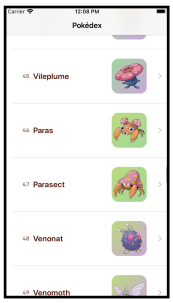
4: Pokédex I

Due: Noon, 8th February

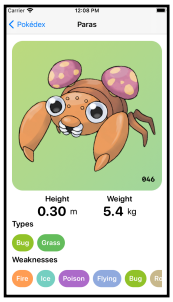
Description

In this assignment you will create a Pokedex app to explore Lists, Navigation, and decoding. Most of the coding involving functionality is fairly short and straightforward, so we will require special attention be given to the visual aspects of views. Developing appreciation for, and the ability to construct, appealing visuals is an important element that you'll need for your final project.



Your app should include the following functionality:

- The app should open to a list of Pokemon, where each row in the list displays the Pokemon's name, number, and a thumbnail-sized image of the Pokemon.
- Selecting a row should bring the user to a detail page, which should display (1) the pokemon's name; (2) the pokemon's image; (3) the pokemon's height and weight data; (4) the pokemon's types and weaknesses.
- Your app should display properly in both Light and Dark modes.



Your view's appearance should be almost identical to the samples provided here. **Part of your grade for this project will be based on the visual aspects of your app.**

Code for each of your views should be relatively short, reusable (via parameters) and have a single responsibility. Look for ways to reuse views in multiple places. **Part of your grade for this project will be based on how well your design and organize your views.**

Walk Through

I. Set up your project. Create a new project for this assignment.

1. Import the provided files via File > Add files to … (Or drag & drop)
   - `pokedex.json`
   - `PokemonType.swift`
   - `Color+PokemonType.swift`
2. Drag the images folder into the asset catalog (Assets.xcassets) below the AppIcon
3. Be sure to include an appropriate app icon

II. Create your model. Your model should include an array of all the Pokemon data, read in from the provided JSON file. To do this you'll need a new type to represent one Pokemon. Inspect the JSON to determine what properties you must include. Note that weaknesses are also Pokemon types. Use the `PokemonType` definition.

III. Create your view model. Because this initial version of the app is all about displaying the Pokemon data, your view model might be very simple.

IV. Create your views.

1. Use `List` to produce a scrollable list of Pokemon. Each row contains the number, name and image of a Pokemon. Put each image on a background that is a rounded rectangle filled with a linear gradient consisting of the colors associated with its types. (See Hints.)
2. The detail view should again display the image on a linear gradient as before. Height and weight should appear below the image, side by side, with an appropriate number of decimal places and units (kg and m) displayed. Types and weaknesses should be displayed on a background consisting of a `Capsule` filled with the color associated with the type. (Remember that weaknesses are just types too.) Using a vertical scroll view for the entire detail will ensure that all details are accessible. Using horizontal scroll views for types and weaknesses will ensure that these will always display well regardless of how many there are.

Testing

Testing of the application is imperative. Be sure that all the required features have been implemented.

1. Test your app in both Light and Dark mode to ensure that it displays properly in both.
2. The Paras Pokemon has a long list of weaknesses. Be sure that its detail view properly scrolls all its weaknesses.

Hints

Most assignments will have hints sprinkled throughout the assignment or in a separate section like this.

1. You can use `decoder.keyDecodingStrategy = .convertFromSnakeCase` to tell your decoder to look for the value of "`propertyName`" under the key "`property_name`" in your JSON
2. Use `String(format:, …)` to get numbers to render properly with leading 0's
3. The image name for a pokemon is the three-digit Pokemon's id, with leading 0's where necessary
4. Create single-responsibility views to keep your code readable and reusable. Avoid any duplicated code by extracting out code to their own type.
5. You might want to extend `LinearGradient` with a new initializer that takes a `Pokemon` as a parameter and creates a linear gradient from the colors associated with each Pokemon type. We've already provided you with an extension to `Color` to assist you. Initializers don't return a value so you'll need to assign `self` to the linear gradient you create inside the initializer.
6. Use `.navigationTitle` in your detail view to set the title of the detail page.
7. The `Font` type has a static method `system(size:weight:design:)` that provides an easy way to customize the standard system font. (I used the rounded design for the Pokemon's height & weight stats.)

8. In the simulator you can switch between Light and Dark appearance in the Settings app. Go to the Developer settings from the main settings page.
9. You can customize the Accent Color (default color for button text, navigation buttons, etc) by going to the `Assets` folder and editing `AccentColor`. Open the Inspector and choose the Attributes Inspector (⌥⌘4) to reveal editing options.

Troubleshooting

Assignments may also have a section for troubleshooting to call out common pitfalls or areas to watch out for.

1. Look in the console for decoding errors if your model is not working as intended. The error should tell you at which key and index the decoder failed.
2. Use Canvas previews while designing your views! Create a `standard` value of `Pokemon` for initializing views that take a Pokemon as a parameter.
3. Remember to enclose your ScrollView/List in a NavigationView and to apply any navigation modifiers to views inside the NavigationView.
4. Navigation destination views are not considered part of the Navigation view's hierarchy and therefore do not receive any environment objects from the Navigation view. You will need to explicitly add them via the `.environment` view modifier. (But ask yourself if you really need to make your entire view model accessible in the detail view.)

Submission

Prepare your project for 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.