Module 4 Portfolio Milestone: My Recipe App

Alexander Ricciardi

Colorado State University Global

CSC475: Platform-Based Development

Professor Herbert Pensado

March 9, 2025

Module 4 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's main goal is to provide a user with access to 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 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.

Architecture Overview

The MVVM approach separates the User Interface (UI) from the business or application logic, by keeping the UI layer from directly manipulating data. This is done by introducing an intermediary layer sitting between the UI layer and the data layer. These layers are defined by three components, which are the *View* (UI layer), the *ModelView* (intermediate layer), and the *Model* (the logic or data layer).

The *Model* component manages the meal recipes data locally by using the *Room Persistence Library*. The library provides an abstraction layer built on top of *SQLite*, converting relational data to object data that can be used by the application. The *Model* component also handles API calls to the TheMealDB's *NoSQL* database to retrieve recipes *JSON* documents using the *Retrofit* library. Then the *Moshi* library will be used to translate these *JSON* documents into *Kotlin* data objects, which in turn can be stored in the *Room* database.

The *ModelView* component acts as an intermediate layer between the *Model* and the *View*, it exposes data from the *Model* to the *View*, and transfers the data from the *View* (user data

inputs) to the *Model*. It also manages asynchronously these data operations using LiveData, to ensure that the UI stays responsive.

For the API call using *Retrofit* library, the following table lists API URL calls provided by TheMealDB. TheMealDB allows the use of its API for testing and educational purposes. This can be done by using API key "1" (TheMealDB, n.d.b). Note that I will end up signing up for TheMealDB Premium API Lifetime account as it only costs €10.00. This will allow me to add to the app the additional functionality and to share recipes on TheMealDB database. At this point of the development, the app does not support login as it is not needed. Additionally, a future integration of a Large Language Model (LLM) chatbot that can provide new recipes or recommend based on ingredients pr other criteria will significantly enhance the app. The table below provides a list of TheMealDB API URLs and their description.

Table 1

TheMealDB API Methods

Method	URL	Premium Only	Notes
Search meal by name	www.themealdb.com/api/json/v1/1/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/v1/1/search.php?f={letter}	No	Replace {letter} with the first letter of the meal.
Lookup meal details by ID	www.themealdb.com/api/json/v1/1/lookup.php?i={meal_id}	No	Replace {meal_id} with the meal's ID.
Single random meal	www.themealdb.com/api/json/v1/1/random.php	No	
10 random meals	www.themealdb.com/api/json/v1/1/randomselection.php	Yes	
List all meal categories	www.themealdb.com/api/json/v1/1/categories.php	No	
Latest Meals	www.themealdb.com/api/json/v1/1/latest.php	Yes	

List Categories/Area /Ingredients	www.themealdb.com/api/json/v1/1/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/v1/1/filter.php?i={ingredient}	No	Replace {ingredient} with the main ingredient (e.g., chicken_breast).
Filter by multi-ingredient	www.themealdb.com/api/json/v1/1/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/v1/1/filter.php?c={category}	No	Replace {category} with the category (e.g., Seafood).
Filter by area	www.themealdb.com/api/json/v1/1/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 and their description. Data from " Free Recipe API" by TheMealDB (n.d.b).

Data Layer (Model)

The following are examples of code representing classes used at the Data Layer level

Code Snippet 1

Data Layer Classes Room

```
//-------------- Room -----------------

// Using Room for local persistence
// wraps the JSON structure from TheMealDB
@Entity(tableName = "meals")
data class Meal(
// ...
}

//-------------------------------------------------------------------------
```

```
// Using Room: Define DAO methods for Meal operations
// Data Access Object (DAO)
// interface to interact with Meal records
@Dao
interface MealDao {
    @Query("SELECT * FROM meals")
    fun getAllMeals(): Flow<List<Meal>>

    @Insert(onConflict = OnConflictStrategy.REPLACE)
    suspend fun insertMeal(meal: Meal)

    @Update
    suspend fun updateMeal(meal: Meal)

    @Delete
    suspend fun deleteMeal(meal: Meal)
}

//-----------------------------------------------------------------------

// Using Room: Define the Room database
// Sets up the local database for Meal storage
@Database(entities = [Meal::class], version = 1)
abstract class MealDatabase : RoomDatabase() {
    abstract fun mealDao(): MealDao

    companion object {
        @Volatile private var instance: MealDatabase? = null

        fun getDatabase(context: Context): MealDatabase =
            instance ?: synchronized(this) {
                instance ?: Room.databaseBuilder(
                    context.applicationContext,
                    MealDatabase::class.java,
                    "meal_database"
                ).build().also { instance = it }
            }
    }
}
```

Note: The code snippet gives examples of classes using the Room library at the Data Layer level.

Code Snippet 2

Data Layer Code Retrofit

```
//-------------- Moshi ------------------

// Using Retrofit for API calls and Moshi for JSON
// Interface for endpoints of the Free Recipe API.
// The base URL includes the test key 1
// (e.g., https://www.themealdb.com/api/json/v1/1/).
interface TheMealDBApiService {
    // Search meal by name: /search.php?s=Arrabiata
    @GET("search.php")
    suspend fun searchMeals(@Query("s") query: String): Response<MealResponse>

    // List meals by first letter: /search.php?f=a
    @GET("search.php")
    suspend fun listMealsByFirstLetter(@Query("f") letter: String):
                                                        Response<MealResponse>

    // Lookup meal details by id: /lookup.php?i=52772
    @GET("lookup.php")
    suspend fun getMealDetails(@Query("i") id: String): Response<MealResponse>

    // Lookup a single random meal: /random.php
    @GET("random.php")
    suspend fun getRandomMeal(): Response<MealResponse>

    // List meal categories: /categories.php
    @GET("categories.php")
    suspend fun getMealCategories(): Response<CategoriesResponse>
}

//-----------------------------------------------------------------------

// Using Retrofit to build a Retrofit instance
// This helper function creates a Retrofit instance using the base URL (with test
key “1”).
object RetrofitInstance {
    private const val BASE_URL = "https://www.themealdb.com/api/json/v1/1/"

    private val moshi = Moshi.Builder().build()  // Moshi for JSON conversion

    val api: TheMealDBApiService by lazy {
```

```
        Retrofit.Builder()
            .baseUrl(BASE_URL)
            // Using Moshi converter
            .addConverterFactory(MoshiConverterFactory.create(moshi))
            .build()
            .create(TheMealDBApiService::class.java)
    }
}
```

Note: The code snippet gives examples of code using the Retrofit library at the Data Layer level.

