CSD3156 Mobile and Cloud Computing

**TEAM PROJECT 1 REPORT**

“SNAP & COOK”



# 

Team Members:

|  |  |
| --- | --- |
| **Name** | **SIT ID** |
| Ernest Ho Yong Heng | 2301223 |
| Lucas Yee Junjie | 2301212 |
| Tan Shun Zhi Tomy | 2301341 |
| Wong Woon Li | 2301308 |
| Yan Yu | 2301213 |
| Muhammad Zikry Bin Zakaria | 2201751 |

# Introduction

## Project Overview

**Snap & Cook** is a 100% native Android application developed in Kotlin. The application leverages computer vision and machine learning to detect available ingredients through image capture or gallery selection. Based on the detected ingredients, the app searches the Spoonacular recipe database to retrieve personalised, step-by-step recipes for the user.

The app also supports barcode scanning using ML Kit, an on-device machine learning library, which detects barcodes from the camera and queries the Open Food Facts API to retrieve the product name. The identified product is then automatically added to the ingredient list before searching for matching recipes via Spoonacular.

The system integrates camera functionality, computer vision, machine learning, networking, barcode product lookup, and local storage to create a smart cooking assistant that enhances everyday meal preparation.

## Objective

The primary goal of Snap & Cook is to reduce food waste and simplify meal planning by helping users make the most of the ingredients they already have.

The application aims to:

* Automatically detect ingredients using the device camera
* Allow users to verify and modify detected items
* Scan packaged product barcodes to identify ingredients instantly
* Retrieve matching recipes from the Spoonacular database based on available ingredients
* Provide step-by-step cooking guidance
* Enable offline saving of recipes

### Target Audience:

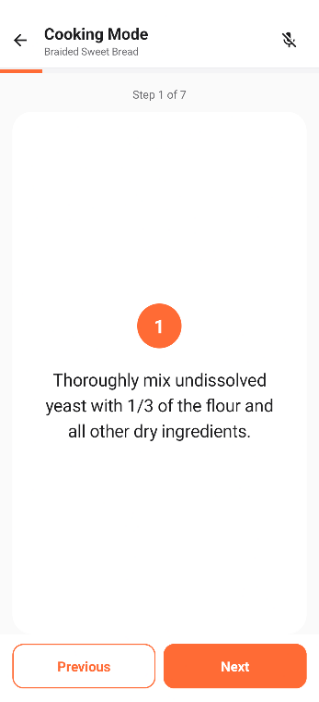
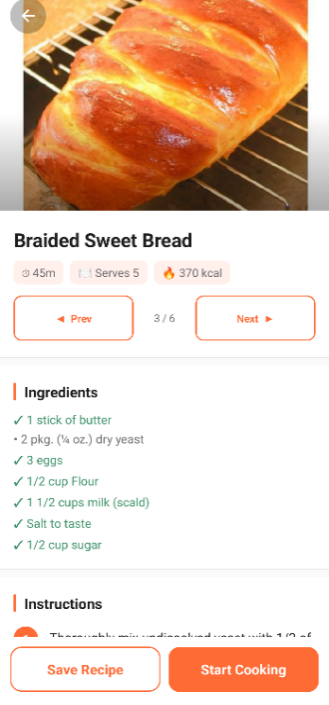
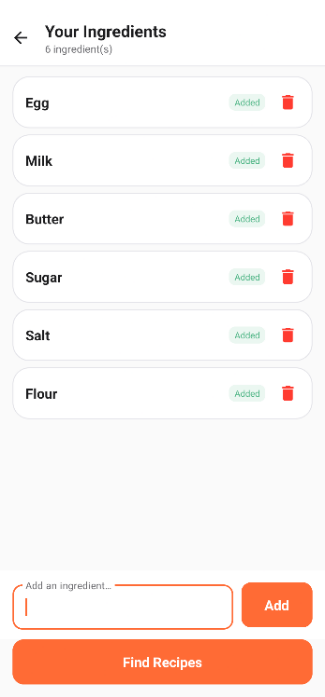
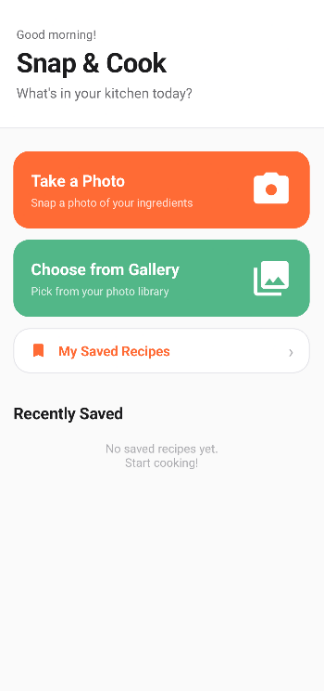
* Students living independently
* Busy working professionals
* Home cooks seeking convenience
* Individuals looking to reduce food waste
* Beginners who need guided cooking instructions

