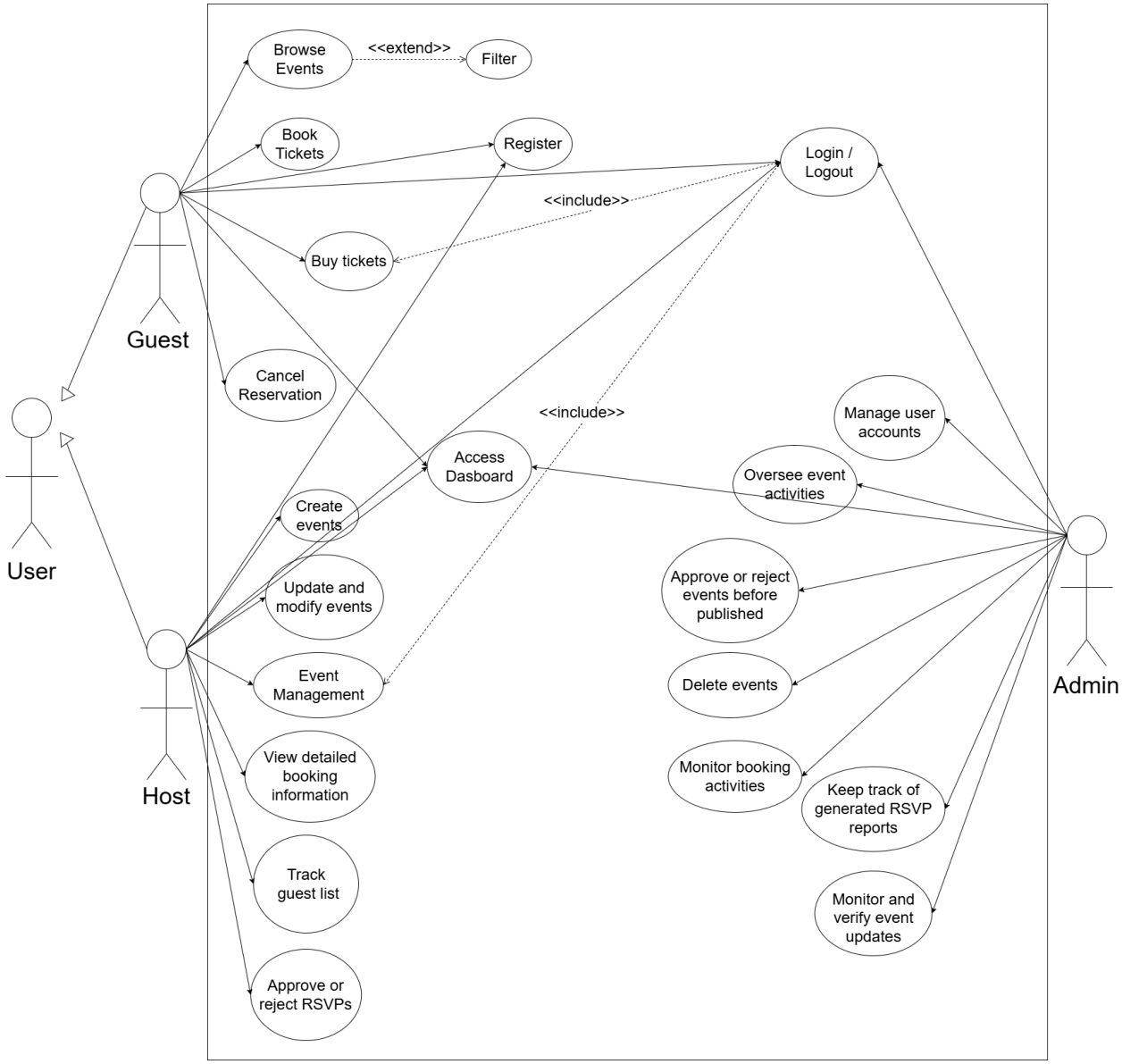



Figure 1: Use-case Diagram

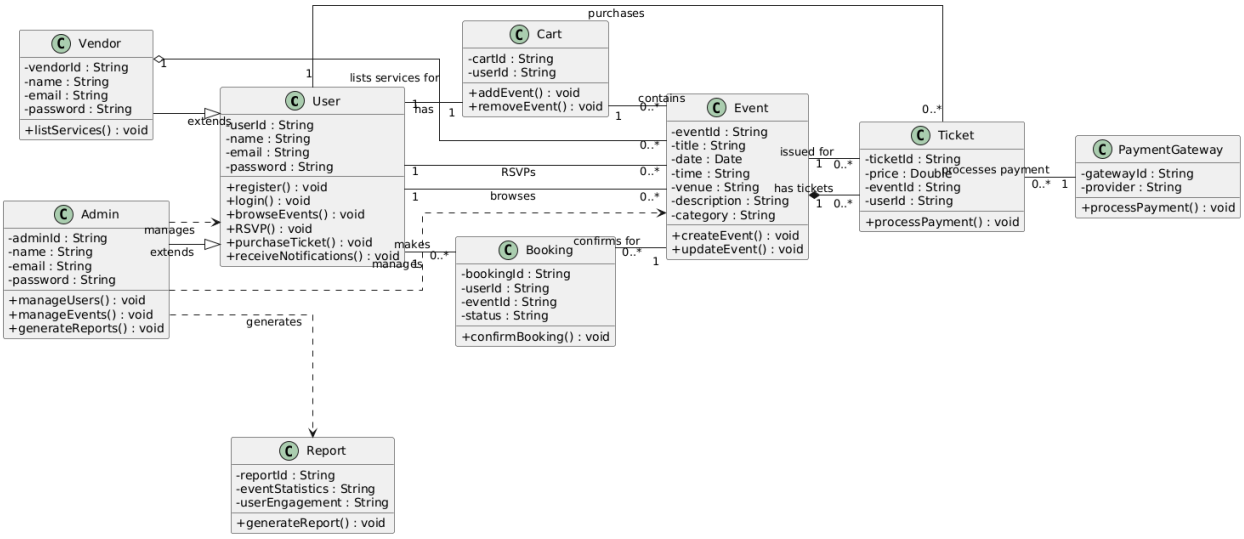



Figure 2: Class Diagram

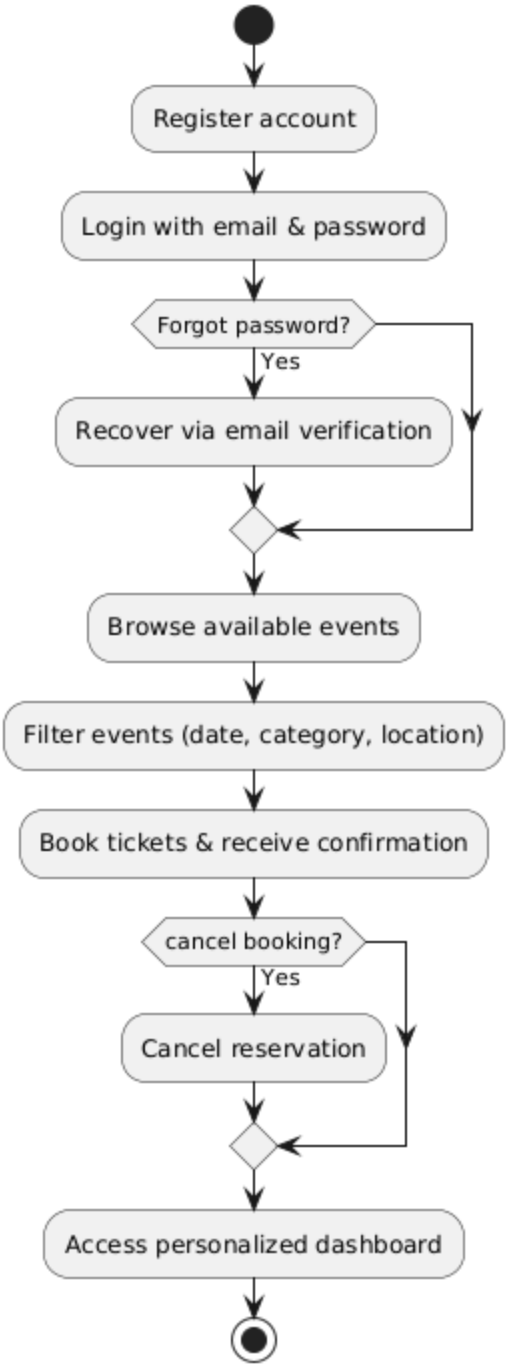



Figure 3: Activity Diagram

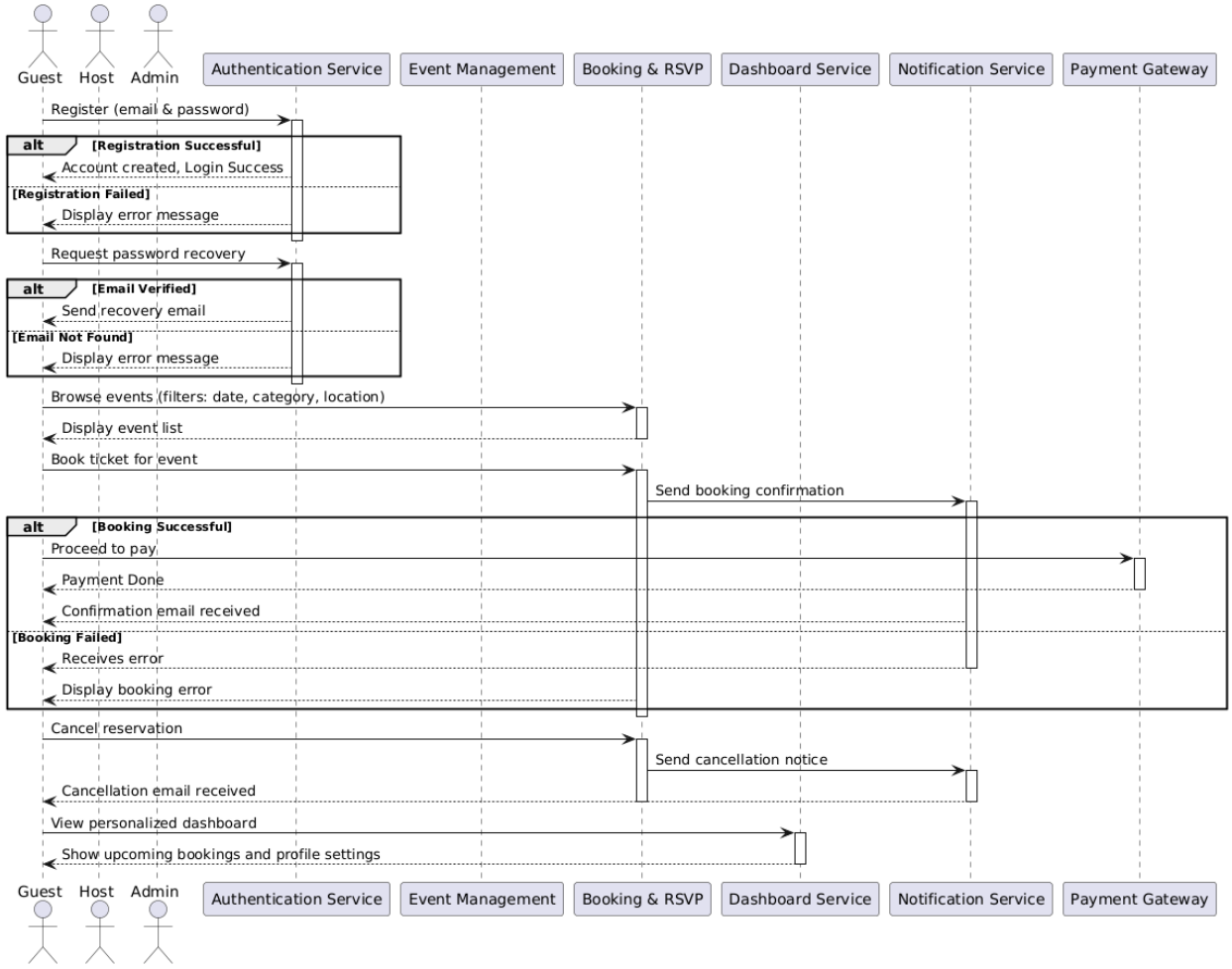



Figure 4: Sequence diagram

# 2.3 Requirement Validation

## 2.3.1 Requirement Reviews

A review board evaluated the documented requirements to ensure alignment with business objectives. Key feedback included:

- Emphasizing mobile responsiveness for increased accessibility.
- Ensuring the checkout process is seamless with multiple payment gateways.