Repository Layer

Repository Layer follows the Single Responsibility Principle, ensuring that the *ViewModel* does not directly fetch data, it acts like a bridge between the *ViewModel* and the Data Layer (*Model*). The following are examples of code representing a class that can be used at the Repository Layer level. It integrates both the local data source (Room) and the remote API (Retrofit/Moshi).

Code Snippet 3

Repository Layer Class

```
// Uses Room-Retrofit/Moshi
// fetches data from the API and updates the local database
class MealRepository(
    // Local data source (Room)
    private val mealDao: MealDao,
    // Remote data source (Retrofit/Moshi)
    private val apiService: TheMealDBApiService
) {
    val allMeals: Flow<List<Meal>> = mealDao.getAllMeals()

    // Fetch meals by name from the remote API and store them on divise
    suspend fun refreshMeals(query: String) {
        val response = apiService.searchMeals(query)
        if (response.isSuccessful) {
            response.body()?.meals?.forEach { meal ->
                mealDao.insertMeal(meal)
            }
```

```
        }
    }

    // repository methods can be added
}
```

Note: The code snippet gives examples of code using a class at the Data Layer level.

ViewModel Layer

The *ViewModel* exposes meal data to the UI and handles asynchronous API calls. The following is an example of code representing a class used at the ViewModel Layer level.

Code Snippet 4

ViewModel Layer Class

```
// Using Android ViewModel & LiveData.
class MealsViewModel(private val repository: MealRepository) : ViewModel() {
    // Expose meals as being LiveData for the UI (observed via Jetpack Compose)
    val meals: LiveData<List<Meal>> = repository.allMeals.asLiveData()

    // Called when the user starts a search.
    fun searchMeals(query: String) {
        viewModelScope.launch {
            repository.refreshMeals(query)
        }
    }
}
```

Note: The code snippet gives examples of code using a class at the ViewModel Layer level.

User Interface Layer (UI) (View)

The following are examples of code representing Composable functions used at the UI Layer level.

Code Snippet 5

UI Layer Composable Functions Examples

```
// Meal List Screen - Using Jetpack Compose
// UI for displaying a list of meals
@Composable
fun MealListScreen(viewModel: MealsViewModel) {
    // Observe meals data from the ViewModel.
    val meals by viewModel.meals.observeAsState(initial = emptyList())

    LazyColumn {
        items(meals) { meal ->
            MealCard(meal = meal)
        }
    }
}

//-------------------------------------------------------------------------

// Meal Card - Using Jetpack Compose
// UI for an individual meal
@Composable
fun MealCard(meal: Meal) {
    Card(modifier = Modifier.padding(8.dp)) {
        Column(modifier = Modifier.padding(16.dp)) {
            Text(text = meal.strMeal, style = MaterialTheme.typography.h6)
            Text(text = meal.strCategory, style = MaterialTheme.typography.body2)
        }
    }
}
```

Note: The code snippet gives examples of code using Composable functions at the UI Layer (*View*) level.

The following table provides a summary of the above code snippets and illustrates how they fit within the MVVM architecture.

Table 2

MVVM Code Examples Description

Layer	Component	Description	Technology Used
Model	*@Entity(tableName = "meals")*	Structure of a meal recipe.	Room
	MealDao	Data Access Object. Defines methods for interacting with the Meal data in the database.	Room
	MealDatabase	The Room database that stores recipes	Room
	MealResponse	JSON structure of a single meal response from the API.	Moshi
	CategoriesResponse	JSON structure of a category list response from the API.	Moshi
	TheMealDBApiService	API call using Retrofit annotations.	Retrofit
	RetrofitInstance	Retrofit client, with a Moshi converter for JSON parsing.	Retrofit, Moshi
Repository	*MealRepository*	Provides data to the *ViewModel.*	Room, Retrofit, Moshi
ViewModel	*MealsViewModel*	Stores and manages UI data. Exposes data to the UI using LiveData.	Android ViewModel, LiveData
UI (View)	*MealListScreen*	Jetpack Compose function that displays a list of meal recipes.	Jetpack Compose
	MealCard	Jetpack Compose function that represents a single recipe.	Jetpack Compose

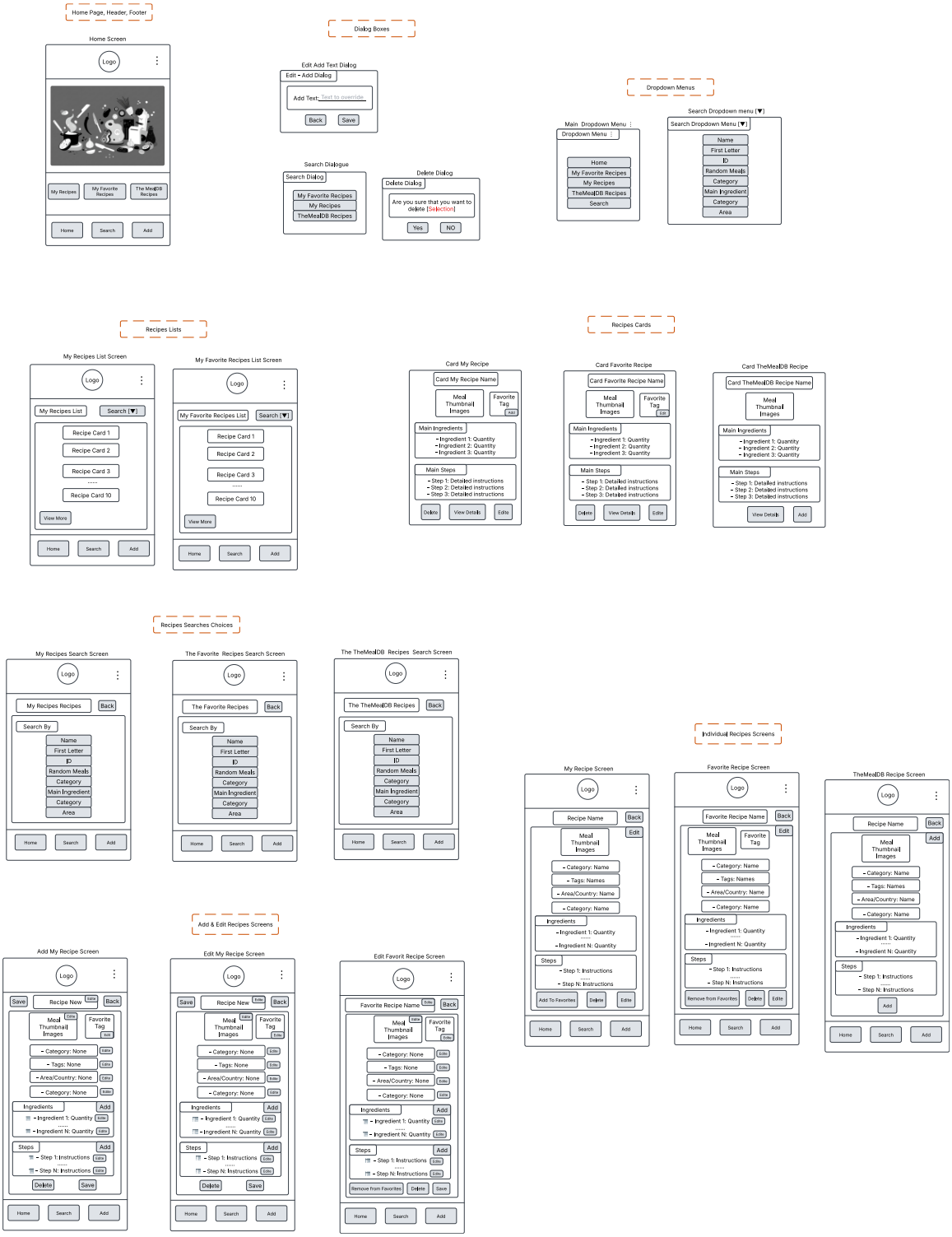
Note: The table provides a summary of the code snippets and illustrates how they fit within the MVVM architecture.