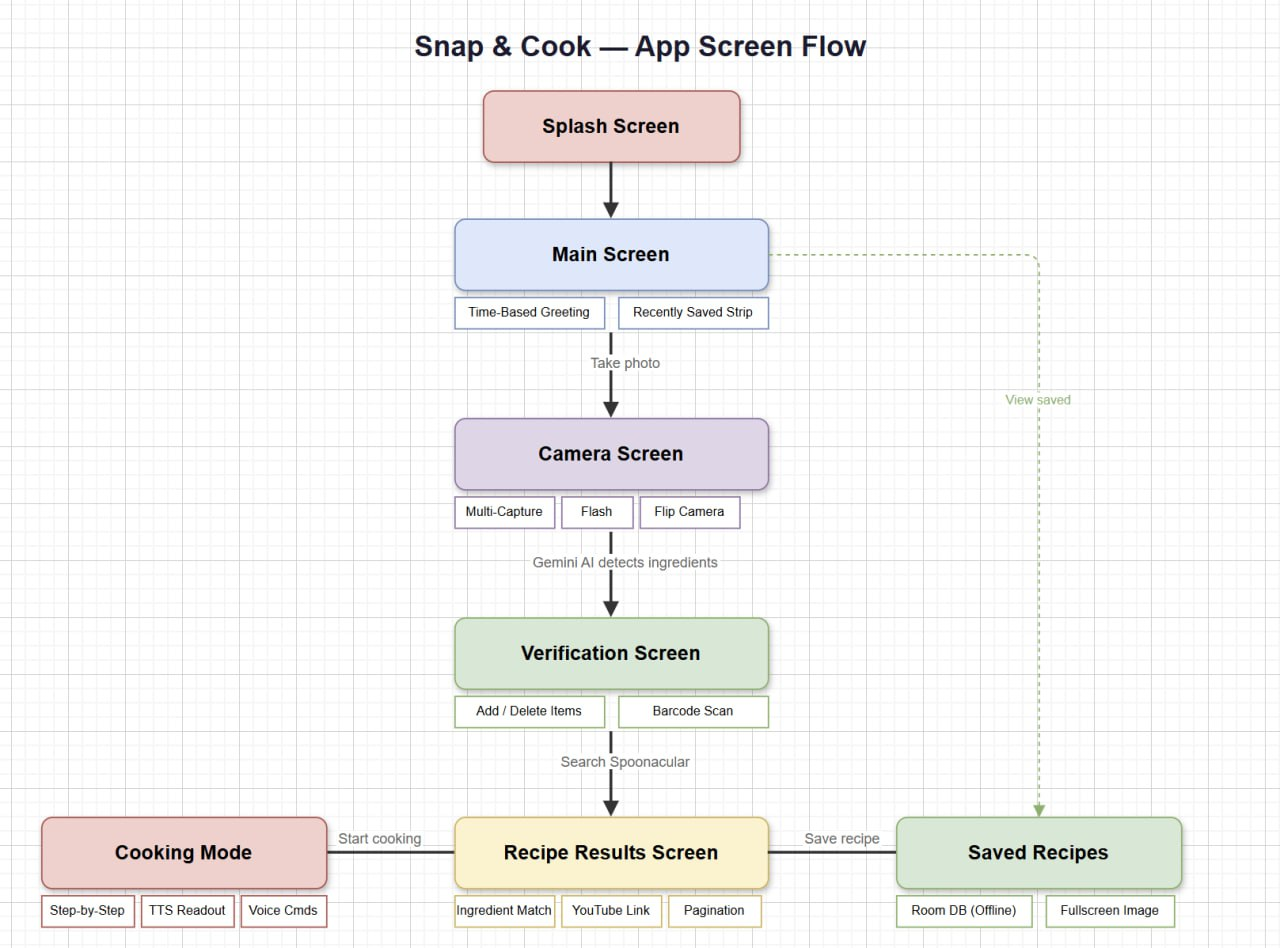
# App Design

## Concept

A utility application utilizing device hardware and machine learning to scan available ingredients using the camera or via gallery and generate custom step-by-step recipes.

