Module 2 Portfolio Milestone: Recipe App

Alexander Ricciardi

Colorado State University Global

CSC475: Platform-Based Development

Professor Herbert Pensado

February 23, 2025

Module 2 Portfolio Milestone: Recipe App

This document is my portfolio milestone module from the Platform-Based Development CSC475 course at Colorado State University Global. It describes the requirements, UI/UX analysis, audience, and scope of an Android recipe app. The app's scope is to provide access to meal recipes to a user through a UI system that includes view, search, add, modify, and favorite recipe functionalities. The app will be developed using Kotlin and Jetpack Compose, with plans for future integration of a Large Language Model (LLM) chatbot and the TheMealDB API (n.d.) to access and share recipe data.

Purpose and Scope

The app's primary purpose is to allow users to view, store on the device, search, add, and modify meal recipes, as well as create a list of favorite recipes. The app will come with store-on-device meal recipe data from TheMealDB (n.d.), which is an open, crowd-sourced database of recipes from around the world. This means that users with a Patreon account can contribute recipes to build the TheMealDB database. They also offer a free API for anyone wanting to use it and a Patreon subscription for extra features. Although the app will only initially accommodate on-device meal recipe data, it will be designed with future scalability in mind, including the integration of the TheMealDB API to access and share recipe data, and an LLM chatbot capable of offering recipe suggestions or answering questions about recipes and other topics. To support these future features, the app needs to be significantly modularized to ensure that integration of the new future components will have a minimal impact on the core structure of the app. This modularity will be designed with the Model-View-ViewModel (MVVM) approach.

MVVM is a modern approach to Android development that separates the application logic from the UI elements of the application by introducing an intermediary layer (Brian, 2023).

The approach is structured around three components which are *Model, View,* and *ViewModel*, and can be described as follows:

View:

- Informs the ViewModel about user actions.
- Observes ViewModel for data updates.
- Contains no business logic.

ViewModel:

- Mediates between View and Model.
- Transforms data from the Model.
- Provides data streams to the View.

Model:

- Holds application data.
- Does not directly talk to the View.
- Exposes data to the ViewModel via Observables.

It is important to understand how MVVM differs from other approaches. For instance, in a Model-View-Controller (MVC) architecture, the Android *Activity class* acts as a *Controller* and the XML file as the *View*; on the other hand, in MVVM, both the *Activity class* and the XML file are treated as the *View* and the ModelView acts as the *Controller*, that is where the business logic is written (Zych, 2023).

Target Audience

The app targets a wide audience of Android users from busy moms and dads wanting to provide cooked meals for their families by easily assessing and saving meal recipes to cooking enthusiasts wanting to store their recipes or explore new and exotic meal recipes, to novice cooks looking for a simple way to learn and organize meal recipes. It can also appeal to health-conscious individuals looking to learn or store recipes that meet specific dietary requirements due to social, cultural, or health considerations, such as gluten-free, vegan, or low-carb requirements. The app is meant to be easy to use and accessible through mobile devices as it targets Android users.

Key Features and Functionalities

As discussed previously, the app will be designed using the MVVM approach. This implies that the feature and the functionality of the app will be highly modularized separating the UI functionality and feature from the data application logic. The app will be developed using Kotlin and Jetpack Compose, which will be used to implement its core functionalities. The features of the as are as follows:

- Search: The user can search for recipes stored on-device by name, ingredient, or category.
- Add: The user can manually input and save their own recipes to the app.
- Modify: The users can modify existing recipes (whether pre-loaded from TheMealDB or user-added).
- Favorites: Users can add and manage a list of favorite recipes.

