Fridgify: A Smart Recipe Recommender Mobile Application

Author: Jingbo Wang, Chengkai Yang

Course: CS 501

Institution: Boston University

Date of Submission: Dec 15, 2024

Abstract

This report presents *Fridgify*, a mobile application developed for Android that aims to reduce food waste, streamline meal planning, and offer personalized culinary experiences. By recommending recipes based on users' available ingredients and preferences, Fridgify alleviates the need for extensive grocery shopping and promotes efficient use of on-hand resources. Built with modern Android development techniques (Jetpack Compose, ViewModel, Room) and integrated with external APIs (FatSecret, OpenFoodFacts, Google Custom Search) as well as Firebase services (Authentication, Realtime Database, Analytics, Cloud Messaging), the application demonstrates robust data handling, offline capability, and a user-centric interface. This report details the app's concept, architecture, implementation, challenges, testing procedures, and future directions. The code snippets and classes referenced throughout provide a deeper technical understanding of Fridgify's core functionality.

Introduction

Food waste and complexity in meal planning are ongoing challenges for many individuals. *Fridgify* addresses these issues by recommending recipes that can be made from the ingredients already in the user's pantry. Additionally, the application personalizes suggestions according to dietary restrictions and culinary preferences. In essence, Fridgify:

- Reduces food waste by prioritizing on-hand ingredients.
- Saves time by simplifying meal planning and grocery decisions.
- Delivers a tailored cooking experience through user preferences, dietary needs, and analytics-driven insights.

The features were developed using current best practices in Android development and guided by a scalable, maintainable architecture. This report provides a detailed overview of the application, references specific code components, and discusses modifications from the initial proposal.

Project Overview

Application Name: Fridgify - Smart Recipe Recommender

Target Audience:

- **Busy Individuals:** Quickly find recipes using ingredients already available.
- **Home Cooks & Enthusiasts:** Explore new recipes without extra grocery runs.
- **Dietary Restriction Cases:** Filter recipes for vegan, vegetarian, or other specific health conditions.

Key Features:

1. **Ingredient-Based Recommendations:**
   Users input their available ingredients, and the app fetches suitable recipes.
2. **Personalized Preferences & Dietary Filters:**
   Customized recommendations based on saved user profiles, dietary restrictions (e.g., vegetarian/vegan), and favorite cuisines.
3. **Step-by-Step Instructions:**
   Detailed cooking steps guide users through the recipe, making the experience accessible to any skill level.
4. **Offline Functionality with Room Caching:**
   Previously viewed recipes, ingredients, and user data remain accessible without an internet connection.
5. **Push Notifications (Firebase Cloud Messaging):**
   Users receive timely reminders about new recipes, seasonal ingredients, and expiring pantry items.
6. **Usage Analytics (Firebase Analytics):**
   Tracking user interactions and preferences helps refine future recommendations and identify feature enhancements.

Technical Architecture and Design

Platform and Tools:

- **Android:** Target minSdkVersion = 24 and targetSdkVersion = 34.
- **Kotlin:** The primary programming language, ensuring concise code and interoperability with Android frameworks.

- **Jetpack Compose:** For building reactive and modern UI components.
- **Architecture Pattern:** MVVM (Model-View-ViewModel) for a clear separation of concerns.

Architectural Layers:

1. **UI Layer (Compose Screens):**
   Screens such as HomeScreen, IngredientScreen, and ProfileScreen are implemented using Jetpack Compose (HomeScreen.kt, IngredientScreen.kt). Composables rely on ViewModels to receive state updates and manage UI changes.
2. **ViewModels:**
   - **HomeViewModel and IngredientViewModel:**
     Handle state management for recipes and ingredients. For instance, HomeViewModel fetches and caches recipe data, while IngredientViewModel manages the user's ingredient list, updates quantities, and syncs with remote data sources.
   - **ProfileViewModel:**
     Manages user authentication, registration, and preference loading via Firebase Authentication and Realtime Database.
3. ViewModels observe data from repositories using StateFlow or LiveData, ensuring the UI reacts to data changes in real-time.
4. **Data & Domain Layer:**
   - **Repositories (e.g., IngredientRepository, RecipeRepository, UserRepository):**
     Central points for data handling, encapsulating logic for fetching, caching, synchronizing, and updating data from both local (Room) and remote (Firebase, APIs).
   - **Room Database:**
     Entities like IngredientEntity and RecipeModel are defined to persist data locally (IngredientDao, RecipeDetailDao). The Room database ensures offline access and improved performance.
   - **Remote Data Fetching:**
     - **Retrofit and OkHttp:** Interact with FatSecret, OpenFoodFacts, and Google CSE APIs. For instance, FatSecretAuthService.kt, GoogleImageSearchService.kt, OpenFoodFactsService.kt handle authentication and data retrieval.
     - **Firebase Realtime Database:** Syncs user data (ingredients, favorites) across devices in real-time.
5. **Networking & Services:**
   - **FatSecret API:** Used for authenticating requests and fetching detailed nutritional info. The code in FatSecretAuthService.kt manages OAuth tokens and RetrofitInstance.kt creates service instances.

- **OpenFoodFacts API:** Integrates with OpenFoodFactsService.kt to retrieve ingredient details from barcodes.
- **Google CSE API:** Invoked by GoogleImageSearchService.kt to fetch images for visual UI enhancements.

Detailed Code References

Data & API Integration Example:

- **RetrofitInstance.kt:**
  Centralizes Retrofit and OkHttp configuration. It sets the base URLs, interceptors, and converters for JSON parsing.
- **FatSecretAuthService.kt:**
  Contains logic to obtain OAuth tokens and call FatSecret endpoints. This service ensures the app can securely fetch detailed nutritional info on demand.

Ingredient Management:

- **IngredientScreen.kt:**
  Demonstrates how users can view, edit, and delete ingredients. Code references show AnimatedVisibilitytransitions to smoothly animate the removal of ingredients.
- **IngredientViewModelFactory and IngredientViewModel:**
  Manage ingredient data retrieval from local IngredientRepository and remote Firebase storage. The updateIngredientQuantity() and deleteIngredient() methods ensure consistent and synchronized updates.
- **IngredientEntity and IngredientDao:**
  Define schema and CRUD operations for storing ingredient details in the local Room database.

Recipe Retrieval and Display:

- **HomeViewModel & HomeScreen.kt:**
  Fetch a list of recommended recipes based on the user's ingredients and preferences. HomeViewModel relies on RecipeRepository and RecipeDetailRepository for data. It uses MutableStateFlowfor loading states and selectedRecipeDetails to handle user-triggered detail requests.
- **RecipeDetailRepository:**
  Retrieves detailed recipe instructions and nutrition. This repository layer fetches data from local and remote sources, caching results to optimize performance.

Offline Support:

- **Local Caching with Room:**
  The IngredientRepository and RecipeRepository first check local caches before making network calls.
- **Offline Modifications:**
  The app marks changes as "pending sync" when offline. When connectivity is restored, a WorkManager DataSyncWorker.kt job triggers synchronization with Firebase. The snippet in DataSyncWorker.kt shows how background tasks ensure continuous data integrity.

Barcode Scanning and Sensors:

- **CameraPreview.kt:**
  Integrates the camera for barcode scanning using ML Kit. The provided code example shows how the gyroscope sensor is used to detect device stability and trigger autofocus. The light sensor data is used to toggle the flashlight automatically.
  Code snippet (as shown in the provided content) demonstrates how the SensorManager and SensorEventListenerhandle real-time sensor events and adjust camera behavior.