User Interface Layout

This section showcases a series of wireframes illustrating the UI layout of the app. The wireframe also illustrates how the different UI components relate to each other.

Figure 1

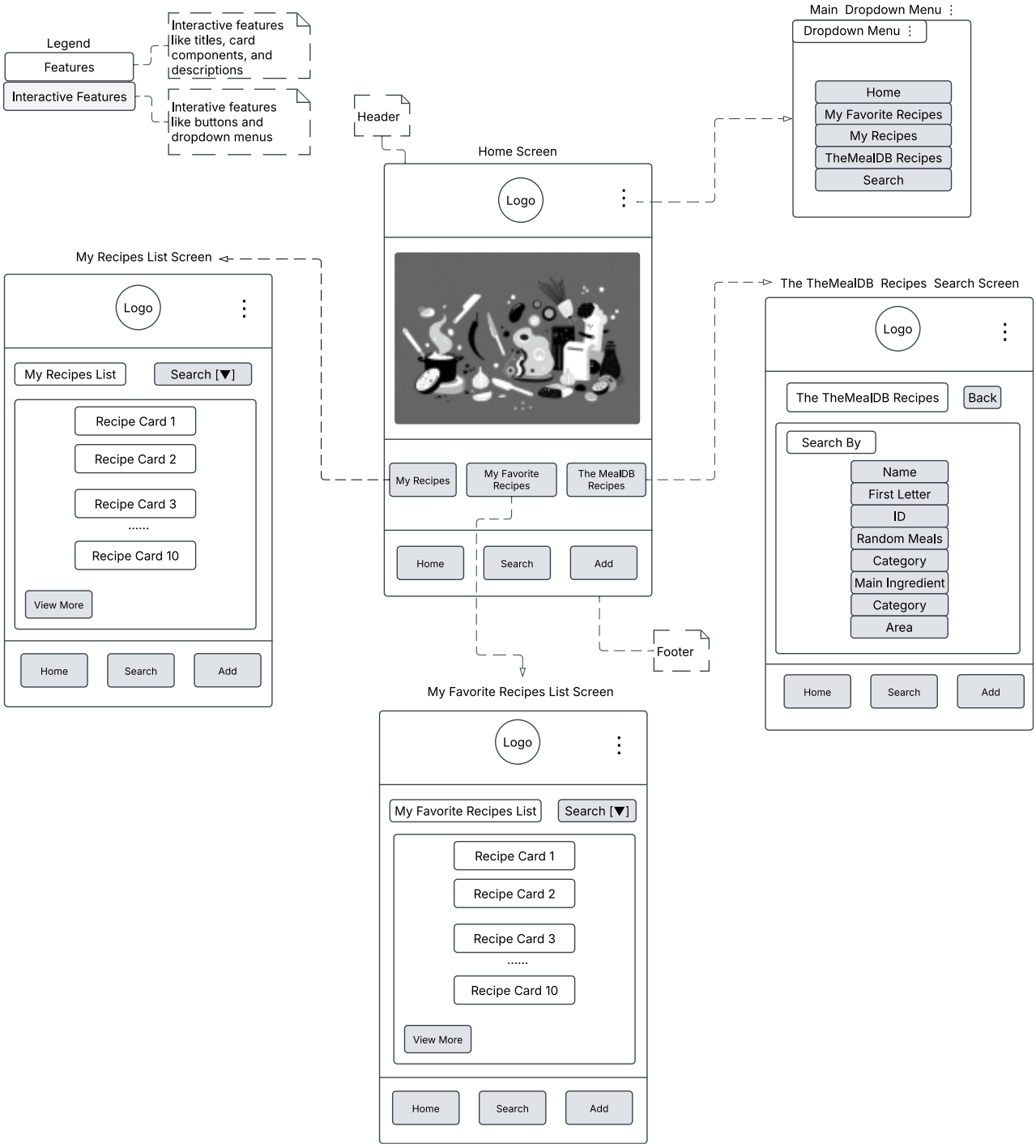
UI Components



Note: This figure illustrates the different UI components of the app.

