iOS Interview questions and answers Part18

Sandeep Reddy Challa · Follow
7 min read · Jun 5

196.What steps do you take to identify and resolve a memory leak?

Suggested approach: Hopefully you have at least some experience with Instruments, so talk about persistent vs transient objects, talk about filtering for your custom data types, and so on. You should also discuss how you can be sure the leak is gone, for example if you push and pop the same view controller 10 times does the memory level remain constant?

197.What steps do you take to identify and resolve performance issues?

Suggested approach: This is a tricky question to answer because "performance" has many forms, so be prepared to adjust your answer as you talk based on interviewer feedback.

If they mean graphical performance then you should probably talk about using the Core Animation instrument to identify slow drawing, but you should also consider venturing into the cost of Auto Layout in things like table view cells where lots of work happens quickly.

If they mean code performance then perhaps talk about stack traces, retain cycles, unnecessary caches, and similar — again make sure and bring Instruments in.

And then there's network performance, where you might talk about things such as batching requests to reduce battery wastage or using compression to save bandwidth.

198.How much experience do you have using Face ID or Touch ID? Can you give examples?

Suggested approach: If you're applying for a job at any company that has secure user data, biometric authentication is almost certainly involved somewhere. Fortunately, it's not hard to learn!

For bonus points mention the need for a password backup in case Face ID/Touch ID fails, but if you're generally stuck here a good approach might be just to say "I haven't used it before, but I have used the keychain for secure storage."

199.How would you explain App Transport Security to a new iOS developer?

Suggested approach: This is your chance to demonstrate your security knowledge: why is HTTPS so important, and in what specific cases might you need to opt out? It also an opportunity to demonstrate your awareness of Apple's app review guidelines, which require secure transmission of user data.

200.How would you calculate the secure hash value for some data?

Suggested approach: Secure hash values use something like SHA-3, which is not the kind of code you'd want to write yourself. Instead, the best approach here is to mention something like Apple's CryptoKit framework, which can do hashing and encryption quickly, efficiently, and correctly.

201.In which situations do Swift functions not need a return keyword?

Suggested approach: There are three: when the function isn't supposed to return a value, when it is supposed to return a value but you've used something like fatalError() to skip that requirement, and when it returns a value using a single expression. That second case is useful when you have placeholder functions you haven't implemented yet, or have created an abstract class where child classes will override your erroring implementations.

202.Apart from the built-in ones, can you give an example of property wrappers?

A wrapper to make sure numbers are never negative, or strings are never empty, or perhaps arrays that silently stay sorted.

1. Use @State for simple properties that belong to a single view. They should usually be marked private.
2. Use @ObservedObject for complex properties that might belong to several views. Most times you're using a reference type you should be using @ObservedObject for it.
3. Use @StateObject once for each observable object you use, in whichever part of your code is responsible for creating it.
4. Use @EnvironmentObject for properties that were created elsewhere in the app, such as shared data.
5. Use @Binding lets us declare that one value actually comes from elsewhere, and should be shared in both places. This is not the same as @ObservedObject or @EnvironmentObject, both of which are designed for reference types to be shared across potentially many views.This is exactly what @Binding is for: it lets us create a property in the add user view that says "this value will be provided from elsewhere, and will be shared between us and that other place."

Of the four you will find that @ObservedObject is both the most useful and the most commonly used, so if you're not sure which to use start there.

203.Can you give useful examples of enum associated values?

Suggested approach: Enum associated values let us attach one or more extra pieces of data to enum cases — that much is easy enough. However, the key word here is "useful", which means you need to provide an example that is even vaguely real world.

For instance, you might describe a weather enum that lists sunny, windy, and rainy as cases, but has an associated value for cloudy so that you can store the cloud coverage. Or you might describe types of houses, with the number of bedrooms being an associated integer.

It doesn't really matter what example you choose, because the point is to show you understand why they are useful outside of a textbook!

