CS290.02
Homework 4 - Recipe App v4 with API Call

So far we have been progressively improving the Recipe app, installing our MVVM architecture in the last homework. We are now about to change out our datastore for just an API call. You are free to continue using your existing recipes code base or to use the posted solution for hw3 (in the Sakai Resources folder). If you use your own code base, you may want to keep a copy of your current Xcode project that you completed for hw3, so you have a pristine version of that. We are going to make enough changes to the ViewModel code that saving the other could be worth it just for your own knowledge and posterity (we will not be using it in future homework or anything).

So, instead of using a RecipeStore, we are going to rely on an API for all our data.

A) Since this is a GraphQL API, this shifts our thinking as to where LoadingState is. Since we're going to be able to fetch all our data all at once, we won't need LoadingState on the Detail, but we do now need it in the ListView. To keep things simple in this homework, we are not going to try and load the API data into the DataStore. We are going to ignore the store and just make an API call on the list view.

B) Right now our RecipeDetailVM updates the dataStore when the lastPreparedAt is modified with an updateRecipe function. Now we are going to "pretend" to send an update command to the API Service. Details will follow. But since we are pretending, your preparedAt checkbox on the list view will not reflect changes from the detail view. This functionality is currently "broken" and that is fine.

C) As always … the Movies code fromLecture 11 is probably pretty helpful.

The baseUrl for the api we are using is
https://learning-swift.gigalixirapp.com/api

To access the browser-based graphiql tool, visit
https://learning-swift.gigalixirapp.com/api/graphiql

A starting recipe GraphQL query **with instructions info missing**:

```
query  { recipes
   { id name credit details componentSections lastPreparedAt mealCourse thumbnailUrl tags
    components { id ingredient { id name } note position quantity section unit }
   }
}
```

In your Recipe App:

1. You will need 4 files in your code: APIClient.swift, APIError.swift, GraphQLAPI.swift, and GraphQLEndpoint.swift. They are in the hw4 folder, or you could grab them from the Movies code from Lecture 11 If you drag those files into your project, you'll want to make sure the "Target Membership" box is checked for them in the right-hand nav. Or you could just create the empty files in your project and copy the content from those files in.
2. In the RecipeListVM, remove any RecipeStore-related code.
3. Include a LoadingState enum for your ListViewModel that looks like this

```
enum LoadingState {
    case notAvailable
```

```
    case loading
    case success
   case failed(error: Error)
}
```

Note that the success state no longer has associated data. When we get data from the API, we will load it right into our RecipeListVM recipes array

4. On the RecipeDetailVM, remove any references to the recipeStore and to state. You most likely have a @Published var recipe: Recipe = Recipe(name: “Loading”, mealCourse: .sideDish). Remove that “Loading” recipe dummy data (but keep the property). In other words, erase the equals sign and everything to the right of that property.
5. On the RecipeDetailVM, include a property to hold onto the APIService. Then adjust the updateRecipe() function you have in your RecipeDetailVM so it calls updateRecipe on the apiService, not the recipeStore.
6. So now your RecipeDetailVM should have only two properties without a default value: an apiService and the recipe. That means they need be assigned in your init function. Adjust your init function in the RecipeDetailVM so it receives an apiService and a recipe and sets sefl.apiService to the apiService and self.recipe to that recipe.
7. With the browser-based graphiql tool referenced in the introduction (and demoed during lecture), copy in the incomplete query provided above. It is missing “instructions” and the two Instruction properties. Include those in the query and ensure they are showing up in the JSON displayed in the graphiql tool. Notice how the braces are used to show nested properties. So instructions is probably at the top level of properties and then its two properties are inside some braces.
8. Create a RecipesEndpoint file to hold the baseUrl and the query for recipes you just completed
9. Create an RecipeAPIService struct with a **fetch** function to fetch Recipes. Also include two other functions:

```
 func updateRecipe(_ recipe: Recipe) {
       print(“Simulating Update Recipe")
}

  func deleteRecipe(_ recipe: Recipe) {
       print(“Simulating Delete Recipe")
}
```

10. Add a property to your RecipeListVM to store the apiService. When you create your RecipeListVM in your App file, instead of injecting the recipeStore, you will inject the ReceipeAPIService.

    NOTE: Your store was an ObservableObject, but the RecipeAPIService is just a struct. So we don’t need to assign it to @StateObject or anything. When we pass an apiService to the viewModel, we will just create an instance as we do so. I bet some of the code will have seen in class will show this.
11. In your RecipeListVM, change the deleteRecipe function so that it calls the apiService, not the recipeStore
12. In your NavigationLink destination on the List view, you are no longer going to pass the store and the recipe.id. Instead, pass the viewModel.apiService and the recipe itself to the RecipeDetailVM.
13. Include an alert modifier on your ListView to display any errors.

14. Make sure to display any errors. Test two types of errors: a) Turn off your network connection and ensure an error displays. b) misspell one of the requested properties in your RecipesEndpoint query and ensure a GraphQL error displays.
15. We are faking any update or delete commands since we won't be making those calls back to the API. So when you click the prepared button or swipe to delete the recipe, you'll see the log commands show up in the console but our data won't change.
16. In all our homework, we never displayed the recipeComponent note if it had one! Please include the recipeComponent note at the end of the componentDisplay string you are generating in the RecipeDetailVM.
17. At the end of it all you should see one lone recipe from the API. Because typing in more than that was tedious.

Congratulations on completing this homework so quickly and proactively on Saturday! Enjoy the Super Bowl (as a sportsball or just a cultural event) knowing you can rest easy.