Module 6 Portfolio Milestone: My Recipe App

Alexander Ricciardi

Colorado State University Global

CSC475: Platform-Based Development

Professor Herbert Pensado

March 23, 2025

Module 6 Portfolio Milestone: My Recipe App

This document is my portfolio milestone module from the Platform-Based Development CSC475 course at Colorado State University Global. It is the second step in developing my Android “My Recipe App.” The app allows a user to access meal recipes. The recipes can be stored on the user's device and fetched from TheMealDB (n.d.a) database using API calls. The UI system includes view, search, add, modify, and favorite recipe functionalities. The UI system will include view, search, add, modify, and favorite recipe functionalities. The app is being developed using Kotlin, Jetpack Compose, and the Model-View-ViewModel (MVVM) architecture. This document provides the design details for the app including the architecture, components, and overall layout of the User Interface.

App Requirements

Below is a list of the app requirements:

- Jetpack Compose (2.7.x)
- Kotlin (2.0.21)
- AndroidX Core KTX (1.15.0): Kotlin extensions for core Android functionality
- Navigation Compose (2.7.7): Navigation between screens
- Material 3: Material Design 3 components and theming system
- Room (2.6.1): Local database for storing recipes with SQLite abstraction
- Lifecycle Components (2.8.7): ViewModel and LiveData for MVVM architecture
- Retrofit (2.9.0): Type-safe HTTP client for API
- Moshi (1.15.0): JSON parser for API
- OkHttp (4.12.0): HTTP client and logging
- Coil (2.5.0): Image loading library

- Compose Runtime LiveData (1.6.2)
- Gson (2.10.1): JSON serialization/deserialization library
- Activity Compose (1.10.1): Compose integration with Activity
- Compose BOM: Bill of materials for consistent Compose dependencies

The app requires a TheMealDB API key. TheMealDB provides a free API for development and education, as the app will be deployed, I pay a $10 lifetime API key that allows “access to the beta version of the API which allows multiple ingredient filters. You also get access to adding your own meals and images. You can also list the full database rather than be limited to 100 items.” TheMealDB (n.d.b). The database uses JSON files and has 302 recipes which is a small number but enough for my application, at this point. The table below lists the API calls that can be used with a lifetime API key.

Table 1

TheMealDB API Lifetime API Key Methods

Method	URL	Premium Only	Notes
Search meal by name	www.themealdb.com/api/json/v2/<Your API Key>/search.php?s={meal_name}	No	Replace {meal_name} with the name of the meal.
List all meals by letter	www.themealdb.com/api/json/v2/<Your API Key>/search.php?f={letter}	No	Replace {letter} with the first letter of the meal.
Lookup meal details by ID	www.themealdb.com/api/json/v2/<Your API Key>/lookup.php?i={meal_id}	No	Replace {meal_id} with the meal's ID.
Single random meal	www.themealdb.com/api/json/v2/<Your API Key>/random.php	No	
10 random meals	www.themealdb.com/api/json/v2/<Your API Key>/randomselection.php	Yes	
List all meal categories	www.themealdb.com/api/json/v2/<Your API Key>/categories.php	No	
Latest Meals	www.themealdb.com/api/json/v2/<Your API Key>/latest.php	Yes	

List Categories/Area/Ingredients	www.themealdb.com/api/json/v2/<Your API Key>/list.php?{c/a/i}=list	No	Use c=list for categories, a=list for areas, i=list for ingredients.
Filter by ingredient	www.themealdb.com/api/json/v2/<Your API Key>/filter.php?i={ingredient}	No	Replace {ingredient} with the main ingredient (e.g., chicken_breast).
Filter by multi-ingredient	www.themealdb.com/api/json/v2/<Your API Key>/filter.php?i={ingr1},{ingr2},{ingr3}	Yes	Replace {ingr1}, etc. with ingredients (e.g., chicken_breast,garlic,salt).
Filter by category	www.themealdb.com/api/json/v2/<Your API Key>/filter.php?c={category}	No	Replace {category} with the category (e.g., Seafood).
Filter by area	www.themealdb.com/api/json/v2/<Your API Key>/filter.php?a={area}	No	Replace {area} with the area (e.g., Canadian).
Meal Thumbnail	/images/media/meals/{image_name}.jpg/{size}	No	Add /preview to the end of the meal image URL. {size} can be small, medium, or large.
Ingredient Thumbnail	www.themealdb.com/images/ingredients/{ingredient}-{size}.png	No	{ingredient} is the name (spaces replaced with underscores). {size} is optional (small, medium or large).

Note: The table provides a list of TheMealDB API URLs using an API key and their description. Data from " Free Recipe API" by TheMealDB (n.d.b).

To secure the API key, the key is stored in `local.properties` rather than hardcoding it in the source code. It provides better security as, if using GitHub, is automatically excluded from version control by using `.gitignore`. It also follows Android best practices by separating the app credentials from the application code. This also allows for CI/CD integration as the automated build Gradle can have different API keys for different environments without needing to modify source code. See examples below:

Code Snippet 1

The key In the local.properties File

```
#This file must *NOT* be checked into Version Control Systems, as it contains information on
your local configuration. Location of the SDK. This is only used by Gradle. For customization
when using a Version Control System, please read the header note.
sdk.dir=C\:\\Users\\User\\AppData\\Local\\Android\\Sdk
MEAL_DB_API_KEY=<Your TheMealDB Key>
```

Note: Code in the local.properties file, the path on your system can be different. The path for my specific property file is MyRecipeApp\local.properties.

Code Snippet 2

Loading API key from local.properties Code in The Gradel.kts:App file

```
android {
    namespace = "com.example.myrecipeapp"
    compileSdk = 35

    defaultConfig {
        applicationId = "com.example.myrecipeapp"
        minSdk = 24
        targetSdk = 35
        versionCode = 1
        versionName = "1.0"

        testInstrumentationRunner = "androidx.test.runner.AndroidJUnitRunner"

        // Load API key from local.properties
        val localProperties = Properties()
        if (rootProject.file("local.properties").exists()) {
            localProperties.load(FileInputStream(rootProject.file("local.properties")))
        }
        buildConfigField("String",
                         "MEAL_DB_API_KEY",
                         "\"${localProperties.getProperty("MEAL_DB_API_KEY", "1")}\"")
    }
}
```

Note: This code needs to be implemented in build.Gradel.kts file. Your Gradle file can look different and have more code within, this is just an example.

