Project Proposal

Project Title

Tour Guide App (Guidee)– Explore Egypt Smartly

Overview

- The Tour Guide App is an intelligent mobile application designed to help users explore tourist attractions in Egypt in an organized and convenient way.
  Through a simple interface and an interactive map, users can view the best nearby places, add them to their visit list, read reviews from other visitors, and receive notifications reminding them of upcoming visits and travel plans.

- The main goal of the application is to enhance the tourism experience in Egypt by organizing trips and providing reliable information in one accessible platform.

Objectives

- Facilitate the process of exploring tourist attractions for users.
- Provide accurate details, images, and reviews for each location.
- Enable users to create and manage their daily trip plans.
- Send notifications to remind users of upcoming visits or suggested destinations.
- Offer a smooth and fast user experience with full support for both Arabic and English languages.

Scope

Technologies Used:
- Frontend: Flutter
- Backend & Database: Supabase (for authentication and data storage)
- Cloud Services: Google Cloud (for APIs and media storage)
- Local Storage: Hive (for caching and offline data)

User Type:
- Visitor/User: Can explore places, view details and reviews, add favorites, and create visit plans.

Core Features:
- Display a list of tourist attractions categorized by type (historical, natural, cultural, etc.)
- Detailed information about each place including images and descriptions.
- Create and manage trip plans (with reminders).
- Receive notifications for upcoming visits or saved plans.
- Multilingual interface (Arabic and English).

Timeline (Gantt Chart Summary)

Phase	Deadline	Description
Project Planning & Management	21/09/2025	Define project goals, structure, and task distribution.
Literature Review	21/09/2025	Research existing tour guide applications and analyze their strengths and weaknesses.
Requirements Gathering	21/09/2025	Identify user needs, features, and system requirements.
System Analysis & Design	21/09/2025	Design the system architecture, data model, and user interface.
Implementation	15/11/2025	Develop the mobile app using Flutter, Supabase, and Google Cloud.
Final Presentation & Testing	20/11/2025	Conduct app testing, fix issues, and prepare final documentation and presentation.

Milestones

- Project Proposal Approval – Confirmed concept and team roles.
- System Design Completed – Database structure, UI sketches, and technical stack decided.
- First Prototype Developed – Basic navigation and data flow implemented.
- Testing Phase – Functional and usability testing.
- Final Submission – Full documentation and presentation delivered.

Team Overview

- The Tour Guide App development team consists of six Flutter developers collaborating equally on all stages of the project. There is no formal team leader — all decisions are made collectively, and tasks are distributed based on skills and interest.

1. Uneven Task Distribution
Risk: Some members may have a heavier workload than others.
Mitigation: Tasks will be discussed and assigned collectively in meetings to ensure balance and fairness.

2. Time Management
Risk: Team members may face delays due to overlapping academic schedules.
Mitigation: Weekly progress check-ins will be held to track updates and reassign delayed tasks if needed.

3. Communication Gaps
Risk: Misunderstanding between team members about requirements or deadlines.
Mitigation: Clear documentation will be maintained in Notion, and team meetings will be held regularly.

4. Technical Challenges
Risk: Integration issues between Flutter and Supabase or unfamiliar packages.
Mitigation: Team members will research solutions together, and allocate time for testing and debugging.

5. Version Control Conflicts
Risk: Code conflicts during GitHub merges.
Mitigation: Members will follow a clear branching strategy and commit frequently with proper messages.

6. Design Inconsistency
Risk: Different coding or design styles across screens.
Mitigation: A shared design system (colors, fonts, components) will be created in Figma and followed by all members.

KPIs (Key Performance Indicators)

KPIs (Key Performance Indicators)

- App Launch Success Rate: The app should run smoothly without crashes in at least 98% of launches.
- Response Time: Data fetched from Supabase or Google Cloud should load in less than 2 seconds to ensure a smooth experience.
- User Adoption Rate: At least 70% of the targeted users should register or start using the app within the first month.
- User Retention Rate: More than 60% of users should return to use the app after one week.
- Error Rate: The percentage of failed requests or crashes should remain below 2%
- Feature Completion: All planned features should be fully implemented and functional by the end of development.
- UI Consistency Score: The app interface should follow the shared Figma design with at least 95% accuracy.
- Team Productivity: Around 90% of the planned tasks and milestones should be completed on time.
- App Rating (if published): The app should achieve an average user rating of 4.5★ or higher based on feedback.