Figure 2

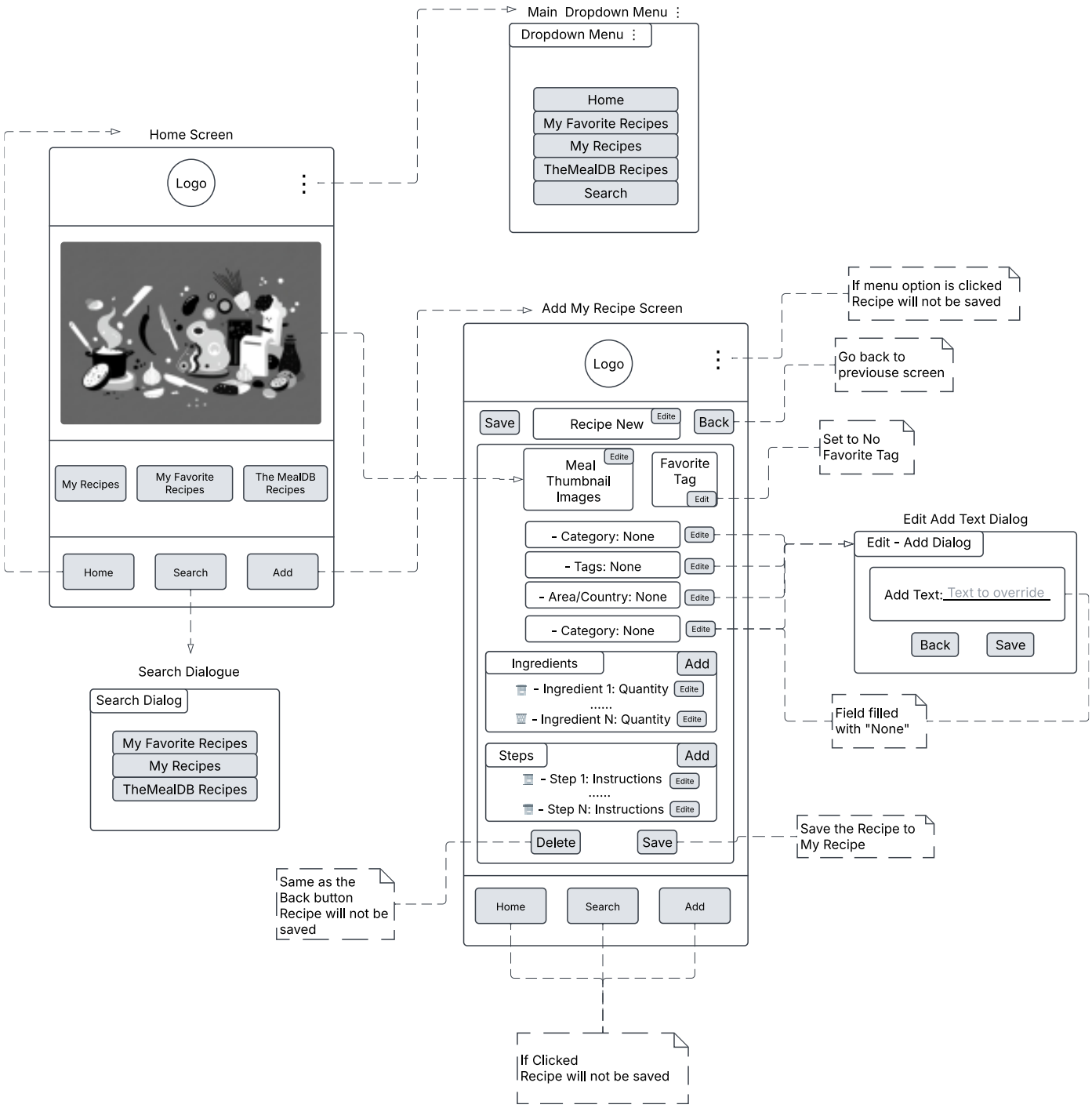
Home Screen Main Features



Note: This figure illustrates the Home Screen and how it relates to the main Dropdown Menu (⋮), My Recipes List Screen, and My Favorite Recipes List Screen.

Figure 3

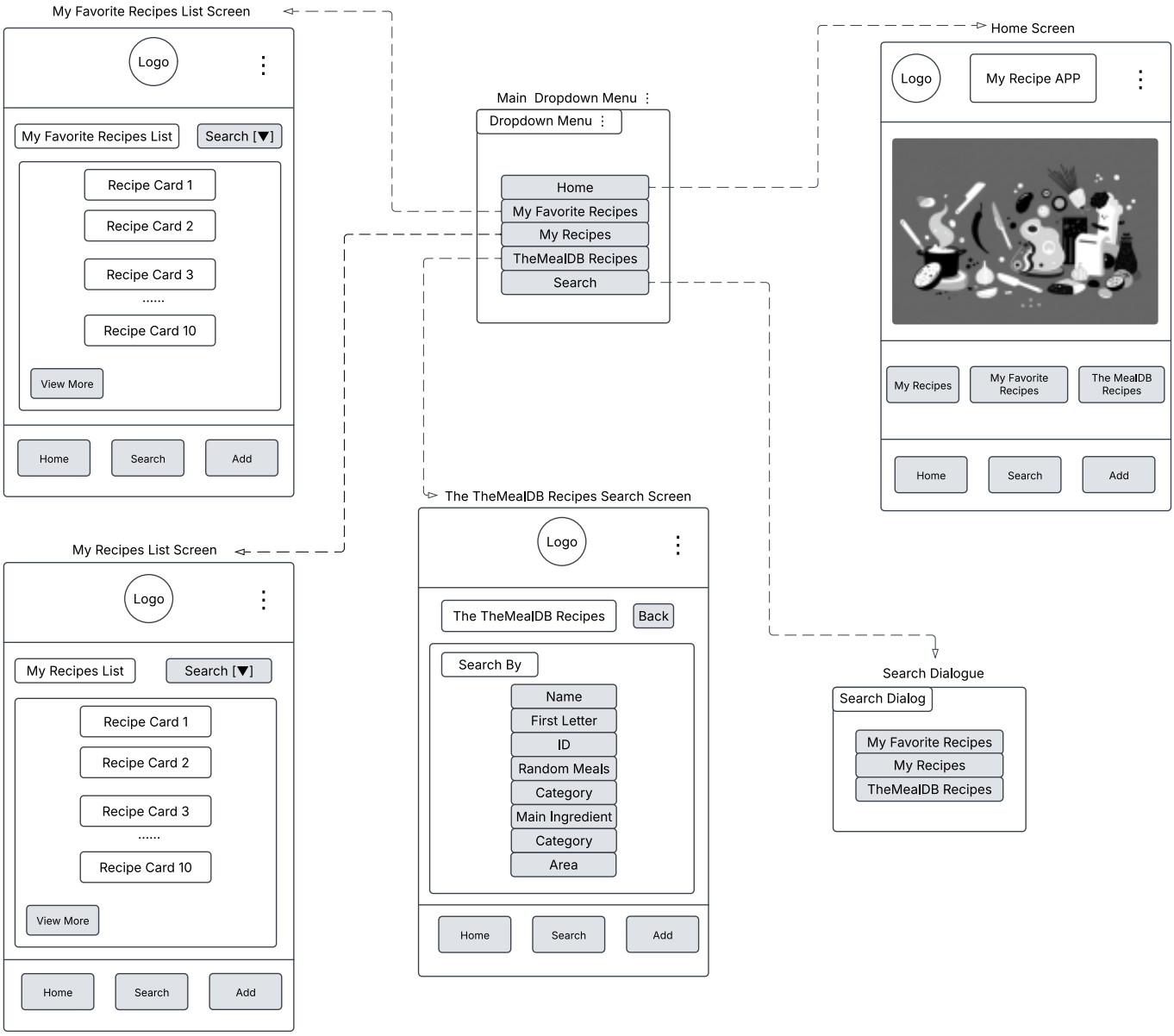
Home Header and Footer Features



Note: This figure illustrates the Home Header and Footer, and how they relate to the Search Dialogue, Main Dropdown Menu (⋮), Add My Recipe Screen, and Edit Add Text Dialog.