Below is the app file structure:

```
├── app
│   ├── main
│   │   ├── AndroidManifest.xml (configuration, permissions, components declaration)
│   │   ├── myrecipapp
│   │   │   ├── MainActivity.kt (Main activity)
│   │   │   ├── RecipeApplication.kt (Application class for initialization)
│   │   │   ├── data/
│   │   │   │   └── api/ (-- MODEL LAYER --)
│   │   │   │       ├── MealDbApiResponse.kt (Data classes for JSON from TheMealDB)
│   │   │   │       └── TheMealDbService.kt (Retrofit service interface for API calls)
│   │   │   ├── model/ (-- MODEL LAYER --)
│   │   │   │   ├── Category.kt (Data class for recipe categories)
│   │   │   │   ├── Converters.kt (Type converters for Room database objects)
│   │   │   │   ├── Ingredient.kt (Data class for recipe ingredients)
│   │   │   │   ├── OnlineRecipe.kt (Data class for recipes from TheMealDB)
│   │   │   │   ├── Recipe.kt (Main data class for recipe, stored locally)
│   │   │   │   ├── RecipeDao.kt (Data Access Object for Room database operations)
│   │   │   │   └── RecipeDatabase.kt (Room database configuration and setup)
│   │   │   ├── repository/ (Repository layer)
```

```
│   │   │   │   ├── LocalRecipeRepository.kt (local database operations)
│   │   │   │   ├── OnlineMealRepository.kt (TheMealDB API operations)
│   │   │   │   └── RecipeRepository.kt (Interface for repository functionality)
│   │   │   ├── ui/ (-- VIEW LAYER --)
│   │   │   │   ├── components/
│   │   │   │   │   ├── CommonComponents.kt (UI components like headers/footers)
│   │   │   │   │   └── RecipeComponents.kt (Recipe-specific UI components)
│   │   │   │   ├── navigation/
│   │   │   │   │   ├── AppFooter.kt (Bottom navigation bar component)
│   │   │   │   │   └── Navigation.kt (Navigation graph and route definitions)
│   │   │   │   ├── screens/
│   │   │   │   │   ├── HomeScreen.kt (Main app home screen with recent recipes)
│   │   │   │   │   ├── OnlineRecipeDetailScreen.kt (Detail recipe for online recipes)
│   │   │   │   │   ├── PlaceholderScreen.kt (placeholder for unimplemented features)
│   │   │   │   │   ├── RecipeFormScreen.kt (Form/screen for adding/editing recipes)
│   │   │   │   │   ├── RecipeListScreen.kt (List recipe of all saved recipes)
│   │   │   │   │   ├── RecipeViewScreen.kt (Detailed recipe of a single recipe)
│   │   │   │   │   ├── SearchDialog.kt (Dialog for search)
│   │   │   │   │   ├── SearchResultsScreen.kt (Display of search results)
│   │   │   │   │   ├── SearchScreens.kt (search screen implementations)
│   │   │   │   │   ├── TheMealDbScreen.kt (Screen for searching TheMealDB recipes)
│   │   │   │   │   └── UnifiedSearchScreen.kt (Combined search)
│   │   │   │   ├── theme/
│   │   │   │   │   ├── Color.kt (Color for the app)
│   │   │   │   │   ├── Theme.kt (Theme including dark/light modes)
│   │   │   │   │   └── Type.kt (Typography definitions)
│   │   │   │   └── viewmodel/ (-- VIEWMODEL Layer --)
│   │   │   │       ├── OnlineRecipeViewModel.kt (ViewModel for TheMealDB recipes)
│   │   │   │       ├── RecipeViewModel.kt (ViewModel for local recipe operations)
│   │   │   │       └── SearchViewModel.kt (ViewModel for search functionality)
│   │   │   └── res/
│   │   │       ├── drawable/
│   │   │       │   └── ic_launcher_background.xml (App icon background vector)
│   │   │       ├── mipmap-*/
│   │   │       │   └── (App icon resources at different resolutions)
│   │   │       ├── values/
│   │   │       │   └── colors.xml (Color for light theme)
│   │   │       ├── values-night/
│   │   │       │   └── colors.xml (Color for dark theme)
│   │   │       └── xml/ (Additional XML configuration files)
```

The list below lists the files by MVVM layers:

Application Layer (handles the app configuration and initialization) :

- `MainActivity.kt`: Entry point of the app, sets up the Jetpack Compose UI and navigation.
- `RecipeApplication.kt`: Custom Application class that initializes app-wide components like the database and repositories.

- AndroidManifest.xml: App components, permissions, and configuration.

Data Layer (Model):

- `Recipe.kt`: The main data class, a recipe with properties like name, ingredients, and instructions.
- `Ingredient.kt`: Data class for recipe ingredients with name and measurement.
- `Category.kt`: Data class for recipe categories (breakfast, dinner, etc.).
- `RecipeDatabase.kt`: Sets up Room database, defines database version, migrations, and provides DAOs.
- RecipeDao.kt: Interface with SQL queries, database operations (insert, update, delete, query recipes).
- `Converters.kt`: Type converters for objects (like lists) that can't be directly stored in SQLite.
- `OnlineRecipe.kt`: Data class for recipes fetched from TheMealDB.

Repository Layer:

- `RecipeRepository.kt`: Interface for methods for recipe operations.
- `LocalRecipeRepository.kt`: Local database operations via Room.
- `OnlineMealRepository.kt`: Feteches recipes from TheMealDB API.
- `TheMealDbService.kt`: Retrofit service interface with API endpoints for TheMealDB.
- `MealDbApiResponse.kt`: Data classes that model JSON responses from TheMealDB.

ViewModel Layer:

- `RecipeViewModel.kt`: Local recipes (adding, editing, deleting).
- `OnlineRecipeViewModel.kt`: Manages state and operations for online recipes from TheMealDB.
- `SearchViewModel.kt`: Ssearch functionality.

UI Layer (View):

- `HomeScreen.kt`: Main screen showing recent recipes.
- `RecipeFormScreen.kt`: UI for adding or editing recipes with validation.

- `RecipeViewScreen.kt`: Detailed recipe of a single recipe with all information.
- `RecipeListScreen.kt`: Grid or list recipes of all saved recipes with filtering options.
- `TheMealDbScreen.kt`: Interface for searching and displaying TheMealDB recipes.
- `SearchScreens.kt`: Search functionality.
- `UnifiedSearchScreen.kt`: Combined search screen.
- `SearchResultsScreen.kt`: Displays search results.
- `SearchDialog.kt`: Dialog popup for search.
- `OnlineRecipeDetailScreen.kt`: Detailed view for online recipes.
- `PlaceholderScreen.kt`: Generic placeholder for features.

Navigation:

- `Navigation.kt`: Sets up the navigation graph with routes to screens.
- `AppFooter.kt`: Bottom navigation bar.

UI Components

- `CommonComponents.kt`: UI components like headers, buttons, and loading indicators.
- `RecipeComponents.kt`: Recipe-specific components like recipe cards, and ingredient lists.

Theming:

- `Theme.kt`: Theme including color schemes for both light and dark modes.
- `Color.kt`: Color constants.
- `Type.kt`: Typography styles (font families, sizes, weights).
- `colors.xml` (values): Color for light theme.
- `colors.xml` (values-night): Color for dark theme.

Reflection

The development of the code source for this app was very challenging; I didn't get much sleep and drank a lot of coffee during that time. However, the documentation from my previous milestone assignments was very helpful, notably the architecture overview and user interface layout sections from my module 4 milestone documentation provided a strong foundation to craft my app source code. I encounter many issues, too many to list. Nonetheless, I implemented several strategies to solve the issue like implementing the source code in phase, I broke down the MVVM architecture into 4 implementation phases starting with the Model (data) layer, followed by the Repository Sub-Model layer, the ViewModel layer, and finally the View layer. As described in my Portfolio Milestone 4 Assignment (Ricciardi, 2025), the MVVM architecture separates the User Interface (UI) from application logic, by keeping the UI layer from directly manipulating data. This is implemented by creating an intermediary layer between the UI layer and the data layer. These layers are defined by three main components, which are the View (UI layer), the ModelView (intermediate layer), and the Model (the logic or data layer). Note that the Repository layer can be considered a Sub-Model layer acting as a mediator between the Model layer data sources and the ViewModel layer adding an extra safety layer protecting the data sources.