## User Interface (UI) Design



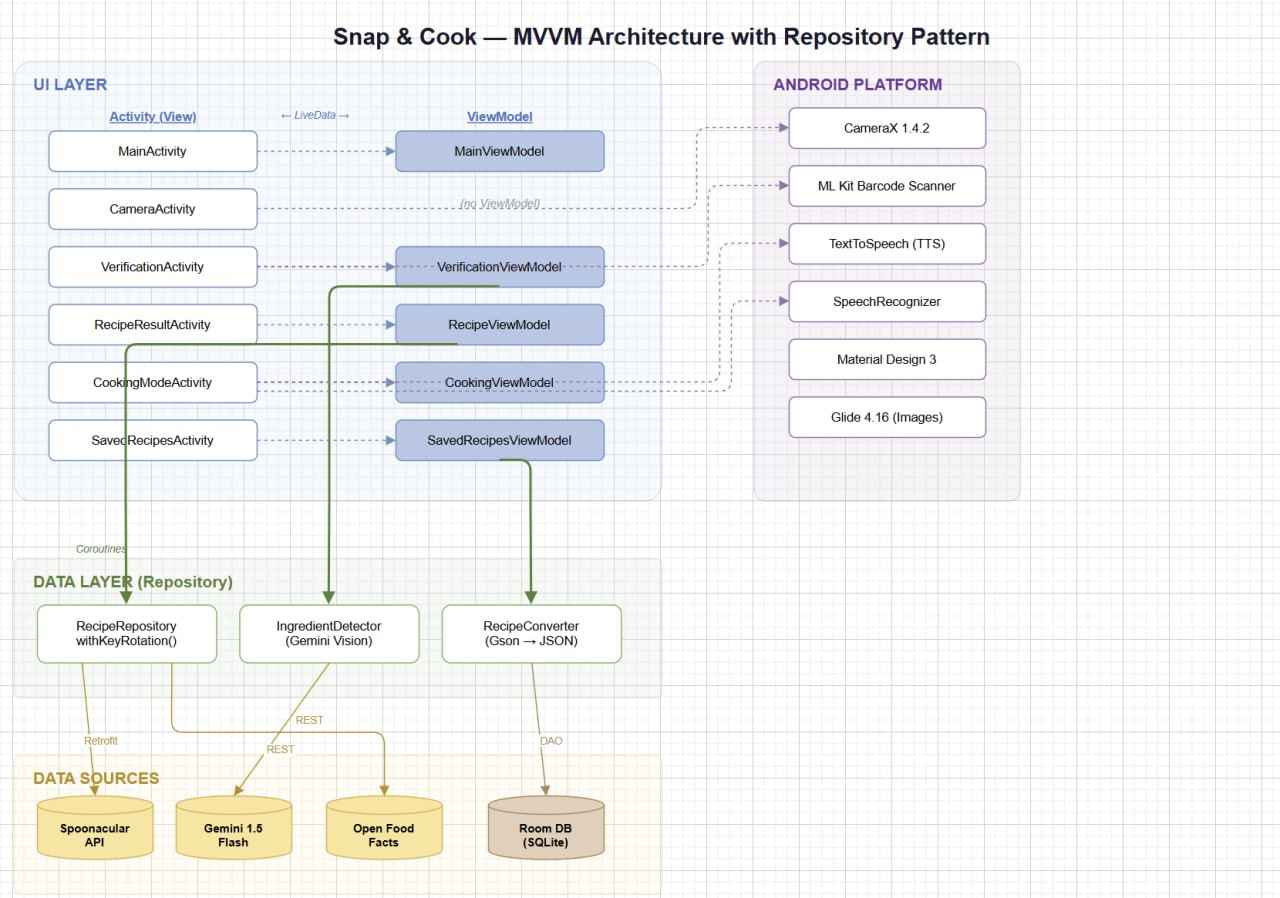


The UI follows a linear navigation flow driven by explicit Android Intents. SplashActivity fades into MainActivity (home screen), which provides entry points to both the Camera and Gallery. From CameraActivity, captured or selected images proceed to VerificationActivity for ingredient review. Confirmed ingredients trigger a recipe search in RecipeResultActivity, where the user can page through up to six matching recipes. From any recipe the user can save it locally or launch CookingModeActivity for hands-free step-by-step guidance.