## 2.3.2 Prototyping

Wireframes and UI mockups were developed using **Figma** and tested with a focus group. Feedback suggested the following improvements:

- A separate section for trending/popular events
- Enhancing the search filter experience with real-time suggestions

### 2.3.3 Test-Case Generation

Test cases were generated for critical functionalities:

- **User Registration & Login:** Verify authentication with valid and invalid credentials.
- **Event Creation:** Ensure that all required fields must be filled before submission.
- **Booking Process:** Validate payment success and failure handling.
- **Notification System:** Check for timely event reminders and alerts.

## 2. 4. Requirement Management

### 2. 4.1 Requirement Change Management

A **Change Control Board (CCB)** was established to evaluate proposed modifications. Changes were documented, and their impact was analyzed before approval. Examples of changes handled:

- Introducing a 'Featured Events' section based on user engagement.
- Adding multi-language support for broader accessibility.

### 2. 4.2 Requirement Evolution

The system will evolve based on:

- **User Feedback:** Regular surveys and analytics to identify feature gaps.
- **Technological Advancements:** Exploring AI-powered recommendations and smart event categorization.
- **Market Trends:** Adapting to new industry demands, such as virtual event hosting capabilities.

## 2. 5. Quick Design

A preliminary design will be created to provide a basic structure of the application. This will include:

- **Wireframes and UI Mockups**: Created using **Figma** to outline the user journey.
- **Database Schema**: MySQL database design to handle user, event, and booking data.
- **Navigation Flow**: Defining how users will navigate through the app, from event discovery to ticket booking.

# 2. 6. Building the prototype

## Tools & Tech Stack

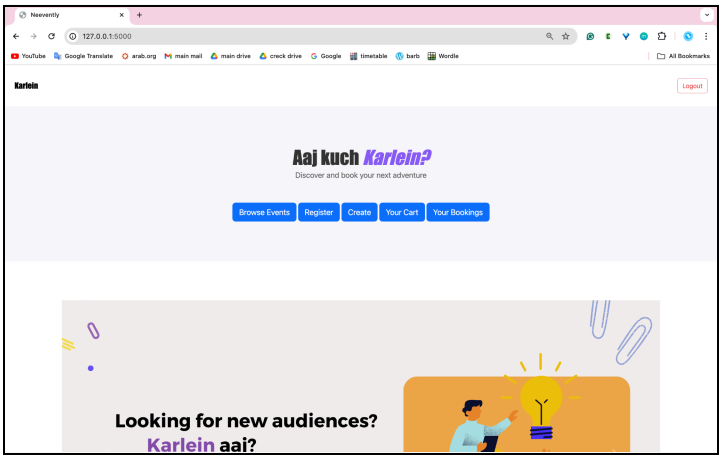
### Frontend Development:

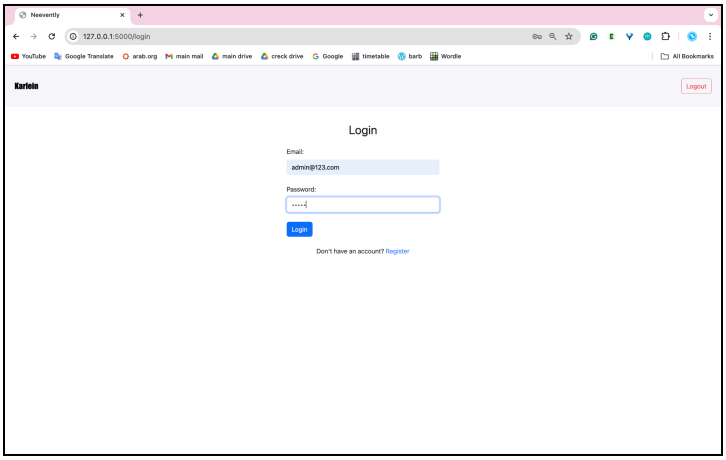
The frontend will be built using **HTML, CSS (Tailwind/Bootstrap), and JavaScript (Alpine.js/Vanilla JS)** for an interactive and visually appealing interface. **Jinja2 templating** in Flask will be used to dynamically render content based on user data. The key pages will include a **homepage** showcasing featured events, an **event details page** with registration options, and an **admin dashboard** for event creation and management.

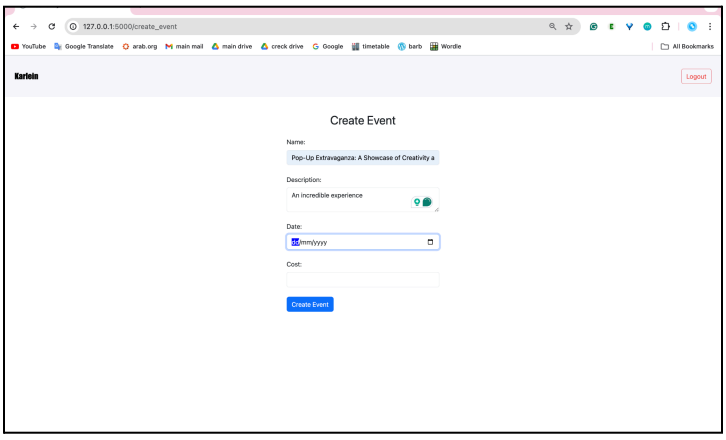
- *HTML, CSS (Tailwind/Bootstrap)* – For clean and modern UI.
- *JavaScript (Alpine.js / Vanilla JS)* – For basic interactivity (e.g., form validation, dynamic content loading).
- *Jinja2 Templating (Flask)* – To dynamically render content from the backend.