Below is a representation of the different MVVM components related to each other and an example of data flow:

- View depends on ViewModel
- ViewModel depends on Repository
- Repository depends on Model (Room database, API responses)

Adding a recipe example:

1. View: User fills the form and taps Save
2. View → ViewModel: UI calls viewModel.addRecipe(recipe)
3. ViewModel → Repository: ViewModel delegates to repository (or directly to DAO in simpler cases)
4. Repository → Model: Recipe is inserted in Room database
5. Model → Repository → ViewModel → View: Flow/LiveData emits updated recipe list
6. View: UI refreshes to show new recipe

In phase 1, I implemented the model layer of the MVVM architecture. In this layer, I handled the local data using the Room library for local data, the Retrofit library for API calls, the OkHttp library as a HTTP client, the Moshi library integrated with the Retrofit library for JSON Mapping, the Gson library used to convert complex JSON for the Room database, Kotlin Coroutines (e.g. the keyword 'suspended') for asynchronous programming (API calls), and the Coil library used for image (thumbnail) loading and caching. The main issue I encountered in this phase is that TheMealDB API required specific formatting for search queries, more specifically for ingredients and area (country) searches. Spaces needed to be replaced with underscores for ingredients, and country names needed to be normalized. My initial implementation of the app only had live search working for recipe names, not other fields. The solution was solved by implementing, in the ViewModel layer, in the module in the viewmodel/OnlineRecipeViewModel.kt file, a method that replaces spaces with underscores and converts to lowercase, making queries like "Olive Oil" become "olive_oil" see code below:

Code Snippet 3

Ingredient Search Formatting (line 416)OnlineRecipeViewModel.kt

```
SearchType.INGREDIENT -> query.trim().replace(" ", "_").lowercase()
```

To handle various countries, the solution is to handle (e.g., "usa", "united states", "american" all map to "American"), this was handled in the same files, see code below:

Code Snippet 4

Area/Country Search Normalization (lines 416-448) OnlineRecipeViewModel.kt

```
SearchType.AREA -> {
    // TheMealDB API is very particular about area names
    // First capitalize correctly
    val capitalizedArea = query.trim().replaceFirstChar {
        if (it.isLowerCase()) it.titlecase(java.util.Locale.getDefault()) else it.toString()
    }

    // Then handle special cases
    when (capitalizedArea.lowercase()) {
        "us", "usa", "united states", "american" -> "American"
        "uk", "united kingdom", "england", "british", "great britain" -> "British"
        ...
        "netherlands", "dutch" -> "Dutch"
        "turkey", "turkish" -> "Turkish"
        "portugal", "portuguese" -> "Portuguese"
        else -> capitalizedArea // Use capitalized version for other countries
    }
}
```

To handle category normalization, it was also implemented in the same file see code below:

Code Snippet 5

Category Search Normalization (lines 392-415) OnlineRecipeViewModel.kt

```
SearchType.CATEGORY -> {
    // TheMealDB API requires exact category names
    val normalizedCategory = query.trim().replaceFirstChar {
        if (it.isLowerCase()) it.titlecase(java.util.Locale.getDefault()) else it.toString()
    }

    // Convert common terms to official TheMealDB categories
    when (normalizedCategory.lowercase()) {
        "breakfast" -> "Breakfast"
        "chicken", "poultry" -> "Chicken"
        "dessert", "sweets", "cake", "cakes" -> "Dessert"
        "goat" -> "Goat"
        "lamb" -> "Lamb"
        "pasta", "noodles" -> "Pasta"
        "pork" -> "Pork"
        "seafood", "fish" -> "Seafood"
        "side", "sides", "side dish" -> "Side"
        "starter", "starters", "appetizer", "appetizers" -> "Starter"
        "vegan" -> "Vegan"
        "vegetarian", "veggie" -> "Vegetarian"
        "beef", "steak" -> "Beef"
        "miscellaneous", "misc" -> "Miscellaneous"
        else -> normalizedCategory  // Use normalized version for other categories
    }
}
```

I also encountered a problem implementing the User Interface (UI), in the view layer, particularly misaligned icons and filter chips that didn't fit on the screen. These issues were addressed by trying different layouts and implementing iterative adjustments to layout containers, padding, and alignment. Another issue I encountered was implementing the Navigation component of the UI, this was a new concept for me making it very interesting but also a little more challenging than expected; this created some issues that I solved by strictly adhering to browse>view>edit pattern and linking to the correct screens, still, some miner linking issue remains that will be fixed (one route links to the wrong screen) during the testing phase of this assignment.

In conclusion, this assignment was both extremely challenging and rewarding. The project MVVM architecture added extra complexity to the project; however, it is good practice to implement it as it separates components, contributing to the safety, robustness, and maintainability of the app. I encountered many issues when implementing this architecture and the app features; it will take many pages to describe and list them all. Nonetheless, they were addressed by implementing the app in phases following the MVVM architecture and by adopting a modular programming approach, which breaks down problems into smaller ones. This is demonstrated by the large number of files that compose this project. Ultimately the application has a solid foundation for testing, deployment, maintenance, and future development and expansion.