All screens enforce portrait orientation. Dark mode is supported via values-night resource directories and Android 12+ dynamic colour theming. Full dark mode styling is actively being extended across all screens.

# System Architecture

## High-Level Architecture



The application follows the MVVM (Model-View-ViewModel) architectural pattern with a Repository layer. Three core layers separate concerns:

View Layer: Seven Activities (SplashActivity, MainActivity, CameraActivity, VerificationActivity, RecipeResultActivity, CookingModeActivity, SavedRecipesActivity) handle all UI using Android View Binding. Activities observe LiveData from their ViewModels and update the UI reactively without holding business logic.

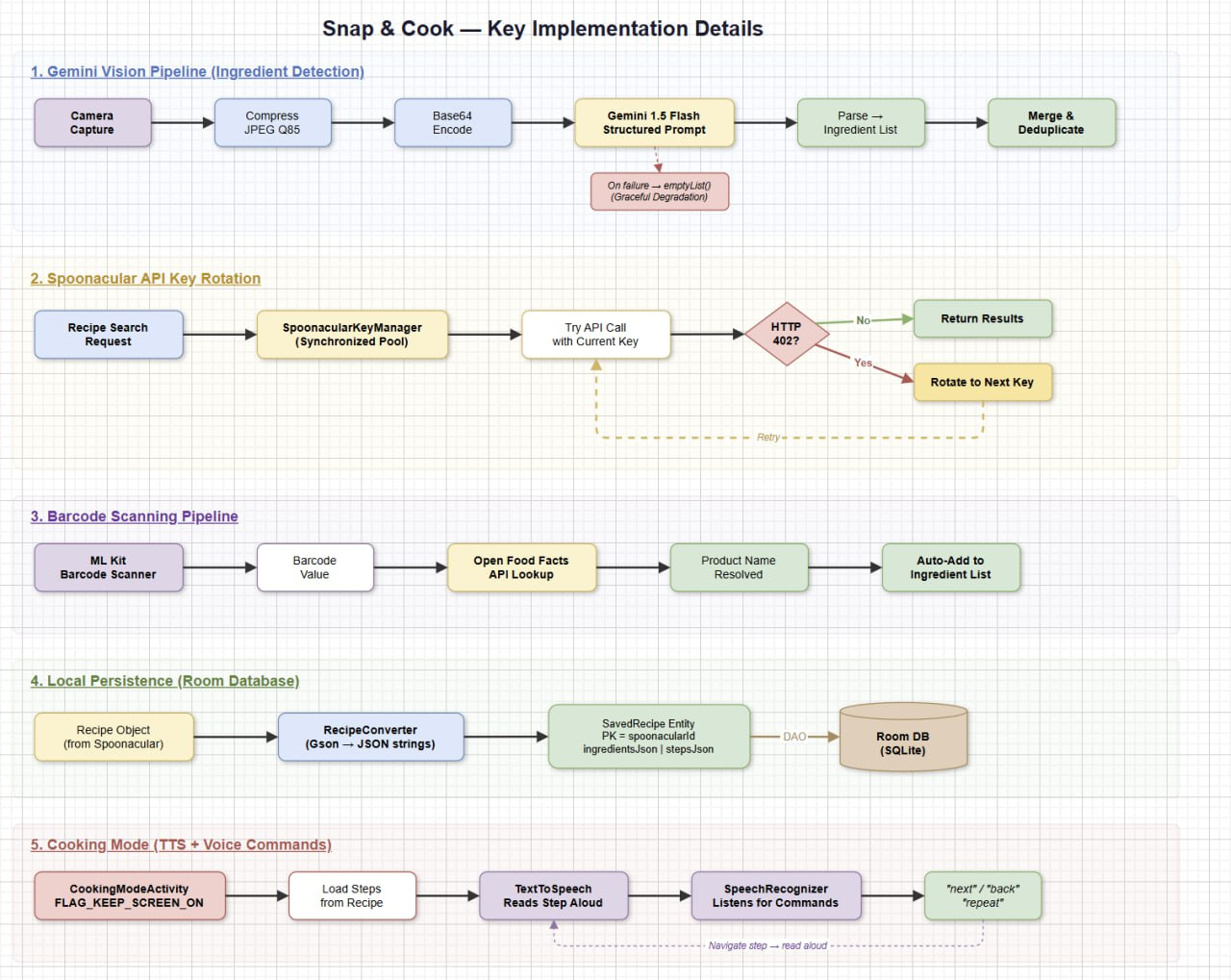
ViewModel Layer: Five ViewModels (MainViewModel, VerificationViewModel, RecipeViewModel, CookingViewModel, SavedRecipesViewModel) manage lifecycle-safe UI state. All network and database operations are launched as Kotlin Coroutines within viewModelScope, with results exposed as LiveData.

