ADDIS ABABA SCIENCE AND TECHNOLOGY UNIVERSITY

COLLEGE OF ENGINEERING

SOFTWARE ENGINEERING

MOBILE COMPUTING AND PROGRAMMING

PROJECT DOCUMENTATION

NAME: MedVault

NAME	ID
ABEL TEFERI	ETS0035/15

SUBMITTED TO:

SUBMISSION DATE:

TABLE OF CONTENT

1. INTRODUCTION

MedVault is a mobile application that allows users to securely store, organize and manage their medical information in one place. It supports features such as medical record storage, medication reminders, appointment tracking and personal health information management.

Think of it as a “digital health diary”.

Core Problem It Solves:

While hospitals and healthcare organizations typically store and manage patient information using their own structured systems, patients themselves often rely on scattered and inconsistent methods to keep track of their medical records. This project aims to address that gap.

In most cases, medical data is scattered: prescriptions in photos, lab results in PDFs, appointment notes in WhatsApp chats, etc. When patients visit a new doctor, they often don’t have their full history available — which leads to confusion or repeated tests.

MedVault fixes this by offering one simple, secure, and organized place for everything related to health.

2. MEDVAULT – REQUIREMENTS SPECIFICATION

2.1. FUNCTIONAL REQUIREMENTS

2.1.1. Authentication & User Account Management

- The system shall allow users to create an account using email and password.
- The system shall allow users to log in securely.
- The system shall allow users to reset password via email.
- The system shall support biometric login (fingerprint/face ID).
- The system shall store authentication data securely (e.g., Firebase Auth).

2.1.2 User Profile Setup

- The system shall allow users to enter personal health details including:
    - ✓ Full Name
    - ✓ Age / Date of Birth
    - ✓ Gender
    - ✓ Height, Weight

- ✓ Blood Group
- ✓ Emergency Contact

➢ The system shall allow users to edit/update their profile information.

➢ The system shall allow users to upload a profile picture.

2.1.3 Medical Record Management

```
The system shall allow users to attach medical documents to diagnoses,
including:
- Prescriptions
- Lab reports
- Medical images
- PDFs
The system shall allow users to:
- Upload documents via camera or file picker
- View attached documents
- Delete documents
- Tag documents with metadata (doctor name, hospital, date)
```

2.1.4 Diagnosis Management

➢ The system shall allow users to add diagnoses (e.g., Diabetes, Asthma).

➢ The system shall allow users to view diagnosis details.

➢ The system shall allow users to attach documents related to a diagnosis.

➢ The system shall allow users to record:

- ✓ Diagnosis description
- ✓ Date diagnosed
- ✓ Doctor's notes

2.1.5 Medication Tracker

- ➢ The system shall allow users to add medications, including:
    - ✓ Medicine name
    - ✓ Dosage
    - ✓ Time schedule
    - ✓ Frequency
- ➢ The system shall send push notifications when it is time to take medication.
- ➢ The system shall allow users to edit or delete medication reminders.

2.1.6 Appointment & Doctor Management

- ➢ The system shall allow users to add appointments with:
    - ✓ Doctor name
    - ✓ Hospital/clinic
    - ✓ Date and time
- ➢ The system shall send reminders for upcoming appointments.
- ➢ The system shall allow users to view a list of all upcoming appointments.

2.1.7 Dashboard

- ➢ The system shall display a user dashboard containing:

    - ✓ Vitals summary (Height, Weight, BMI, optional BP or Sugar)
    - ✓ Next appointment
    - ✓ Upcoming medication reminder
    - ✓ Recently added diagnoses

- ➢ The system shall allow users to navigate to all other sections from the dashboard.

2.1.8 Settings & Security

- ➢ The system shall provide logout functionality.
- ➢ The system shall allow users to switch between Light/Dark mode.
- ➢ The system shall allow users to enable/disable biometric authentication.

2.2. NON-FUNCTIONAL REQUIREMENTS

2.2.1 Performance

- ➢ The app shall load the dashboard in under 3 seconds.
- ➢ Notifications shall trigger within ± 30 seconds of scheduled time.

2.2.2 Security

- ➢ All personal and medical data shall be encrypted.
- ➢ Authentication shall follow secure standards (Firebase, JWT, etc.).

- The app shall lock automatically after a period of inactivity.

2.2.3 Reliability

- The app shall cache user data locally for improved performance and display cached data when offline. Full functionality requires internet connectivity for Firebase operations.