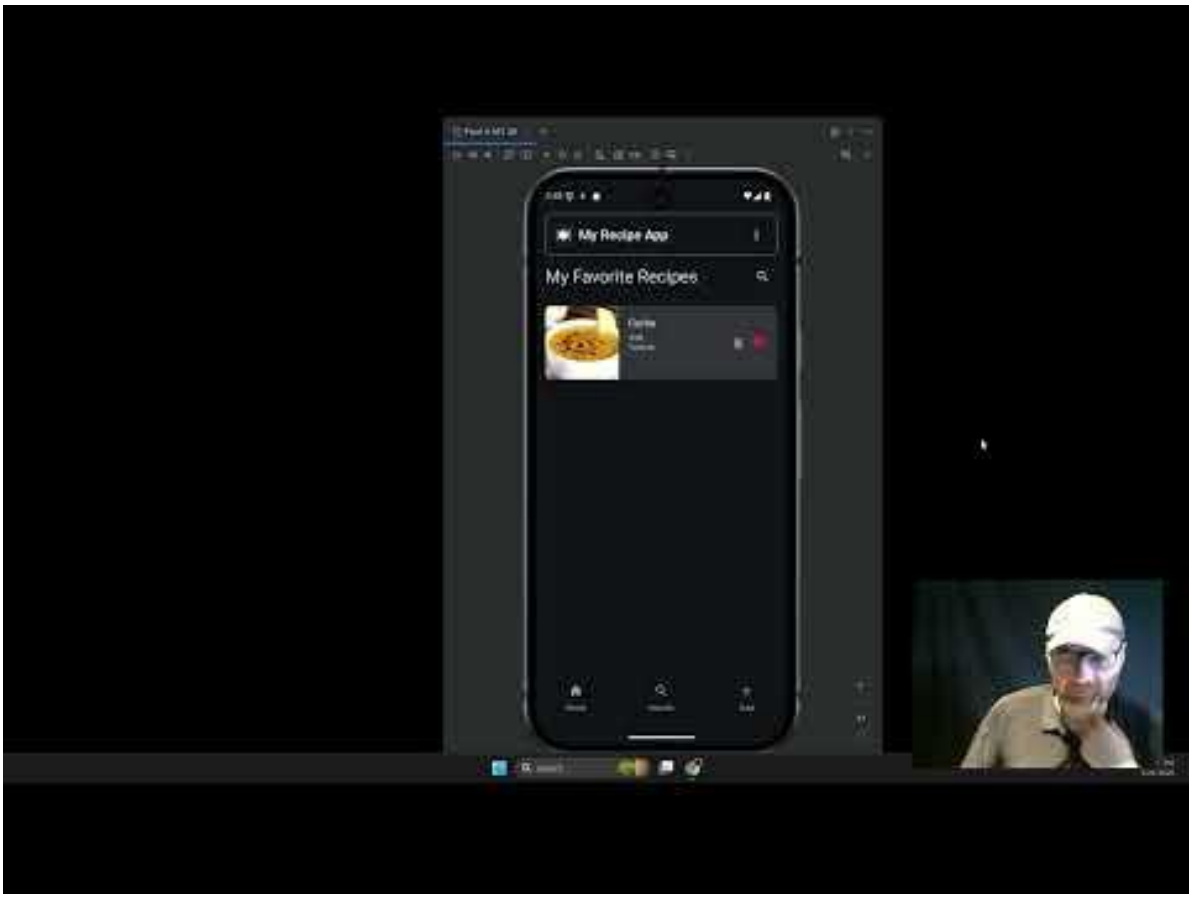
Screenshots

This section showcases the functionality and features of the app. The screenshots showcase the various UI screens of the app in light and dark modes (see next page); however, the

app is quite complex making it difficult to showcase the entire functionality of the app with just screenshots, so I created the following video showcasing the functionalities and features of the app.

Video 1

My Android Recipe App Version 1

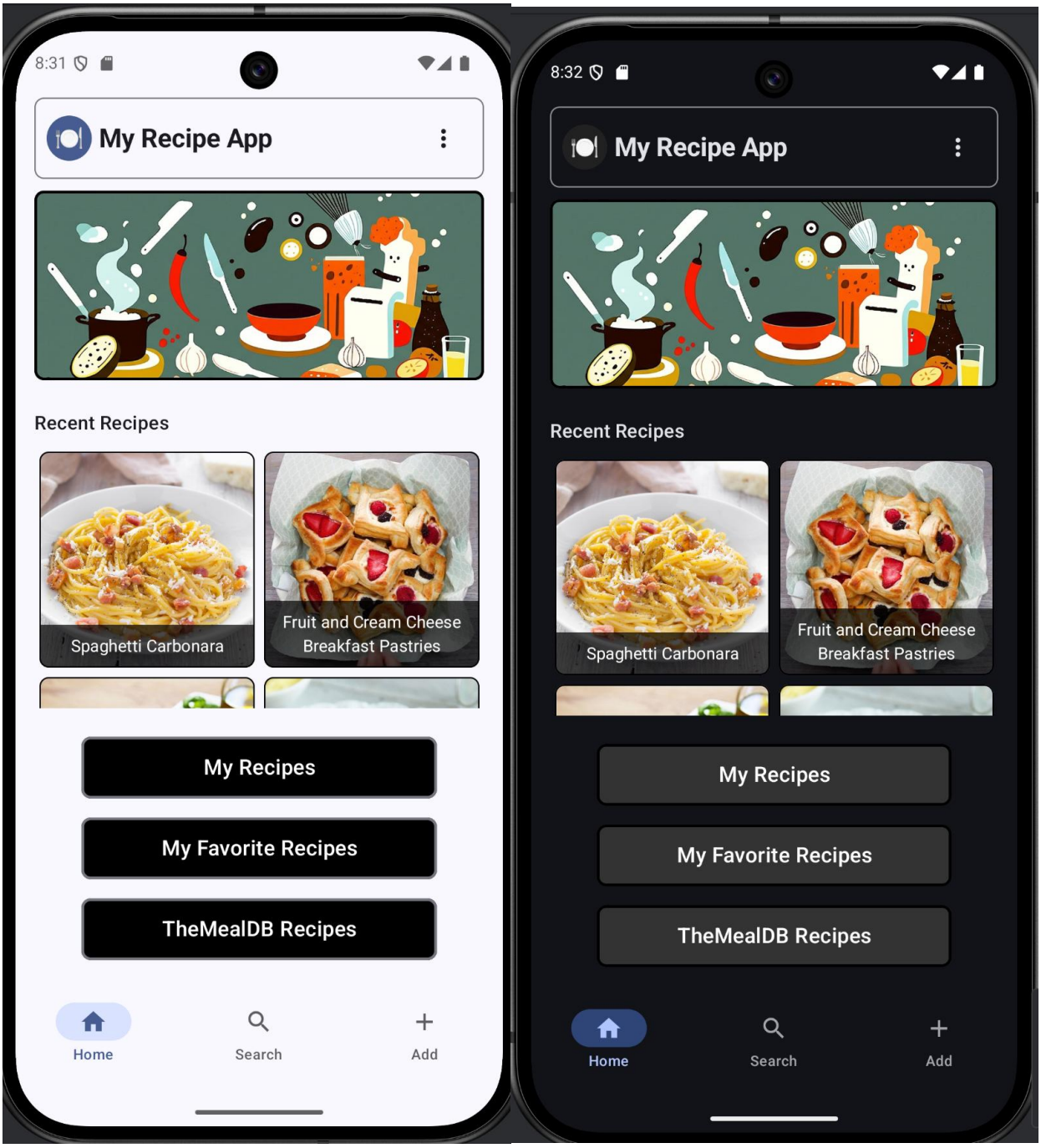


Note: The video showcases the app functionalities and features. From “My Android Recipe App Version 1” by Omegapy (2025).