Figure 4

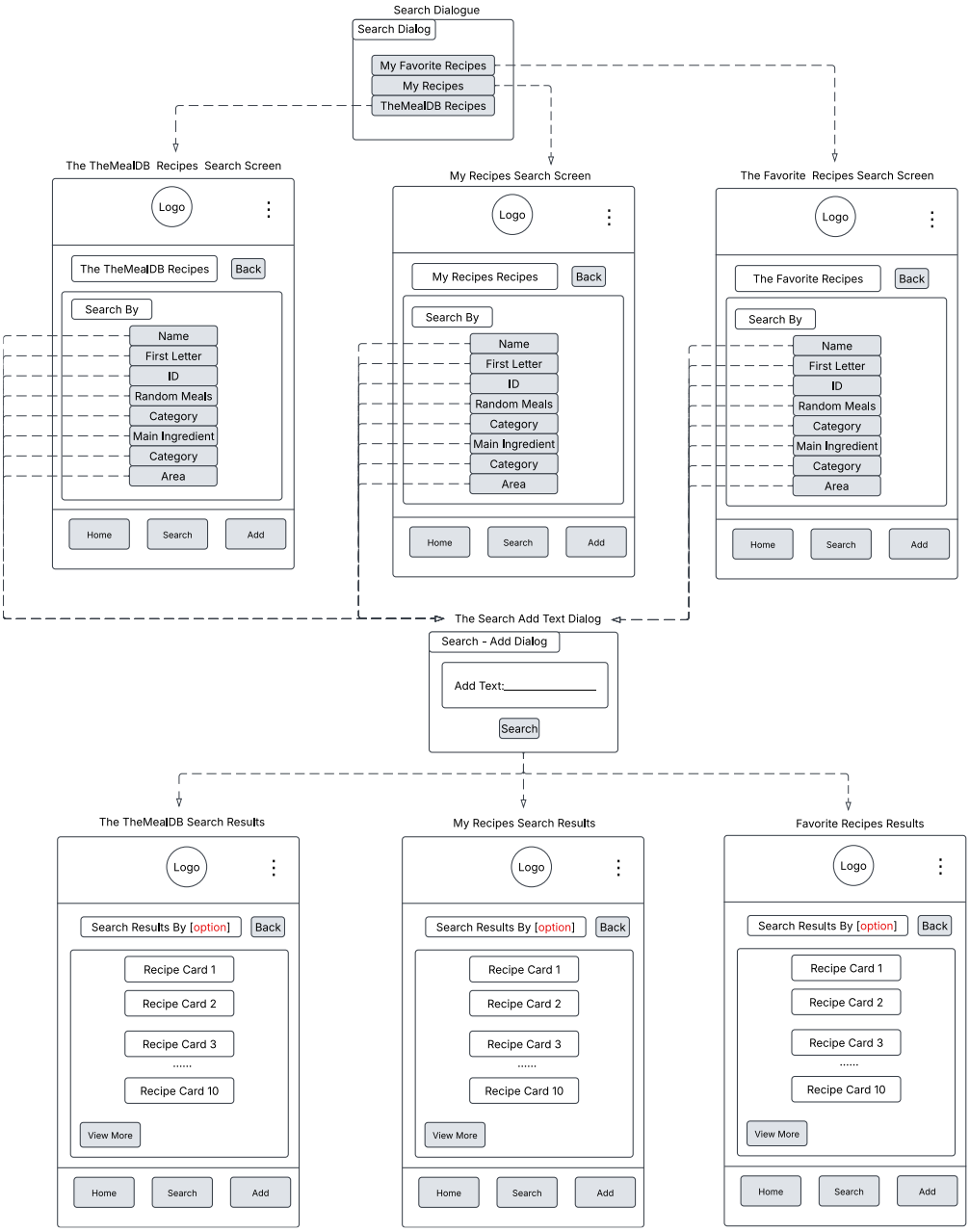
Main Dropdown Menu (:) Features



Note: This figure illustrates the Main Dropdown Menu (⋮), and how it relates to the Search Dialogue, Home Screen, TheMealDB Recipes Search Screen, My Favorite Recipes List Screen, and My Recipes List Screen.

