Assignment VI: Memorize Themes

Objective

The goal of this assignment is to learn about how to have multiple MVVMs in your application and how to present groups of `Views` via navigation or modally.

Be sure to review the Hints section below!

Also, check out the latest in the Evaluation section to make sure you understand what you are going to be evaluated on with this assignment.

Due

This is the last assignment of the quarter before you start on your final project. It is similar in scope to your Assignment 3, so be sure to budget your time accordingly.

Materials

- You will start this assignment with your Memorize from Assignment 5.

Required Tasks

1. Your Memorize application should now show a "theme chooser" UI when it launches. See attached images for examples, but you can be creative with how you show each theme.
2. Use a `List` to display the themes.
3. Each row in the `List` shows the name of the theme, the color of the theme, how many cards in the theme and some sampling of the emoji in the theme.
4. Touching on a theme in the `List` navigates (i.e. the `List` is in a `NavigationView`) to playing a game with that theme.
5. While playing a game, the name of the theme should be on screen somewhere and you should also continue to support existing functionality like score, new game, etc. (but you may rearrange the UI if you wish).
6. It is okay if going from playing a game back to the chooser and then back to the game in progress restarts the game, though savvy implementations would probably not do that (except when the theme in question is modified (see below) since that would almost certainly want to restart the game).
7. Provide some UI to add a new theme to the `List` in your chooser.
8. The chooser must support an Edit Mode where you can delete themes and where you can access some UI (i.e. a button or image in each row) which will bring up a Theme Editor UI for that theme modally (i.e. via sheet or popover).
9. The Theme Editor must use a `Form`.
10. In the Theme Editor, allow the user to edit the name of the theme, to add emoji to the theme, to remove emoji from the theme and to specify how many cards are in the theme. (It is Extra Credit to be able to edit the color of the theme.)
11. The themes must be persistent (i.e. relaunching your app should not cause all the theme editing you've done to be lost).
12. Your UI should work and look nice on both iPhone and iPad.
13. Get your application work on a physical iOS device of your choice.

Hints

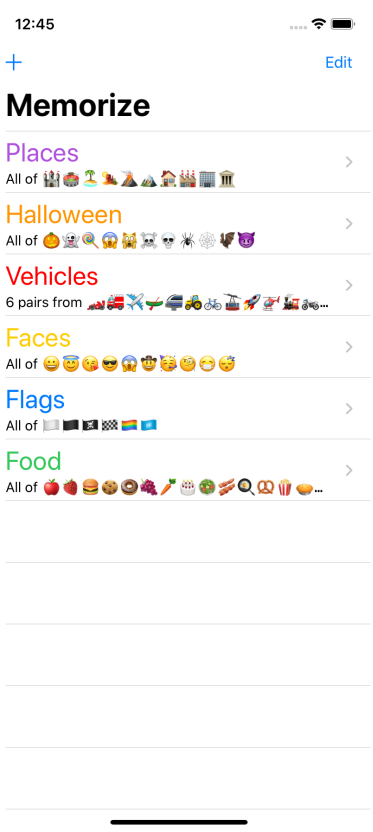
1. You'll likely want to start by creating a ViewModel for your chooser. This is just a store for your themes (even simpler than the one for EmojiArt since you don't support renaming in place, so you don't even need a names dictionary). A simple array of themes that you persist into `UserDefaults` as JSON (which you made possible via your Homework 5) is all you'll need.
2. Since this new ViewModel is only a store (i.e. it does not do any logic), there's no reason not to make its array of themes non-`private` (i.e. you don't need to cover every kind of change you could make to a theme with a `function` in this new ViewModel, just let the `View` manipulate the array of themes directly).
3. You'll definitely want to "autosave" any changes to your array of themes (which we did two different times in EmojiArt, so you should be familiar with how to use a `Publisher` to do this).
4. In EmojiArt's document chooser we just had a `Text` (and then an `EditableText`) representing the rows in the `List`. But there's absolutely no reason you can't create your own custom `View` to go there instead. In fact you will certainly want to do this.
5. You can make "add a theme" a lot easier by just creating some simple, default theme (similar to an "Untitled" document) with a few emoji in it and then the user can then simply edit it to their taste by bringing up the Palette Editor later. In other words, the add theme button does not have to bring up a modal panel of its own.
6. Don't make the code in your Theme Editor `View` gigantic. Break it down into smaller `Views` (maybe one for each `Section`, for example). Similarly, cleanly organize the code in your custom `View` that shows each row in the `List`.
7. There are two general strategies for a modal editor like your Theme Editor. The first is "live editing" and the second is "done/cancel" …
8. In "live editing," changes the user makes to the theme are immediately put in the store. If you use this approach (and this is generally preferred in iOS), you might pass your ViewModel (your store) along with the theme you want to edit to your Theme Editor (and think of the changes the user is making as "user intentions" and have Intent functions in your store to execute them). Or you could use a `Binding` directly into your array of themes in your ViewModel. This latter approach can be a bit tricky though, since you also support deleting themes from that array and so you might have circumstances where you delete a theme right out from some other `View`'s `Binding` (depending on whether you pass your theme `Binding` around to `Views`).
9. In a "done/cancel" style modal `View`, you'd have a Done button somewhere in your Theme Editor that dismisses the Theme Editor and copies all changes made to the theme while the Theme Editor was up back into the store. You'd then also have a Cancel button which would dismiss the Theme Editor *without* updating the store.