Please see the next page

Figure 1

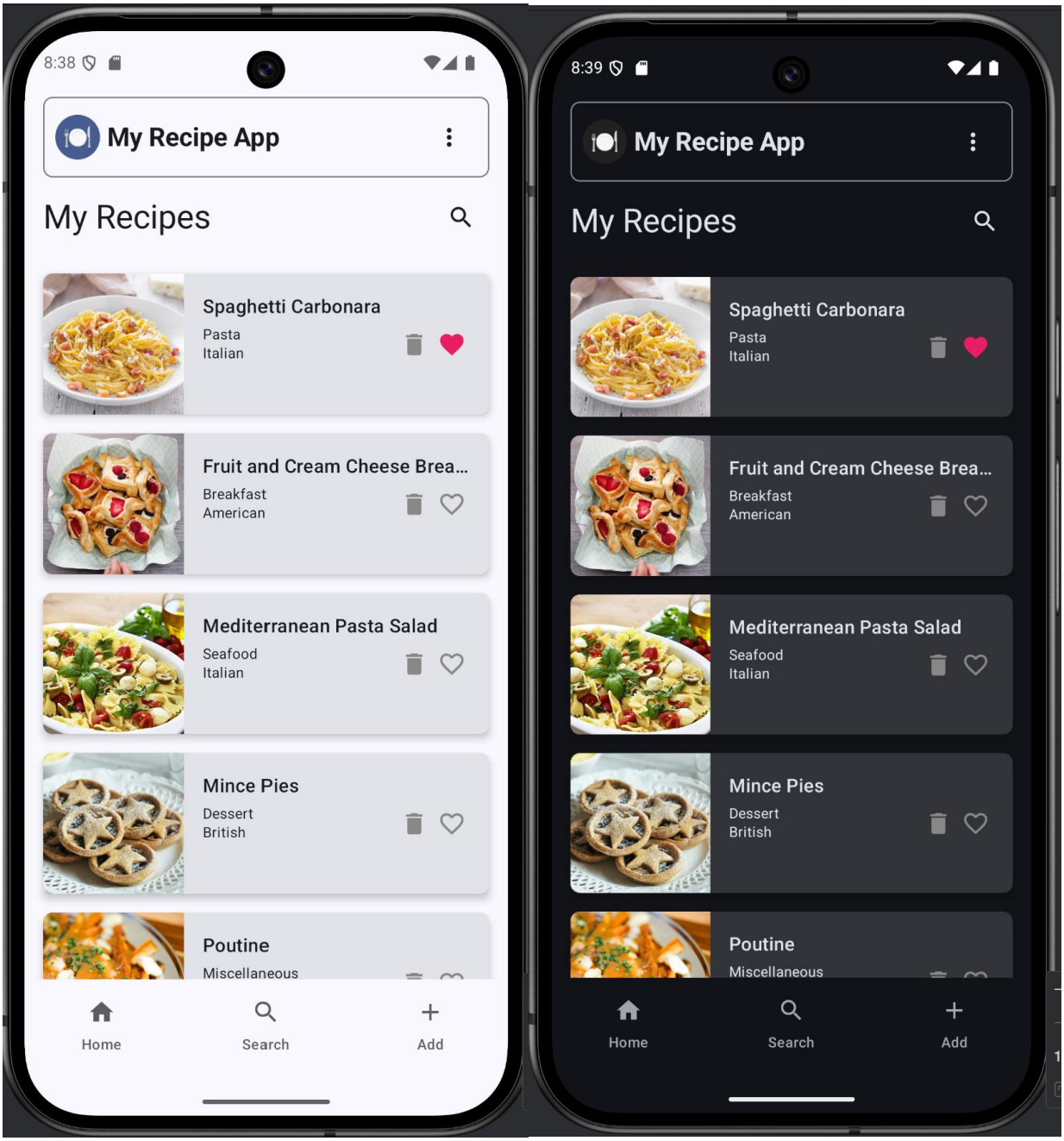
Home Screen



Note: Home screen Light and Dark theme.

Figure 2

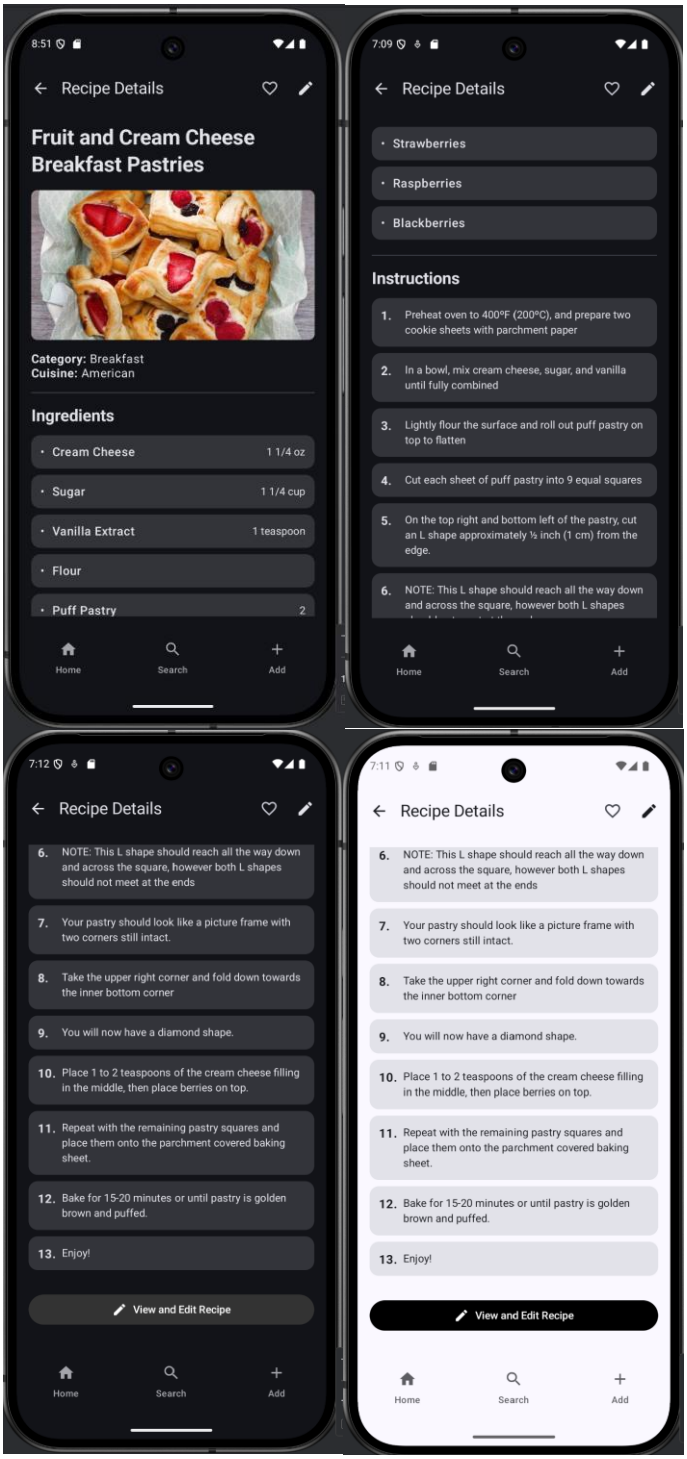
My Recipe Screen



Note: My Recipe Screen is in light and dark modes.

