Casey Tirrell

15 May 2024

For this project I decided to employ Model View ViewModel architecture. To do this I created a group named “Meals” which contained three swift files: Meals.swift, MealsVew.swift and MealsViewModel.swift. Along with these files I created a StartView screen, this being the screen that the user will be greeted with when they open the application.

**Meals.swift**

* This is the model.

struct Meal

* This struct represents a single meal with the basic details that are given by the API which returns all of the current desserts available.
* It’s identifiable and decodable which allows for instances to be decoded from JSON.
* It has three properties:
  + “idMeal”
    - This represents the ID of the dessert.
  + “strMeal”
    - This represents the name of the dessert.
  + “strMealThumb”
    - This represents the URL to the dessert’s image thumbnail.
* It also has one computed property:
  + “id”
    - This lets “idMeal” be returned as a unique ID, which is needed by the identifiable protocol.

struct MealsResponse

* This represents the response given from the API call which gives the list of all of the desserts currently available.
* It's decodable to allow for instances to be decoded from JSON.
* It has just one property
  + “Meals”
    - This represents an array of “Meal” instances that is returned from the API.

struct MealDetails

* This struct represents the detailed information that is returned from the API that takes a dessert ID as a parameter.
* It’s identifiable and decodable which allows for instances to be decoded from JSON.
* It has 11 properties
  + “idMeal”
    - This represents the ID of the dessert.
  + “strMeal”
    - This represents the name of the dessert.
  + “strDrinkAlternative”
    - I’m not entirely sure what this is supposed to represent as the value is always returned to be null. But, it isn’t used within my app; it was just returned in the JSON package so I have it just to avoid errors.
  + “strCategory”
    - This represents the category that the dessert falls under.
  + “strArea”
    - This represents the area/region the meal is from.
  + “strInstructions”
    - This represents the set of instructions used to create said dessert.
  + “strMealThumb”
    - This represents the URL to the dessert’s image thumbnail.
  + “strTags”
    - This represents the tags that the dessert falls under.
  + “strYoutube”
    - This represents the youtube URL to watch a tutorial on how to make the dessert.
  + “strSource”
    - This represents the source URL of the dessert recipe if the user would like to visit it.
* It also has just one computed property
  + “Id”
    - This lets “idMeal” be returned as a unique ID which is required by the identifiable protocol.
* This struct has a set of CodingKeys to make sure that key names match the JSON data to avoid an errors.
* This struct has a custom decoding set for the package: *init(from decoder: Decoder):* since it will be easier to store the information within a container for display on the view.
  + It decodes each property using the provided keys.
  + Some of these properties don’t always get filled like “strDrinkAlternate”, “strTags”, “strYoutube” and “strSource”. For these the use of decodeIfPresent is necessary.
  + It stores the ingredients list along with their respective measurements within a dictionary.
    - It does this with a for loop that iterates 1 through 20 since that's how many ingredients are listed in the JSON package.

struct MealDetailsResponse

* This struct represents the response given from the API call that provides the specific details of a certain dessert.
* It’s decodable to allow for instances to be decoded from JSON.
* It has just one property
  + “Meals”
    - This represents the array of “MealDetails” instances that is returned from the API.

**MealsView.swift**

* This is the view.

struct MealsView

* This view is the list of all the current desserts that are offered. The user will have the option to scroll through this list and click on any of the desserts listed to see its specific details.
* It give the user the name of the dessert, along with a photo.

struct MealDetailsView

* This view is a dessert detail view that the user will interact with whenever they click on an item in the list.
* It gives the user information on the dessert like:
  + Name
  + Photo
  + Category
  + Area/Region
  + Tags
  + Instructions on how to make the dessert
  + Ingredients along with the measurements
  + A youtube tutorial video
  + A button that brings the user to the source URL.
* For styling this view takes advantage of the Fetch color scheme.

struct StartView

* This view is the first view the user sees when they open the app.
* The user can tap anywhere on the screen to continue to the MealsView

**MealsViewModel.swift**

* This is the view model.

func fetchMeals()

* This function's purpose is to fetch the list of desserts that is returned from the API and then update the application’s state to represent that.
* It is asynchronous.
* It uses a loading state to indicate that the data is in the process of being fetched.
* It uses a URLSession to make an asynchronous network connection.
* It uses a JSONDecoder to decode the fetched data into the “MealsResponse” struct discussed earlier.
  + If this fails an error is thrown.
* From there it sorts the list of fetched data into alphabetical order and updates the “meals” property with an updated list.
* It then sets the loading state back to false to indicate that the fetching process is complete.
* It then prints the number of fetched desserts in the terminal to confirm its process.

func fetchMealDetails()

* This function's purpose is to fetch detailed information about a specific meal by its ID and update the application's state to represent that.
  + It takes “id” as a parameter.
* It is asynchronous.
* It uses a loading state to indicate that the data is in the process of being fetched.
* It uses a URLSession to make an asynchronous network connection.
* It uses a JSONDecoder to decode the fetched data into the “MealDetailsResponse” struct discussed earlier.
  + If this fails an error is thrown.
* From there it checks to see if there is at least one meal in the response given by the API call and then updates the “mealDetails” property with the first item in the array.
* It then sets the loading state back to false to indicate that the fetching process is complete.