LambdaSchool / **ios-projects**

Branch: **master** **ios-projects** / Sprint 9 / **Creating Frameworks** / Create new file | Upload files | Find file | History

armadsen Add Module 9.3 project Latest commit `914cdd0` on Sep 19, 2018

..		
IndeterminateLoadingView.swift	Add Module 9.3 project	5 months ago
LoadingUI.gif	Add Module 9.3 project	5 months ago
PolarPoint.swift	Add Module 9.3 project	5 months ago
README.md	Add Module 9.3 project	5 months ago

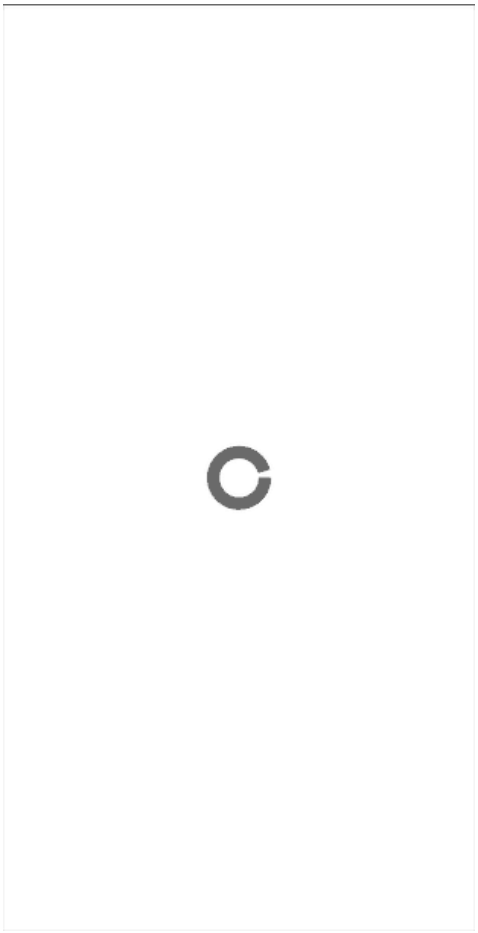
README.md

Loading UI Framework

The goal of this project is practice creating a framework, including thinking about the public API you want to expose. This project covers the concepts covered in the Creating Frameworks module. After completing the lesson material and this project, you should be able to:

- create a framework for sharing code between multiple apps
- include sub-projects with a framework in an Xcode app project
- use Swift's access control modifiers appropriately to make code private, internal, or public
- make good decisions about abstraction, API surface, and modularity when creating a framework

Your assignment is to write a framework called "Loading UI" or similar. At a minimum, the framework should include a view controller that can be presented to show a loading animation like the one in the animated GIF below:



You can expand on this by completing the tasks in the Go Farther section. As you work on this project, think about the API you want to expose to users of the framework. It should be as simple as possible to use the framework in an app for various scenarios.

Part 0 - Animation Help

You're encouraged to figure out how to implement the animation itself on your own using the concepts you learned in Sprint 6 - User Interface. However, an animation like this will take a little time to get right, and will stretch you beyond what you've already learned. This project is focused on creating a framework, and if you feel it would be a better use of your time to focus on that, you are welcome to use the `IndeterminateLoadingView` class in `IndeterminateLoadingView.swift` in this repo. If you decide to write the animation on your own, here are some hints:

- There are a number of ways to do this, but `CAShapeLayer` is a good one. Note the `strokeStart` and `strokeEnd` properties which are animatable.
- You can create a circular `UIBezierPath` using `UIBezierPath(ovalIn: rect)` where `rect` is a square `CGRect`.
- `CAAnimation` has a delegate property that allows you to be notified when an animation finishes. You can use this to trigger another animation.
- You may find the `PolarPoint` struct found in this repo helpful. It makes circular geometry and drawing code easier. Note that I didn't use it in my solution to this project, but depending on the approach to drawing you take, you might find it useful.

Part 1 - Create a Framework and Setup a Test App

Start by creating a new framework project. You should also set up a test app so you can excersise, test, and debug your framework code.

1. Create a parent directory for this project. Call it e.g. `LoadingUIFrameworkDevelopment`
2. Inside `LoadingUIFrameworkDevelopment`, create a new Cocoa Touch Framework project called `LoadingUI` (or whatever name you'd prefer).
3. Inside `LoadingUIFrameworkDevelopment`, create another app project called `LoadingUITest` (or similar). The folder for `LoadingUITest` should be a sibling to the `LoadingUI` folder.
4. Open `LoadingUITest`, and add the `LoadingUI` framework project as a subproject.
5. Add the LoadingUI.framework as an embedded binary in the target settings for `LoadingUITest`.
6. `import LoadingUI` in `ViewController`. Make sure the project builds.

Part 2 - Define the Interface

When creating a new API component for a framework, or even in app code, it's helpful to start by figuring out the external interface you want to expose *before* you start implementing the API's functionality.

1. Create a subclass of `UIViewController` called `LoadingViewController`. Should this be public, internal, or private?
2. Add stubs for the `public` methods you want to expose on `LoadingViewController`. These need not be final. You can always refine the interface as you go, but you want to start with something to guide you
3. Are there any other types, functions, or methods you should expose publicly in your framework?
4. Write code in your app (`LoadingUITest`) to use `LoadingViewController`. How you do this is up to you, but you might make it so that a button on the view controller modally presents a `LoadingViewController` and starts animating it. Your goal here is to allow you to test your framework code, but also to get an idea of how the API you're creating feels in practice.

Part 3 - Implement `LoadingViewController`

Fill out the stub methods you created for `LoadingViewController`. Mark any methods and properties that shouldn't be exposed to users of the framework using either `private` or `internal` as appropriate. As you develop this code, build and run your `LoadingUITest` app to test and debug it.

Go Farther

If you finish early or want to challenge yourself, think about completing one or more of the following tasks:

- Add the ability for the loading animation to show *determinate* progress. That is, instead of spinning until you tell it to stop, make it so the circle can be filled in from 0-100% as the progress of a task completes.
- Building on the last task, add support for `Progress`, which is a Foundation class for doing progress reporting.
- Implement a more complicated loading animation. The sky's the limit. Any relatively simple, interesting, repeating animation can be used. Take a look at `CAReplicatorLayer` which can do some very interesting things.
- Add support for installing your framework using Carthage. Hint: This is likely easier than you think.
- Add support for installing your framework using CocoaPods. Hint: This is likely harder than you think 😉.