**YOYZI**

**Submitted by:**

Isha Chavada – 241H0660039

Abhivyakti Johari - 241H0660003

For the award of the degree of MCA

**Under the supervision of**

Ms. Juli Jyotika Tudu



Department of Mathematics and Computing

**Banasthali Vidyapith**

**Banasthali – 304022**

**Session: 2024-25**

INDEX:

1. **Introduction**

1.1 Purpose

1.2 Scope

1.3 Definition, Acronyms, and Abbreviations

1.4 References

**2. Functional Requirements**

2.1 Organizer Features

2.2 User Features

2.3 General Features

**3. Non-Functional Requirements**

3.1 Performance

3.2 Scalability

3.3 Usability

3.4 Security

**4. System Design Constraints**

**5. Assumptions and Dependencies**

**6. Diagrams and Models**

6.1 Sequence Diagram

6.2 Data Flow Diagram (DFD)

6.3 ER Diagram

6.4 Data Model Diagram

**Introduction**

**Purpose**

The purpose of VOYZI is to develop an event organization app that connects users to local and mainstream events. The app caters to both event organizers and users, providing tools for event creation, management, discovery, and booking. By combining a sleek user interface and advanced features, VOYZI aims to become a go-to platform for event enthusiasts.

**Scope**

VOYZI allows:

* **Organizers**: To create, manage, and promote events, monitor ticket sales.
* **Users**: To discover events, book tickets, and engage with real-time updates and notifications.

Key features include:

* Event categorization by type (e.g., "Art and Crafts", "Workshops", "Gatherings" etc.).
* Secure payment integration for ticket bookings.
* Real-time data updates powered by Firebase.
* Intuitive design for seamless user interaction.

**Definitions, Acronyms,**

**and Abbreviations**

* **React Native**: Framework for cross-platform mobile app development.
* **Firebase**: Backend-as-a-Service platform for authentication, real-time database, and cloud notifications.
* **Events**: Events tailored to younger audiences, such as pop-up markets, street-style gatherings, and music battles etc.

**System Overview**

VOYZI is designed to:

* Connect organizers with potential attendees efficiently.
* Allow users to discover events based on preferences like category, location, and price.
* Provide a seamless experience for event ticket booking and management.

The system is role-based with the following key profiles:

1. **Organizer**: Manages event creation, promotion, and analytics.
2. **User**: Explores, saves, and books events, while receiving notifications.

**Functional Requirements**

**3.1 Organizer Features**

* **Create Events**: Add event details (name, description, category, date, time, location, and ticket pricing).
* **Manage Events**: Edit, update, or delete events, monitor attendee lists, and view ticket availability.
* **Analytics Dashboard**: View data like ticket sales, attendee demographics, and revenue insights.
* **Notifications**: Send announcements, event reminders, and updates to users.
* **Payment Tracking**: Monitor payments received for tickets.

**3.2 User Features**

* **Event Browsing**: Search and filter events by type, location, date, or price.
* **Ticket Booking**: Securely purchase tickets with payment gateway integration.
* **Favorite Events**: Save events for future reference.
* **Real-Time Notifications**: Get alerts for event updates, reminders, or cancellations.
* **User Profile Management**: Maintain booking history and saved preferences.

**3.3 General Features**

* **Login/Sign-up**: Using email, phone number, or social accounts with OAuth 2.0.
* **Search and Filter**: Flexible search options by keywords, categories, or filters like date and price range.
* **Feedback Mechanism**: Users can leave reviews and ratings for events.
* **Cross-Platform Support**: Available on both Android and iOS devices.

**Non-Functional Requirements**

**4.1 Performance**

* Data synchronization for events and ticket updates should occur in real-time.

**4.2 Scalability**

* The system should be capable of scaling to accommodate a growing number of users and events.
* Firebase’s auto-scaling capabilities will be utilized for handling database and notification loads.

**4.3 Usability**

* The UI/UX design should prioritize intuitive navigation with minimal learning curve.

**4.4 Security**

* **Data Protection**: Encryption of user and payment data during storage and transmission.
* **Authentication**: Secure login with two-factor authentication.
* **Payment Security**: Integration of PCI-compliant payment gateways.