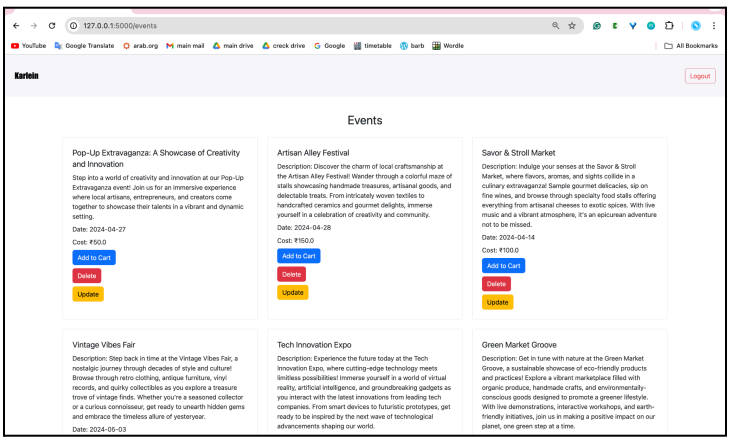
**Frontend Features in Prototype:**

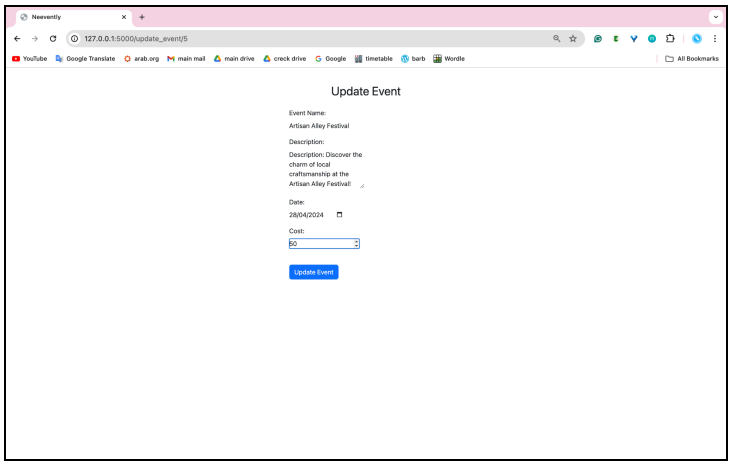
- Home page with menu, upcoming event posters, etc.
- Event detail pages with descriptions, hosting & registration options.
- Admin dashboard for event creation and management.

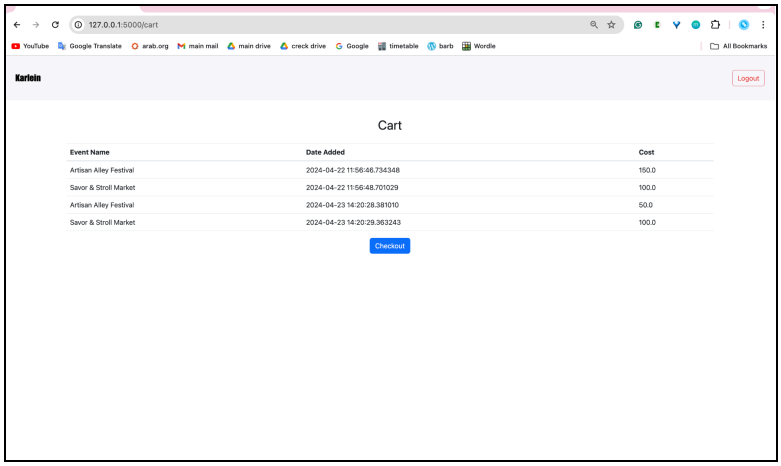


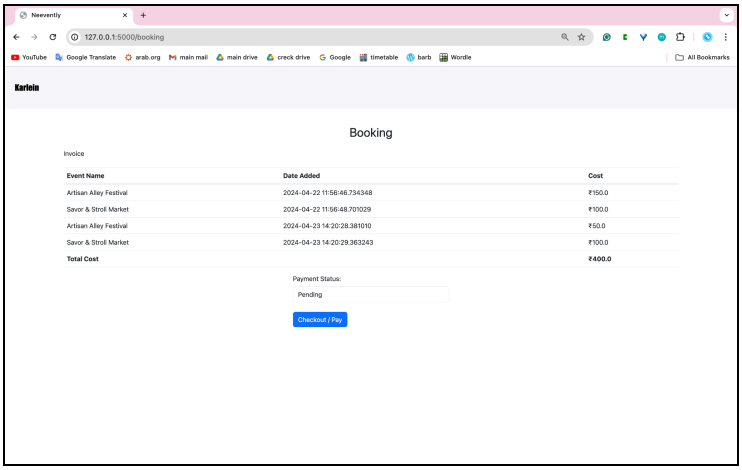






**Backend Development (Flask Framework)**

Flask will serve as the backend framework, handling routing, authentication, and business logic. **Flask-Login** will be used for managing user authentication (for both attendees and organizers), while **Flask-WTF** will enable secure form handling. Database interactions will be managed using **Flask-SQLAlchemy**, and **Flask-Migrate** will handle database migrations, ensuring easy updates to the schema.