Figure 3

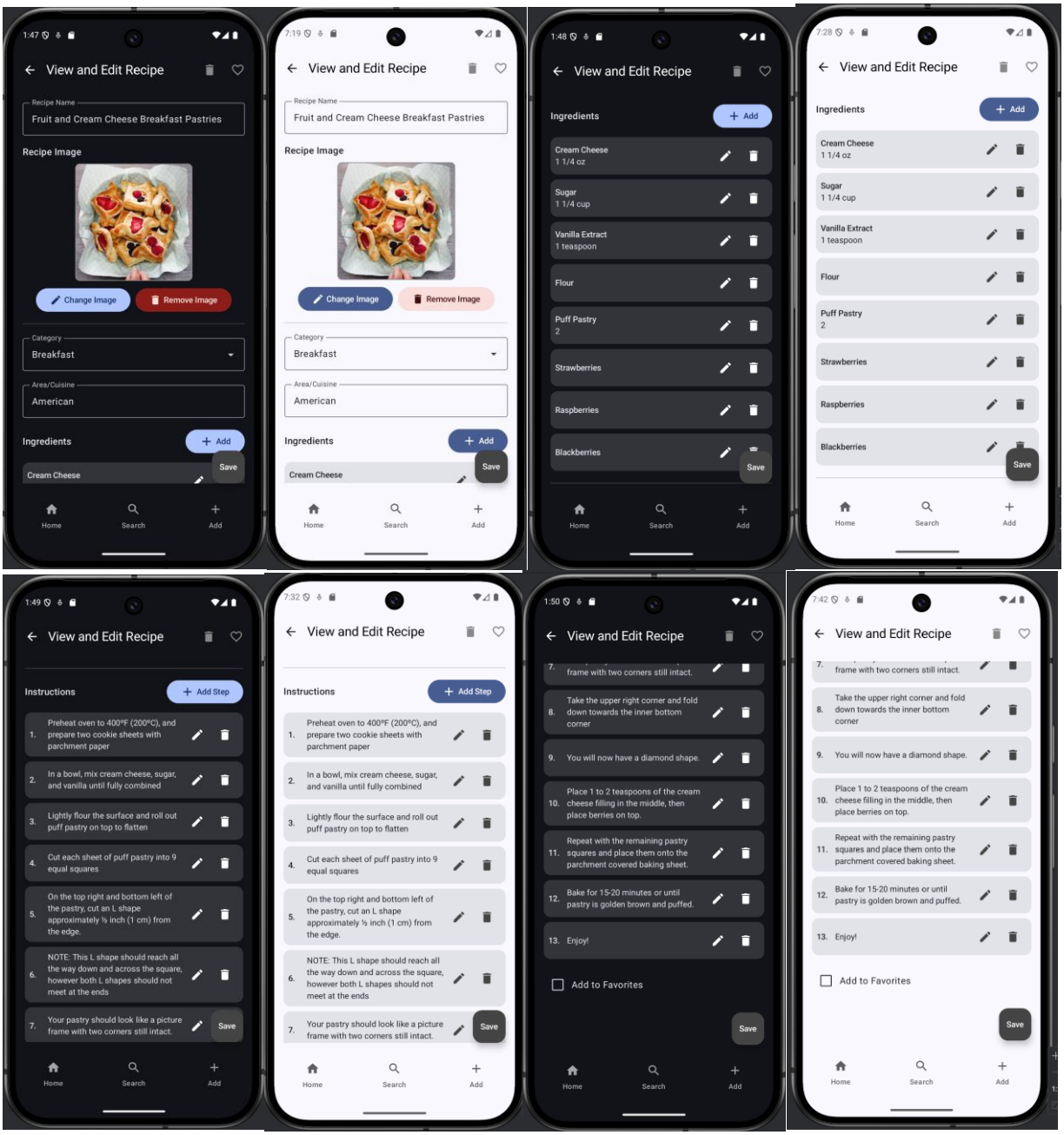
My Recipe Detail Screen



Note: My Recipe Details Screen light and dark modes.

Figure 4

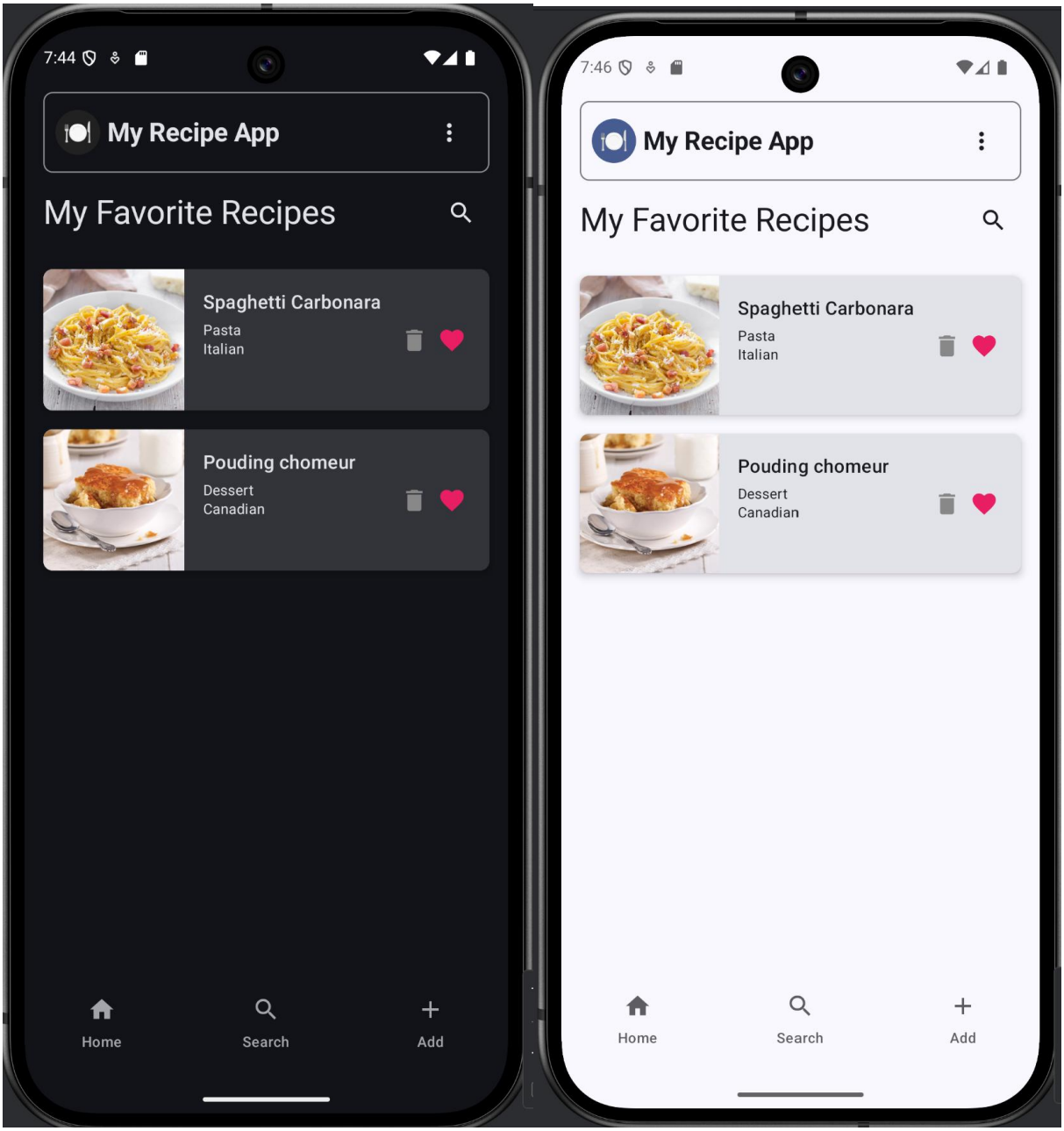
My Recipe View and Edit Recipe Screen



Note: My Recipe View and Edit Recipe Screen light and dark mode.