Future Features

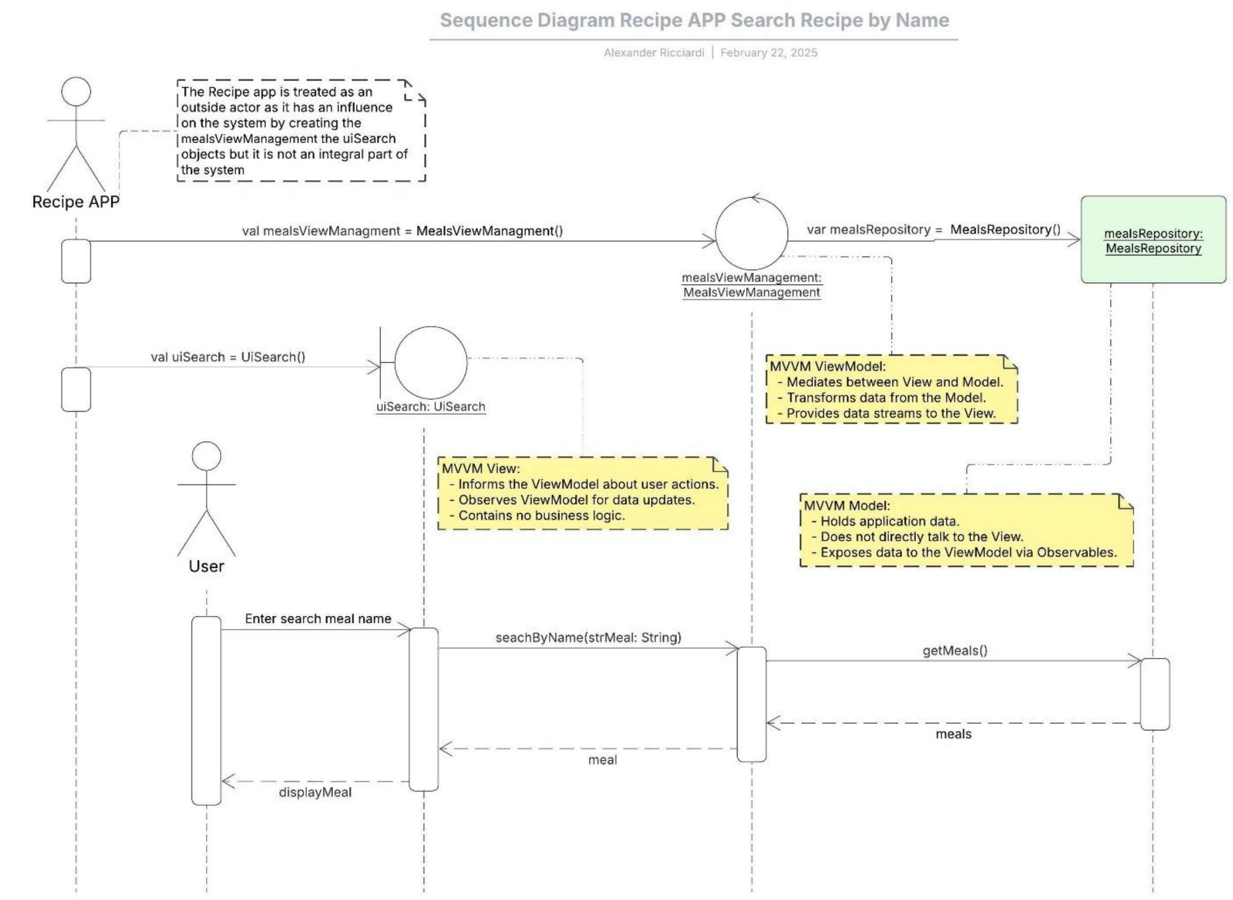
- Integration with the TheMealDB API the app will potentially connect to TheMealDB API to retrieve data, allowing users to access the crowd-sourced library of recipes from around the world and share their recipes.

- LLM Chatbot: An on-device (when Available) or cloud-based LLM chatbot could be added to suggest recipes from the local dataset (e.g., “What can I make with these ingredients?”), or retrieve recipes directly from TheMealDB Database, or answer cooking questions or other questions.

The sequence diagram below illustrates the search recipe by name functionality that could be implemented within the app using the MVVM approach.

Figure 1

Sequence Diagram Recipe APP Search Recipe by Name


Sequence Diagram Recipe APP Search Recipe by Name
Alexander Ricciardi | February 22, 2025
The Recipe app is treated as an outside actor as it has an influence on the system by creating the mealsViewManagement the uiSearch objects but it is not an integral part of the system
Recipe APP
val mealsViewManagment = MealsViewManagment()
mealsViewManagement: MealsViewManagement
var mealsRepository = MealsRepository()
mealsRepository: MealsRepository
val uiSearch = UiSearch()
uiSearch: UiSearch
MVVM ViewModel:
- Mediates between View and Model.
- Transforms data from the Model.
- Provides data streams to the View.
MVVM View:
- Informs the ViewModel about user actions.
- Observes ViewModel for data updates.
- Contains no business logic.
MVVM Model:
- Holds application data.
- Does not directly talk to the View.
- Exposes data to the ViewModel via Observables.
User
Enter search meal name
seachByName(strMeal: String)
getMeals()
meals
meal
displayMeal

Note: the UML sequence diagram illustrates the user utilizing the search recipe by name functionality query for a recipe.