Repository & Data Layer: RecipeRepository is the single source of truth for remote data. It queries the Spoonacular REST API via Retrofit and implements automatic API key rotation on HTTP 402 (quota exhausted) responses. Local persistence uses Room (SQLite). Ingredient detection is handled separately by IngredientDetector, which calls the Google Gemini 1.5 Flash multimodal API.

## Component Diagram

CameraActivity / Gallery Picker -> image URIs -> VerificationActivity -> IngredientDetector (ML layer) -> Gemini API -> ingredient list -> RecipeResultActivity -> RecipeRepository -> Spoonacular API -> recipe summaries & detail -> RecipeResultActivity -> (save) Room Database / (cook) CookingModeActivity -> Android TTS Engine + SpeechRecognizer.

RecipeRepository also calls Spoonacular's food/videos/search endpoint to fetch a relevant YouTube tutorial alongside each recipe. SpoonacularKeyManager maintains the pool of API keys; when HTTP 402 is received, the exhausted key is marked used and the API call is automatically retried with the next available key.



## Technology Stack

Language & Build: Kotlin 2.0.21, Android Gradle Plugin 8.13.2 (Kotlin DSL), Java 11 compatibility. Min SDK 26 (Android 8.0) / Target SDK 35 (Android 15).

Jetpack Components: ViewModel 2.8.7, LiveData, Room 2.6.1 (SQLite ORM), CameraX 1.4.2 (camera-core, camera-camera2, camera-lifecycle, camera-view), View Binding, Activity Result Contracts.

Networking: Retrofit 2.11.0 with Gson converter 2.11.0, OkHttp 4.12.0 with logging interceptor.

UI: Material Design 3 (1.12.0), ConstraintLayout 2.2.1, RecyclerView 1.3.2, CardView, SwipeRefreshLayout, Glide 4.16.0 (async image loading).

Async: Kotlin Coroutines 1.8.1 (coroutines-core + coroutines-android).

AI & Voice: Google Gemini 1.5 Flash REST API (via Retrofit/OkHttp), Android TextToSpeech (built-in), Android SpeechRecognizer (built-in).

Barcode & Product Lookup: Google ML Kit play-services-code-scanner 16.1.0 (GMS-based bottom-sheet barcode scanner UI), Open Food Facts REST API queried via the existing Retrofit/OkHttp stack.

# Feature Implementation

## Feature 1: Camera

The application uses the device's built-in camera to capture images of available ingredients. Users may also select existing images from the gallery. CameraActivity is powered by CameraX 1.4.2 and supports: live viewfinder preview via PreviewView, single-photo capture with ImageCapture, flash mode toggle (auto / off), front/back camera switching, and accumulation of multiple photos in one session. All captured or selected image URIs are forwarded to VerificationActivity for analysis. Orientation is tracked via OrientationEventListener so captured photos are correctly rotated regardless of how the device is held.

## Feature 2: Machine Learning

Ingredient detection is powered by the Google Gemini 1.5 Flash multimodal model via the IngredientDetector class. Each Bitmap is JPEG-compressed at 85% quality, Base64-encoded, and sent to the Gemini REST API alongside an engineered prompt instructing the model to return only a comma-separated list of specific food ingredient names (e.g. "egg, carrot, soy sauce"). Generic category labels such as "food" or "vegetable" are explicitly forbidden by the prompt.

The model response is split on commas, trimmed, and deduplicated by case-insensitive comparison. When multiple images are analysed, results from each are merged into a single consolidated ingredient list. The detector degrades gracefully - on any API or network failure it returns an empty list, allowing the user to add ingredients manually on the verification screen.