2.2.4 Usability

- The UI shall be clean, simple, and accessible.
- Icons and labels shall clearly represent medical items.

2.2.5 Compatibility

- The app shall run on Android 8.0+ and iOS 12+.

2.3. SYSTEM CONSTRAINTS

- Mobile only (Android & iOS).
- Built with Flutter framework
- Uses Firebase (Auth, Firestore, Storage) or equivalent.
- File uploads limited to supported formats: PDF, PNG, JPG.
- Reminders depend on Android/iOS notification systems.

3. DETAILED USE CASES

3.1. ACTORS (USERS IN THE SYSTEM)

Actor	Description
User (Patient)	The main user who stores, views, and manages their health-related information.
System	Handles authentication, storage, reminders, and security operations.
Notification Service	Sends scheduled medication and appointment reminders.

Use Case 1 — User Registration

Actor: User

Goal: Create a new account

Steps:

- ➢ User opens app and selects “Sign Up”.
- ➢ User provides email and password.

- System validates input.
- System creates account and saves UID.
- System redirects to Profile Setup page.

Use Case 2 — Upload Medical Record

Actor: User

Goal: Store a medical document

Steps:

- User selects "Add Document".
- User chooses camera or file upload.
- System uploads file to storage.
- System saves metadata in database.
- System displays the document in the list.

Use Case 3 — Add Medication Reminder

Actor: User, Notification Service

Goal: Notify user to take medicine

Steps:

- User opens Medication Tracker.

- User adds medicine name, dosage, time, and frequency.
- System saves the reminder.
- Notification Service schedules local/push notification.

Use Case 4 — Add Appointment

Actor: User

Goal: Remind user of an upcoming appointment

Steps similar to medication reminder, but for date/time of visit.

Use Case 5 — View Dashboard

Actor: User

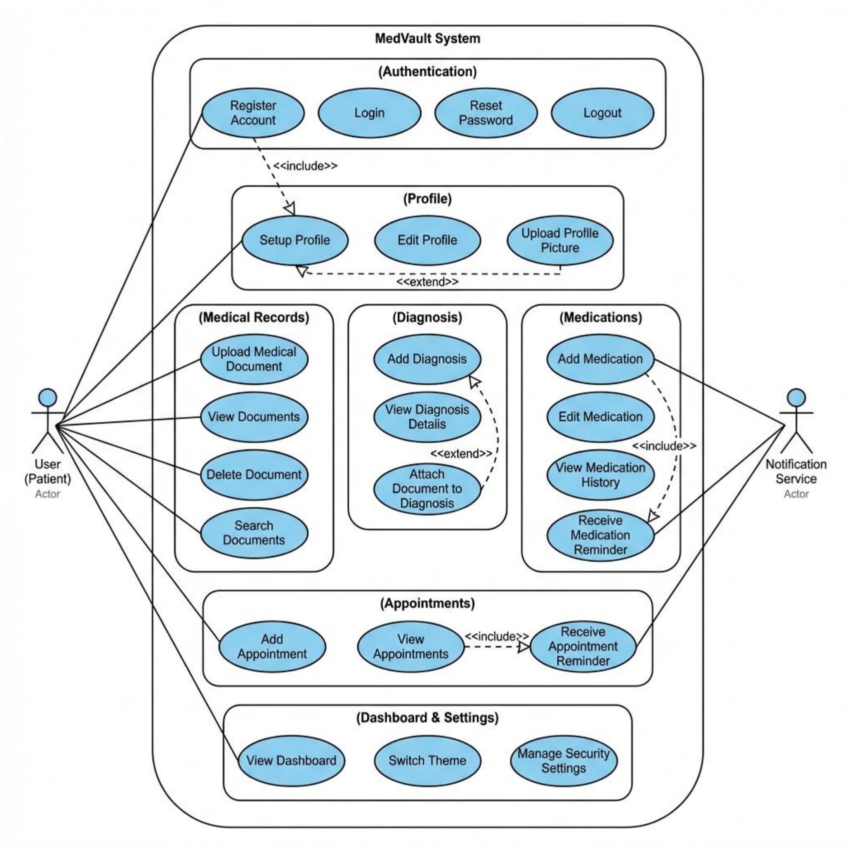
Goal: See overall health summary

System shows:

- Profile info
- Next appointment
- Next medication
- Diagnoses list
- Quick navigation items