Figure 5

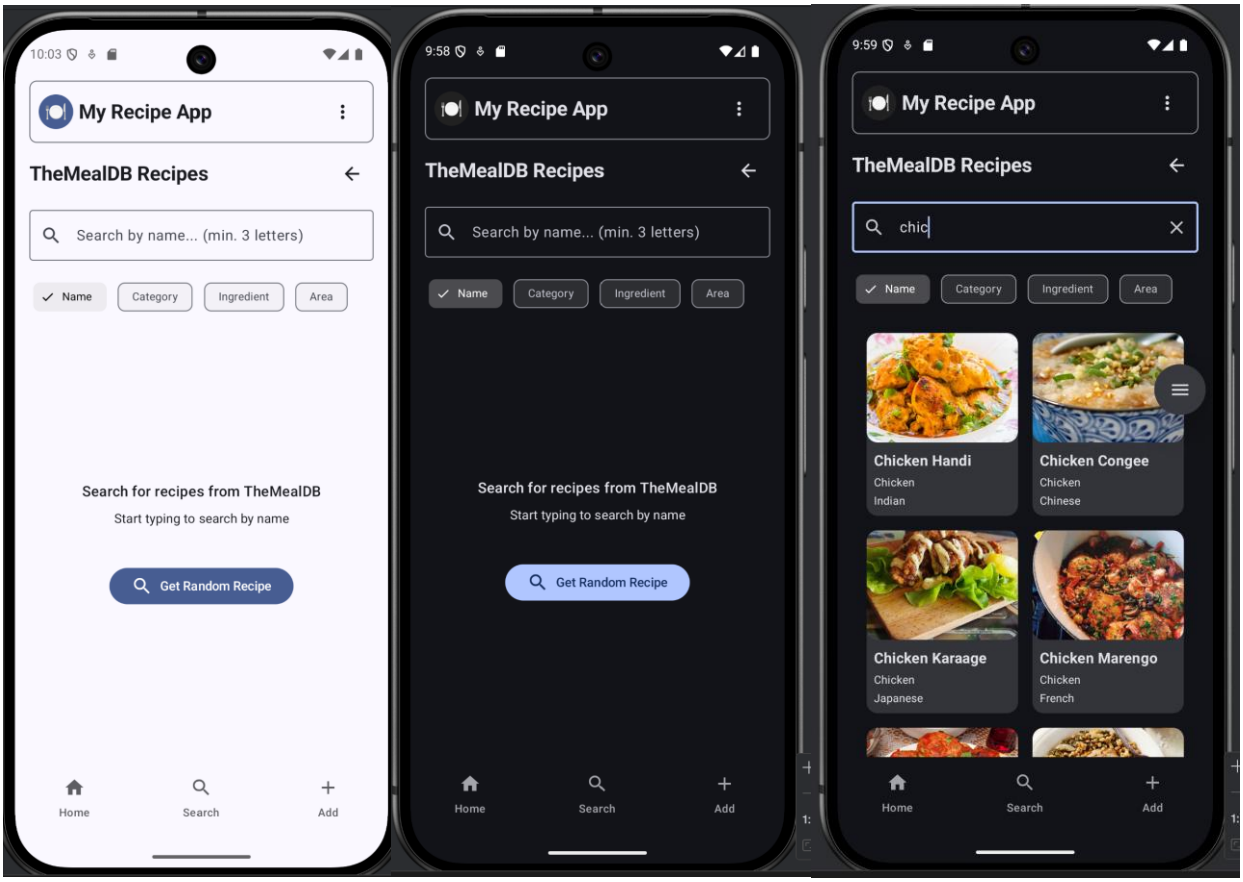
My Favorite Recipe Screen



Note: My Favorite Recipe View and Edit Recipe Screen light and dark mode. Not that the My Recipe Favorite Details and My Favorite Recipe View and Edit Recipe screens use the same UI features as the ones used by the My Recipe UI features.

Figure 6

TheMealDB Recipes Screen

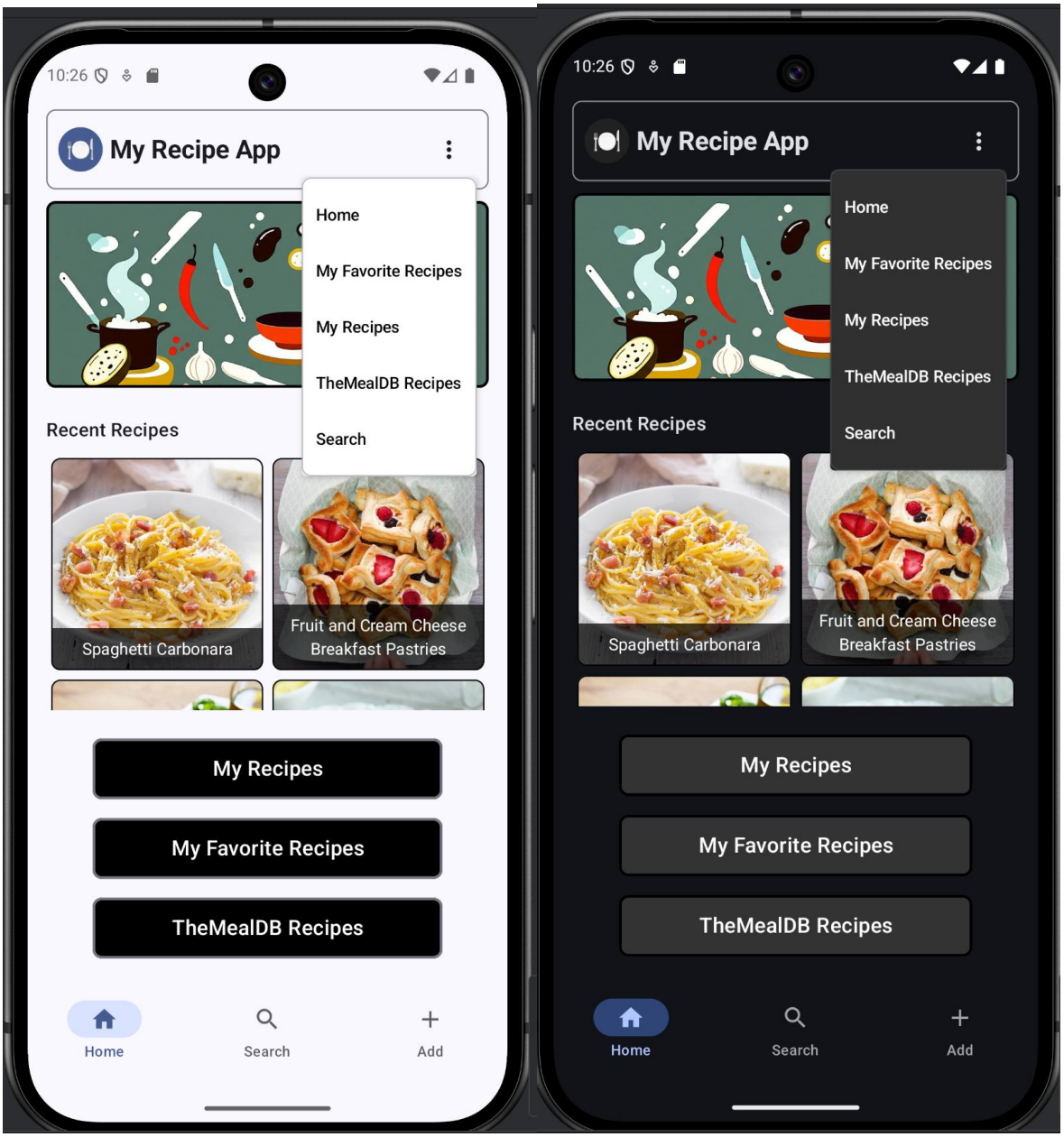


Note: My TheMealDB Recipes screen is also the TheMealDB Recipe Search screen, shown in light and dark mode.