1. Stakeholder Analysis

- The key stakeholders involved in the Tour Guide App are:
  - **Users (Tourists):** Individuals who use the application to explore tourist attractions, view details, and plan their visits.
  - **System (Application):** The mobile app that provides destination information, personalized suggestions, and notifications.
  - **Developers (Team Members):** The Flutter development team responsible for designing, implementing, and maintaining the app.
  - **Supervisor:** The academic advisor who reviews and evaluates the project's progress and outcomes.

2.1 User Stories

User stories describe how users will interact with the app in short, simple sentences.

- As a user, I want to view nearby tourist attractions so that I can decide where to visit.
- As a user, I want to see detailed information and images of each place so that I can learn more about it.
- As a user, I want to save places I'm interested in so I can plan my future trips.
- As a user, I want to get notifications reminding me about upcoming visits.
- As a user, I want to filter or search for attractions by city or category (e.g., museums, beaches, landmarks).
- As a user, I want to switch between Arabic and English, so I can use the app in my preferred language.

2.2 Use Cases

Use cases describe the main actions users can perform within the system:

- Use Case 1: View Tourist Places
  - Actor: User
  - Description: The user browses a list of tourist attractions and views details about each one.
- Use Case 2: Save Favorite Place
  - Actor: User
  - Description: The user adds a place to their favorites list to view it later.
- Use Case 3: Receive Notifications
  - Actor: User
  - Description: The system sends notifications reminding the user of scheduled visits.
- Use Case 4: View Details
  - Actor: User
  - Description: The user opens a specific place to read information, reviews, and view images.
- Use Case 5: Switch Language
  - Actor: User
  - Description: The user changes the app language between Arabic and English.

3. Functional Requirements

The functional requirements define what the system should do — the features and behaviors that the app must include.

1. The system shall allow users to browse and view tourist attractions.
2. The system shall display detailed information, including description, location, and images for each attraction.
3. The system shall allow users to save attractions to a favorites list.
4. The system shall allow users to add visit reminders and send notifications before the visit date.
5. The system shall allow users to search and filter attractions by location or category.
6. The system shall support two languages — Arabic and English.
7. The system shall store and retrieve user data locally using Hive.
8. The system shall integrate with Google Cloud API and Supabase to fetch and manage data.
9. The system shall work offline for previously viewed data using local storage.
10. The system shall provide a simple and user-friendly interface following UI/UX best practices.

4. Non-Functional Requirements

These requirements describe how the system should perform, not what it does. They focus on quality, performance, and usability.

1. Performance:
   The app should load data and respond to user actions in less than 2 seconds.
2. Usability:
   The user interface should be simple, intuitive, and consistent across all screens, allowing smooth navigation for both Arabic and English users.
3. Reliability:
   The app should work correctly even if the internet connection is slow or temporarily unavailable (using Hive for local data storage).
4. Scalability:
   The system architecture should support future updates, such as adding more destinations or new features, without major code changes.
5. Security:
   User data stored locally or retrieved from Supabase must be protected from unauthorized access.
6. Compatibility:
   The app must run smoothly on both Android and iOS devices.
7. Maintainability:
   The codebase should follow clean architecture and standard Flutter practices, making it easy to update and maintain.
8. Localization:
   All text, labels, and messages should be properly translated and switchable between Arabic and English.

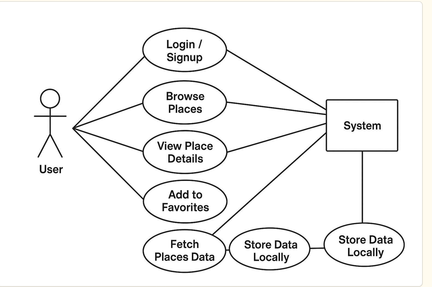
Problem Statement

Tourists often struggle to find reliable, organized, and localized information about tourist attractions in Egypt.
Most existing apps either lack Arabic language support, have outdated data, or require a continuous internet connection.
This makes trip planning harder, especially for local users who want quick and offline access to trusted information.

Objectives

- To create a mobile application that helps users explore tourist attractions easily.
- To provide detailed information, images, and reminders for each place.
- To allow users to store their favorite places and upcoming visits locally (using Hive).
- To support both Arabic and English languages.
- To ensure smooth performance, even with limited internet access.
- To use modern technologies like Flutter, Google Cloud, and Supabase for data handling and scalability.

