EasyChef Final Report

Ronan Green

C00270395

Contents

1. Introduction 2
1.1 Purpose of This Document 2
1.2 Project Motivation 2
1.3 Report Structure 2
2. Description of Submitted Project 2
2.1 Project Overview 2
2.2 Key Features and Functions 3
2.3 Technologies Used 4
2.4 Screenshots 6
3. Description of Conformance to Specification and Design 11
3.1 Original Specification vs. Final Submission. 11
3.2 Changes from Original Plan 11
- 3.3 Justification for Differences 12
4. Description of Learning 12
4.1 Technical Learning 12
4.4.1 New Technologies I Used 12
4.1.2 Challenges Faced 13
4.2 Personal Learning 13
5. Review of the Project 14

6. Acknowledgements 14
7. Extra Reading..... 15

1. Introduction

1.1 Purpose of This Document

This document summarises the aims, scope and results of the final-year project. It describes the development process, shows how the work meets module requirements, and provides a record for future reference or improvement.

1.2 Project Motivation

The project was selected to combine mobile image recognition with the chance to learn Kotlin for Android. Using on-device image classification made it possible to explore practical AI techniques while creating an application with clear real-world value. The work could not have been attempted earlier because machine-learning modules that were covered in the final year along with AI modules were crucial in helping me understand what was necessary and the necessary experience with APIs and large codebases were only after the third year internship.

1.3 Report Structure

The remainder of this report is organised as follows:

• Section 2 – Submitted Project: outlines the application, its key features, the technologies used and supporting screenshots.

• Section 3 – Conformance to Specification: compares the final build with the original plan and explains any differences.

• Section 4 – Learning: reviews the technical and personal skills developed during the project.

• Section 5 – Project Review: discusses successes, difficulties and lessons learned.

• Section 6 – Acknowledgements: records those who contributed advice or support.

2. Description of Submitted Project

2.1 Project Overview

This project delivers an Android application designed to assist users in

discovering recipes based on ingredients they have available. The core functionality revolves around identifying ingredients, either through image recognition of photographed items or via manual text input. Using these inputs, the application retrieves relevant recipes from Spoonacular, ranking them to prioritize those that best match the user's provided ingredients. Beyond recipe discovery, the application incorporates features for personalisation, including dietary preference and intolerance settings, the ability to save favourite recipes, and a fully functional shopping list manager. There is a choice offered to the user between two distinct image recognition technologies:
Google's powerful Gemini model or a TensorFlow Lite model called mobilenet that had been tweaked so it only returned food it recognised, enabling exploration of different AI capabilities.
The application aims to streamline meal planning, potentially reduce food waste, and offer a personalised cooking inspiration tool.

2.2 Key Features and Functions

Ingredient Input Flexibility: Users can add ingredients to their query list in two ways:

- **Image Recognition**: By taking a picture of ingredients, the app utili**s**es an AI model to identify them automatically.
- **Manual Entry:** Users can type ingredient names directly into the application.

Dual Image Recognition Models: The application offers users a choice between two different AI models for ingredient identification from images:

- **Google Gemini:** Leverages a powerful, cloud-based AI model for potentially higher accuracy and broader recogni**tion** capabilities.
- **TensorFlow Lite Model:** Utilises a on device model trained on a large dataset but has been tweaked to only identify food.

Recipe Retrieval and Ranking (via Spoonacular): Based on the identified or inputted ingredients, the application queries the **Spoonacular API** to search for matching recipes. Results are presented in a ranked list, ordered by the number of matching ingredients, ensuring the most relevant recipes appear first.

User Personalisation:

- **Dietary Preferences & Intolerances:** Users can specify dietary requirements (e.g., vegetarian, vegan) and intolerances (e.g., gluten-free, dairy-free) to filter recipe results accordingly via Spoonacular's filtering capabilities.
- **Saved Recipes:** Functionality is provided to bookmark or save favourite recipes (retrieved from Spoonacular) for quick and easy future access within the app.

- **Shopping List Management:** A dedicated section allows users to create, read, update, and delete items on a shopping list, aiding meal preparation and grocery shopping.
- **Personalised Recipe Recommendations:** The application analyses frequently used ingredients (tracked internally) to proactively suggest new recipes from Spoonacular that the user might enjoy based on their cooking habits.