Figure 5

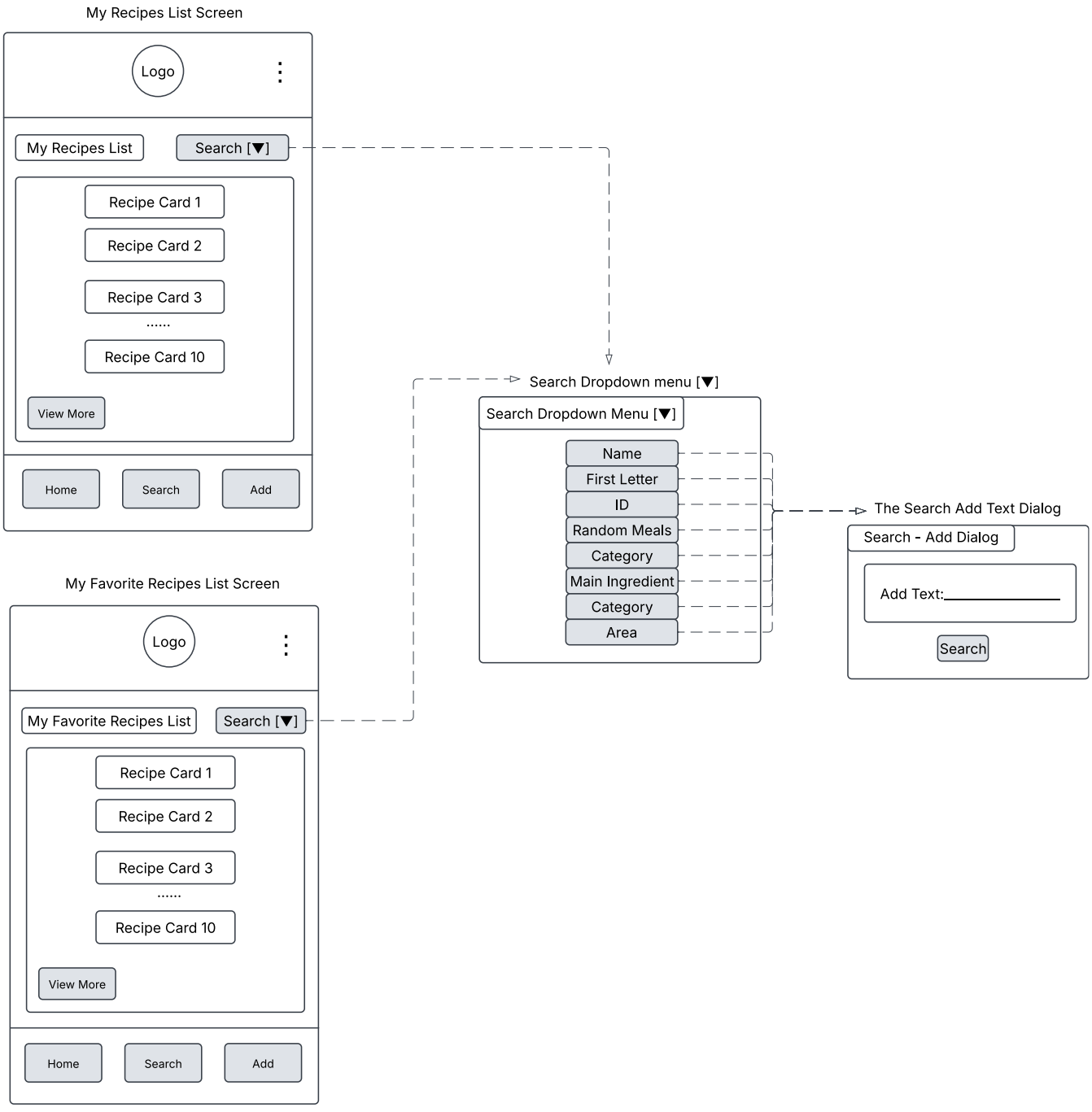
Search Dialogue Features



Note: This figure illustrates the Search Dialogue, and how it relates to the TheMealDB Recipes Search Screen, My Recipes Search Screen, Favorite Recipes Search Screen, The Search Add Text Dialog, TheMealDB Search Results, My Recipes Search Results, and Favorite Recipes Results.

Figure 6

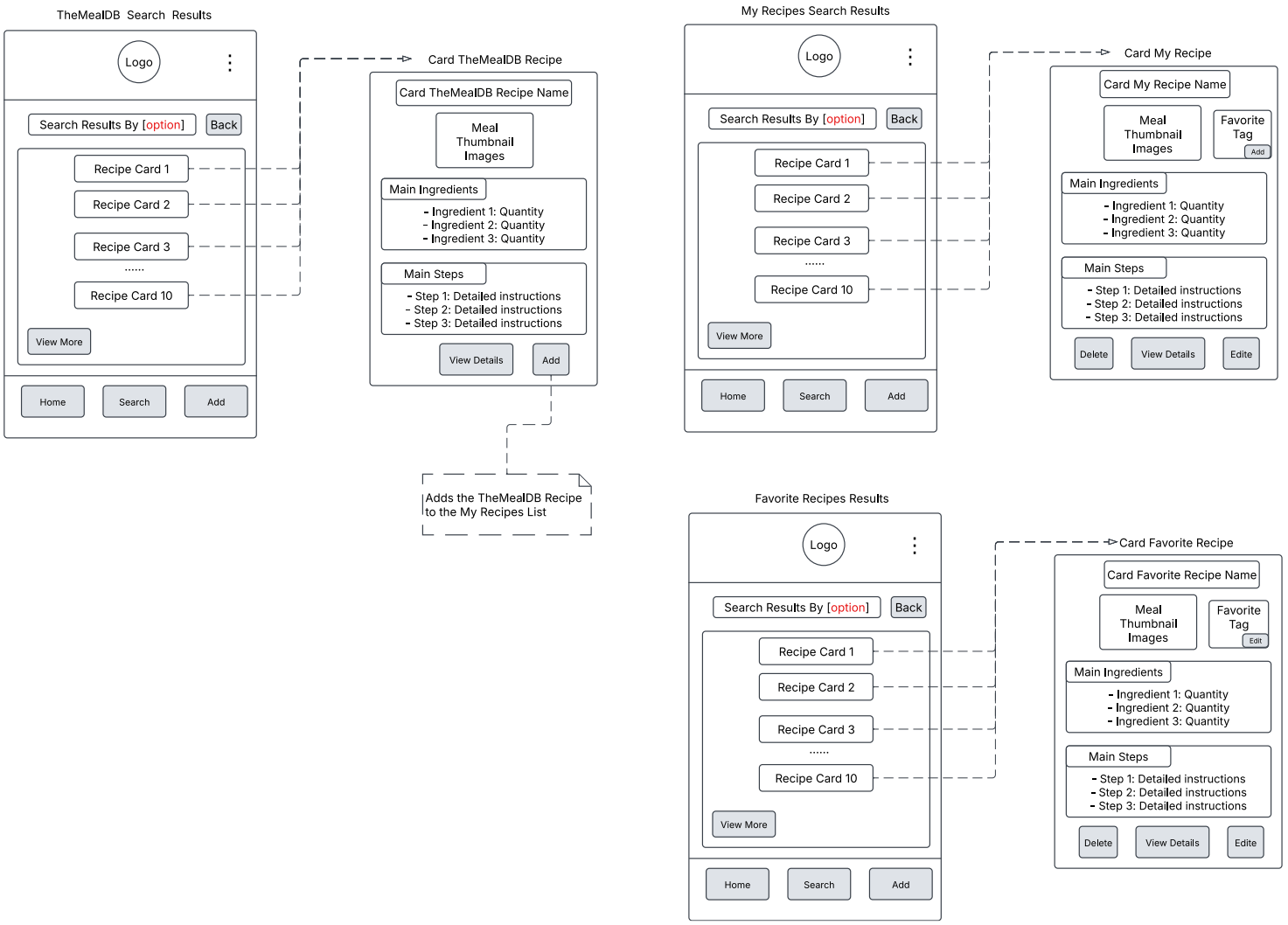
Search Dropdown menu [▼] Features



Note: This figure illustrates Search Dropdown menu [▼], and how it relates to the My Favorite Recipes List Screen, My Recipes List Screen, and The Search Add Text Dialog.

