Roll No: 22SW064 & 22SW019 **Section: 1**

BMI Calculator – Project Report

1. Real World Problem Identification

Problem Statement

Many people struggle to monitor and maintain a healthy weight due to the lack of simple, accessible, and accurate tools for calculating Body Mass Index (BMI). While numerous health apps exist, they are often cluttered with unnecessary features, require an internet connection, or store sensitive data remotely.

Key issues include:

- **Complexity:** Many apps are overloaded with ads and unnecessary features.
- **Data Privacy:** Users are uncomfortable storing health data online.
- **Inconsistency:** Inaccurate conversions between metric and imperial units.
- **No Persistence:** Most simple calculators don't store previous results.

Impact

- Users lose motivation due to the inability to track progress over time.
- Lack of persistence limits meaningful self-assessment.
- Health awareness is reduced among casual users who prefer lightweight tools.

2. Proposed Solution

Overview

The **BMI Calculator** app provides a simple, offline, and user-friendly solution for calculating and tracking BMI. It supports multiple units, stores results locally using SQLite, and displays BMI category feedback for better understanding.

Core Functionalities

1. **BMI Calculation**
    - Accepts height and weight in various units (cm/m/ft, kg/lb).
    - Converts all units internally to metric for accurate results.
    - Displays BMI value and classification (e.g., Underweight, Normal, Overweight, Obese).
2. **Unit Conversion System**
    - Height Units: Centimeters (cm), Meters (m), Feet (ft).
    - Weight Units: Kilograms (kg), Pounds (lb).
    - Automatic conversion ensures consistency between display and calculation.
3. **Data Persistence with SQLite**
    - Stores BMI history in a local database (`bmi_records` table).
    - Each record includes ID, BMI value, category, height, weight, gender, and timestamp.
    - Allows users to **view and clear** saved BMI records.
4. **User Interface Features**
    - Intuitive sliders and text fields synchronized in real time.
    - Dropdown menus for unit selection.
    - History screen for record visualization.

5. **Cross-Platform Compatibility**
    - Works on Android, iOS, and web (with in-memory DB fallback).
    - Lightweight and responsive Flutter-based UI.

3. Responsive User Interfaces

- **Main Screen:**
  Displays height, weight, and BMI result with real-time feedback and a “Save to History” button.

- **History Screen:**
  Lists previously saved BMI entries with options to clear all.

4. Data Storage

Database Used: SQLite

Justification

1. **Local Persistence:** Works offline without internet dependency.
2. **Lightweight:** Perfect for small datasets like BMI history.
3. **Structured Data:** Ideal for relational tables (records with multiple fields).
4. **Cross-Platform Support:** Works seamlessly with Flutter via the `sqflite` package.
5. **Ease of Integration:** Simple CRUD operations and fast queries.

Alternatives Considered:

- **SharedPreferences:** Limited to key-value pairs, not suitable for structured data.
- **Firebase:** Overkill for local-only storage and requires network connectivity.

5. APIs / Packages / Plug-ins Used

Flutter Packages

1. **sqflite (^2.0.0)**
    - Purpose: Local SQLite database engine.
    - Usage: Stores and retrieves BMI history records.
2. **path (^1.8.0)**
    - Purpose: File system path management.
    - Usage: Provides directory paths for storing the local database.

6. Issues and Bugs Encountered and Resolved

1. **Unit Conversion Rounding Errors**
    - Fixed by normalizing conversions before saving.
    - *Lesson:* Always store internal values in a single unit (metric).
2. **SQLite Schema Errors**
    - Resolved by ensuring consistent column types and primary keys.
    - *Lesson:* Always define schema before initializing DB.
3. **Slider Synchronization Issues**

    - Fixed by binding slider values to model state updates.
    - *Lesson:* Maintain reactive consistency between UI and data model.
4. **History Screen Not Updating**
    - Solved using `setState()` after DB operations.
    - *Lesson:* Refresh UI after CRUD actions.
5. **Manual Save Button Delay**
    - Added debounce logic to prevent multiple rapid writes.
    - *Lesson:* Optimize DB writes for better performance.

7. Conclusion

The **BMI Calculator** project demonstrates an effective and lightweight solution for tracking health metrics using Flutter.
It combines cross-platform development, local data persistence, and clean UI design principles.

Key Learnings:

- Implementing SQLite with Flutter (`sqflite`).
- Managing unit conversions accurately.
- Designing responsive and intuitive UIs.
- Maintaining separation of concerns (Model, ViewModel, View).