Usecase Diagram



Use Case Descriptions

1. Use Case: View Tourist Attractions
   - Actor: User
   - Description: The user can browse all available tourist places.
   - Precondition: The app must have internet access or cached data.
   - Postcondition: The list of attractions is displayed.
2. Use Case: View Details of a Place
   - Actor: User
   - Description: The user selects a place to view its description, images, and other info.
   - Precondition: The attraction exists in the database.
   - Postcondition: The details screen is displayed.
3. Use Case: Save Favorite Places
   - Actor: User
   - Description: The user can mark certain places as favorites to access them later.
   - Precondition: The user must be browsing an attraction.
   - Postcondition: The attraction is saved locally in Hive.
4. Use Case: Set Visit Reminder
   - Actor: User
   - Description: The user can set a reminder for a future visit to a place.
   - Postcondition: A notification is scheduled locally.
5. Use Case: Receive Notifications
   - Actor: System
   - Description: The system sends notifications reminding the user of scheduled visits.
6. Use Case: Search or Filter Attractions
   - Actor: User
   - Description: The user can search for attractions by city or category.
   - Postcondition: The list is filtered based on the selected criteria.
7. Use Case: Switch App Language
   - Actor: User
   - Description: The user can switch between Arabic and English from the settings.

Functional & Non-Functional Requirements

Functional Requirements

1. User authentication (Login / Signup)
2. Browse places by category
3. Display place details (photos, description, reviews)
4. Add/remove places from Favorites & Visit List
5. Schedule visits with date selection
6. Receive Push Notifications
7. Fetch data from Google Cloud API
8. Store data locally using Hive

Non-Functional Requirements

1. Performance: Fast data loading and smooth navigation
2. Scalability: Easy to add more categories and places
3. Security: Protect user data (Login/Signup, Local Storage)
4. Usability: Simple and intuitive UI
5. Offline Access: Show cached data without internet
6. Cross-Platform: Android and iOS support

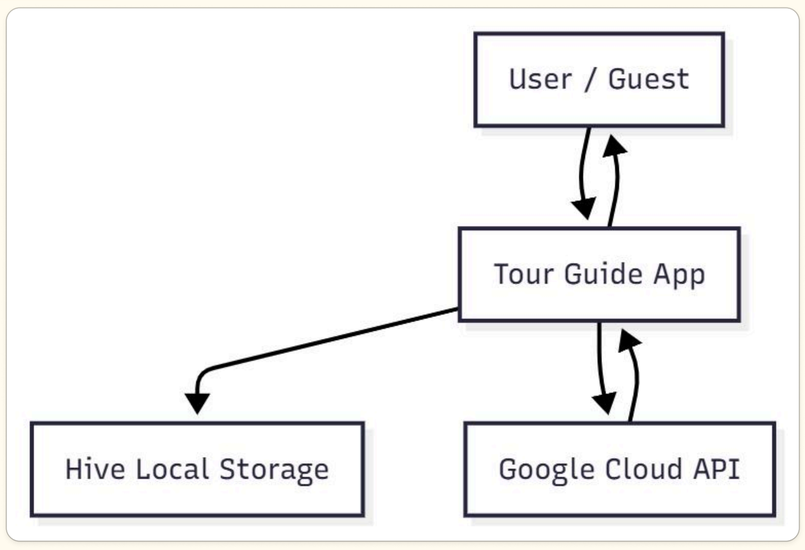
Software Architecture
Architecture Style: MVVM / Clean Architecture

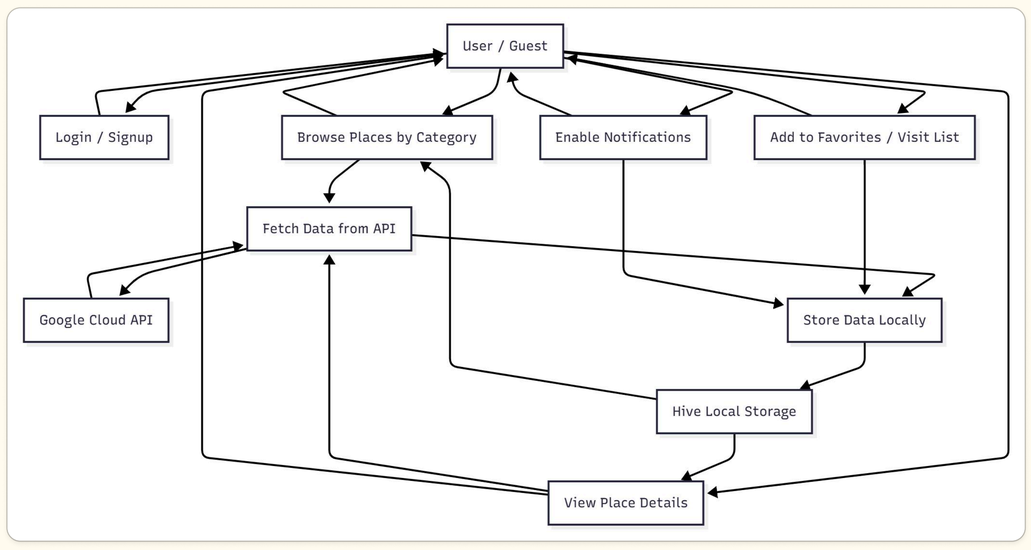
Components
1. Presentation Layer (UI):
   - Screens: Home, Place Details, Favorites, Visit List
   - Handles user interactions
2. ViewModel / Controller:
   - Connects UI with Data Layer
   - Manages app state (GetX / Bloc)
3. Data Layer:
   - Remote Data Source: Google Cloud API
   - Local Data Source: Hive
4. Notification Service:
   - Handles scheduling and sending Push Notifications