4. UML USE CASE DIAGRAM

Figure 1: MedVault Use Case Diagram



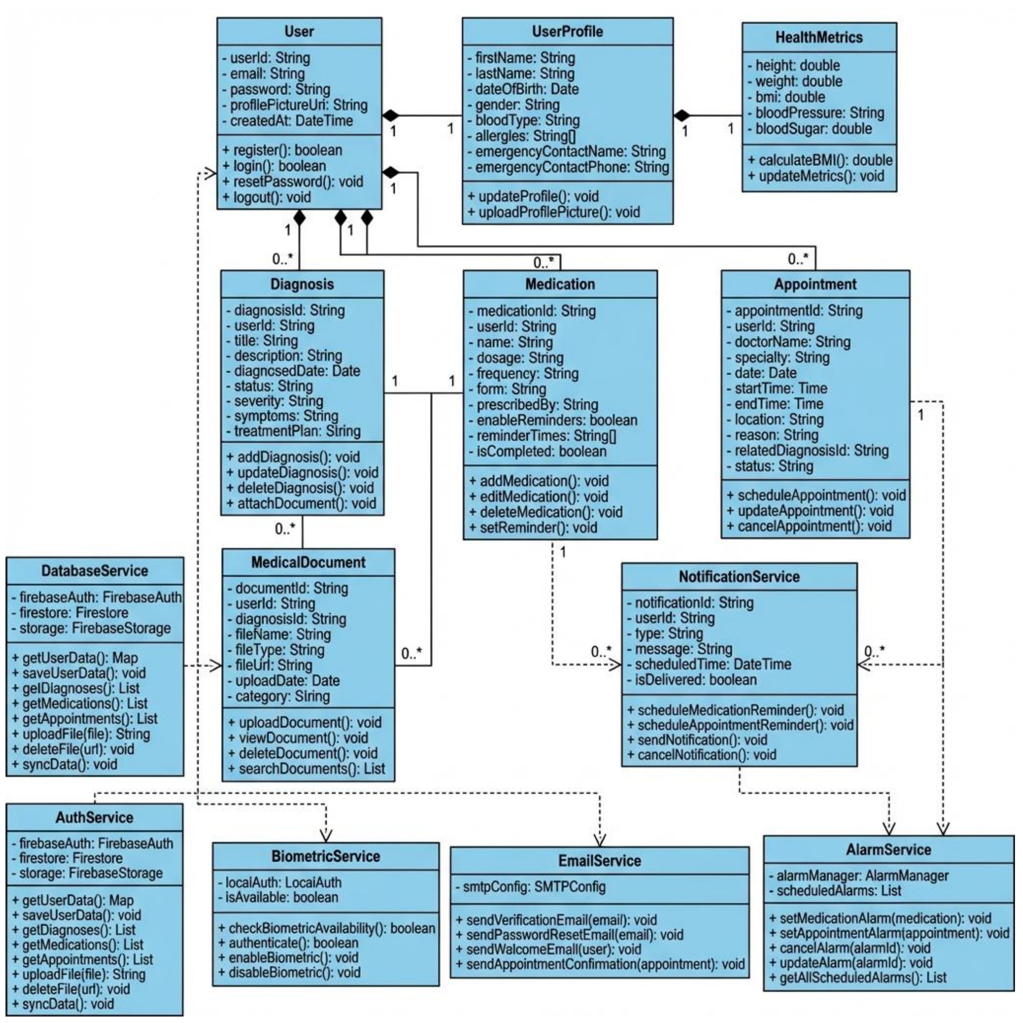
The Use Case Diagram above illustrates the interactions between the primary actors (User and Notification Service) and the MedVault system. The diagram shows:

- Primary Actor: User (Patient) - interacts with all core functionalities
- Secondary Actor: Notification Service - handles automated reminders
- System Boundary: MedVault System - contains all use cases
- Key Relationships:
  - ✓ <<include>> relationships show mandatory dependencies

- ✓ <<extend>> relationships show optional features
- ✓ Association lines connect actors to their use cases

5. CLASS DIAGRAM

Figure 2: MedVault Class Diagram



The Class Diagram above illustrates the object-oriented structure of the MedVault application. It shows the main classes, their attributes, methods, and relationships that form the foundation of the system architecture.

5.1. CORE CLASSES

User & Profile Management:

- User: Represents the authenticated user account with login credentials and authentication methods.
- UserProfile: Contains personal information including name, date of birth, gender, blood type, allergies, and emergency contact details.
- HealthMetrics: Stores and manages health measurements (height, weight, BMI, blood pressure, blood sugar) with automatic BMI calculation.