Please, see the next page

Figure 7

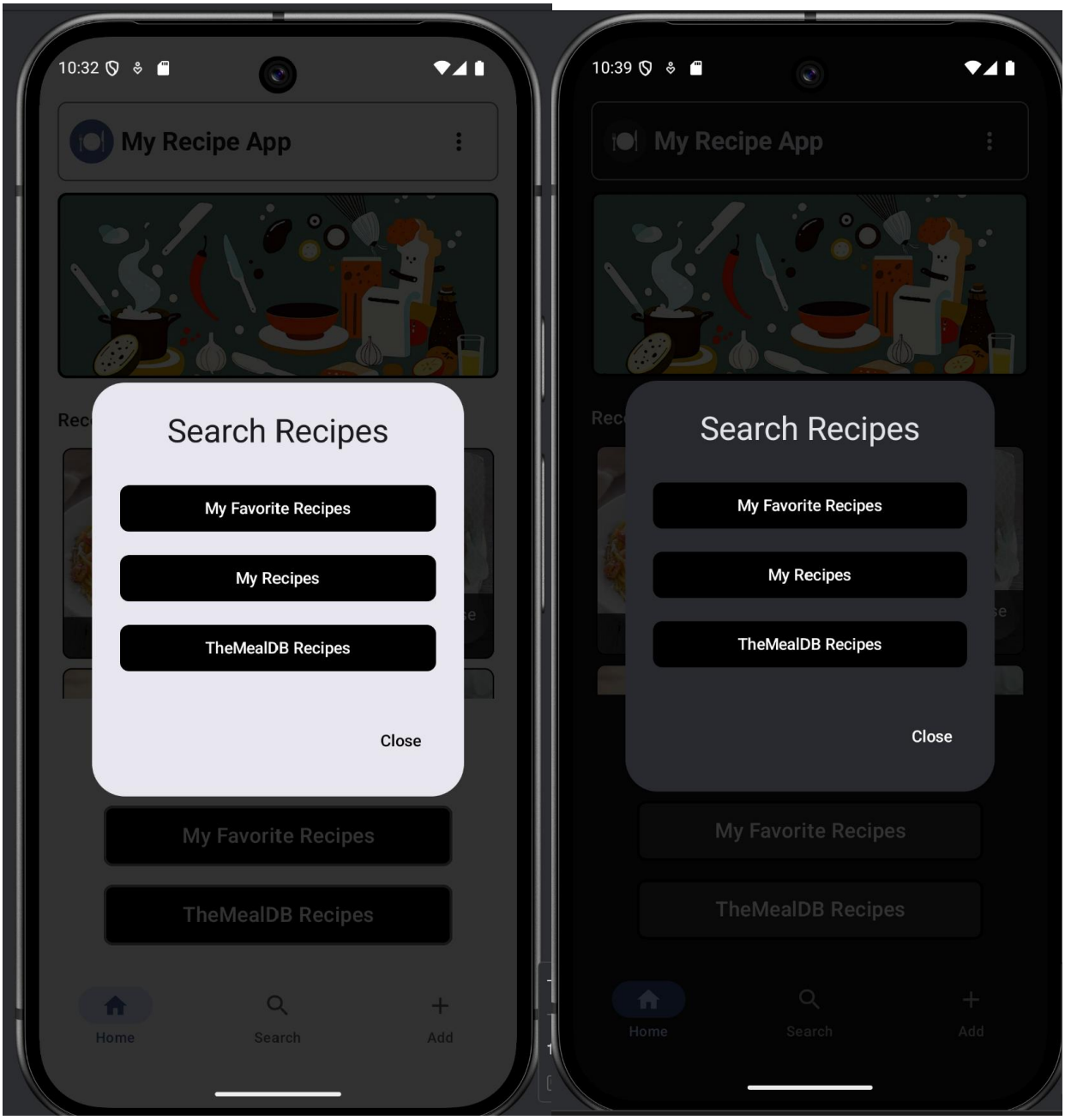
Header Dropdown menu (⋮)



Note: Header Dropdown menu (⋮) Link the Home, My Favorite Recipes, My Recipes, and Seach dialog screens, in light and dark modes.

Figure 8

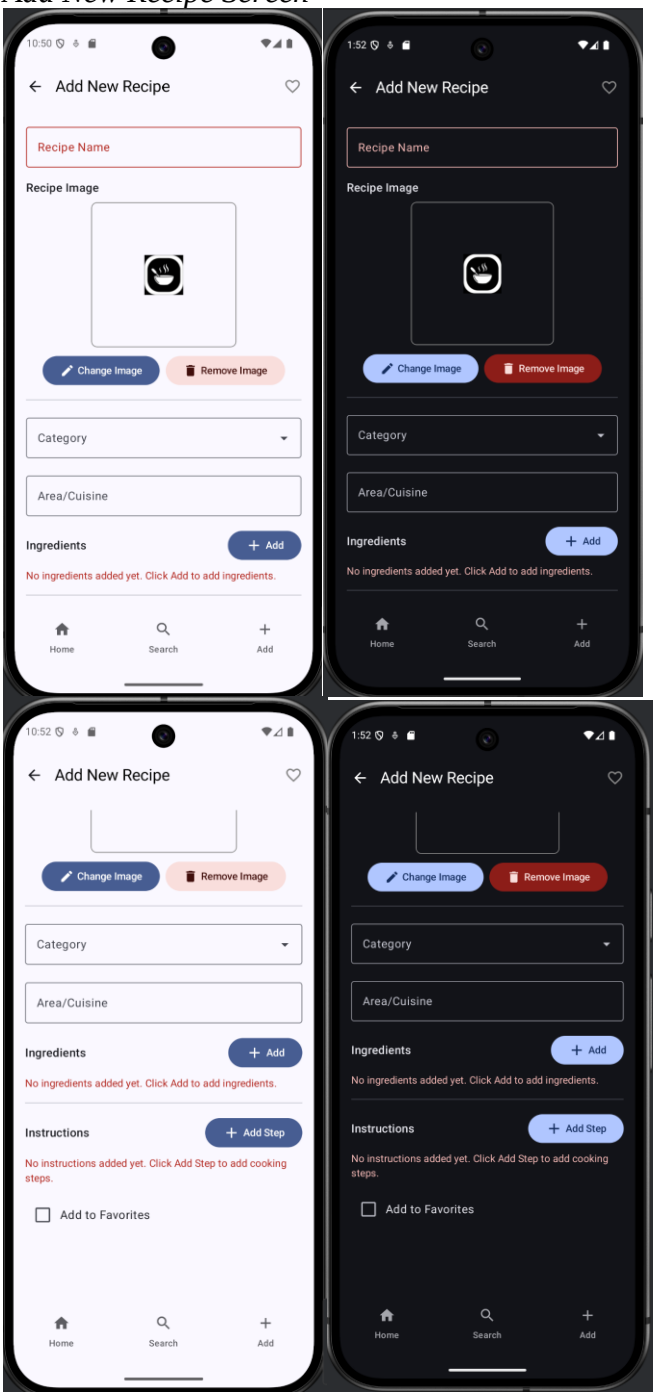
Search Dialog



Note: Search Dialog in light and dark modes, the dialog can be opened from the Header Dropdown menu (⋮) or by using the ‘ 🔍 ’ icon from the Footer or the top of the My Recipes and My Favorite screens.

Figure 9

Add New Recipe Screen



Note: Add New Recipe Screen in light and dark modes. The user can access the Add New Recipe screen by clicking on the (+) within the Footer. Note that a Save button will appear, when all the warnings, in red, are addressed by the user.

Figures 1 to 9 illustrate the various UI screens, and Video 1 illustrates the functionalities and features of the app. This demonstrates that the app works as intended.

Conclusion

This document showcases the architecture and the file structure of the Android "My Recipe App" app. It reflects the issues encountered during the development of the source code, as well as the solutions that were implemented to address those issues. It also provides a video showcasing the functionality and features of the app as well as screenshots illustrating the various UI screens in light and dark modes.

References

Omegapy(2025, March 21). My Android Recipe App Version 1 [Video]. YouTube. https://www.youtube.com/watch?v=Ad7-ovMeGUQ&t=2s

TheMealDB (n.d.a). *Welcome to TheMelDB*. https://www.themealdb.com/

TheMealDB (n.d.b). *Free recipe API*. https://www.themealdb.com/api.php

Ricciardi, A. (2025, March 9). Module 4 Portfolio Milestone: My Recipe App [Assignment]. CSU Global, CSC475 Platform-Based Development, Department of Computer Science.