2.3 Technologies Used

The development of this Android application relied upon a carefully selected stack of technologies, chosen for their suitability to the project's requirements, alignment with modern Android practices, and relevance to the learning objectives.

Programming Language: Kotlin

- **Reasoning:** Kotlin was the primary language, favoured for its status as Google's preferred language for Android. It has many upsides such as null safety, and easy to use coroutines. Gaining proficiency in Kotlin was also a core learning goal.

Development Environment: Android Studio

- **Reasoning:** The official IDE for Android, Android Studio, provided essential tools for coding, debugging (using tools like Logcat), layout design, and building the application via the integrated Gradle system.

Core Android SDK and Framework:

- **Reasoning:** The fundamental Android SDK provided the APIs for core application components (AppCompatActivity, Context, Intent for screen navigation), managing permissions (PackageManager, Manifest), interacting with device hardware/storage (MediaStore, ContentValues, Uri), and building the user interface using the traditional Android View system (Widgets like Button, EditText, Spinner, TextView, LinearLayout, menu systems via Menu, MenuItem).

Android Jetpack Libraries:

- **Reasoning:** Several Jetpack libraries were incorporated to follow best practices and simplify development:
- **CameraX:** Used to streamline camera operations.
- **Lifecycle:** Utilised to manage asynchronous operations (coroutines) tied to the lifecycle of UI components, preventing memory leaks and ensuring tasks are cancelled appropriately.
- **AppCompat:** Ensured backward compatibility for UI elements and features across different Android versions.

User Interface:

- **Reasoning:** The application utilises Android's traditional XML-based View system for defining layouts, coupled with Kotlin code for handling user interactions and dynamically updating UI elements.

Data Persistence: Firebase Firestore

- **Reasoning:** Firestore, a cloud-based NoSQL database from Google. Firebase, was chosen for storing user-specific data such as saved recipes and the shopping list.

User Authentication: Firebase Authentication

- **Reasoning:** Integrated to manage user accounts, providing sign-in capabilities and securing user-specific data stored in Firestore.

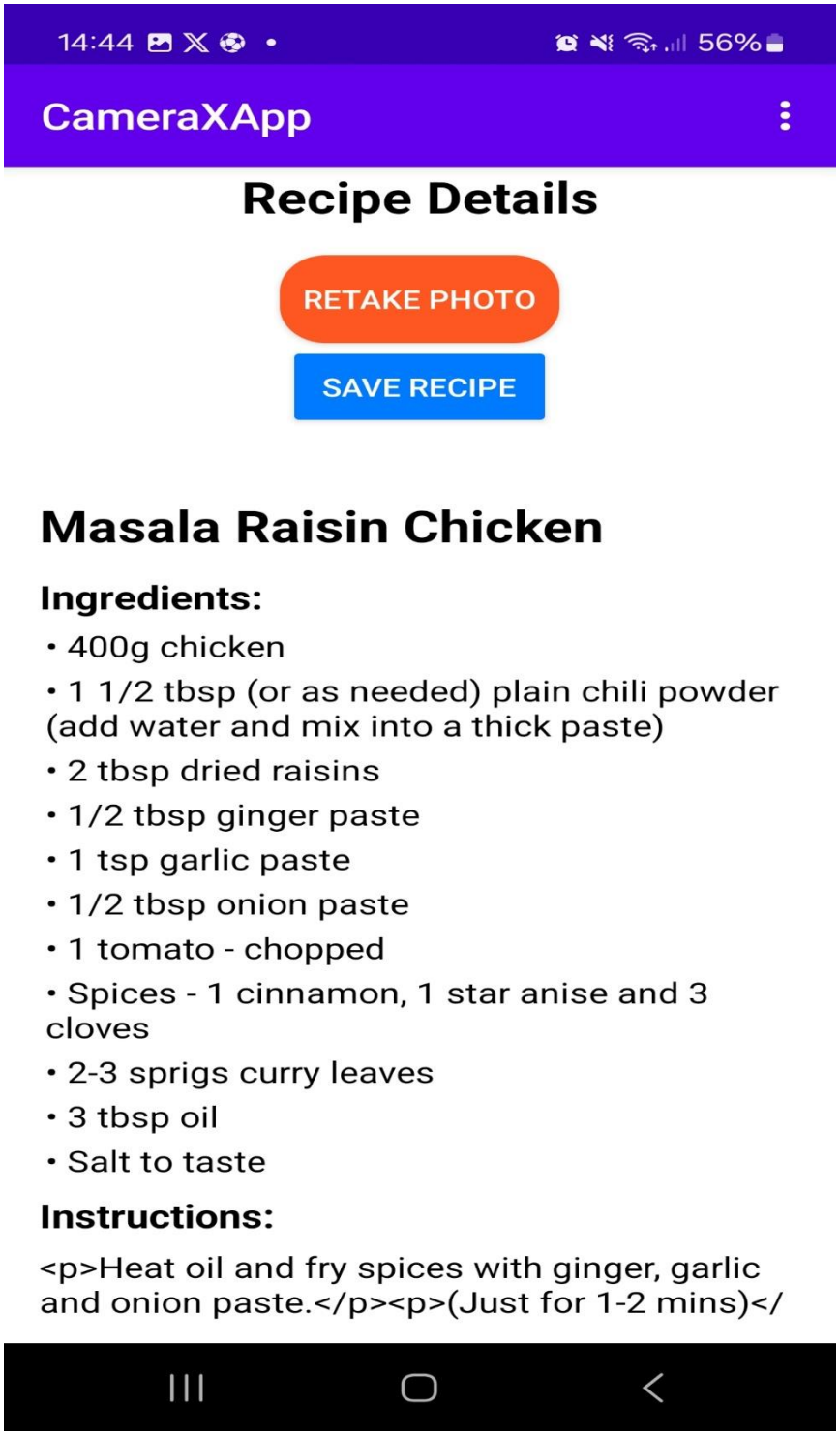
Networking and APIs:

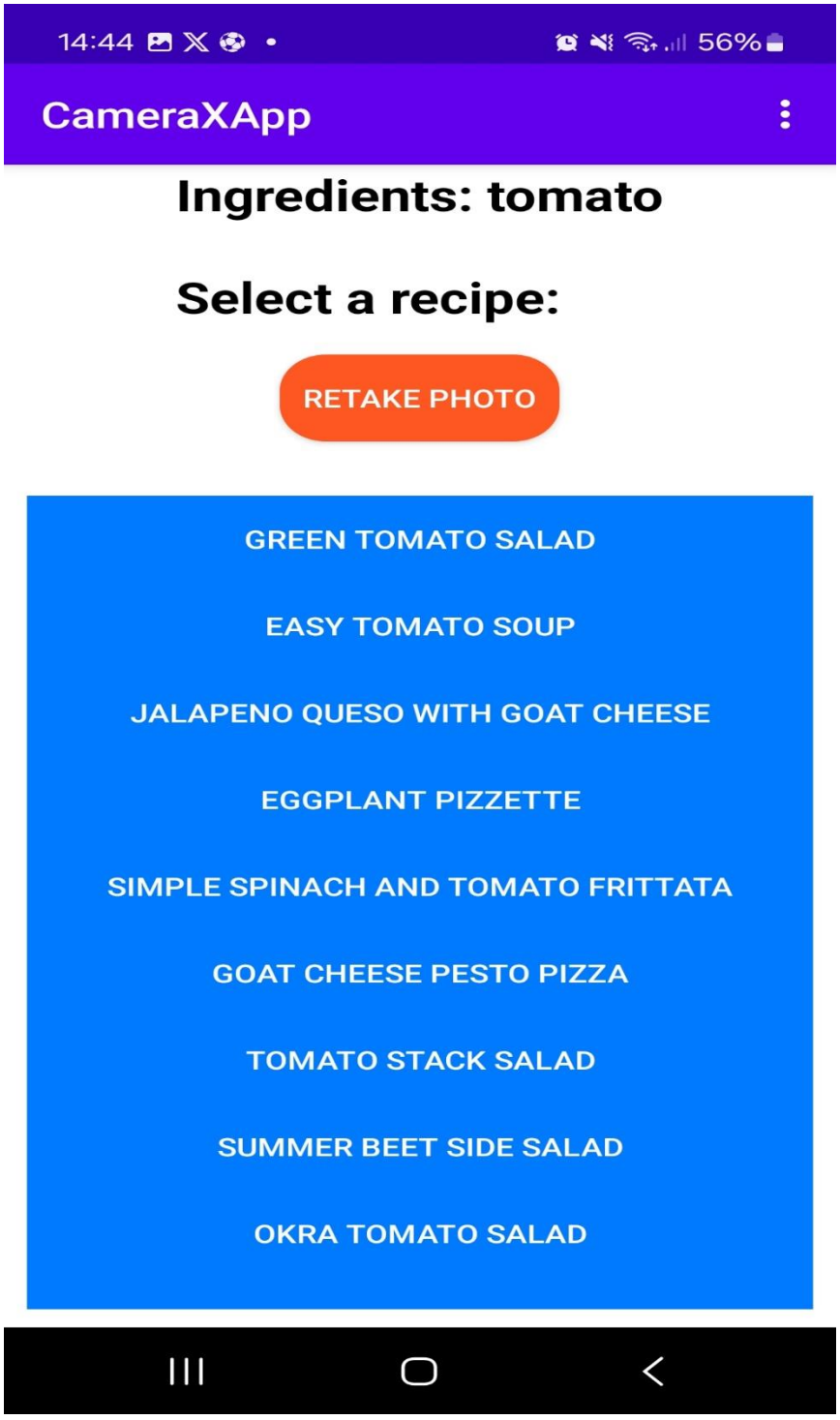
- **Spoonacular API:** The primary source for recipe data, providing extensive search and filtering capabilities based on ingredients and dietary needs. Libraries like Retrofit were used internally within the service class to handle HTTP requests.
- **Google Gemini API:** Used as one of the image recognition options, leveraging Google's cloud-based AI for ingredient identification.

