iOS Engineer Interview

SwiftUI

1. What is SwiftUI and how does it differ from UIKit?
   - SwiftUI is Apple's modern framework for building user interfaces, offering a declarative syntax compared to UIKit's imperative approach. It simplifies UI creation and updates.
2. Explain the concept of declarative UI in SwiftUI.
   - Declarative UI describes the desired outcome, not the steps to achieve it. SwiftUI builds and updates the UI based on the declared state.
3. How do you create a custom view in SwiftUI?
   - Create a new struct conforming to the `View` protocol and define its content within a `body` property.
4. What are the benefits of using SwiftUI over UIKit?
   - Benefits include declarative syntax, easier state management, and unified interface for macOS, iOS, and other Apple platforms.
5. How do you handle state management in SwiftUI?
   - Use `@State` for local state, `@ObservedObject` for observable classes, and `@EnvironmentObject` for global state.
6. Explain the difference between `@State` and `@Binding`.
   - `@State` is used for local state management, while `@Binding` is used to share state between views.
7. How do you use `@EnvironmentObject` in SwiftUI?
   - `@EnvironmentObject` is used to access an object that is passed down through the view hierarchy.
8. What is the purpose of `@ObservedObject` and `@StateObject`?
   - `@ObservedObject` observes changes in an object, while `@StateObject` manages the lifecycle of an object.
9. How do you handle view animations in SwiftUI?
   - Use animation modifiers like `.animation()` or `withAnimation {}` to animate UI changes.
10. What is the difference between `ViewBuilder` and `@ViewBuilder`?
    - `ViewBuilder` is a protocol for building views, while `@ViewBuilder` is a property wrapper for functions returning views.

CocoaPods and Dependencies

11. What is CocoaPods and how is it used in iOS development?
    - CocoaPods is a dependency manager for Swift and Objective-C Cocoa projects, simplifying library integration.
12. How do you install CocoaPods?
    - Install via Ruby gem: `sudo gem install cocoapods`.
13. What is a Podfile and how do you configure it?
    - A Podfile lists project dependencies. Configure by specifying pods and their versions.
14. How do you add a dependency to your project using CocoaPods?
    - Add the pod to the Podfile and run `pod install`.
15. What is the difference between `pod install` and `pod update`?
    - `pod install` installs dependencies as specified, while `pod update` updates to the latest versions.
16. How do you resolve conflicts between different pods?
    - Use pod versions that are compatible or specify versions in the Podfile.
17. What is Carthage and how does it differ from CocoaPods?
    - Carthage is another dependency manager that builds and links libraries without integrating deeply into the project.
18. How do you manage different pods for different build configurations?
    - Use conditional statements in the Podfile based on build configurations.
19. What is a podspec file and how is it used?
    - A podspec file describes a pod's version, source, dependencies, and other metadata.
20. How do you troubleshoot issues with CocoaPods?
    - Check pod versions, clean the project, and consult the CocoaPods issue tracker.

UI Layout

21. How do you create a responsive layout in iOS?
    - Use Auto Layout and constraints to make views adapt to different screen sizes.
22. Explain the difference between `Stack View` and `Auto Layout`.
    - Stack Views simplify laying out views in a row or column, while Auto Layout provides precise control over positioning.
23. How do you use `UIStackView` in iOS?

- Add views to a Stack View and configure its axis, distribution, and alignment.

24. What is the difference between `frame` and `bounds` in iOS?
    - `frame` defines the view's position and size relative to its superview, while `bounds` defines the view's own coordinate system.
25. How do you handle different screen sizes and orientations in iOS?
    - Use Auto Layout and size classes to adapt the UI to various devices and orientations.
26. Explain how to use `Auto Layout` constraints in iOS.
    - Set constraints between views to define their relationships and positions.
27. What is the difference between `leading` and `trailing` in Auto Layout?
    - Leading and trailing adapt to text direction, while left and right do not.
28. How do you create a custom layout in iOS?
    - Subclass `UIView` and override `layoutSubviews()` to position subviews manually.
29. Explain how to use `UIPinchGestureRecognizer` and `UIRotationGestureRecognizer`.
    - Attach gesture recognizers to views and handle their actions in delegate methods.
30. How do you handle layout changes for different device types (iPhone, iPad)?
    - Use size classes and adaptive layouts to adjust the UI for different devices.