Interactions
- UI → Controller/ViewModel → Data Layer (Remote API / Local Hive)
- Visit List → Notification Service → Push Notifications

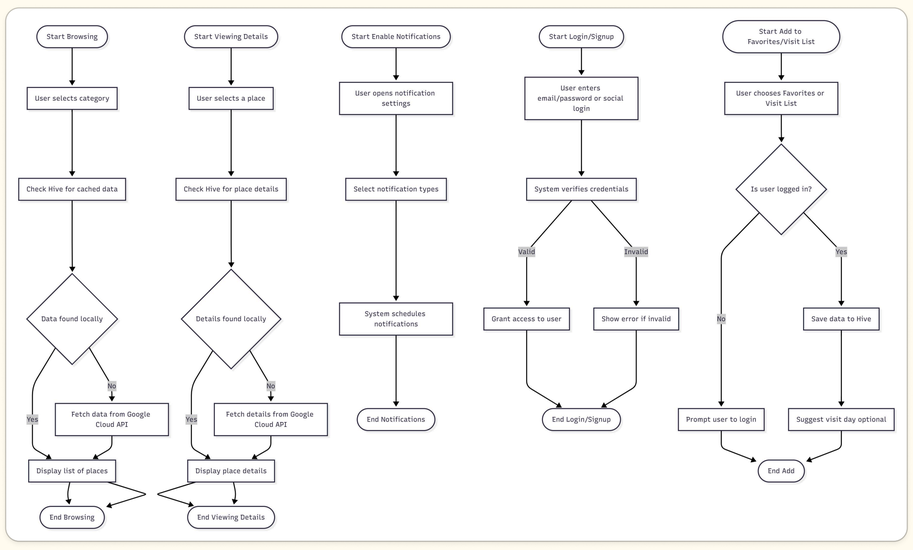
Data Flow & System Behavior