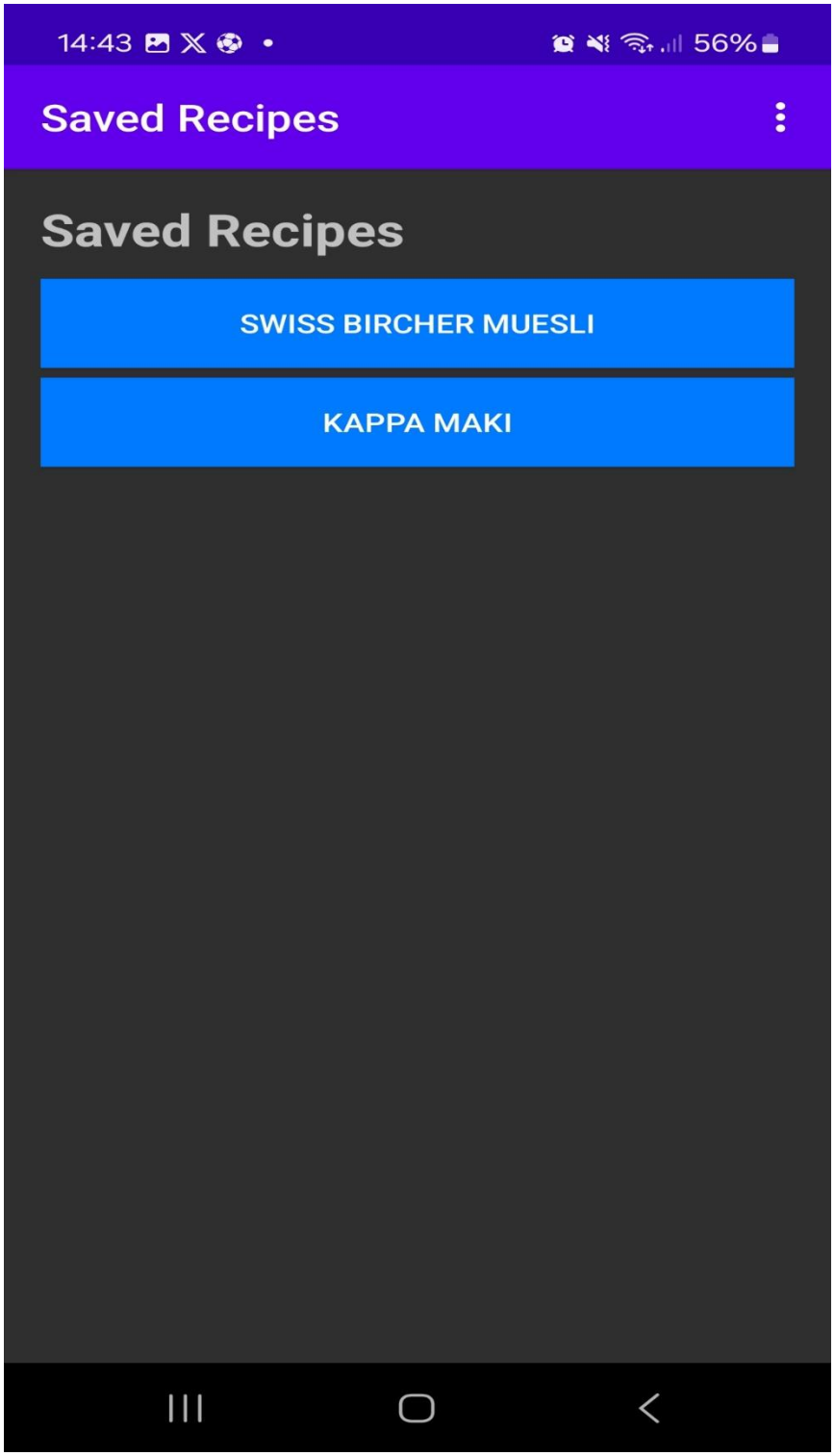
Image Processing and Machine Learning:

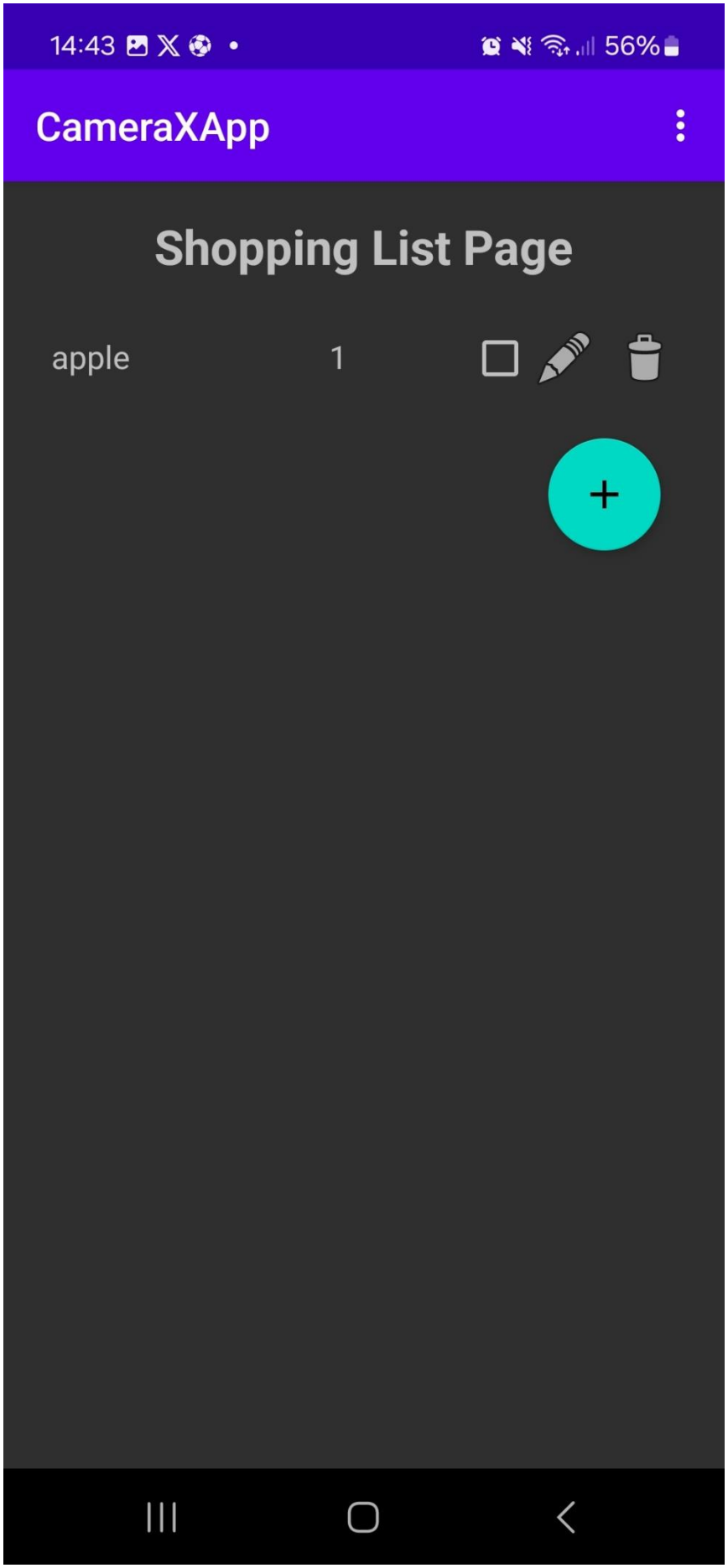
- **Android Graphics:** Standard Android classes used for handling and manipulating image data (Bitmaps) captured from the camera or gallery before feeding it to the ML models.
- **TensorFlow:** Employed for the on-device image recognition option. This involved integrating a TensorFlow Lite model via a helper class to identify ingredients directly on the user's device.

