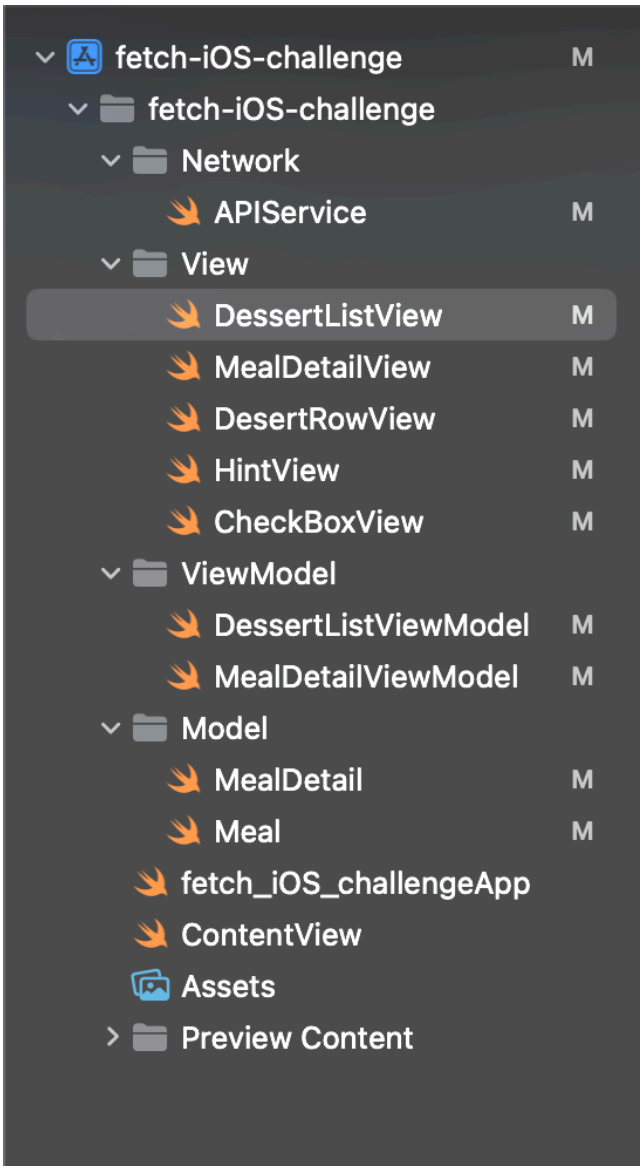
Fetch iOS Challenge application

- *By Aashi Shrimal*

This is a documentation to help understand each file in the codebase.

This is a native iOS application made using swift and swiftUI. It follows the MVVM design pattern.

The folder system looks like this:



Code files in detail :

NETWORK **1. APIService**	The `MealService` class in the code is designed to interact with TheMealDB API to fetch meal-related data. It contains two main functionalities: 1. `fetchDesserts(completion:)` - This method fetches a list of desserts from TheMealDB. It constructs a URL for the dessert category, makes a network request, and decodes the JSON response into a `MealList` object. The completion handler returns either the `MealList` on success or an `Error` on failure. 2. `fetchMealDetails(by:completion:)` - This method fetches details for a specific meal by its ID. Similar to `fetchDesserts`, it constructs a URL using the meal ID, performs a network request, and decodes the response into a `MealDetailList` object. The completion handler again returns either the `MealDetailList` or an `Error`. The `NetworkError` enum defines possible errors that might occur during these network operations, such as `invalidUrl` and `noData`. This class uses asynchronous network calls with completion handlers to return data or errors, making it suitable for modern asynchronous programming practices in Swift. The use of generics in completion handlers (`Result<MealList, Error>` and `Result<MealDetailList, Error>`) provides a clean and flexible way to handle the network response.
VIEW **2. DessertListView**	The `DessertListView` struct presents a list of desserts fetched from an API. It uses a `DessertListViewModel` to handle data fetching and state management. The view displays a list of desserts, grouped by their first letter, in a `NavigationView`. It also includes a loading overlay and an error alert to handle loading states and errors, respectively. Additionally, there's a hint overlay presented on the initial load to guide users. The `MealDetail` extension provides a default value for a meal detail,