204.What are opaque return types?

Suggested approach: Whenever you see some in a return type, it's an opaque return type — when you want to specify that some kind of type will be returned, but you don't want to say what.

It's important that you try to explain the difference between an opaque return type and returning a protocol, because in the latter your returned value can be absolutely anything whereas in the former the compiler knows what data was actually returned even if you don't get access to that.

For bonus points, talk about how SwiftUI uses @ViewBuilder to silently allow us to return different view types from a view body.

205.How would you explain SwiftUI's environment to a new developer?

Suggested approach: I would suggest you start off nice and broad, and say that the environment acts a bit like a singleton manager — you place objects in there and share them in many places. But then you want to dive into the details a little more, perhaps saying that actually you can subdivide the environment if you want, allowing some views to have different environment objects.

I would recommend you try to mention how the environment differs from just injecting an ObservableObject instance in an initializer.

206.What does the @State, @Published property wrapper do?

This creates a property inside a view, but it uses the @State property wrapper to ask SwiftUI to manage the memory. This matters: all our views are structs, which means they can't be changed, and if we weren't even able to modify an integer in our apps then there wouldn't be much we could do.

So, when we say @State to make a property, we hand control over it to SwiftUI so that it remains persistent in memory for as long as the view exists. When that state changes, SwiftUI knows to automatically reload the view with the latest changes so it can reflect its new information.

@State is great for simple properties that belong to a specific view and never get used outside that view, so as a result it's important to mark those properties as being private to re-enforce the idea that such state is specifically designed never to escape its view.

When used inside an ObservableObject an @Published property will automatically send out change notifications when its value changes.

@Published is one of the most useful property wrappers in SwiftUI, allowing us to create observable objects that automatically announce when changes occur. SwiftUI will automatically monitor for such changes, and re-invoke the body property of any views that rely on the data.

In practical terms, that means whenever an object with a property marked @Published is changed, all views using that object will be reloaded to reflect those changes.

207.When would you use @StateObject versus @ObservedObject?

Both of these property wrappers monitor an observable object for changes, and refresh SwiftUI views when changes happen. However, @StateObject is used when you create an object for the first time and want to retain ownership of it, whereas @ObservableObject is used in other places where you pass the object and does not retain ownership.

There is one important difference between @StateObject and @ObservedObject, which is ownership — which view created the object, and which view is just watching it.

The rule is this: whichever view is the first to create your object must use @StateObject, to tell SwiftUI it is the owner of the data and is responsible for keeping it alive. All other views must use @ObservedObject, to tell SwiftUI they want to watch the object for changes but don't own it directly.

208.What's the difference between a view's initializer and onAppear()?

Suggested approach: Using init() and onAppear() both let us run some code early in a view's lifecycle, however it's important to understand the difference between them.

SwiftUI creates all its view structs immediately, even creating destination views for navigation links, which means that initializers are run immediately and that's probably not something you want. In comparison, code placed in an onAppear() modifier is called only when the view is shown for the first time, so it's the right place to do complex work.

209.How can an observable object announce changes to SwiftUI?

Suggested approach: There are two primary ways this is done: using the @Published property wrapper, or by calling objectWillChange.send() directly.

Try to provide examples of when one is preferable over the other, such as saying that you might use @Published by default, switching over to objectWillChange.send() for times when you need more fine-grained control.

210.How would you create programmatic navigation in SwiftUI?

Suggested approach: SwiftUI makes simple navigation as easy as it should be, but programmatic navigation is trickier because you need to declare all your states up front.

If you want to talk about tags for NavigationLink views you can, but I would say the important thing here is to think about why it's important — handling deep links from Spotlight or widgets are both good places where you need to navigate programmatically.

IOS App Development · IOS · Swiftui · Swift · Xcode

Written by Sandeep Reddy Challa
Follow
171 Followers
I have around 14 years of experience in iOS development and lead/senior developer