**System Design**

**Constraints**

* **• Frontend:** Developed using React Native (with Expo) for seamless cross-platform compatibility.  
  **• Backend:** Uses Firebase services to handle backend logic such as authentication, data storage, and real-time updates — eliminating the need for a separate server-side language.  
  **• Database:** Firebase Firestore (or Real-Time Database, depending on what you're using) for instant syncing and easy integration with React Native. Third-Party Integrations:
  + Payment gateway for secure transactions.
  + Google Maps API for event location tagging and navigation.

**Assumptions and Dependencies**

* Users will have stable internet connectivity to access app features.
* Firebase will handle key backend functionalities, including real-time data sync and notifications.
* Payment gateways will ensure secure and seamless ticket booking.
* Availability of development resources like React Native, Firebase SDK, and Java tools.

**Sequence Diagram**

**Definition:**  
A sequence diagram shows the step-by-step interactions between objects, systems, or actors over time to complete a specific task or use case. It is part of UML (Unified Modeling Language) and focuses on the flow of messages.

**Purpose:**

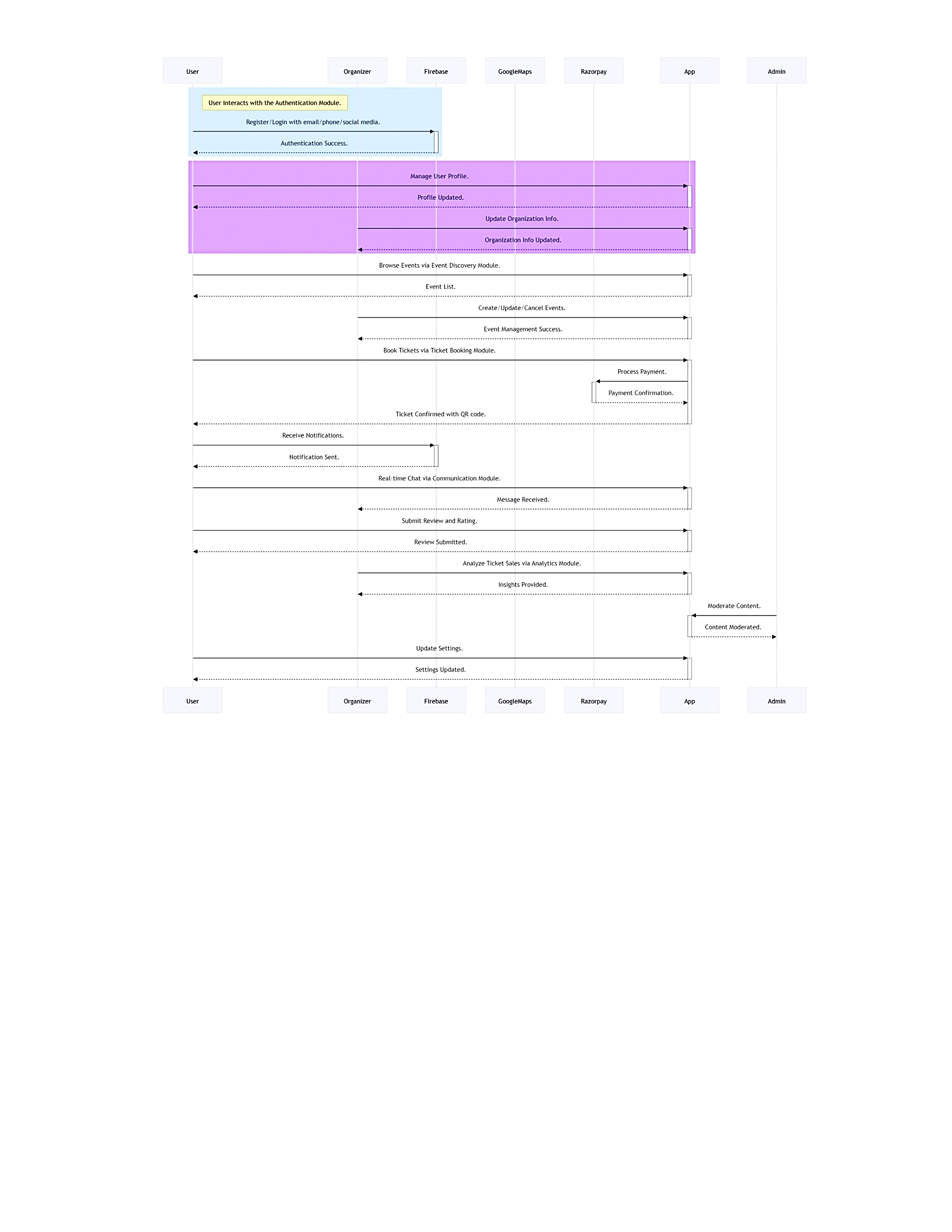
* To visualize the dynamic behavior of a system.
* To understand how components interact to achieve a specific goal.