Figure 7

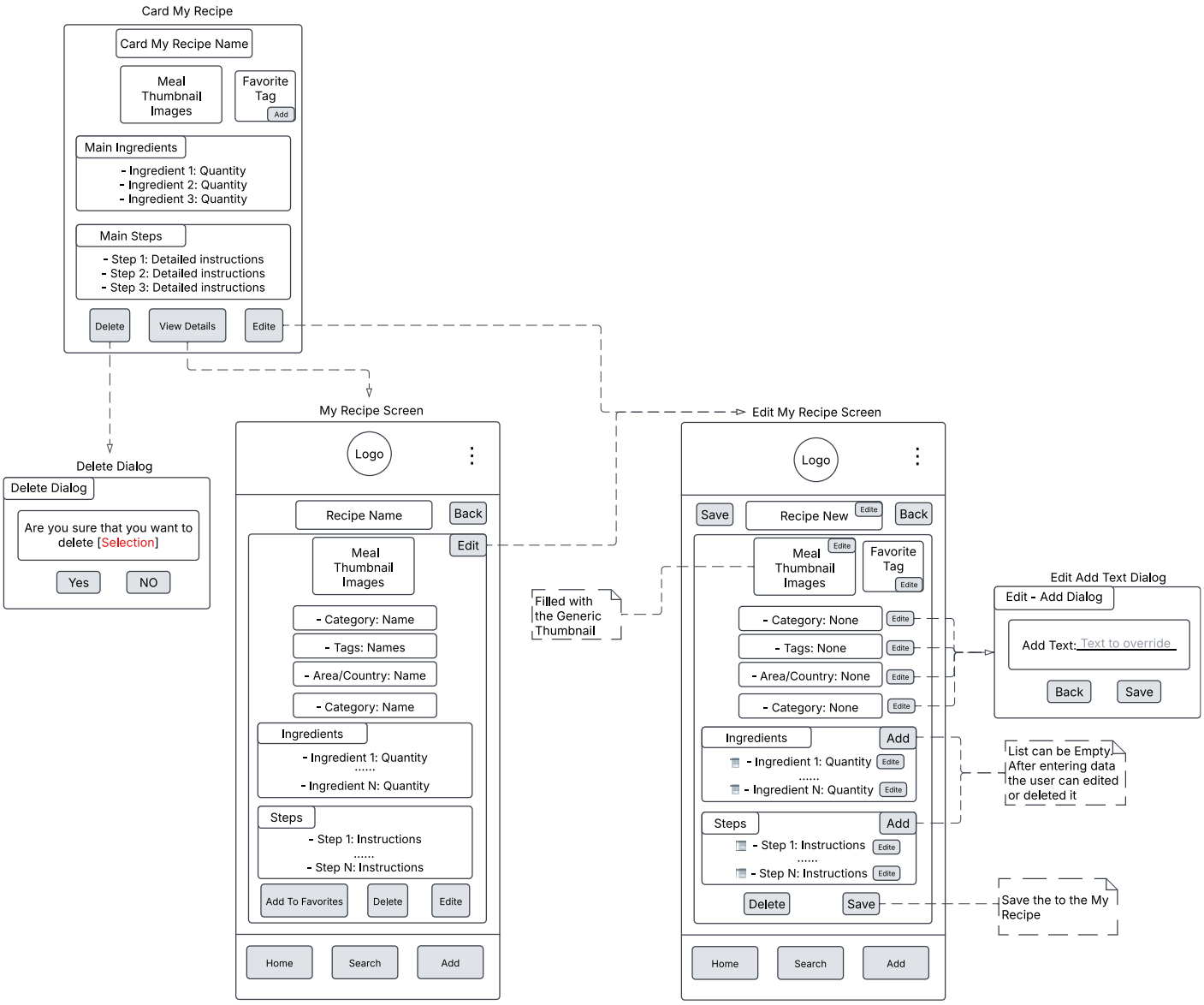
Search Results Features



Note: This figure illustrates the Search Results Features, and how each relates to their corresponding card features.

Figure 8

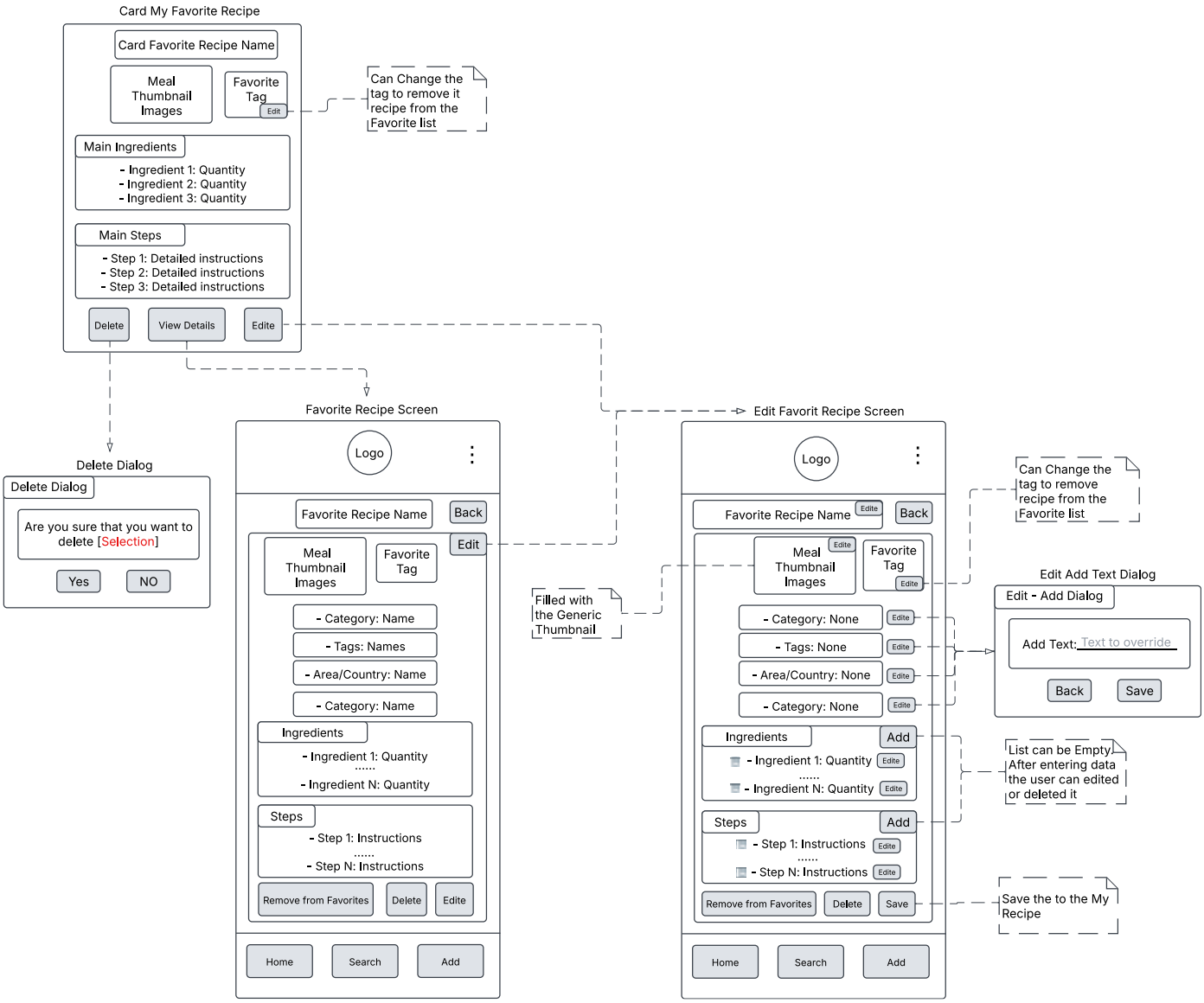
My Recipe Card, View Screen, and Edit Screen Features