DFD Diagrams

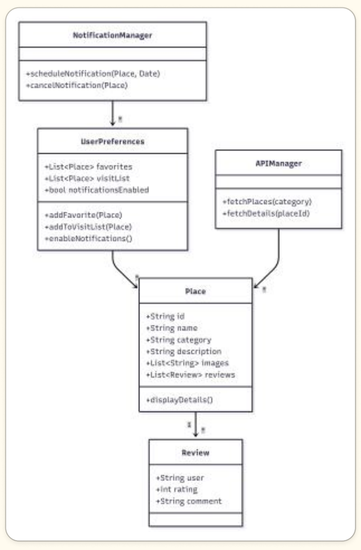




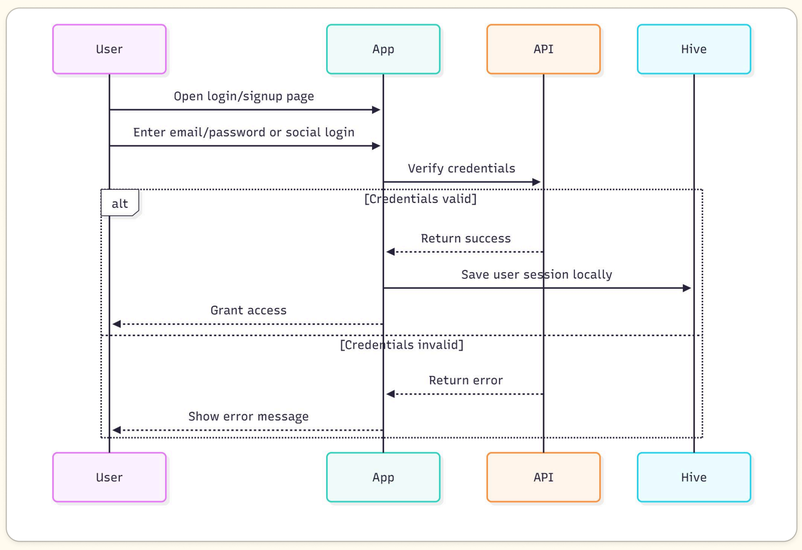
Activity Diagrams

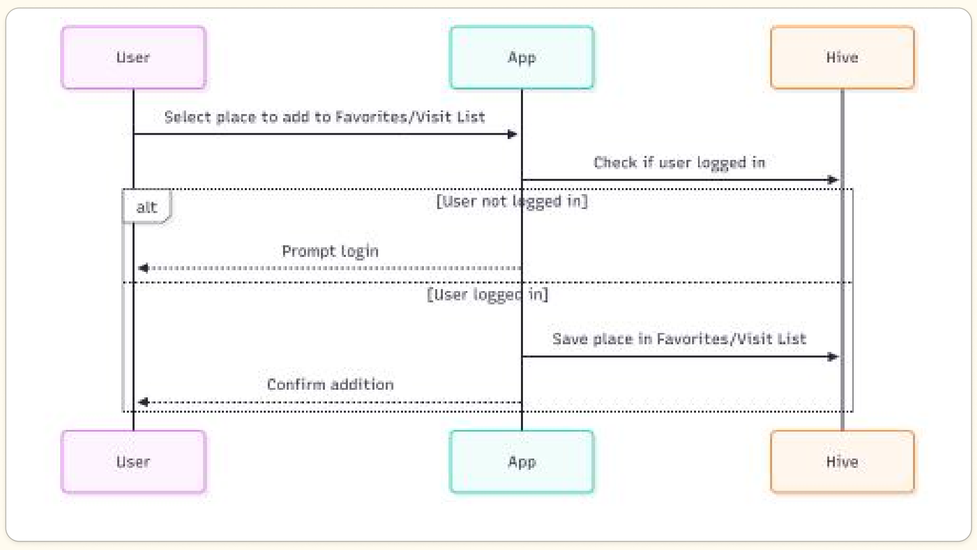


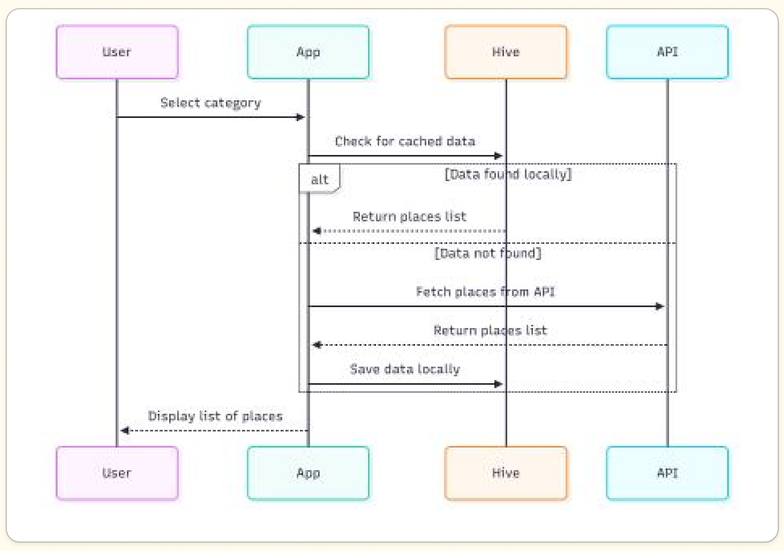