**Example Use Case:**

* *Event Ticket Booking*: A user searches for an event, views detail, selects tickets, pays, and receives confirmation.

**Key Elements:**

* **Actors**: Represent external users or systems that interact with the app (e.g., User, Organizer, Firebase).
* **Objects**: Represent entities within the system (e.g., Database, Payment Gateway).
* **Messages**: Arrows indicating the interactions or requests between actors and objects.
* **Lifelines**: Dotted lines extending downward that represent the lifespan of each object/actor.
* **Activation Bars**: Represent when an object/actor is actively performing a task.



**Data Flow Diagram**

**Definition:**  
A flow diagram, or **data flow diagram (DFD)**, represents how data moves through a system. It highlights the flow of information between processes, users, and external entities.

**Purpose:**

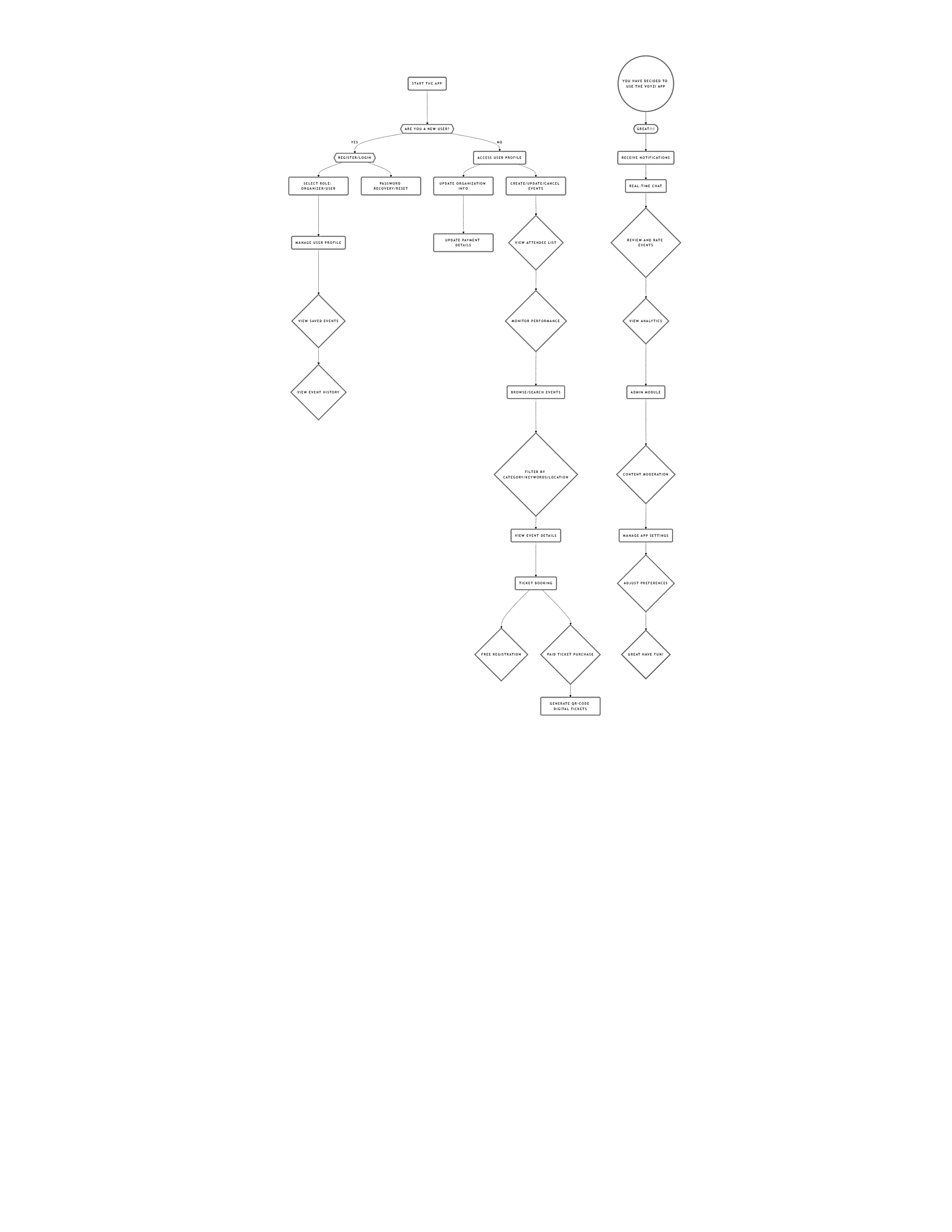
* To understand how data is input, processed, stored, and output.
* To design a system with an optimal flow of information.