The TheMealDB database is a NoSQL database that stores semi-structured data using JSON document format. The TheMealDB JSON format is structured as follows:

Code Snippet 1

TheMealDB JSON Structure

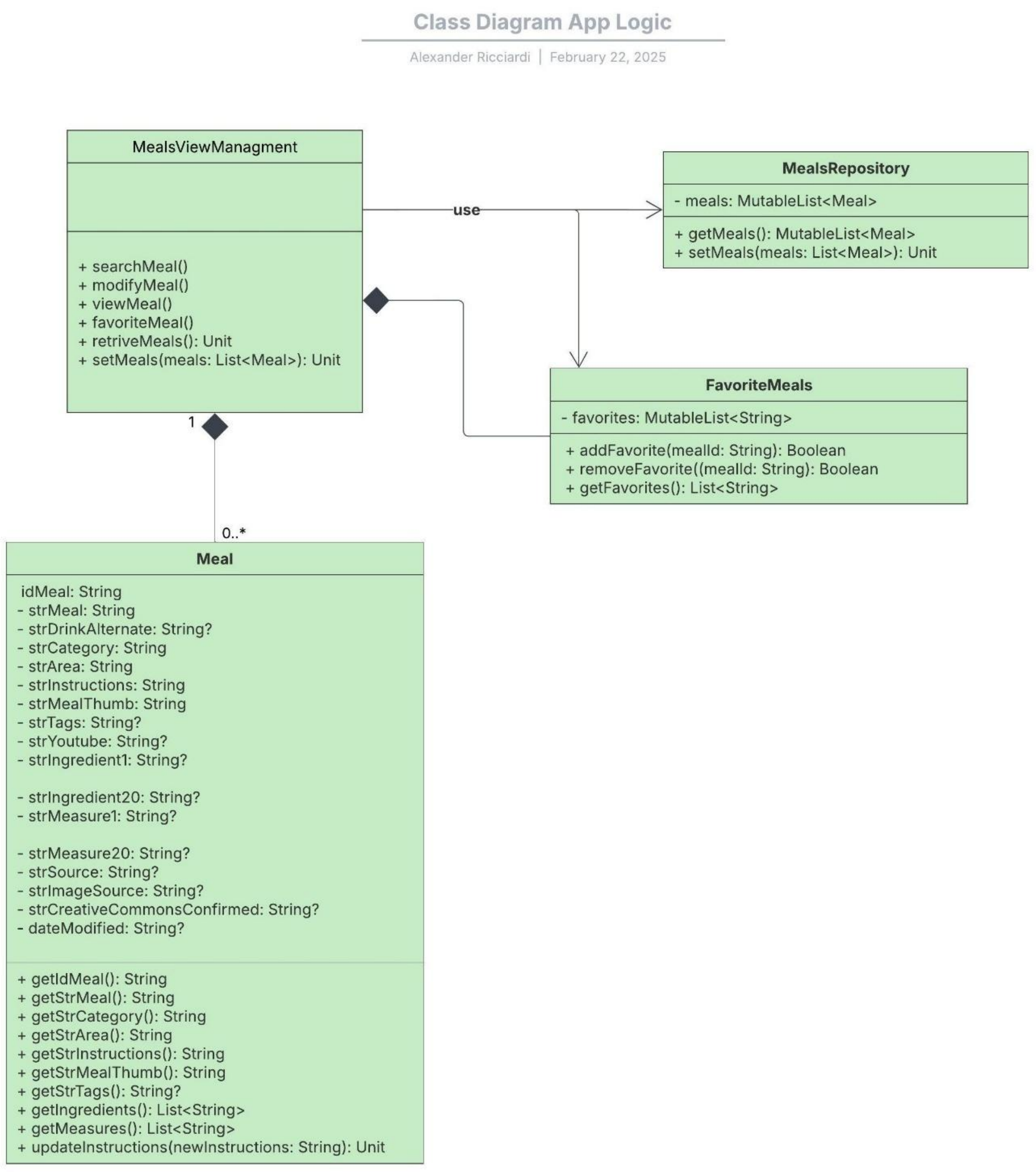
```
{
  "meals": [
    {
      "idMeal": "52977", // A unique id for the meal
      "strMeal": "Corba", // The name of the meal
      "strDrinkAlternate": null,
      "strCategory": "Side", // The category the meal belongs to
      "strArea": "Turkish", // The region or country the meal originates from
      // Instructions for preparing the meal
      "strInstructions": "Pick through your lentils for any foreign debris, rinse them 2 or 3
                          times with fresh water and put them in a pan on the heat. Add a
                          little salt, cover with water and bring to a boil...",
      // A URL to a thumbnail image of the meal
      "strMealThumb": "https://www.themealdb.com/images/media/meals/58oia61564916529.jpg",
      "strTags": "Soup",
      "strYoutube": "https://www.youtube.com/watch?v=VVnZd8A84z4",
      // The ingredients used in the meal
      "strIngredient1": "Lentils",
      "strIngredient2": "Onion",
      "strIngredient3": "Carrots",
      ...
      "strIngredient17": "",
      "strIngredient18": "",
      "strIngredient19": "",
      "strIngredient20": "",
      // The measurements for each ingredient
      "strMeasure1": "1 cup ",
      "strMeasure2": "1 large",
      "strMeasure3": "1 large",
      ...
      "strMeasure17": "",
      "strMeasure18": "",
      "strMeasure19": "",
      "strMeasure20": "",
      "strSource": null,
      "strImageSource": null,
      "strCreativeCommonsConfirmed": null,
      "dateModified": null
    },
    // ... more meals
  ]
}
```