Medical Records:

- Diagnosis: Manages medical conditions with details such as diagnosis date, status (Ongoing/Resolved/Monitoring), severity level, symptoms, and treatment plans.
- Medication: Tracks prescribed medications including dosage, frequency, form, and reminder settings for adherence.
- Appointment: Handles doctor appointments with scheduling information, specialty, location, and optional linkage to related diagnoses.
- MedicalDocument: Stores uploaded medical files (prescriptions, lab reports, images, PDFs) with metadata and categorization.

5.2. SERVICE CLASSES

The application follows a service-oriented architecture with specialized service classes:

- DatabaseService: Manages all Firebase Firestore operations including user data retrieval, storage, and synchronization. Also handles file uploads to Firebase Storage.
- AuthService: Handles user authentication through multiple methods (email/password, Google Sign-In, Apple Sign-In) and password reset functionality.
- BiometricService: Provides biometric authentication (fingerprint/face recognition) for enhanced security and quick app access.
- NotificationService: Manages push notifications for medication reminders and appointment alerts to ensure users stay on track.
- AlarmService: Schedules and manages system-level alarms for time-critical reminders, ensuring notifications are delivered even when the app is closed.
- EmailService: Handles email communications including verification emails, password reset links, welcome messages, and appointment confirmations.

5.3. KEY RELATIONSHIPS

Composition Relationships (filled diamond):

- User owns UserProfile, HealthMetrics, and all medical records (Diagnosis, Medication, Appointment, MedicalDocument). When a user account is deleted, all associated data is also removed.

Association Relationships (solid line):

- Diagnosis can be linked to multiple MedicalDocuments (e.g., test results, X-rays).
- Diagnosis can be associated with multiple Appointments (follow-up visits).
- This enables comprehensive tracking of medical conditions and their related documentation.

Dependency Relationships (dashed arrow):

- Service classes depend on entity classes they manage.
- Medication and Appointment depend on NotificationService and AlarmService for reminder functionality.
- AuthService depends on EmailService for password reset and verification emails.

5.4. DESIGN PATTERNS

The class structure implements several design patterns:

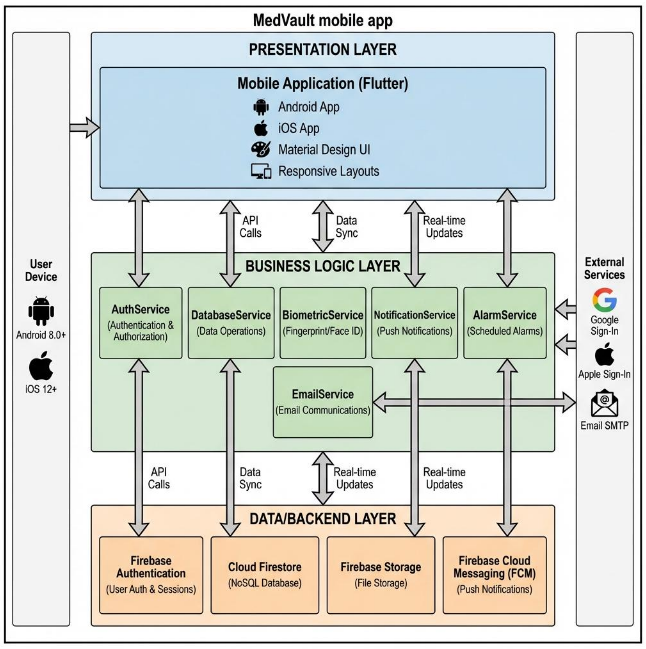
➢ Repository Pattern: DatabaseService acts as a repository for all data operations.

➢ Service Layer Pattern: Business logic is encapsulated in service classes.

➢ Composition over Inheritance: User aggregates related data through composition rather than inheritance.

6. SYSTEM ARCHITECTURE DIAGRAM

Figure 3: MedVault System Architecture


MedVault mobile app
PRESENTATION LAYER
Mobile Application (Flutter)
Android App
iOS App
Material Design UI
Responsive Layouts
API Calls
Data Sync
Real-time Updates
BUSINESS LOGIC LAYER
AuthService (Authentication & Authorization)
DatabaseService (Data Operations)
BiometricService (Fingerprint/Face ID)
NotificationService (Push Notifications)
AlarmService (Scheduled Alarms)
EmailService (Email Communications)
User Device
Android 8.0+
iOS 12+
External Services
Google Sign-In
Apple Sign-In
Email SMTP
API Calls
Data Sync
Real-time Updates
Real-time Updates
DATA/BACKEND LAYER
Firebase Authentication (User Auth & Sessions)
Cloud Firestore (NoSQL Database)
Firebase Storage (File Storage)
Firebase Cloud Messaging (FCM) (Push Notifications)