**Example:**

* A flow diagram for *event creation* could show how the Organizer enters details, the system validates the input, and data is saved to the database.

**Key Elements:**

* **Processes**: Represent tasks or functions (e.g., "Search Events" or "Book Tickets").
* **Data Stores**: Represent where data is stored (e.g., Database, Cache).
* **External Entities**: Represent actors interacting with the system (e.g., User, Payment Gateway).
* **Data Flows**: Arrows showing how data moves between entities and processes.



**Entity-Relationship (ER) Diagram**

**Definition:**  
An ER diagram models the relationships between entities in a database. It is a blueprint for how data will be stored and connected.

**Purpose:**

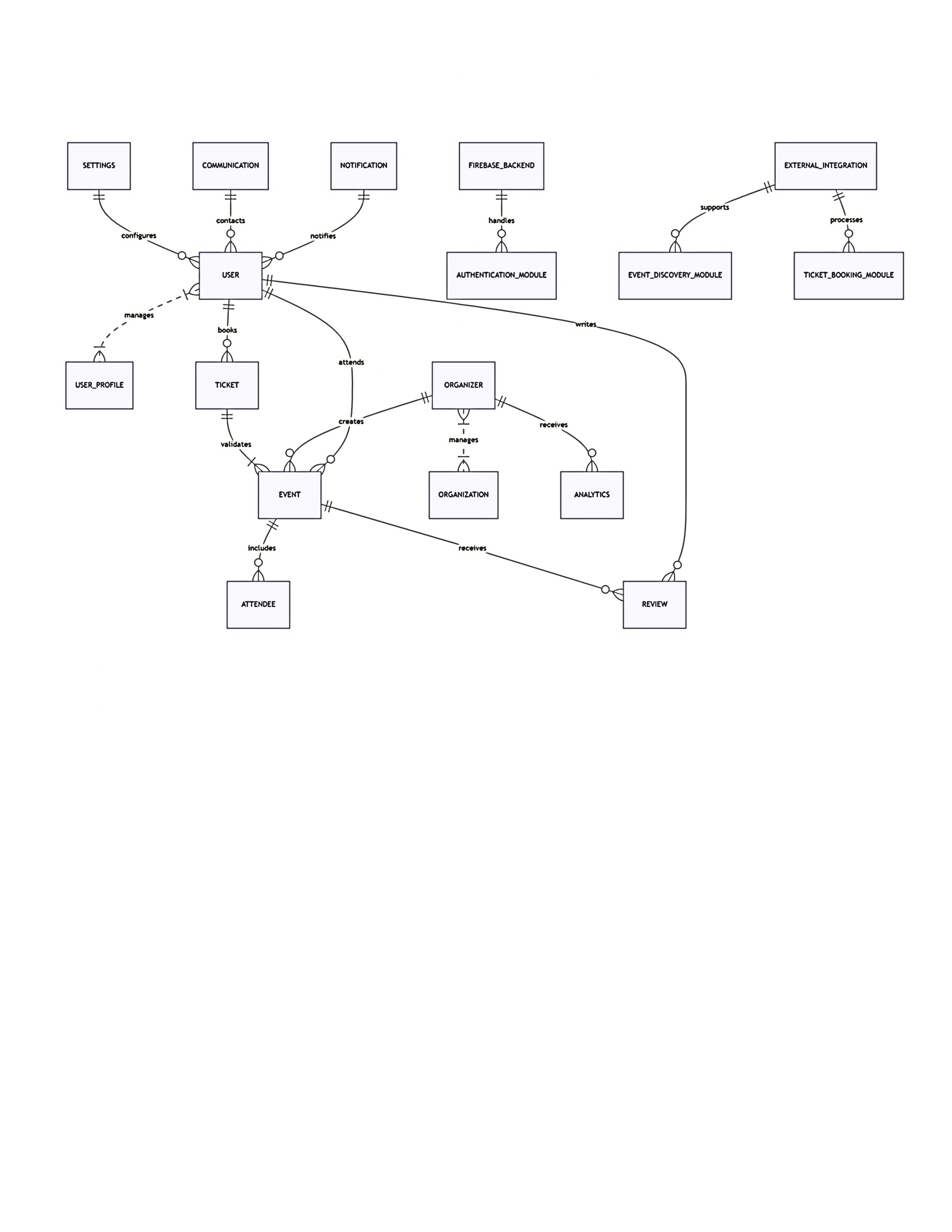
* To design the database schema for the application.
* To ensure efficient data storage and retrieval by clearly defining relationships.

