5: Pokédex II

Due: Noon, 15th February

Description

In this assignment you will add features to your Pokedex app. You will again be graded on the visual aspects of your app, along with the functionality, style, and organization of your code. You will be penalized for not creating views that can be used in multiple places in your app.

- **Home view:**
  - The home view for your app will now display rows of Pokemon 'cards.' There should be a row for each `PokemonType.` Each row should be appropriately titled. Because Pokemon have multiple types, they will appear in multiple rows.
  - Each card should *somehow* indicate whether a Pokemon has been captured.
  - Tapping a card should navigate to that Pokemon's detail page.
  - If any Pokemon have been captured, they should be displayed in a row at the top. The row should be appropriately titled. If no Pokemon have been captured, do not display the row at all (it would be empty anyway).
  - Rows should be scrollable to ensure all Pokemon in a row can be displayed.
  - From the home view there should be an option to view a full list of all Pokemon.
- **Pokemon list:**
  - Each row in the list should *somehow* indicate whether a Pokemon has been captured
  - The user should be able to filter Pokemon in the list by `PokemonType,` e.g., filter the list so it only displays Pokemon of type Dragon. The same mechanism for filtering should also allow no filter to be applied so that all Pokemon are again in the list.
- **Pokemon detail screen:**
  - There should be a button which allows the user to mark the Pokemon as captured/released.
    - The user should know what action they are taking (capturing/releasing).
  - There should be sections displaying the Pokemon cards for the evolutionary predecessors (`prevEvolution`) and successors (`nextEvolution`) of the Pokemon.
    - Clicking on a Pokemon in either section should navigate to its detail page.
- **Persistence:**
  - The captured Pokemon should be saved when the user terminates the app.

Walk Through

I. Update your model. Your model needs to support the capture status of each Pokemon. You *must* store this information in your struct that represents a Pokemon. You will need to modify how you decode the json file because it does not contain a field for the capture status. You *may not* modify the json file. Support persistence via a file in your app's document directory. Save your data whenever your app enters the `.inactive` phase.

II. Create/Update Views

1. Create your home view. Break your design into single responsibility views that are small, general, and reusable. Implement all the functionality described above.
2. Add filtering to your list. To allow the user to *optionally* filter by PokemonType you will need to use a Picker with a binding to a state variable of type `PokemonType?` and include an option for None (no filtering). You will need to provide an explicit tag to each item in the picker with an appropriate value of type `PokemonType?`.
3. Update your detail view. Display evolution info in your detail view. The values for `prevEvolution` and `nextEvolution` are lists of IDs referencing other Pokemon. Display them here as Pokemon cards. Add a button that toggles the `captured` property of the Pokemon. The button should change appearance based on the value of `captured`.

Testing

1. Include appropriate catch clauses with your decoder to identify errors.
2. Test persistence by marking some Pokemon as captured and closing/reopening the app.
3. Check each of the requirements in the Description section above.

Hints

1. Remember that the model creates & maintains the data. The view model just interprets or provides this data. Avoid using SwiftUI in the view model.
2. You will need to have `PokemonType` conform to `CaseIterable` to use `.allCases`.
3. Create single-responsibility views to keep your code readable and reusable. For example, your Pokemon 'card' View can be used on the 'home' View and in your detail pages.
4. Placing a `Picker` directly in a `ToolbarItem` will allow it to adapt and give a nice visual when tapped. Use the picker style `MenuPickerStyle()`.
5. Be sure that your home view and all navigation destinations have a navigation bar title.

Submission

1. Remove all breakpoints, print statements and any other cruft.
2. Make sure your project compiles cleanly from scratch: choose **Clean** from the Product menu in Xcode, then Build and Run your program. Be sure to remove all warnings.
3. Check that all resources were copied into your project folder.
4. Ensure that you have merged your work back onto the master branch. **Commit and push your project**
5. Verify your project! If you are able, use the continuous integration process to verify your project on the server builds. Otherwise, try pulling your project from a different machine in an ITS lab. We will be deducting points for project that do not build correctly.