The system follows a three-tier architecture:

6.1. PRESENTATION LAYER (MOBILE APP)

➢ Built with Flutter framework

- Implements Material Design UI components
- Supports both Android and iOS platforms
- Provides responsive layouts for different screen sizes

6.2. BUSINESS LOGIC LAYER (SERVICES)

- AuthService: Handles authentication and authorization
- DatabaseService: Manages data operations with Firestore
- BiometricService: Provides biometric authentication
- NotificationService: Manages push notifications
- AlarmService: Schedules system-level alarms
- EmailService: Handles email communications

6.3. DATA LAYER (FIREBASE BACKEND)

- Firebase Authentication: User authentication and session management
- Cloud Firestore: NoSQL database for storing user data
- Firebase Storage: File storage for medical documents and images
- Firebase Cloud Messaging: Push notification delivery

6.4. DATA FLOW

1. User interacts with Flutter UI
2. UI calls appropriate service classes

3. Services communicate with Firebase backend

4. Firebase processes and stores data

5. Real-time updates sync back to the app

6. UI updates to reflect changes

7. IMPLEMENTATION DETAILS

7.1. TECHNOLOGY STACK

➢ Framework: Flutter 3.9.2+

➢ Programming Language: Dart

➢ Backend: Firebase (Auth, Firestore, Storage, Cloud Messaging)

➢ State Management: StatefulWidgets

➢ Local Authentication: local_auth package

➢ Notifications: flutter_local_notifications package

➢ Image Handling: image_picker package

➢ File Handling: file_picker package

7.2. KEY FEATURES IMPLEMENTED

✓ Multi-method authentication (Email, Google, Apple)

✓ Biometric app lock

✓ Comprehensive diagnosis tracking

✓ Medication management with reminders

✓ Appointment scheduling with notifications

✓ Document attachment to diagnoses

✓ Health metrics tracking with BMI calculation

✓ Dark mode support

✓ Real-time data synchronization

7.3. SECURITY MEASURES

➢ Firebase Authentication with secure token management

➢ Encrypted data transmission (HTTPS)

➢ Biometric authentication for app access

➢ Secure password reset via email

➢ Data isolation per user account

8. TESTING & VALIDATION

8.1. Testing Approach

The application was tested using manual testing methodologies covering:

- Functional testing of all features

- UI/UX testing across different screen sizes

- Authentication flow testing

- Data persistence testing

- Notification delivery testing

- Dark mode compatibility testing

8.2. Test Scenarios Covered

✓ User registration and login

✓ Profile setup and editing

✓ Adding, editing, and deleting diagnoses

✓ Adding, editing, and deleting medications

✓ Medication reminder notifications

✓ Appointment scheduling and reminders

✓ Document upload and viewing

✓ Biometric authentication

✓ Theme switching (Light/Dark mode)

✓ Data synchronization with Firebase

8.3. Known Limitations

➢ Requires internet connectivity for full functionality

- Notification delivery depends on device settings
- Biometric authentication availability varies by device

9. CONCLUSION

MedVault successfully addresses the critical need for personal medical record management by providing a comprehensive, secure, and user-friendly mobile application. The project demonstrates the practical application of mobile computing principles, cloud-based backend services, and modern UI/UX design.

9.1. PROJECT ACHIEVEMENTS

The application successfully implements:

- Secure multi-method authentication system
- Comprehensive medical record management
- Intelligent medication and appointment reminders
- Document storage and organization
- Real-time data synchronization
- Cross-platform compatibility (Android & iOS)

9.2. LEARNING OUTCOMES

Through this project, valuable experience was gained in:

- Flutter framework and Dart programming
- Firebase backend integration
- Mobile app architecture design
- UML modeling and documentation
- User-centered design principles
- Mobile notification systems

9.3. FUTURE ENHANCEMENTS

- Potential improvements for future versions include:
- Data export to PDF for sharing with healthcare providers
- Integration with wearable devices for automatic health data import
- AI-powered medication interaction checker
- Telemedicine integration for virtual consultations
- Family account management
- Multi-language support
- Offline mode with full functionality

9.4. FINAL REMARKS

MedVault demonstrates how mobile technology can empower individuals to take control of their healthcare information. By consolidating scattered medical data into one secure, accessible platform, the application improves healthcare outcomes and patient engagement.