**Example:**  
For VOYZI, an ER diagram would define:

* User table with attributes like UserID, Name, Email.
* Event table with attributes like EventID, Name, Location.
* A Booking table with a many-to-many relationship between Users and Events.

**Key Elements:**

* **Entities**: Represent tables in the database (e.g., User, Event, Booking).
* **Attributes**: Represent fields within an entity (e.g., UserID, EventID, Date, etc.).
* **Relationships**: Show how entities are connected (e.g., a User can book multiple Events).



**Data Model Diagram**

**Definition:**  
A data model diagram illustrates the structure and relationships within a database. It extends an ER diagram by detailing primary keys, foreign keys, and normalization.

**Purpose:**

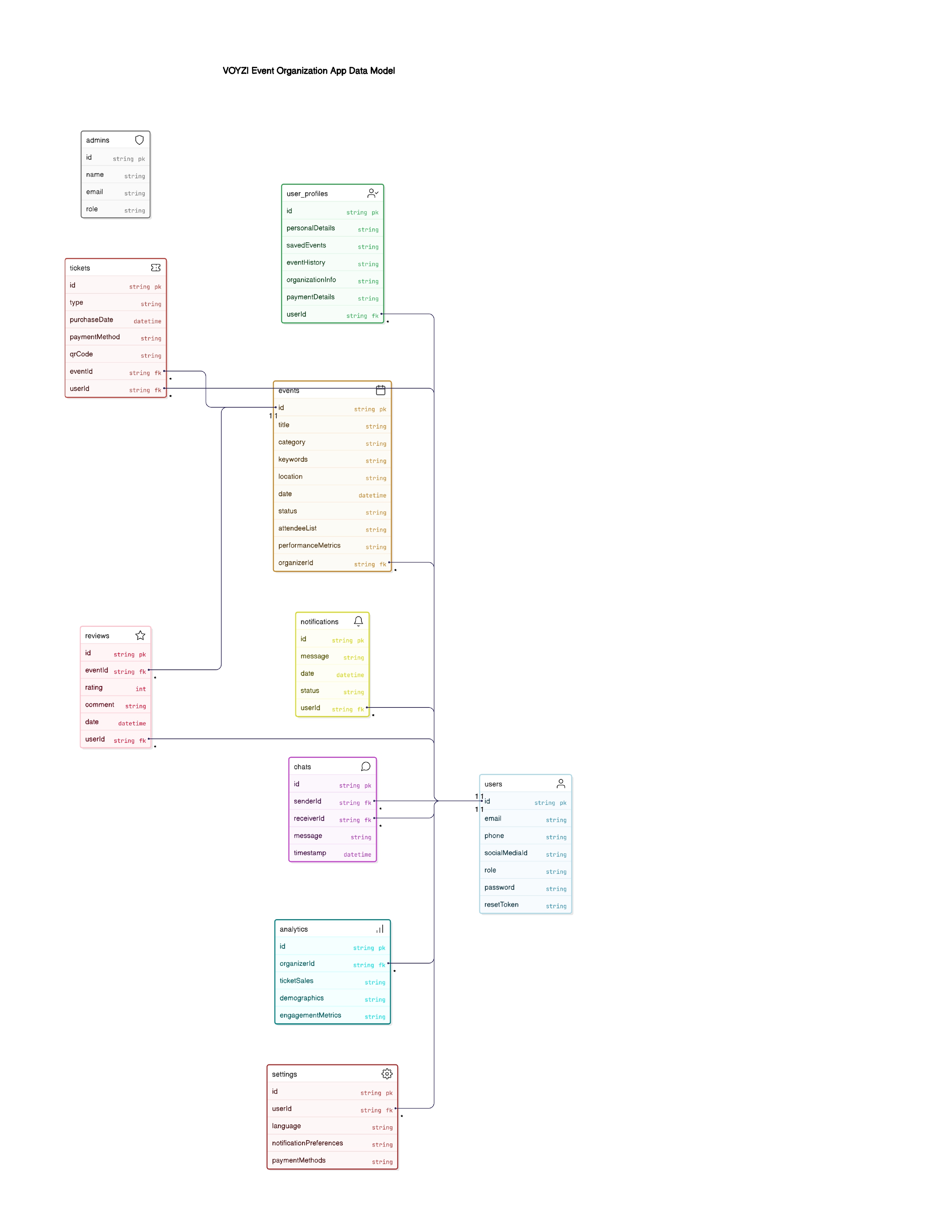
* To map out the database design.
* To ensure data integrity and normalization.

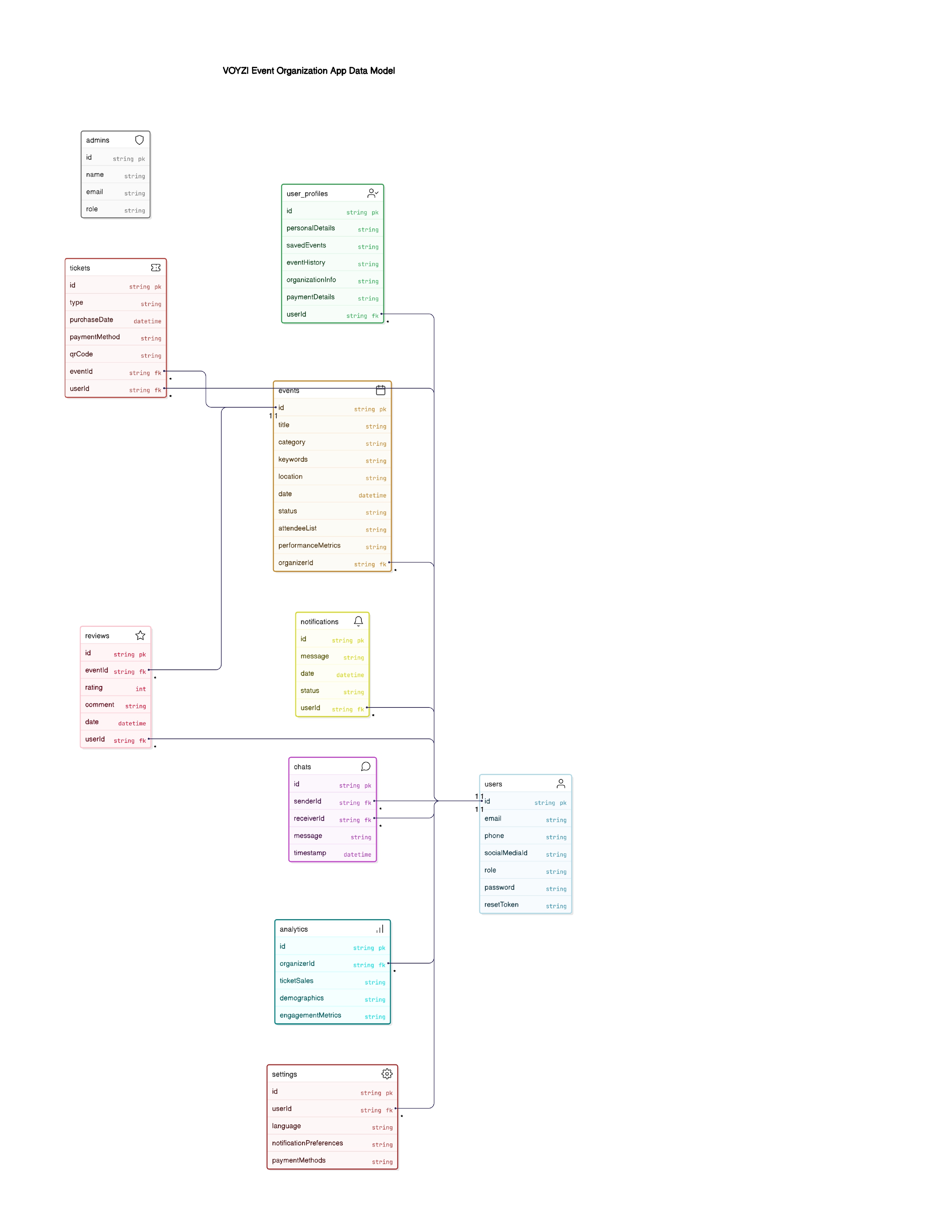
**Example:**  
A logical data model for VOYZI might specify:

* UserID as the primary key for the User table.
* EventID as the primary key for the Event table.
* BookingID as the primary key for the Booking table, with UserID and EventID as foreign keys.

**Key Types:**

1. **Conceptual Data Model**: High-level, focusing on what data is stored (e.g., User, Event).
2. **Logical Data Model**: More detailed, specifying attributes and relationships without considering database technology.
3. **Physical Data Model**: The most detailed, defining tables, columns, data types, and constraints for implementation.





**Use Case Diagram**

**Definition:**  
A Use Case Diagram represents the functional requirements of a system from the end-user's perspective. It illustrates the interactions between users (actors) and the system through various use cases (functions).

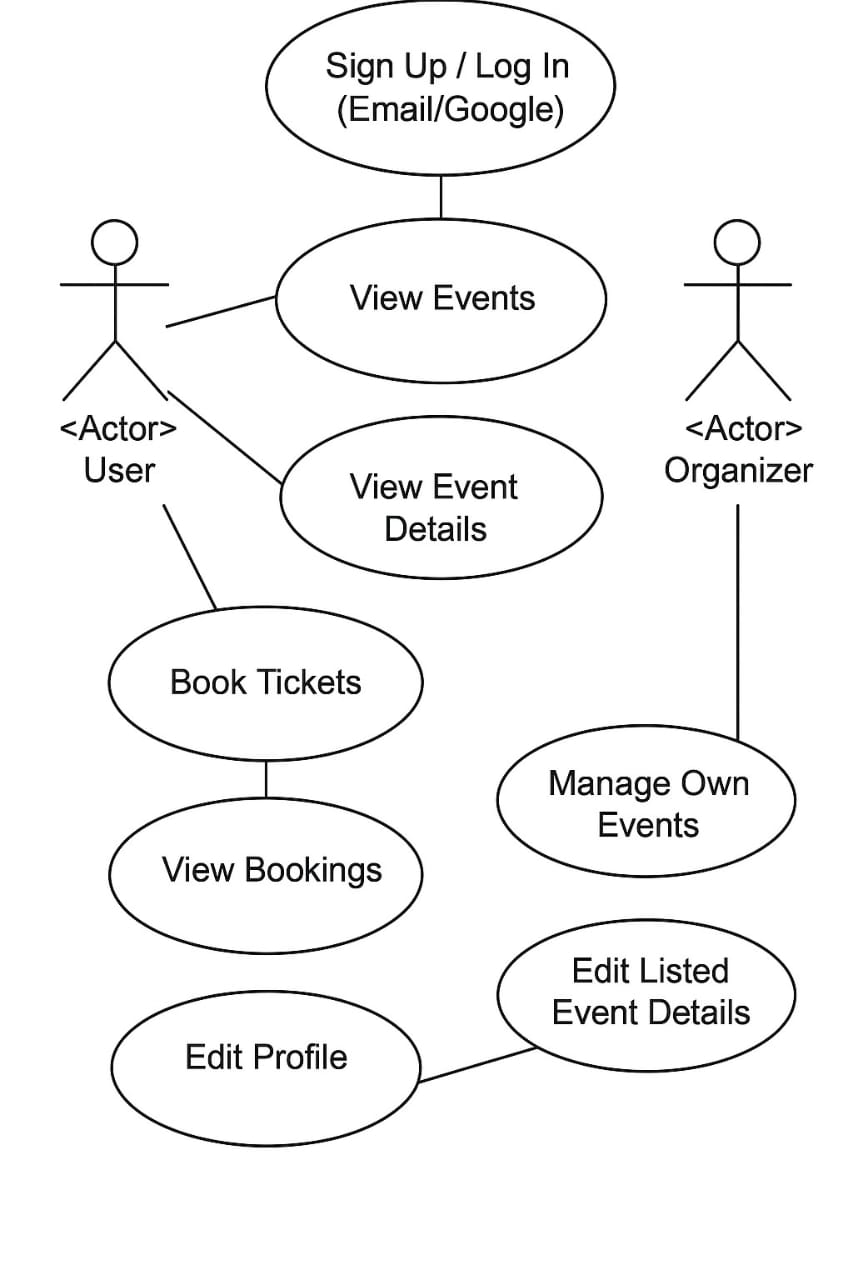
**Purpose:**

• To visualize how users interact with the system.  
• To identify the core functionalities of the application.  
• To serve as a foundation for system design and development.

**Example:**  
For **VOYZI**, a use case diagram would include:  
• Actors like **User** and **Organizer**.  
• Use cases such as **Register/Login**, **Browse Events**, **Book Ticket**, **Create Event**, etc.  
• Relationships showing which actor performs which actions.

**Key Elements:**

**• Actors:** Represent users or external systems that interact with the application (e.g., User, Organizer).  
**• Use Cases:** Represent the system's functions or services (e.g., Book Ticket, View Event, Make Payment).  
**• System Boundary:** A box that defines the scope of the application being modeled.  
• Relationships:  
