CS290.02
Homework 1 - Start modeling the Recipe Domain

Please refer to the accompanying recipe example, Gong Bao Chicken with Peanuts, for additional context. Do not be disturbed by the length of the text that follows. Much of it is commentary.

During the first half of the semester we'll be working to build a recipe app in SwiftUI. So our first task will be to get a good data model in place. Our app will eventually have a searchable list of recipes as well as a detail page about each. Gong Bao Chicken with Peanuts is an example.

How do we break down the information we are seeing into types within Swift? And once we build those types with structs, what additional functions and computed properties might be useful going forward as we look to build out our app?

Sometimes coming up with a data model is "easy" as you are mirroring an API's data structures. But we don't yet have that luxury here. Let's start from scratch, drawing inspiration from that Gong Bao recipe.

Certainly we will have a Recipe type. Are there other types we should break out? Let's create a notion of an Ingredient, as well as an Instruction. And, to get comfortable with enums, let's include a MealCourse enum.

This assignment will be composed and delivered inside a single Swift Playground.

DATA MODEL

1. In your Swift Playground, create the following 3 structs and 1 enum:

a) Recipe struct containing

Id	UUID
name	string
Details	optional string
credit	optional string
thumbnailURL	optional url
mealCourse	MealCourse enum
instructions	an array of Instructions
ingredients	an array of Ingredients
tags	an array of Strings

b) MealCourse enum with the cases **appetizer**, **mainDish**, **sideDish**, **dessert**

c) Instruction struct containing

position	integer with a default value of 999999
instructionText	string

d) Ingredient struct containing

Id	UUID
name	string
position	integer with a default value of 999999
quantity	optional double (i.e., 1.5)
unit	optional string (i.e., “cups”)
note	optional string

A note: It might seem like a good idea to make the unit property an enum with cases like with cups*,* tablespoons*, etc.). But should this enum include* slices*? A* pinch*? There really isn’t a defined list of options here. Let’s keep it a string.*

2. Reflect on the data model you just created. In its current version, ingredient names are simple strings that are coupled with quantity, unit, and position. This doesn’t easily allow us to roll-up ingredients into shopping lists when the same ingredient is used multiple times in a recipe (sliced garlic in the oil, minced garlic in the sauce, etc.). Let’s decouple the ingredient from it’s order and measurement:

a) *Rename* the current Ingredient struct to RecipeComponent and create a separate Ingredient struct which only has a **name** property which is a String.
b) Include the Ingredient struct within the RecipeComponent struct as a property named **ingredient**.
c) Change the Recipe struct so that it references an array of **components: [RecipeComponent]**, not **ingredients: [Ingredient]**

3. Note in the Gong Bao Chicken recipe example that the ingredients are broken up into sections according to particular component of the dish (marinade, sauce, etc.) Our current data model does not easily support breaking the ingredients into sections. Consider a couple of choices:

Option 1

First, instead of a recipe having many components, it could have many sections. We could consider inserting a “layer” so that a recipe would have sections and sections would have components, like so:

Recipe
name: String
// other properties…

sections: [RecipeSection]

RecipeSection:
name: String
components: [RecipeComponents]

That way we could just iterate through the sections when we are displaying a detail view of the recipe. However, there are some downsides to this approach. What happens if the recipe ingredients are not divided into sections? Should we create a dummy empty section to "hold" them? And, when it comes time to think about database persistence, are we creating an extra table of "Sections" just to handle what in the end is a formatting exercise?

Option 2

Instead, let's think about the sections as simple labels that we need to manage. Include a String property on RecipeComponent called **section**. This will tell us if this recipe component will be displayed in a particular section … or not. In other words, *that label could be (and often will be) nil.* Also, as we think about displaying the ingredients, we will need to keep track of the order of the sections. Do we want the sauce components first or the marinade? How should we do that?

a) Go with Option 2. Include an array of these section header strings on Recipe and call that property **componentSections** … we will then be able to see if there are strings in the sections array and iterate through those to display the components in the sections.

4. Conform Recipe, RecipeComponent, and your new Ingredient structs to the Identifiable protocol. While in some instances you might use an Int for the id, use a UUID type and initialize a UUID value as the default.

5. Copy and paste the 3 sample recipes provided into the playground.

CREATE SOME HELPFUL FUNCTION AND COMPUTED PROPERTIES

6. As we look towards displaying the recipe list on the screen in our UI, we'll need to consider how to deliver sectioned ingredients (the marinade ingredients, the sauce ingredients, etc.) to the UI. And, as discussed above, some ingredients belong to a section while others do not.

- We will need to break the **recipeComponents** into their sections if they exist. So it's likely that we will need a *function* that takes a section label string ("Marinade") and returns an position-ordered array of RecipeComponents in that section. Within the Recipe struct, create that instance function titled **componentsForSection**.
- We will need to return recipe components that do not have a section so we can include those as well. Create a *computed property* on Recipe called **unsectionedComponents** that returns recipe components with no section.

Note that the first task here will need to be a function since it takes a parameter (the section label) while the second task does not have a parameter and can be a computed property

For both task, t*hink declaratively here rather than imperatively. Sure you could loop over the items with a counter or something, but is there a more declarative way to do so?*

7. Create 2 static functions with the signatures and functionality:

```
static func mealCourseFilter(course: MealCourse, recipes: [Recipe]) -> [Recipe]
```

Inside the body of this function, filter the recipes to return only those which have the mealCourse that was provided to the function

```
static func courseSearchAndContinue(course: MealCourse, recipes: [Recipe], completion: ([Recipe]) -> Void)
```

Inside the body of the function use the **mealCourseFilter** function you just created to get an array of recipes and call then call the completion closure on the results.

EXECUTE AT THE BOTTOM OF THE PLAYGROUND

Now let's use the models and tools that we built. Note that you can use the command "print" to print information to the playground log. For example, print(recipe.name) will print the **name** property of the variable **recipe** in the log.

Make sure to download the sample data and copy and paste it into your Playground.

8. In the playground, execute the **courseSearchAndContinue** method with the .sideDish mealCourse and the sample recipes and provide a closure that prints the recipe names to the playground console with the command

```
print( [*however you are referring to the recipe*].name)
```

You will succeed if the log results look like this:

Green Beans with Miso Butter
Dry-fried Green Beans

9. Create a standalone function with the signature

```
printIngredientComponents(recipe: Recipe)
```

Within the body of that function, first print out the unsectioned recipe components for the recipe followed by the sectioned components, including their section label in capital letters.

If you then enter the command

```
printIngredientComponents(recipe: Recipe.previewData[0])
```

You will succeed if the log results look like this:

CHICKEN
boneless chicken breasts
garlic
ginger

spring onions
dried chiles
cooking oil
Sichuan Pepper
roasted peanuts
MARINADE
salt
light soy sauce
Shaoxing wine
potato starch or corn starch
SAUCE
sugar
potato starch or corn starch
dark soy sauce
light soy sauce
Chinkiang vinegar
seasame oil
chicken stock