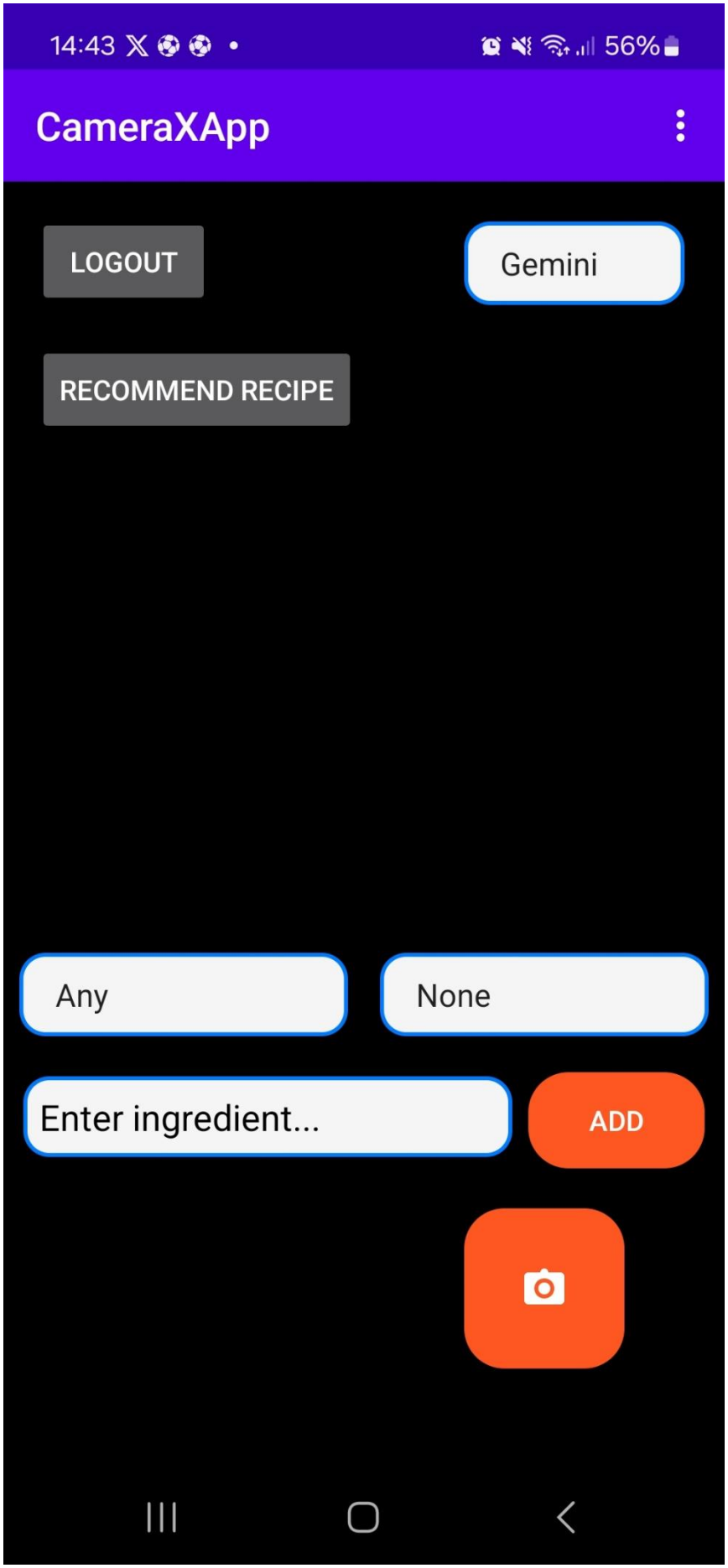
2.4 Screenshots




14:44
56%
CameraXApp
Ingredients: tomato
Select a recipe:
RETAKE PHOTO
GREEN TOMATO SALAD
EASY TOMATO SOUP
JALAPENO QUESO WITH GOAT CHEESE
EGGPLANT PIZZETTE
SIMPLE SPINACH AND TOMATO FRITTATA
GOAT CHEESE PESTO PIZZA
TOMATO STACK SALAD
SUMMER BEET SIDE SALAD
OKRA TOMATO SALAD


14:43
56%
Saved Recipes
Saved Recipes
SWISS BIRCHER MUESLI
KAPPA MAKI


14:43
56%
CameraXApp
Shopping List Page
apple
1


14:43
56%
CameraXApp
LOGOUT
Gemini
RECOMMEND RECIPE
Any
None
Enter ingredient...
ADD

3. Description of Conformance to Specification and Design

3.1 Original Specification vs. Final Submission.

Area / Feature	Specification Promised	Final App Delivers	Status
Ingredient capture	• Take one or many photos, accept/reject detections • Barcode scanning • Manual entry	• Photo input supported (choice of Google Gemini or custom TFLite) • Manual entry retained • Barcode route dropped	Core met (barcode omitted)
Ingredient recognition backend	Single custom on-device model	Dual-path: cloud Gemini and on-device TFLite	Enhanced
Ingredient verification	Call Open Food Facts for detailed breakdown	Not implemented	Missing
Recipe search & ranking	Use Spoonacular; rank by best ingredient match	Implemented exactly as specified	Met
Dietary / intolerance filters	User-set filters applied to search	Implemented (vegetarian, vegan, gluten-free, etc.)	Met
Healthy alternative suggestions	AI swaps high-calorie items for healthier ones	Not implemented	Missing
Personalised recipe recommender	Recommend based on saved / frequent ingredients	Implemented (uses ingredient-usage history)	Met
Shopping list	Full CRUD; add items from recipes, photo, barcode; AI suggestions	CRUD and add-from-recipe implemented; AI list suggestions & barcode path omitted	Partially met
User profile	Store favourites, dietary prefs, username & UI theme	Favourites + dietary prefs present; username/theme customisation not delivered	Partially met