User Interaction and Animations

Fridgify employs Jetpack Compose animations for a polished user experience:

- **AnimatedVisibility:**
  Used in IngredientItemCard() (in IngredientScreen.kt) for smooth fade-in/out transitions during ingredient deletion or updates.
- **Real-Time Search Feedback:**
  As users type ingredient names, qrScannerViewModel executes delayed network calls (via coroutineScope.launch { ... delay(1000) }) and updates search results live. This is evident in IngredientScreen.kt where searchedFoods and selectedFood states are tracked.

Testing and Quality Assurance

- **Unit Testing:**
  Repositories and ViewModels have JUnit-based tests. For example, unit tests validate IngredientRepository to ensure correct handling of adding, updating, and deleting ingredients.
- **Firebase Crashlytics:**
  Implemented to capture runtime exceptions. For instance, try-catch blocks in HomeViewModel log errors locally and call FirebaseCrashlytics.getInstance().recordException(e) for real-time crash reporting.
- **Firebase Analytics:**
  Custom events (e.g., view_recipe) are logged to understand user behavior, recipe

popularity, and ingredient usage trends. This data guides future refinements, validated by code in AnalyticsLogger classes.

Challenges and Solutions

1. **Data Synchronization Between Offline and Online States:**
   - **Challenge:** Users can update ingredients offline, requiring correct merging of changes once online.
   - **Solution:** Used a two-way sync strategy. Operations are either applied immediately if online or marked as pending. DataSyncWorker.kt ensures synchronization once the network is available.
2. **Accurate Personalized Recommendations:**
   - **Challenge:** Balancing complexity (dietary restrictions, available ingredients) with quick, relevant results.
   - **Solution:** Reliance on FatSecret for nutritional data and OpenFoodFacts for product identification. Future iterations plan to employ machine learning for even more refined recommendations.
3. **Complex Dietary Filtering & Advanced Features:**
   - **Challenge:** Advanced dietary filters and social features were initially planned but proved time-consuming.
   - **Solution:** The current version supports basic dietary restrictions. The codebase (notably ProfileViewModeland UserRepository) was structured to facilitate future expansions.

Modifications from Original Proposal

- **Dietary Filtering:**
  Reduced complexity in filtering logic. Basic restrictions (vegan, vegetarian) are implemented; advanced filters (e.g., low-FODMAP) postponed.
- **Machine Learning Integration:**
  Deferred due to project time constraints. Current code is modular, allowing integration of ML-based personalization in the future.
- **User-Generated Content & Social Sharing:**
  Community features and social integrations have been delayed. The focus remains on stable core functionalities and reliable data handling.

Future Work

- **Enhanced Dietary and Allergy Filters:**
  Extend current filters for more nuanced recommendations based on user health goals.
- **Machine Learning Models:**
  Implement ML-based ranking to improve recommendation relevance over time using accumulated analytics data.
- **Community Features & Social Sharing:**
  Allow users to share recipes, rate them, and build a community-driven recipe recommendation ecosystem.
- **Adaptive Notifications:**
  Refine push notifications to consider user behavior, seasonal trends, and expiring items for more context-aware alerts.

Project Link

Github Link: https://github.com/NullPointer-coder/Chengkai-Jingbo-JK-FinalProject.git

Conclusion

Fridgify successfully meets its core objectives: it helps users make the most of their existing ingredients, provides personalized and informed recipe recommendations, and maintains seamless offline functionality. By leveraging Jetpack Compose, Room, Firebase, and external APIs, the app achieves a robust architecture, efficient data handling, and a fluid user experience.

Although certain advanced features were deferred, the current codebase lays a strong foundation for future enhancements. As user insights from Firebase Analytics are interpreted, subsequent versions will incorporate more complex filtering, machine learning-driven personalization, and community-driven content sharing.

References

- Android Developers Documentation: https://developer.android.com/
- FatSecret API Documentation: https://platform.fatsecret.com/api/
- OpenFoodFacts API Documentation: https://world.openfoodfacts.org/data
- Google Custom Search Engine (CSE) API: https://developers.google.com/custom-search
- Firebase Documentation: https://firebase.google.com/docs