MyRecipe App Overview

MyRecipes is an Android application developed in Kotlin using Jetpack Compose, which allows to discover, browse, and save their favorite recipes. The app fetches recipes from the TheMealDB API and stores favorites locally in a **JSON file**.

The app includes the following key features:

- **Home Screen**: Displays a selection of recipes, allowing to explore new recipes.
- **Recipe Details Screen**: Provides detailed information for each recipe, including an image, ingredients, origin and instructions, with the option to save the recipe to the Favorites list.
- **Favorites Screen**: Displays a list of saved recipes and allows to access their details.
- **Search Functionality**: Users can search for recipes by name, which leads to a screen showing the search results.

Home Screen

The key aspects of the Home Screen include:

- **Random Recipe Fetching**: The app displays a selection of four random recipes fetched from TheMealDB every time navigates to the home page. This is handled by the **HomeViewModel**, which retrieves the recipes asynchronously using **ViewModelScope** and **Coroutines**.
- **User Interface**: The random recipes are displayed in a **LazyColumn**, which efficiently handles large datasets by only rendering visible items. Each recipe is shown in a simple card layout with essential details such as the recipe's name and image.
- **Navigation**: Users can click on any recipe card to view its details. This interaction is managed by the **onItemClick**function, which navigates to the **Recipe Details Screen**.
- **State Management**: The app maintains the state of the random meals in the **HomeViewModel**. The meals are fetched and stored in a mutable state property fourRandomMeals, ensuring the UI reflects the most up-to-date random recipe suggestions.

Search Functionality

The **Search Screen** allows to search for recipes by name. The implementation involves a **TextField** where users can enter search queries. The search functionality is powered by **ViewModel** and utilizes **StateFlow** to handle the search state.

- **Debouncing**: The input text is debounced with a delay of 500 milliseconds, ensuring that API calls are made only after the user stops typing, reducing unnecessary requests.
- **API Integration**: Once a valid query is entered, the app queries the TheMealDB for matching recipes. Results are displayed in a **LazyColumn**, with each recipe shown in a clickable **Card**.

- **Loading State**: While the search is in progress, a **CircularProgressIndicator** is displayed. If no results are found after a search completes, a friendly message appears to let users know that no matching recipes were found.
- **State Management**: The **SearchViewModel** is responsible for managing the search state, including the search query, results, and loading state. It uses **StateFlow** to update the UI reactively based on the search text.

The **Favorites Screen** allows to view and manage the recipes they have marked as favorites. Here are the key features of the Favorites Screen:

- **Displaying Favorite Recipes**: The app shows a list of recipes that has been saved to their favorites. This list is managed by the **FavoritesViewModel**, which interacts with an internal repository to retrieve the saved recipes. These recipes are fetched from a local data source (stored in a .json file).
- **State Management**: The **favoriteMeals** state is stored in the **FavoritesViewModel**, which is initialized by calling getAllFavorites(). This fetches the list of favorite meals and updates the UI. If the user returns to the Favorites Screen, the list is refreshed to ensure it shows the most current data.
- **Lazy Loading**: The list of favorite recipes is displayed using a **LazyColumn**, ensuring efficient rendering even if there are many recipes. Each recipe is presented as a card, allowing easily tap and view the recipe details.
- **Navigation**: Just like the Home and Search screens, the Favorites Screen uses **onItemClick** to navigate to the Recipe Details screen.
- **Refreshing Favorites**: Whenever the Favorites screen is opened or updated, the **refreshFavorites** function is called, ensuring that the list of saved recipes is always up to date.

API Integration for Data Fetching

The app integrates with the **TheMealDB API** to fetch recipe data, including random recipes and search results based on input. The API is accessed through a web service implemented using **Retrofit** and **Moshi** for data parsing and network communication.

Key Components:

1. **WebService**:
    - The **WebService** class is responsible for setting up and making network requests to the API. It uses **Retrofit**for API communication and **Moshi** for JSON parsing.
    - The **MealsApi** interface defines two key API endpoints:
        - **getRandomMeal**: Fetches a random recipe.
        - **searchMealsByName**: Searches for recipes by name.
    - The **MoshiMealResponse** data class is used to parse the response from the API, which is then mapped into a list of Meal objects.
2. **MainRepository**:
    - The **MainRepository** acts as the intermediary between the UI and the WebService. It contains methods like getRandomMeals and searchMeal that fetch data asynchronously using **Kotlin Coroutines**.

    - The repository ensures data is fetched off the main thread (using `Dispatchers.IO`), ensuring smooth app performance.
    - The repository is also responsible for handling unique random meals, ensuring that duplicate meals are not added to the list.
3. **Moshi JSON Parsing**:
    - The **Moshi** library is used for JSON parsing and serialization. It is configured with the **KotlinJsonAdapterFactory** to handle Kotlin-specific data structures.
    - This allows the app to easily map JSON responses from the API into Kotlin objects (`Meal`).
4. **Asynchronous Data Fetching**:
    - The app fetches data asynchronously using coroutines, ensuring that the UI remains responsive while waiting for network responses.
      The `getRandomMeals` method ensures multiple unique meals are fetched by repeatedly requesting random meals from the API.
5. **Data Handling**:
    - Once the data is fetched from the API, it is transformed into a format usable by the app (`Meal` objects). This is done using the `asMeals()` extension function, which converts the raw API response into a list of `Meal`objects.

Conclusion

The **MyRecipes** app allows users to discover, browse, and save recipes. It is built with Kotlin and Jetpack Compose, providing a better experience. The app fetches recipes from **TheMealDB API**.

The app uses modern tools like **ViewModel**, **Coroutines**, and **Retrofit** to handle data and API requests. It also features an intuitive interface, making it easy to explore new recipes or check saved favorites.