The backend will support **event CRUD operations**, allowing organizers to create, edit, and delete events. Attendees will be able to register for events, and admins will have a separate dashboard for event moderation.

- *Flask* – Lightweight web framework to handle routing & backend logic.
- *Flask-Login* – For user authentication (organizers, admin & attendees).
- *Flask-WTF* – For secure event creation/update forms.
- *Flask-SQLAlchemy* – ORM for database interactions.
- *Flask-Migrate* – For database migrations.

**Backend Features in Prototype:**

- *User Authentication* – Sign up/login for organizers and attendees.
- *Event Management* – CRUD (Create, Read, Update, Delete) for events.
- *Event Registration* – Attendees can register for events.
- *Admin Panel* – Separate dashboard for event organizers.
- *Payment Integration (Future Feature)* – Placeholder for Stripe/PayPal integration.

**Database Options:**

- *SQLite (For Local Development & Testing):* For quick prototyping, SQLite serves as a lightweight, file-based database. However, it's **not suitable for production** due to limited scalability and concurrency handling.
- *Firebase Firestore (For Real-Time Updates)*: If real-time updates are needed (e.g., tracking available event seats dynamically), **Firebase Firestore** can be integrated as a NoSQL database. It offers automatic syncing across devices, but lacks strong relational querying capabilities. A **hybrid approach** can be used: PostgreSQL for structured data and Firebase for real-time event status updates.

# 2. 7. User Evaluation and Feedback

Gather user feedback on the **event management platform prototype** to identify usability issues, missing features, and areas for improvement. The goal is to ensure an intuitive and seamless experience for both event organizers and attendees.

## Approach:

1. Conduct usability testing with a small group of users (event organizers, vendors, and attendees).
2. Monitor user interactions with key features like event creation, ticket booking, RSVP tracking, and payment processing.
3. Identify pain points and collect qualitative and quantitative feedback.

## Tools:

1. Usability Testing Platforms
    - *Maze, UserTesting* → Conduct interactive usability tests where users perform specific tasks while being recorded for analysis.
    - Use A/B testing to compare different UI designs and workflow structures.

2. Feedback Collection
    - *Google Forms, Typeform* → Collect structured feedback from users through surveys after testing the platform.

- Questions focus on ease of navigation, feature utility, and overall experience.
- Open-ended responses help identify additional needs.

3. Bug Tracking & Issue Management
    - *Jira, Trello, GitHub Issues* → Log and categorize bugs, usability issues, and feature requests.
    - Assign severity levels (e.g., critical, major, minor) to prioritize fixes.

4. Analytics & UI Behavior Tracking
    - *Hotjar* → Heatmaps and session recordings help analyze user behavior on the platform.
    - *Google Analytics* → Tracks metrics like page engagement, bounce rates, and conversion rates (e.g., percentage of users completing event bookings).

## 2. 8. Refinement and Reiteration

Enhance the prototype based on user feedback, optimize performance, and add missing functionalities before final deployment.

### Approach:

- Implement an **agile** development approach to address feedback in iterative sprints.
- Focus on improving user workflows, reducing friction in booking and event management, and enhancing performance.
- Optimize the database and backend logic to improve scalability and response times.

### Tools:

1. Version Control
    - *Git, GitHub/GitLab* → Manage code updates, feature branches, and version releases.
    - Track changes efficiently and allow multiple developers to collaborate.

2. Agile Project Management
    - *ClickUp, Trello, Jira* → Organize feature improvements and bug fixes into sprints.
    - Assign tasks to developers, monitor progress, and set deadlines for releases.

3. CI/CD Pipelines
    - *GitHub Actions, Jenkins* → Automate testing, code integration, and deployment for continuous delivery.
    - Ensures new updates do not break existing functionality.