3.2 Changes from Original Plan

- **Dual Image-Recognition Path (Added):** Gemini was integrated to boost accuracy, act as a fail-safe if the on-device model fails, and allow future multi-ingredient detection in a single photo.
- **Barcode Scanning & OpenFoodFacts Lookup (Removed):** Spoonacular already returns detailed nutritional info, making OpenFoodFacts redundant. Barcode input was low-priority given photo and text entry; cut for time.

- **Username & UI Theme Personalisation (Removed):** Time underestimated; effort re-allocated to stabilising core flows.
- **Healthy-Alternative Suggestions (Removed):** Dietary/intolerance filters satisfy the main use-case; feature dropped to meet deadline.
- **AI-Driven Shopping-List Suggestions (Deferred):** Requires additional usage analytics and recommendation logic not completed within schedule.

3.3 Justification for Differences

Real-world time pressure – Development began in early **October**, running in parallel with other final-year module projects.

- **Steep learning curve** – This was my first large project in **Kotlin**, my first time wielding **CameraX**, **Firebase Auth / Firestore**, and an on-device **TensorFlow Lite** pipeline.
- **Dataset scarcity → Gemini** – A public, well-labeled image dataset for raw food ingredients doesn't really exist. Collecting and annotating thousands of photos would have taken weeks. Integrating Gemini Vision solved the coverage problem.
- **API focus, not fragmentation** – **Spoonacular** already returns full macro-/micronutrient panels. Adding OpenFoodFacts would have duplicated data and introduced an extra failure points.
- **Healthy-swap engine postponed** – Credible "light" substitutions require either a large nutrition-substitution corpus or a handcrafted rule base—neither of which was obtainable inside the project window.