Note: The code snippet provides an example of the TheMealDB JSON Structure. From "TheMealDB Arrabiata Meal" by TheMealDB (n.d.). https://www.themealdb.com/api/json/v1/1/search.php?s=Arrabiata

The recipe data will be stored on-device using the TheMealDB JSON Structure. The following class diagram illustrates the application logic handling the data. Note that this is an initial class diagram, and it is missing many of the classes, methods, and attributes that will be refined later on in the app development process.

Figure 2

Class Diagram App Logic


Class Diagram App Logic
Alexander Ricciardi | February 22, 2025
MealsViewManagment
+ searchMeal()
+ modifyMeal()
+ viewMeal()
+ favoriteMeal()
+ retriveMeals(): Unit
+ setMeals(meals: List<Meal>): Unit
use
MealsRepository
- meals: MutableList<Meal>
+ getMeals(): MutableList<Meal>
+ setMeals(meals: List<Meal>): Unit
FavoriteMeals
- favorites: MutableList<String>
+ addFavorite(mealId: String): Boolean
+ removeFavorite((mealId: String): Boolean
+ getFavorites(): List<String>
1
0..*
Meal
idMeal: String
- strMeal: String
- strDrinkAlternate: String?
- strCategory: String
- strArea: String
- strInstructions: String
- strMealThumb: String
- strTags: String?
- strYoutube: String?
- strIngredient1: String?
- strIngredient20: String?
- strMeasure1: String?
- strMeasure20: String?
- strSource: String?
- strImageSource: String?
- strCreativeCommonsConfirmed: String?
- dateModified: String?
+ getIdMeal(): String
+ getStrMeal(): String
+ getStrCategory(): String
+ getStrArea(): String
+ getStrInstructions(): String
+ getStrMealThumb(): String
+ getStrTags(): String?
+ getIngredients(): List<String>
+ getMeasures(): List<String>
+ updateInstructions(newInstructions: String): Unit

Note: The figures illustrate the class diagram app logic that handles the recipe data.

The following Kotlin code snippet is a potential representation of the meal class, wrapping the TheMealDB JSON structure.