## Feature 3: Computer Vision (CV)

Computer vision is delivered through Gemini's multimodal understanding capability. Unlike traditional on-device CV models constrained to a fixed object-label vocabulary, Gemini understands food context semantically and can identify a wide variety of ingredients - including condiments, prepared foods, and packaged items - without any custom model training.

The detection prompt enforces actionable output by prohibiting generic category labels, ensuring returned ingredient names are directly usable as Spoonacular search terms. Before transmission, the bitmap is JPEG-compressed at 85% quality to reduce latency and data usage while maintaining sufficient detail for accurate identification.

## Additional Features

**Networking (Spoonacular API):** The Spoonacular Food API is queried via Retrofit 2 with Gson deserialisation. Three endpoints are used: /recipes/findByIngredients (ranked by ingredient coverage, capped at 15 ingredients and returning up to 6 results), /recipes/{id}/information (full recipe detail including nutrition), and /food/videos/search (YouTube tutorial lookup). RecipeResultActivity displays matched ingredients with a green checkmark for items the user already owns and a bullet point for items they still need to buy. Tapping the recipe hero image opens a full-screen viewer. A relevant YouTube cooking video is fetched and displayed as a tappable thumbnail that opens directly in the YouTube app or browser.

**Local Storage (Room Database):** Users can save any recipe for offline access. The SavedRecipe Room entity stores recipe title, image URL, cooking time, servings, calories, a JSON-serialised ingredient list, and a JSON-serialised step list in a single SQLite table (saved\_recipes). RecipeConverter handles Gson serialisation. The primary key is the Spoonacular recipe ID, so saving the same recipe twice replaces the existing entry without duplication. MainActivity displays a 'Recently Saved' strip showing the last three saved recipes for quick access. A time-based greeting (Good Morning / Afternoon / Evening) is also shown based on the current hour.

**Barcode Scanning (Open Food Facts):** A barcode scanner button in the add-ingredient row of VerificationActivity launches the Google ML Kit GMS Code Scanner, which displays a bottom-sheet UI powered by Play Services. On a successful scan the raw barcode value (UPC/EAN) is passed to VerificationViewModel.lookupBarcode(), which calls the Open Food Facts public API on a background coroutine (Dispatchers.IO). If a product\_name is found in the response, addManualIngredient() appends it to the ingredient list and a toast confirms the addition. If the barcode is not in the API database, the user is prompted to type the name manually. A custom OkHttp interceptor attaches a User-Agent header as required by the Open Food Facts API terms of use.

**Hands-Free Cooking Mode:** CookingModeActivity keeps the screen on (FLAG\_KEEP\_SCREEN\_ON) and reads each instruction step aloud using Android TextToSpeech. Simultaneously, SpeechRecognizer listens continuously for the voice commands 'next', 'previous' (or 'back'), and 'repeat' (or 'again') to enable completely hands-free navigation. Equipment required for each step is displayed as chip labels below the instruction text, and a progress bar tracks overall completion across all steps.

# Software Engineering Practices

## Unit Testing

The project includes 81 unit tests covering critical business logic components. RecipeConverterTest (27 tests) verifies data conversion between API models and database entities, including JSON serialization and null-safety handling. SpoonacularKeyManagerTest (11 tests) validates the API key rotation mechanism and thread-safe exhaustion tracking. ExtensionsTest (11 tests) covers string and integer utility functions used throughout the app. Additional test files for RecipeRepository and RecipeViewModel document the intended testing patterns for these components.

The test suite uses MockK 1.13.13 for Kotlin-friendly mocking, Kotlinx Coroutines Test 1.8.1 for testing suspend functions, Google Truth 1.4.4 for readable assertions, and AndroidX Arch Core Testing 2.2.0 for LiveData testing. All tests follow the Arrange-Act-Assert pattern with Given-When-Then naming to clearly document expected behavior. Tests execute via ./gradlew test and are automatically run by the GitHub Actions CI pipeline on every commit, with results viewable in app/build/reports/tests/testDebugUnitTest/index.html.