Swift

31. What are the key differences between Swift and Objective-C?
    - Swift is safer, more concise, and supports modern language features like closures and generics.
32. Explain the concept of optionals in Swift.
    - Optionals represent values that can be `nil`, indicating the absence of a value.
33. What is the difference between `nil` and `optional`?
    - `nil` is the absence of a value, while an optional can either hold a value or be `nil`.
34. How do you handle errors in Swift?
    - Use `do-catch` blocks or propagate errors using `throw`.
35. Explain the difference between `let` and `var`.
    - `let` declares constants, while `var` declares variables that can be modified.
36. What is the difference between a class and a struct in Swift?
    - Classes support inheritance and are reference types, while structs are value types.
37. How do you create an enum in Swift?

- Define an enum with `enum` keyword and cases, which can have associated values.

38. Explain the concept of protocol-oriented programming in Swift.
    - Protocols define methods, properties, and requirements that conforming types must implement.
39. What is the difference between a protocol and a delegate?
    - Protocols define methods, while delegates implement protocol methods for specific interactions.
40. How do you use generics in Swift?
    - Use generic types to write flexible, reusable code that works with any data type.

Networking

41. How do you handle network requests in iOS?
    - Use URLSession for network tasks, or libraries like Alamofire for higher-level abstractions.
42. What is URLSession?
    - URLSession handles network requests, providing data tasks, upload tasks, and download tasks.
43. How do you handle JSON parsing in Swift?
    - Use `Codable` protocol to decode JSON data into Swift structs or classes.
44. Explain the difference between synchronous and asynchronous requests.
    - Synchronous requests block the calling thread, while asynchronous requests do not.
45. How do you manage network requests in a background thread?
    - Use GCD or OperationQueue to perform requests off the main thread.
46. What is Alamofire and how does it differ from URLSession?
    - Alamofire is a third-party networking library that simplifies HTTP requests compared to URLSession.
47. How do you handle network errors and retries?
    - Implement error handling in completion handlers and consider retry mechanisms for transient errors.
48. Explain how to use `URLSessionDataDelegate` methods.
    - Implement delegate methods to handle request progress, authentication, and more.
49. What is the difference between GET and POST requests?
    - GET retrieves data, while POST sends data to a server to create or update resources.
50. How do you secure network communications?
    - Use HTTPS to encrypt data in transit and handle certificates properly.

Best Practices and Problem Solving

51. How do you ensure code quality in your projects?
    - Use linting tools, write unit tests, and follow coding standards.
52. Explain how you would debug a SwiftUI view.
    - Use Xcode's debugging tools, preview canvas, and print statements to identify issues.
53. What strategies do you use for optimizing app performance?
    - Profile the app using Instruments, optimize data fetching, and reduce UI layer counts.
54. How do you handle memory management in Swift?
    - Use ARC (Automatic Reference Counting) and avoid retain cycles.
55. Explain how you would approach refactoring legacy code.
    - Identify code smells, write tests, and refactor incrementally.
56. What is your experience with CI/CD pipelines?
    - Set up pipelines using tools like Jenkins, GitHub Actions, or Fastlane for automated builds and deployments.
57. How do you stay updated with the latest iOS developments?
    - Follow Apple's developer resources, attend conferences, and participate in developer communities.
58. Explain a time you solved a difficult bug in your project.
    - Describe the process of identifying, isolating, and fixing the issue.
59. What is your approach to version control?
    - Use Git for branching, committing, and collaborating effectively.
60. How do you handle deadlines and pressure in a project?
    - Prioritize tasks, communicate effectively, and manage time efficiently.