Code Snippet 2

Meal Class - Kotlin

```kotlin
// Meal data class
// Wrapper of the TheMealDB JSON
// Version 1
data class Meal(
    var idMeal: String,
    var strMeal: String,
    var strDrinkAlternate: String?,
    var strCategory: String,
    var strArea: String?,
    var strInstructions: String?,
    var strMealThumb: String?,
    var strTags: String?,
    var strYoutube: String?,
    // The ingredients used in the meal
    var strIngredient1: String?,
    var strIngredient2: String?,
    var strIngredient3: String?,
    var strIngredient4: String?,
    var strIngredient5: String?,
    var strIngredient6: String?,
    var strIngredient7: String?,
    var strIngredient8: String?,
    var strIngredient9: String?,
    var strIngredient10: String?,
    var strIngredient11: String?,
    var strIngredient12: String?,
    var strIngredient13: String?,
    var strIngredient14: String?,
    var strIngredient15: String?,
    var strIngredient16: String?,
    var strIngredient17: String?,
    var strIngredient18: String?,
    var strIngredient19: String?,
    var strIngredient20: String?,
    // The measurements for each ingredient
    var strMeasure1: String?,
    var strMeasure2: String?,
    var strMeasure3: String?,
    var strMeasure4: String?,
    var strMeasure5: String?,
    var strMeasure6: String?,
    var strMeasure7: String?,
    var strMeasure8: String?,
    var strMeasure9: String?,
    var strMeasure10: String?,
    var strMeasure11: String?,
    var strMeasure12: String?,
    var strMeasure13: String?,
    var strMeasure14: String?,
    var strMeasure15: String?,
    var strMeasure16: String?,
    var strMeasure17: String?,
    var strMeasure18: String?,
    var strMeasure19: String?,
    var strMeasure20: String?,
    // miscellaneous Metadata
    var strSource: String?,
    var strImageSource: String?,
    var strCreativeCommonsConfirmed: String?,
    var dateModified: String?
) {
    // Getters for key fields
    fun getIdMeal() = idMeal
    fun getStrMeal() = strMeal
    fun getStrCategory() = strCategory
    fun getStrArea() = strArea
    fun getStrInstructions() = strInstructions
    fun getStrMealThumb() = strMealThumb
    fun getStrTags() = strTags

    // Retrieve a list of ingredients (ignoring empty or blank entries)
    fun getIngredients(): List<String> {
        return listOfNotNull(
            strIngredient1, strIngredient2, strIngredient3, strIngredient4, strIngredient5,
            strIngredient6, strIngredient7, strIngredient8, strIngredient9, strIngredient10,
            strIngredient11, strIngredient12, strIngredient13, strIngredient14, strIngredient15,
            strIngredient16, strIngredient17, strIngredient18, strIngredient19, strIngredient20
        ).filter { it.isNotBlank() }
    }

    // Retrievea list of measures (ignoring empty or blank entries)
    fun getMeasures(): List<String> {
        return listOfNotNull(
            strMeasure1, strMeasure2, strMeasure3, strMeasure4, strMeasure5,
            strMeasure6, strMeasure7, strMeasure8, strMeasure9, strMeasure10,
            strMeasure11, strMeasure12, strMeasure13, strMeasure14, strMeasure15,
            strMeasure16, strMeasure17, strMeasure18, strMeasure19, strMeasure20
        ).filter { it.isNotBlank() }
    }

    // Example setter to update instructions
    fun updateInstructions(newInstructions: String) {
        this.strInstructions = newInstructions
    }
}
```

Note: The Kotin conde snippet represents the Meal Class wrapping the JSON structure from TheMealDB.

The Meal class can be used to warp recipe data enabling it to be shared from the *Model* (*MealRepository*) to the *View* (*UiSeach*) through the *ViewModel* (*MealsViewManagment*).