**– Association** (lines connecting actors and use cases)  
**– Include** (common steps reused across multiple use cases) **– Extend** (optional or conditional behaviors)



**How These Diagrams Interconnect in a Project**

1. **Sequence Diagrams** guide the dynamic behavior of the system for specific use cases, influencing how data flows and is processed.
2. **Flow Diagrams** provide a higher-level view of processes and data movement, which aids in designing the system architecture.
3. **ER Diagrams** and **Data Models** ensure the database aligns with system requirements and supports all use cases and data flows.

**Conclusion**

The VOYZI **Event Organization App** is designed to bridge the gap between event organizers and attendees by providing a seamless platform for event discovery, management, and ticket booking. Through the use of **React Native, Java, and Firebase**, the system ensures efficient performance, scalability, and a user-friendly experience.

The **Software Requirement Specification (SRS)** document outlines the functional and non-functional requirements, system constraints, and architectural design of the application. Key system models, including **Sequence Diagrams, Flow Diagrams, ER Diagrams, and Data Models**, provide a structured view of how data and processes interact within the system. These diagrams play a crucial role in defining the flow of information, ensuring data integrity, and optimizing the overall functionality of VOYZI.

By following the detailed specifications and system models provided in this document, the development team will have a clear roadmap for implementing a robust, scalable, and intuitive event management platform. This SRS serves as a foundation for future enhancements and ensures that VOYZI meets the needs of both event organizers and attendees efficiently.

**References**

* https://firebase.google.com/docs
* https://reactnative.dev/docs/getting-started