Technical Documentation

# 1. System Architecture

The Event Mate application is developed using Kotlin on Native Android, leveraging Jetpack Compose for modern, declarative UI development. The overall architecture adheres to a clean, modular, and layered structure, promoting scalability, testability, and separation of concerns.

## a. Presentation Layer (UI Layer)

Built with Jetpack Compose, providing a reactive and component-based UI. Handles user interactions, state rendering, and navigation between screens.

Core screens include:

• Splash Screen – Entry point of the app.  
• Onboarding Flow – Guides new users through the app features.  
• Authentication (Login/Sign Up) – Supports Google Sign-In via Firebase.  
• Home – Displays a dynamic list of events with countdowns.  
• Add/Edit Event – Enables users to create or modify events with custom details.  
• Event Details – Shows full information and actions related to an event.  
• Favorites – Dedicated screen for user-marked favorite events.  
• Notifications – Lists reminders and triggered alerts.  
• Settings & Profile – User preferences and account information.

## b. Domain Layer (Business Logic Layer)

Encapsulates core business rules and decision logic. Manages use cases such as:  
• Event creation, editing, and deletion.  
• Reminder scheduling and validation.  
• Favorites handling.  
Maintains independence from platform-specific implementations.

## c. Data Layer

Employs Room Database for local persistence and offline-first behavior. All event-related data is stored locally, ensuring full functionality without internet access. Uses DAO interfaces and entities to define schema and operations. A Repository pattern is implemented to abstract data sources from business logic.

## d. Authentication Module

Integrated with Firebase Authentication using Google Sign-In provider. Handles user sessions, login state management, and automatic re-authentication for returning users.

## e. Notification System

Events can trigger notifications based on user-defined reminder times. Utilizes Android’s AlarmManager or WorkManager (depending on API level and timing requirements) for reliable scheduling and background execution.

# 2. Database Schema

The Event Mate application leverages Room Database for local data persistence, enabling seamless offline access and robust performance without reliance on remote servers. The schema is designed to support core functionalities such as event creation, modification, deletion, reminders, and favorites management.

## Core Entity: Event

The primary data model in the application is the Event entity, representing user-defined events. It encapsulates the following attributes:

• Identifier (ID): A unique, auto-generated key used internally for data operations.  
• Title: A short, descriptive name of the event.  
• Description: Optional, user-provided details about the event.  
• Category: A tag or label to classify the event (e.g., Personal, Work, Health).  
• Scheduled Date and Time: A timestamp indicating when the event is set to occur.  
• Reminder Configuration: An optional timestamp defining when the user should be notified prior to the event.  
• Image Reference: A URI or path referencing a user-selected image associated with the event.  
• Favorite Flag: A boolean indicator specifying whether the event is marked as a favorite.  
• Creation Timestamp: A stored record of when the event was originally added.

## User Preferences (Optional Component)

An auxiliary structure may be used to capture user-specific application settings, such as theme preference (dark/light mode) or notification permissions. This configuration is stored locally and applies only to the active user session.

## Data Integrity & Constraints

All fields are strongly typed and validated before persistence. Events are scoped to the authenticated session but stored locally; no user-identifiable data is persisted within the Room schema. Referential integrity is maintained internally through application logic, with no foreign keys or relational mappings, as the schema is intentionally flat and single-user focused.

## Data Access Strategy

The application implements a Repository Pattern, ensuring that all data operations—whether reading, writing, or updating—are abstracted from the presentation layer. This separation enhances maintainability, testability, and modular design. Data is consumed reactively using observable streams, ensuring that UI components reflect the latest state without requiring manual refreshes.

# 3. API & Backend Integration

The Event Mate application does not utilize external APIs or remote backends. However, it integrates essential services to support user authentication and event-based reminders.

## a. Firebase Authentication

Integrated solely for Google Sign-In to authenticate users securely. Ensures a seamless login experience and automatic session handling. No additional Firebase services (e.g., Firestore or Realtime Database) are used.

## b. Local Notification System

Utilizes AlarmManager and/or WorkManager to schedule event reminders. Notifications are managed entirely offline, based on the user's configured reminder time. Designed to be battery-efficient and resilient across device reboots.

This lightweight integration ensures the app remains responsive, private, and functional even without internet access.