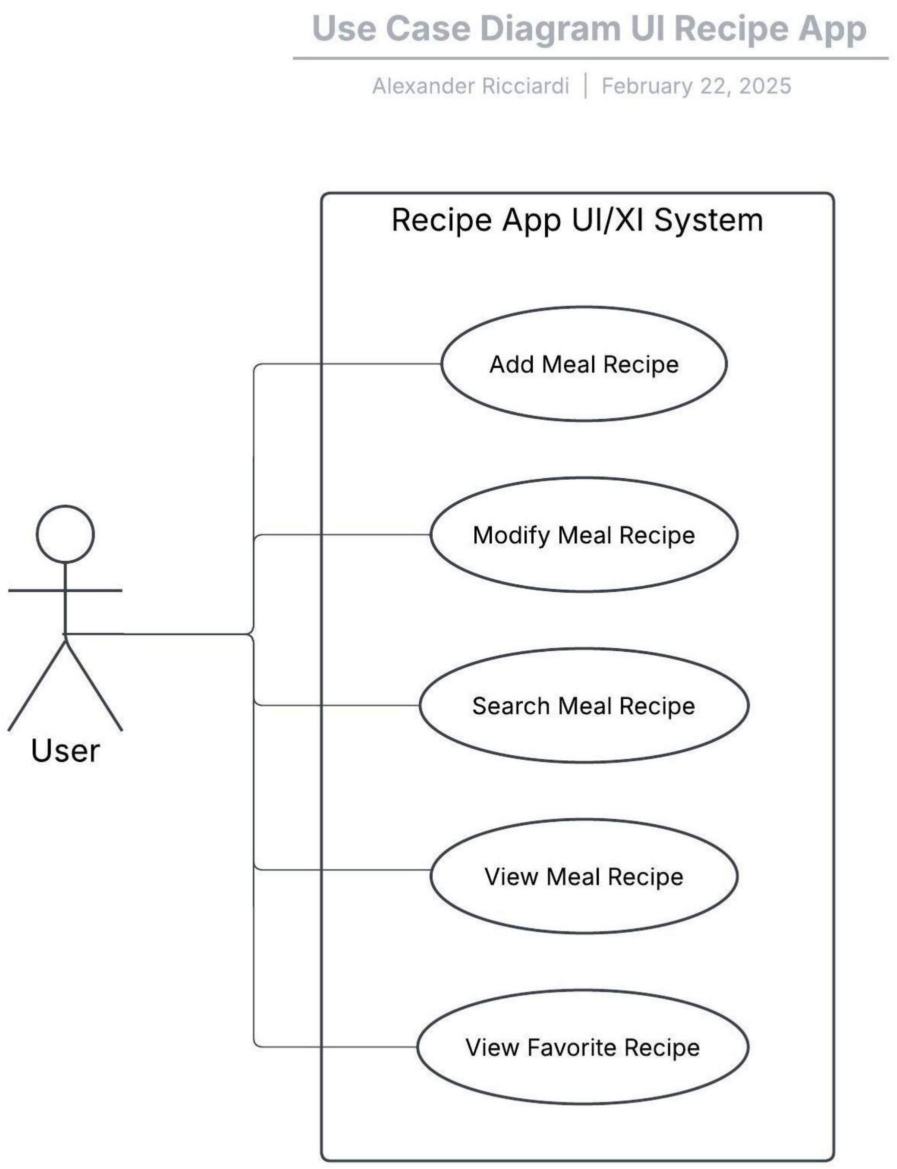
UI/UX Requirements and Analysis

The app's interface will be built using Jetpack Compose and the principal of the MVVM approach where the UI is separate from the app logic meaning that both the *Activity* class and the XML file are treated as the *View* and the *ModelView* acts as the *Controller*. The UI will enable the user to interact with the recipe data by viewing, storing, searching, adding, and modifying it, as well as by creating a list of favorite recipes. The following figure of a use case diagram illustrates how users interact with the app UI system, capturing the functionalities such as adding, modifying, searching, viewing, and favoriting recipes.

Please see next page.