Class Diagram

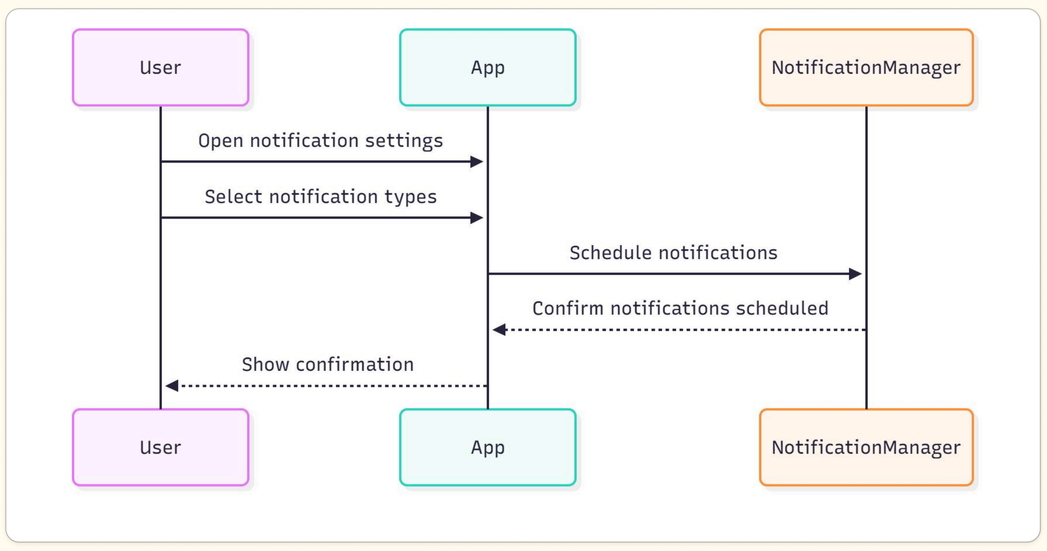


Sequence Diagrms

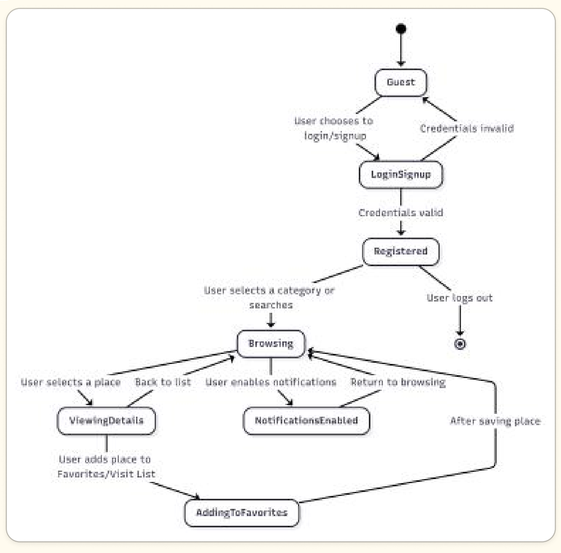
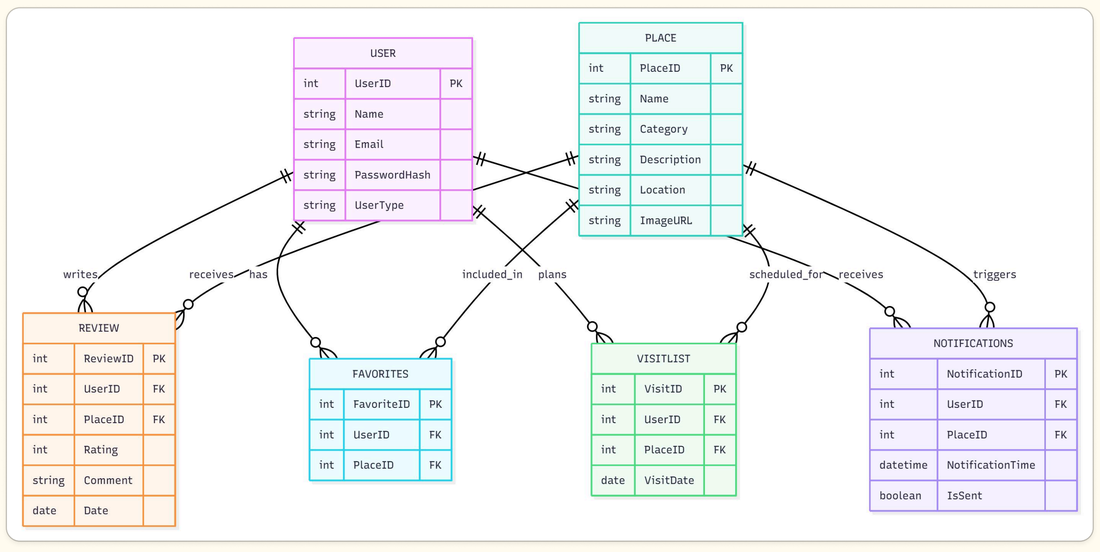








State Diagram



ERD