	useful for placeholders or default states. This struct and extension embody key principles of MVVM architecture and SwiftUI's declarative UI approach, enhancing code readability and maintainability. The *updated* DessertListView now includes a search feature, allowing users to filter desserts by name. The view dynamically updates to show only the sections and desserts that match the search query. The search functionality is integrated via the .searchable(text:) modifier on the dessert list. This addition enhances the user experience by providing a more interactive and user-friendly interface for finding specific desserts within the list.
3. MealDetailView	The `MealDetailView` struct provides a detailed view of a selected meal, including its image, title, ingredients, and preparation instructions. It utilizes the `MealDetailViewModel` to fetch and store meal details. The view supports marking meals as favorites with a heart icon, integrates a checkbox for ingredients, and uses a scrollable layout to accommodate content. It also includes a conditional hint overlay to enhance user engagement.
4. DessertRowView	The `DessertRowView` struct defines a row view for displaying a dessert. It consists of an asynchronous image loader for the dessert's thumbnail and the dessert's name text. The thumbnail is styled with a specific frame size and corner radius, and a placeholder is shown during image loading. This view is designed to be used in a list, presenting a concise and visually appealing summary of each dessert item.
5. HintView	The `HintView` struct is designed to display a hint or instructional overlay to the user. It uses a `@Binding` property to control its visibility, allowing it to be shown or hidden based on user interaction. The view includes a custom `TextBubbleView` for presenting a title, subtitle, and description in a styled bubble format. A button labeled "Got it!" allows users to dismiss the hint. This structure enhances user experience by providing contextual information in an engaging and interactive manner.

6. CheckboxView	The `CheckBoxView` struct represents a custom checkbox component. It toggles its state between checked and unchecked upon tapping, visually represented by changing the system image and color. The `@State` property `isChecked` tracks the checkbox's state, allowing for a dynamic UI response to user interaction. This view is useful for tasks requiring user selection or indicating completion status.
ViewModel **7. DessertListViewModel**	The `DessertListViewModel` class implements the observable object protocol to manage the state and data for a list of desserts in a SwiftUI application. It utilizes Combine for data binding and publishes several properties to update the UI based on data changes. These properties include the list of desserts, loading status, error visibility, and an error message. Additionally, it holds a selected meal detail for further actions. The class provides functionality to fetch dessert data from a service, handle the result, and manage the data for display, such as grouping desserts by their first letter and sorting section titles. The updated `DessertListViewModel` now includes functionality to filter desserts based on user input in a search field. It uses Combine to reactively update the `filteredDesserts` array whenever the `searchText` changes, applying a debounce to reduce the frequency of updates. This model provides a dynamic way to filter and display desserts, enhancing the user experience by allowing for real-time search within the dessert list. The search results are grouped and sorted, maintaining the alphabetical organization even within the filtered view.
8. MealDetailViewModel	The `MealDetailViewModel` class is responsible for fetching and storing the details of a specific meal from a meal service. It utilizes the `@Published` property wrapper to allow SwiftUI views to react to changes in the meal detail data and the loading state. This class simplifies the process of asynchronously loading meal details by ID, updating the UI accordingly based on the success or failure of the fetch operation.
Model	The `MealDetail` struct represents detailed information about a meal, including its ID, name, cooking instructions, image URL, ingredients, and measurements.

9. MealDetail	It conforms to the `Decodable` protocol to facilitate decoding from JSON data, and implements a custom initializer to handle the decoding process. The struct also includes a computed property to combine ingredients with their corresponding measurements for easier display. This structure is designed to efficiently parse and store detailed meal data retrieved from an API, making it a crucial part of the meal detail fetching process in the application.
10. Meal	The `MealList` struct is designed to decode a list of meals, specifically for the Dessert category, from JSON data. It contains an array of `Meal` structs. Each `Meal` struct represents an individual meal and includes properties for the meal's ID, name, and thumbnail URL. This struct conforms to both `Codable` and `Identifiable`, with the latter allowing it to be easily used in SwiftUI lists where each item requires a unique identifier.