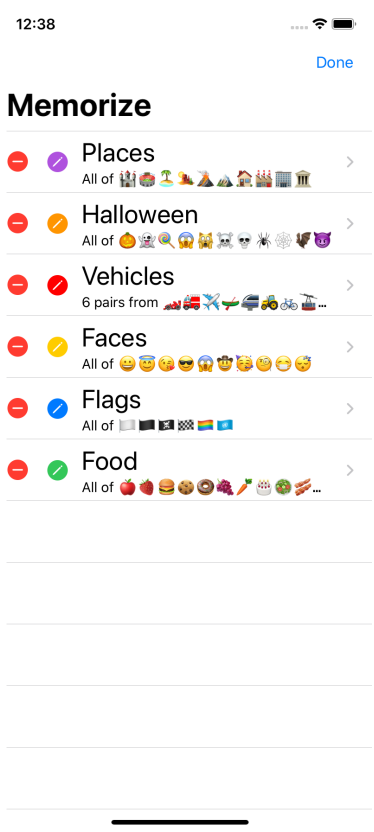
10. If you do end up doing "live editing," pass your ViewModel (the theme store) to your Theme Editor using `.environmentObject()` (i.e. don't pass it as an argument and use `@ObservedObject`). This is (currently) required when passing ViewModels to modally presented `View`s and, besides, the `View`s that make up your Theme Editor would all need it anyway so this way they can all just use `@EnvironmentObject` to get it.
11. Don't let your Theme Editor UI that chooses a number of pairs of cards to show in the theme choose a number that is more than the number of emoji available in the theme! Nor let it choose a single pair (that'd be too easy of a game to play).
12. You'll have to decide what to do if there are (or threaten to be) fewer than two emoji in the theme at any point during editing. There are multiple reasonable approaches to this situation.
13. The Required Tasks don't say anything about what sort of UI you have to employ to add or remove emoji from your theme in your Theme Editor. That's up to you to decide.

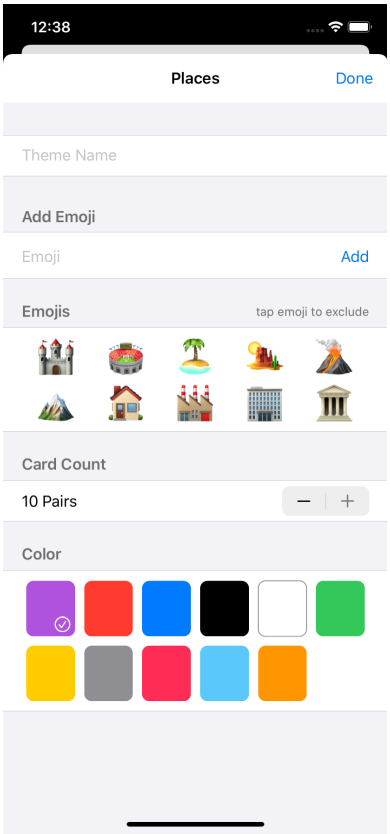
Screenshots

We hate including screen shots or videos of assignments. This is because we don't like biasing you towards one solution or another. However we realize that many of you have a difficult time "picturing" what is being asked of you from written descriptions. So we are making an exception for this assignment.

However, note carefully that this Screenshots section of the writeup is **not** a Required Task. The Required Tasks are included above, not here. These are purely "example" UIs to give you a *general* idea of the kind of thing you're being asked to do. You do **not** have to build your UIs to look like this (in fact, we'd love to see some other creative approaches)!







Things to Learn

Here is a partial list of concepts this assignment is intended to let you gain practice with or otherwise demonstrate your knowledge of.

1. `List`
2. `Form`
3. `NavigationView`
4. Modal presentation
5. `TextField`
6. `EditMode`
7. Multiple MVVMs
8. `Publisher`
9. `UserDefaults`

Evaluation

In all of the assignments this quarter, writing quality code that builds without warnings or errors, and then testing the resulting application and iterating until it functions properly is the goal.

Here are the most common reasons assignments are marked down:

- Project does not build.
- One or more items in the Required Tasks section was not satisfied.
- A fundamental concept was not understood.
- Project does not build without warnings.
- Code is visually sloppy and hard to read (e.g. indentation is not consistent, etc.).
- Your solution is difficult (or impossible) for someone reading the code to understand due to lack of comments, poor variable/method names, poor solution structure, long methods, etc.

Often students ask "how much commenting of my code do I need to do?" The answer is that your code must be easily and completely understandable by anyone reading it.

Extra Credit

We try to make Extra Credit be opportunities to expand on what you've learned this week. Attempting at least some of these each week is highly recommended to get the most out of this course. How much Extra Credit you earn depends on the scope of the item in question.

If you choose to tackle an Extra Credit item, mark it in your code with comments so your grader can find it.

1. Support choosing a theme's color in the Theme Editor.
2. Keep track of any emoji that a user removes from a theme as a "removed" or "not included" emoji. Then enhance your Theme Editor to allow them to put removed emoji back if they change their mind. Remember these removed emoji forever (i.e. you will have to add state to your theme `struct`).