## Version Control

The team uses Git for source control hosted on GitHub. The repository follows a feature-branch workflow: each new feature or bug fix is developed on a dedicated branch and merged into main via a Pull Request requiring at least one peer review before merging. Commits are made per logical unit of work. Tasks are assigned through GitHub Issues with named assignees, and progress is tracked on a GitHub Project Board with milestones tied to submission deadlines.

## CI/CD Pipeline

The Gradle build system automates compilation, resource generation, and APK packaging. A build\_debug.bat convenience script wraps ./gradlew assembleDebug for Windows environments. API keys are excluded from source control by storing them in local.properties (git-ignored) and injecting them into BuildConfig fields at compile time via a dynamic key-collection loop in app/build.gradle.kts, ensuring no secrets are ever committed to the repository.

A GitHub Actions workflow (.github/workflows/android-ci.yml) runs automatically on every push and pull request to the main branch. The workflow executes all 81 unit tests on Ubuntu with JDK 11, then builds a debug APK if tests pass. Test reports and the compiled APK are uploaded as artifacts for download. A build status badge in the README displays the current test status, providing immediate feedback when tests fail and preventing broken code from being merged.

Unit tests reside in app/src/test/ and Android instrumented tests in app/src/androidTest/. Manual end-to-end testing is performed on physical Android devices running API levels 26 through 35 throughout the development cycle to verify behaviour across different OS versions.

## Third-Party Libraries

CameraX 1.4.2 - Lifecycle-aware camera preview, image capture, and orientation management.

Room 2.6.1 - SQLite ORM for structured offline recipe storage.

Retrofit 2.11.0 - Type-safe HTTP client for all Spoonacular API calls.

OkHttp 4.12.0 - HTTP engine and request/response logging interceptor.

Gson 2.11.0 - JSON serialisation and deserialisation for API responses and local Room storage.

Glide 4.16.0 - Asynchronous image loading and in-memory/disk caching.

Kotlin Coroutines 1.8.1 - Structured concurrency for all async network and database operations.

AndroidX Lifecycle (ViewModel + LiveData) 2.8.7 - MVVM reactive architecture components.

Google Gemini 1.5 Flash REST API - Multimodal AI for ingredient recognition from images.

Android TextToSpeech & SpeechRecognizer - Built-in TTS narration and voice command recognition for cooking mode.

Google ML Kit Code Scanner (play-services-code-scanner) 16.1.0 - GMS-based bottom-sheet barcode scanner.

Open Food Facts REST API - Free public product database used to resolve UPC/EAN barcodes to product names.

JUnit 4.13.2, MockK 1.13.13, Kotlinx Coroutines Test 1.8.1, AndroidX Arch Core Testing 2.2.0, and Google Truth 1.4.4 are used for unit testing.

# AI Usage Declaration

Google Gemini 1.5 Flash is used as the core AI component for real-time ingredient detection. Camera images are sent to the Gemini multimodal API with a structured prompt; the model returns a text list of visible food ingredients which is parsed and used to query the Spoonacular recipe database. Gemini was selected over on-device ML models for its superior food-domain understanding and ability to identify a wide variety of ingredients without custom model training.

Generative AI tools (Claude by Anthropic) were used during development to assist with code suggestions, debugging, and architectural decisions. It was also used to set up the unit test framework for RecipeConverter, SpoonacularKeyManager, and extension functions. The GitHub Actions CI/CD workflow configuration was created with AI assistance. All AI-generated code was reviewed, tested, and validated by team members before integration into the project.

# Conclusion

Snap & Cook demonstrates how a modern Android application can integrate multiple mobile platform capabilities - camera hardware, cloud AI, REST APIs, and local persistence - into a cohesive, practical experience. The MVVM architecture with Repository pattern kept the codebase maintainable as features were added iteratively throughout the development cycle.

The most technically challenging aspects were: prompt-engineering the Gemini ingredient detector to produce consistently clean output; managing Spoonacular API quota across multiple team members' devices through the key-rotation mechanism; and implementing the concurrent TTS and voice recognition pipeline in CookingModeActivity without the two systems interfering with each other.

The application successfully meets its objective of helping users reduce food waste and simplify meal planning. Future improvements could include on-device ML fallback for offline ingredient detection, and support for user-created custom recipes.

**Links:**

* GitHub Repository: https://github.com/sTsenre/MobileComputingProject
* App Demo Video: https://youtu.be/JWJMpj2Y8c0
* Presentation Video: https://youtu.be/TkZ5L8IxXCM