# Pantry Pal: Smart Recipe Discovery App - Final Report

## 1. App Features and Design Decisions

## Core Features

**1. Ingredient-Based Recipe Search**
Pantry Pal enables users to discover recipes based on ingredients they already have. This primary feature helps reduce food waste and simplifies meal planning. Users can input up to five ingredients, and the app retrieves recipes that use these ingredients through the Edamam Recipe API integration.

Design Decision: We implemented a flexible multi-ingredient input system that allows users to add and remove ingredients dynamically, with a capped maximum of five ingredients to keep the UI clean and the API calls efficient. This strikes a balance between flexibility and usability.

**2. "Shake for Surprise" Feature**
This playful feature utilizes the device's accelerometer sensor to provide random recipe suggestions when the user shakes their phone. It adds an element of fun and serendipity to the meal planning process.

Design Decision: We implemented a `ShakeSensorManager` class to detect shake gestures with adjustable sensitivity. We included a cooldown period between shakes to prevent accidental triggers and added a visual feedback system via Snackbar notifications to confirm the action was recognized.

**3. Personal Cookbook (Saves)**
Users can save favorite recipes locally for offline access, organizing them in a digital cookbook that's available anytime without requiring internet connectivity. Also, we have a remote database in case you want to access it on a different platform.

Design Decision: We chose a Room database implementation for local storage, using a clearly defined entity structure for saved recipes with efficient query methods. We implemented a visually appealing grid layout for the cookbook view that adapts to different screen sizes. And for different platforms save we used firebase remote.

**4. Pantry & Grocery Helper**
This feature allows users to track pantry items and automatically generate shopping lists based on missing ingredients for recipes they want to try.

Design Decision: We integrated inventory management with recipe exploration by cross-referencing pantry items against recipe ingredients. The grocery list feature includes CRUD operations with intuitive swipe-to-delete functionality and checkboxes for shopping convenience.

**5. Multi-Device Support with Responsive UI**
The app provides distinct layouts optimized for both phones and tablets, ensuring a great user experience across different device types.

Design Decision: We implemented conditional UI rendering using Jetpack Compose's responsive design principles. The app detects screen size and adjusts the layout accordingly—phones get a vertical, stack-based UI, while tablets utilize a multi-pane layout that takes advantage of the additional screen real estate.

## UI/UX Design Philosophy

Our design approach focused on creating a clean, intuitive interface that emphasizes content and functionality without overwhelming users. Key UI/UX design decisions include:

1. Material Design Implementation: Adhering to Material Design 3 principles for consistent visual language and behavior
2. Accessibility Considerations: Supporting screen readers, maintaining proper contrast ratios, implementing dark mode, and providing text scaling compatibility
3. Progressive Disclosure: Presenting information in stages to avoid cognitive overload
4. Visual Feedback: Incorporating loading indicators, transition animations, and clear error states
5. Intuitive Navigation: Using a bottom navigation bar for primary sections with logical screen flows

# 2. Architecture and Technologies Used

## Architecture Pattern: MVVM

We implemented the Model-View-ViewModel (MVVM) architecture pattern for several key reasons:

1. Separation of Concerns: The clear separation between UI (View), data logic (Model), and UI logic (ViewModel) made the codebase more maintainable and testable
2. UI State Management: ViewModels preserve UI state during configuration changes
3. Testability: Business logic in ViewModels can be unit tested independent of Android framework components
4. Jetpack Compose Integration: MVVM pairs naturally with Compose's declarative UI paradigm

The architecture consists of:

- Model: Database entities and network response objects
- View: Composable functions that render UI elements
- ViewModel: State holders that coordinate data flow and transform it for display

# 3. Challenges Faced and Solutions

## Challenge 1: Responsive UI for Different Screen Sizes

Problem: Creating a UI that works well on both phone and tablet form factors without duplicating code.

Solution: We implemented a responsive design system using Jetpack Compose that:

- Detects screen size at runtime using `LocalConfiguration`
- Conditionally renders different layouts based on screen width (threshold set at 600dp)
- Uses shared component functions with adaptive parameters
- Maintains a consistent component library across form factors

This approach allowed us to reuse much of our UI code while still providing optimal experiences for each device type.

## Challenge 2: Shake Detection Reliability

Problem: Initial shake detection was either too sensitive (triggering accidentally) or not sensitive enough (requiring excessive force).

Solution: We implemented a customizable sensitivity threshold and debounce mechanism in the `ShakeSensorManager` class. We:

- Added a configurable shake threshold (initially set to 11.5f)
- Implemented a time-based filter to prevent multiple detections within 1000ms
- Calculated acceleration using a delta between previous and current readings
- Used vector magnitude from all three axes to accurately detect shakes from any direction

These improvements created a much more reliable and satisfying shake detection experience.

## Challenge 3: Data Synchronization

Problem: Maintaining consistency between the local database and in-memory state, especially with multiple ViewModels potentially accessing the same data.

Solution: We implemented a centralized data repository pattern with:

- Single source of truth for each data type
- Kotlin Flow to emit updates to interested components
- Clear separation between read and write operations
- Furthermore, integrating the local Room database with Firebase enabled robust cloud synchronization, enhancing consistency across devices and user sessions.

This approach ensured data consistency throughout the app and simplified the debugging process when data-related issues occurred.

# 4. User Testing and Feedback

We tested our product on a total of 6 friends around us. Here's a summary of their feedback and our proposed improvements:

## Positive Feedback

1. Intuitive Interface: They found the app easy to navigate without instruction
2. Shake Feature: Most of them enjoyed the "shake for surprise" feature, finding it novel and engaging
3. Recipe Variety: All users appreciated the diversity of recipes available
4. Data Access: They valued being able to access saved recipes without an internet connection or on a different platform.
5. Responsive Design: They specifically praised the tablet multi-pane layout as efficient and well-organized

## Areas for Improvement

1. Search Refinement: users requested more filtering options (dietary restrictions, cuisine type, meal type)
    - Planned Solution: Add advanced filtering options in the search screen with collapsible filter panels
2. Recipe Scaling: users wanted the ability to scale recipes for different serving sizes
    - Planned Solution: Add a serving size adjustment feature that recalculates ingredient quantities
3. Cooking Mode: users requested a dedicated cooking mode with step-by-step instructions and timers
    - Planned Solution: The current API(Edamam) we are using does not support this feature, we are planning to use a more robust API.
4. Social Sharing: users wanted to share recipes with friends and family
    - Planned Solution: Implement sharing functionality using Android's share sheet API