**System Architecture Documentation**

**Personalised AI Meal Guidance Application**

**1. Architecture Overview**

The application follows a **hybrid client-server architecture** with emphasis on client-side processing to support offline functionality and maintain user privacy.

**Architecture Pattern:** Model-View-ViewModel (MVVM) within Flutter framework

**Key Architectural Principles:**

* Privacy-first design with local data storage
* Offline-capable for core food logging functions
* API integration for nutrition data retrieval
* Modular components for maintainability

**2. System Components**

**2.1 Client Layer (Flutter Mobile App)**

**Presentation Layer:**

* UI Components (Flutter Widgets)
* Screen Navigation (Flutter Navigator)
* State Management (Provider/Riverpod/Bloc)
* User Input Handlers

**Business Logic Layer:**

* Nutrient Classification Engine
* User Profile Manager
* Meal Builder Logic
* Recommendation Algorithm

**Data Access Layer:**

* Local Database Manager (SQLite via sqflite package)
* API Service Layer
* Cache Manager

**2.2 Data Storage**

**Local Storage (SQLite Database):**

* User profiles and preferences
* Food logging history
* Cached nutrition data
* Offline food database subset

**Tables:**

* users - User profiles (age, height, weight, activity level, goals)
* food\_items - Cached nutrition data from APIs
* meal\_logs - User food entries with timestamps
* user\_preferences - Dietary restrictions and preferences

**2.3 External Services**

**UK Nutrition APIs:**

* CoFID/Foodle API - Primary UK nutrition data source
* Open Food Facts UK - Supplementary barcode product data

**Barcode Scanning:**

* mobile\_scanner package - Native camera integration

**2.4 Core Modules**

**Module 1: Food Entry System**

* Barcode scanner integration
* Text search functionality
* Quantity input interface
* Manual food entry

**Module 2: Nutrient Analysis Engine**

* Receives food item + quantity
* Calculates total nutrients
* Compares against user-specific thresholds
* Generates qualitative labels ("High Sodium", "Low Iron")

**Module 3: User Profile System**

* Collects user data (age, weight, height, activity)
* Calculates personalized nutrient targets
* Manages primary/secondary health goals
* Updates thresholds dynamically

**Module 4: AI Meal Builder**

* Natural Language Parser (NLP)
* Meal description interpreter
* Ingredient database matcher
* Substitution recommendation engine

**3. Data Flow Architecture**

**3.1 Food Logging Flow**

User Input → Barcode/Search → API Query → Local Cache Check

↓

Nutrition Data Retrieved → Quantity Applied → Nutrient Calculation

↓

User Profile Retrieved → Threshold Comparison → Qualitative Classification

↓

Display Results → Store in meal\_logs

**3.2 AI Meal Builder Flow**

User Meal Description → NLP Parser → Extract Ingredients

↓

Match Ingredients → Nutrition Database Lookup → Calculate Meal Nutrients

↓

User Profile Retrieved → Identify Nutritional Gaps → Generate Recommendations

↓

Substitution Algorithm → Rank Suggestions → Display to User

**3.3 API Integration Flow**

App Request → Check Local Cache → If miss: API Call

↓

CoFID/Foodle API → Parse Response → Store in Local DB → Return to App

↓

If API Unavailable → Use Cached Data → Flag as Potentially Outdated

**4. Technology Stack**

| **Component** |  | **Technology** |  | **Justification** |
| --- | --- | --- | --- | --- |
| Mobile Framework |  | Flutter (Dart) |  | Single codebase, native performance, hot reload |
| Local Database |  | SQLite (sqflite) |  | Lightweight, offline support, SQL queries |
| State Management |  | Provider/Riverpod |  | Reactive UI updates, scalable |
| HTTP Client |  | dio package |  | API calls, interceptors, error handling |
| Barcode Scanning |  | mobile\_scanner |  | Native performance, multi-format support |
| NLP Processing |  | Basic string parsing + keyword matching |  | Lightweight, offline-capable |
| Storage |  | shared\_preferences |  | User settings persistence |

**5. Security & Privacy Architecture**

**Data Protection:**

* All user data stored locally on device
* No cloud synchronization (GDPR compliance)
* Encrypted local database (sqlcipher if needed)
* API calls over HTTPS only

**Privacy Considerations:**

* No calorie tracking to prevent obsessive behavior
* Qualitative feedback only
* User data never transmitted to external servers
* Option to export/delete all personal data

**6. Scalability Considerations**

**Current Scope:**

* Single-user local application
* Offline-first architecture
* Limited to UK nutrition database

**Future Extensions:**

* Cloud backup (optional, user-controlled)
* Multi-device synchronization
* Expanded international food databases
* Machine learning model for improved recommendations

**7. Module Dependencies**

UI Layer

↓

Business Logic Layer

↓

Data Access Layer

↓

[Local SQLite DB] ← → [External APIs]

**Key Dependencies:**

* UI depends on Business Logic for processed data
* Business Logic depends on Data Access for raw data
* Data Access manages both local and remote sources
* All layers can function independently for testability

**8. Error Handling Strategy**

**API Failures:**

* Graceful degradation to cached data
* User notification of outdated information
* Retry mechanism with exponential backoff

**Barcode Scanning Failures:**

* Fallback to manual search
* Clear error messaging
* Alternative entry methods

**Data Validation:**

* Input sanitization at all entry points
* Range validation for quantities
* Null safety throughout application

**9. Performance Considerations**

**Optimization Strategies:**

* Lazy loading of food database
* Image caching for barcode results
* Debounced search queries
* Background API prefetching for common foods
* Database indexing on frequently queried fields

**Target Performance Metrics:**

* Barcode scan response: < 2 seconds
* Search results: < 1 second
* Nutrient classification: < 500ms
* App launch time: < 3 seconds

**10. Testing Architecture**

**Unit Testing:**

* Nutrient classification algorithms
* Threshold calculation logic
* NLP parsing functions

**Integration Testing:**

* API communication
* Database operations
* Module interactions

**UI Testing:**

* User flow testing
* Widget testing
* Accessibility testing

**User Acceptance Testing:**

* Target demographic testing (eating disorder awareness)
* Usability evaluation
* Psychological safety assessment