Note: This figure illustrates My Recipe Card, View Screen, and Edit Screen Features, and how they relate to each other and the Delete Dialog and Edit Add Text Dialog.

Figure 9

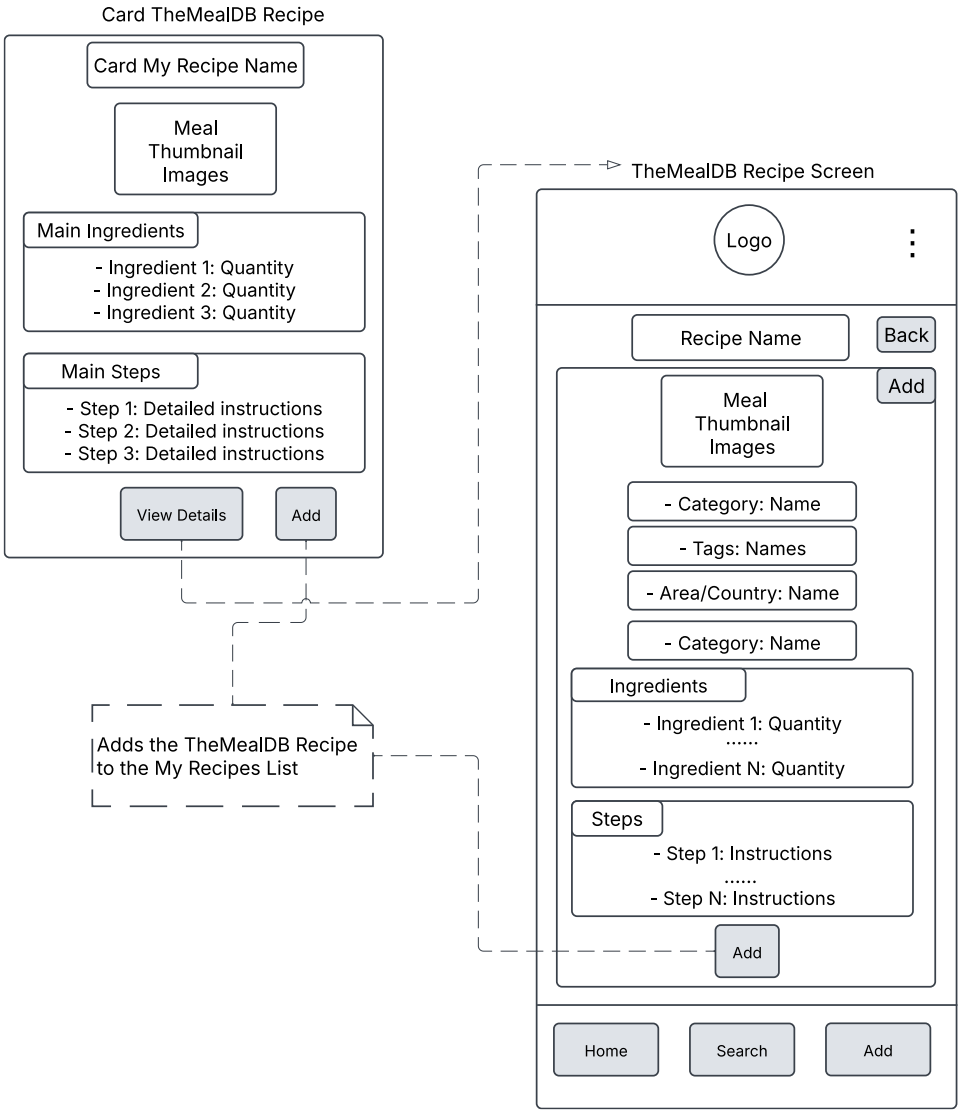
Favorite Recipe Card, View Screen, and Edit Screen Features



Note: This figure illustrates Favorite Recipe Card, View Screen, and Edit Screen Features, and how they relate to each other and the Delete Dialog and Edit Add Text Dialog.

Figure 10

TheMealDB Recipe Card and TheMealDB Recipe Screen Features



Note: This figure illustrates the TheMealDB Recipe Card and TheMealDB Recipe Screen Features, and how they relate to each other.

Figures 1 to 10 illustrate the entire user journey through the app UI, from the various screens to interacting with individual recipe details. It covers user input (search, add, edit), navigation (menus, lists), and data display (recipe cards, detailed views). It also provides insight into the app structure and design.

Conclusion

This document showcases the architectural, data flow, component descriptions, and UI design for my "My Recipe App" Android application. The MVVM architecture, combined with the use of Kotlin, Jetpack Compose, Room, Retrofit, and Moshi, is used to design a robust and stable system that ultimately will result in a maintainable, testable, and scalable app. Time permitting, I will implement every feature listed in this document in my app, which probably goes beyond the course portfolio assignment requirements

References

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

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