4. Description of Learning

4.1 Technical Learning

4.4.1 New Technologies I Used

- Android + Kotlin coroutines/Flow, ViewBinding.
- TensorFlow Lite & Keras model editing
- Google Gemini Vision
- Retrofit / OkHttp
- Firebase
- App architecture & UX

4.1.2 Challenges Faced

- **Ingredient-image dataset scarcity** – No public dataset covered enough raw ingredients for an on-device model.
  Solution: Used Google Gemini Vision as a fall-back recogniser, guaranteeing ingredient coverage without weeks of data gathering.
- **Capturing multiple ingredients in one go** – Early builds only handled a single item per shot.
  Solution: added a “Retake” button so users can snap multiple images. In the future they will be combined into one Gemini request.
- **ChatGPT API unavailable during early tests** – Likely tied to my personal subscription tier.
  Solution: Stopped ChatGPT integration, focused on Spoonacular + Gemini.
- **Time management while juggling four other final-year modules** – Small tasks (gradle conflicts, layout tweaks)took longer than expected.
- **First time planning a full stack alone** – Estimation errors and scope creep were common.
- **Under-estimating Firebase setup** – Security-rule errors and auth-state timing bugs blocked progress.
  Solution: worked through the emulator, added AuthStateListener.
- **Getting used to Kotlin’s import / Gradle dependency flow** – Mis-scoped imports and duplicate modules at the start.
- **Code becoming unmanageable** – Everything lived in one activity file.
  Solution: refactored into ViewModels, Repositories, and separate Activities/Fragments, improving readability and testability.
- **Ingredient list not ordered by relevance** – The “search by ingredient” call produced noisy results.
  Solution: replaced it with Spoonacular’s complex query
- **Miscellaneous troubleshooting** – rotation crashes fixed by handling configChanges, duplicate-view IllegalStateException solved by guarding addView, null-safe Bitmap loads prevented decode crashes.

4.2 Personal Learning

This was the first time I had to build and finish a full mobile application on my own, and at the start I felt completely swamped. The code base grew faster than I could keep track of, crashes piled up, and I wasn’t sure which feature to tackle next. After a couple of chaotic weeks I stepped back and applied a few habits I’d picked up during my internship and from chats with my mentor, **Chris Staff**. I set aside fixed, weekly blocks—just a few hours, but always the same hours—and wrote a short checklist for everything that had to be done before each deadline. Breaking the work into small,

clear items (“camera capture”, “save to Firestore”) meant I could finish one piece, tick it off, and move on without losing the thread. Regular catch-ups with Chris helped me judge which tasks really mattered and which could wait, so scope stayed realistic.

Working this way taught me a lot in a short time: how to size tasks more accurately, how to keep code tidy by splitting huge files into separate classes early, and how to ask focused questions when I hit a wall. Those skills—basic scheduling, task slicing, seeking quick feedback, and organising code—are exactly what let me deliver a complete app this year; earlier in the course I simply didn’t have the routines or confidence to manage work of this scale.

5. Review of the Project

- **What went right**: The core “photo → ingredient → recipe” loop works smoothly, the app stays stable on rotation, and shipping both a TFLite model and Gemini Vision gave coverage for almost every food item I tried.
- **What went wrong**: I badly underestimated the time needed for a lot of the smaller tasks. Early code lived in one file and became a tangle; crashes and API-key issues cost several unplanned days. Feature creep (barcode scan, healthy swaps) forced last-minute cuts.
- **Outstanding / missing:** Barcode input, nutrition “healthy swap” suggestions, dark mode, biometric login, and UI theming are still on the wish list. The Gemini model handles single-item photos well, but a multi-ingredient classifier remains future work.
- **If starting again:** I’d spend the first week drawing up a realistic feature list and project scope, design the package structure before writing code, and set hard mini-deadlines for data, UI, and testing. A smaller, clearer target would have saved many late fixes.

6. Acknowledgements

I would like to thank my supervisor, **Chris Staff**, for providing invaluable guidance and support throughout the development of this project. Additionally, the use of AI tools such as ChatGPT significantly aided in bug fixes and assisted with various implementation details.

7. Extra Reading

For additional information on technologies and APIs used within this project, refer to the following documentation:

- Android Developer Documentation (for Android development best practices and coding guidelines)
- Spoonacular API Documentation (for comprehensive API reference and implementation examples)
- TensorFlow Lite Documentation (TensorFlow implementation and integration details)