Figure 3

Use Case Diagram UI Recipe App

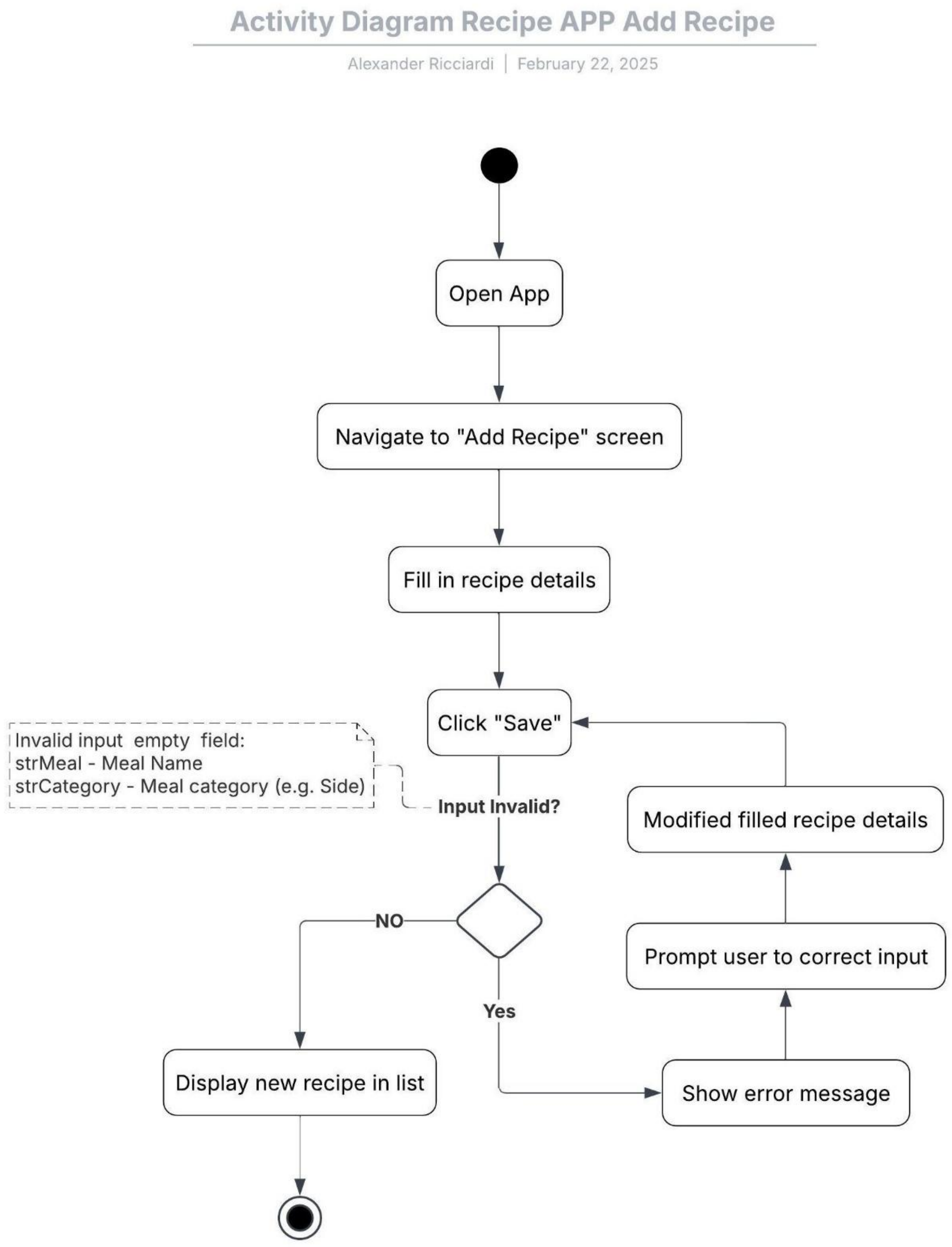


Note: The figures illustrate UI use case diagram capturing how the users interact with the app UI system, and the UI functionalities such as adding, modifying, searching, viewing, and favoriting recipes.

The following activity diagram captures the UX workflow for adding a new recipe, illustrating the sequence of actions from opening the app to saving the recipe.

Figure 4

Activity Diagram Recipe APP Add Recipe


Activity Diagram Recipe APP Add Recipe
Alexander Ricciardi | February 22, 2025
Open App
Navigate to "Add Recipe" screen
Fill in recipe details
Click "Save"
Invalid input empty field:
strMeal - Meal Name
strCategory - Meal category (e.g. Side)
Input Invalid?
NO
Yes
Display new recipe in list
Show error message
Prompt user to correct input
Modified filled recipe details

Note: The figure illustrates the activity diagram capturing the UX workflow for adding a new recipe and illustrating the sequence of actions from opening the app to saving the recipe.

The User Experience (UX) in the Recipe App will focus on making the app easy to use by designing an intuitive and enjoyable interface for the users to navigate through the app and interact with recipes.

Summary

The Android recipe app's goal is to provide access to meal recipes to a user through a UI system that includes view, search, add, modify, and favorite recipe functionalities with a UX that focuses on making the app easy to use. The target audience is wide, including busy moms and dads wanting quick, mobile, and easy access to meal recipes, cooking enthusiasts wanting to store their recipes and access to new and exotic meal recipes, novice cooks looking for a simple way to learn and organize meal recipes, and health-conscious individuals looking to learn or store recipes that meet specific dietary requirements. The app will be developed using Kotlin and Jetpack Compose, with plans for future integration of a LLM chatbot and the TheMealDB API to access and share recipe data. Additionally, using the MVVM approach, the app will be highly modularized by separating the UI functionality and features from the data application logic, allowing future features to be integrated without affecting the core structure of the app.

References

Brian, M. (2023, August 7). *Mastering Android MVVM architecture: Developers' guide*. Medium. https://medium.com/@mutebibrian256/mastering-android-mvvm-architecture-developers-guide-3271e4c8908b

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

Zych, F. (2023, November 15). *Getting to grips with MVVM Architecture*. Netguru. https://www.netguru.com/blog/mvvm-architecture