4. Testing & Performance Optimization
    1. *Unit & Integration Testing*: Verify that individual units of the application (like login, event creation, cart functionality) and their integration (e.g., creating an event and adding it to the cart) work as intended.
        - Tools: `unittest` or `pytest` (for Flask backend) and `Flask-Testing` extension for simulating routes and sessions

| Test Cases | Description |
|---|---|
| User Authentication | Test login with valid/invalid credentials |
| Event Creation | Admin can add a new event and retrieve it from DB |
| Cart Operations | Add/remove events to cart; check total price updates |
| Booking | Submitting payment status updates booking info |

    2. *Load & Stress Testing*: Evaluate how well the platform handles concurrent users and high-volume traffic, particularly around critical actions (browsing, booking, payment).
        - Tools: **Apache JMeter**, GUI-based, good for simulating HTTP requests with parameters, and **k6 (by Grafana)**: Lightweight, script-based modern tool ideal for CI integration.

| Scenario | Metrics to Monitor |
|---|---|
| 100 concurrent users browsing | Response time < 500ms |
| 500 users submitting bookings | Server uptime, CPU/memory usage |
| 1000 simultaneous login attempts | Auth system stability |

        - Key metrics: response time, error rate, throughput (requests/sec), system and resource utilization (CPU/RAM).

## 2.9. Finalization and Deployment

Convert the **refined prototype** into a **fully functional event management system** and deploy it for real-world use.

Approach:

- Perform security and optimization checks before launch.
- Set up cloud-based hosting and ensure high availability.
- Provide documentation and support resources for users.

Tools:

1. Code Review & Optimization
    - *SonarQube, ESLint* → Analyze code for bugs, vulnerabilities, and efficiency improvements.
    - Ensure clean, maintainable code before deployment.

2. Security Testing
    - Scan for security vulnerabilities, SQL injection risks, and authentication flaws.
    - Implement security best practices, including *role-based access control (RBAC) and data encryption.*

3. Hosting & Deployment
    - *AWS, Vercel, DigitalOcean* → Deploy the platform on a cloud infrastructure with auto-scaling capabilities.
    - Use *Docker & Kubernetes* for containerized deployment, ensuring *high availability.*
      s
4. Documentation & Support
    - *Confluence, Notion* → Maintain detailed documentation covering system architecture, API endpoints, and troubleshooting guides.
    - Provide a *helpdesk/support channel* for users to report issues post-launch.

# 3. Advantages of Using the Prototype Model for this Project

## 3.1 Better Understanding of User Needs

As the target users (small business owners and event organizers) may not be tech-savvy, a working prototype helps them understand how the system works, allowing them to refine their requirements effectively.

## 3.2 Increased User Involvement

Since this app requires maximum user interaction, gathering continuous feedback ensures that the final product aligns with the users' expectations and improves customer satisfaction.

### 3.3 Early Detection of Issues

By iteratively testing and refining the prototype, potential usability issues, performance bottlenecks, or missing functionalities can be identified and resolved before full-scale development.

### 3.4 Cost and Time Efficiency

Instead of spending resources on building a complete system with unknown requirements, the prototype model allows for incremental development, reducing rework and ensuring that only necessary features are implemented.

### 3.5 Risk Mitigation

This model helps minimize the risk of developing a product that does not meet user expectations. By incorporating user feedback early, the project remains adaptable and relevant.

# 4. AI Application

**4.1. Post-Event Insights**: AI analyzes feedback and attendance data to evaluate event success and plan improvements.

**4.2. Personalized Recommendations**: AI-powered systems suggest events based on user preferences, boosting engagement.

**4.3. Chatbots and Virtual Assistants**: Real-time support and ticket booking through AI chatbots improve user experience.

**4.4. Predictive Analytics**: Forecast attendance and optimize resources with AI-driven predictions.

**4.5. Crowd Monitoring**: Real-time analysis of crowd density and movement ensures safety and smooth operations.

# 5. Conclusion

The Prototype Model is the most suitable approach for developing the **Event Management App for Small Businesses and Pop-ups**, given the uncertainty of initial user requirements and the

high need for user engagement. By following an iterative process, the app will evolve in alignment with real-world needs, ensuring a well-designed, user-friendly, and market-ready product upon final release.