# Ingeniero de iOS Entrevista

### SwiftUI

1. ¿Qué es SwiftUI y cómo se diferencia de UIKit?
    - SwiftUI es el marco moderno de Apple para construir interfaces de usuario, ofreciendo una sintaxis declarativa en comparación con el enfoque imperativo de UIKit. Simplifica la creación y actualización de la interfaz de usuario.
2. Explique el concepto de interfaz de usuario declarativa en SwiftUI.
    - La interfaz de usuario declarativa describe el resultado deseado, no los pasos para lograrlo. SwiftUI construye y actualiza la interfaz de usuario en función del estado declarado.
3. ¿Cómo se crea una vista personalizada en SwiftUI?
    - Cree una nueva estructura que cumpla con el protocolo `View` y defina su contenido dentro de una propiedad `body`.
4. ¿Cuáles son los beneficios de usar SwiftUI sobre UIKit?
    - Los beneficios incluyen sintaxis declarativa, gestión de estado más fácil y una interfaz unificada para macOS, iOS y otras plataformas de Apple.
5. ¿Cómo se maneja la gestión de estado en SwiftUI?
    - Use `@State` para el estado local, `@ObservedObject` para clases observables y `@EnvironmentObject` para el estado global.
6. Explique la diferencia entre `@State` y `@Binding`.
    - `@State` se usa para la gestión de estado local, mientras que `@Binding` se usa para compartir estado entre vistas.
7. ¿Cómo se usa `@EnvironmentObject` en SwiftUI?
    - `@EnvironmentObject` se usa para acceder a un objeto que se pasa a través de la jerarquía de vistas.
8. ¿Cuál es el propósito de `@ObservedObject` y `@StateObject`?
    - `@ObservedObject` observa cambios en un objeto, mientras que `@StateObject` maneja el ciclo de vida de un objeto.
9. ¿Cómo se manejan las animaciones de vistas en SwiftUI?
    - Use modificadores de animación como `.animation()` o `withAnimation {}` para animar cambios en la interfaz de usuario.
10. ¿Cuál es la diferencia entre `ViewBuilder` y `@ViewBuilder`?
    - `ViewBuilder` es un protocolo para construir vistas, mientras que `@ViewBuilder` es un envoltorio de propiedad para funciones que devuelven vistas.

**CocoaPods y Dependencias**

11. ¿Qué es CocoaPods y cómo se usa en el desarrollo de iOS?
    - CocoaPods es un gestor de dependencias para proyectos Cocoa de Swift y Objective-C, simplificando la integración de bibliotecas.
12. ¿Cómo se instala CocoaPods?
    - Instale a través de la gema de Ruby: `sudo gem install cocoapods`.
13. ¿Qué es un archivo Podfile y cómo se configura?
    - Un archivo Podfile enumera las dependencias del proyecto. Configure especificando pods y sus versiones.
14. ¿Cómo se agrega una dependencia a su proyecto usando CocoaPods?
    - Agregue el pod al archivo Podfile y ejecute `pod install`.
15. ¿Cuál es la diferencia entre `pod install` y `pod update`?
    - `pod install` instala dependencias según lo especificado, mientras que `pod update` actualiza a las versiones más recientes.
16. ¿Cómo se resuelven los conflictos entre diferentes pods?
    - Use versiones de pods que sean compatibles o especifique versiones en el archivo Podfile.
17. ¿Qué es Carthage y cómo se diferencia de CocoaPods?
    - Carthage es otro gestor de dependencias que construye y enlaza bibliotecas sin integrarse profundamente en el proyecto.
18. ¿Cómo se gestionan diferentes pods para diferentes configuraciones de compilación?
    - Use declaraciones condicionales en el archivo Podfile según las configuraciones de compilación.
19. ¿Qué es un archivo podspec y cómo se usa?
    - Un archivo podspec describe la versión, fuente, dependencias y otros metadatos de un pod.
20. ¿Cómo se solucionan problemas con CocoaPods?
    - Verifique las versiones de los pods, limpie el proyecto y consulte el rastreador de problemas de CocoaPods.

**Diseño de la Interfaz de Usuario**

21. ¿Cómo se crea un diseño adaptable en iOS?
    - Use Auto Layout y restricciones para que las vistas se adapten a diferentes tamaños de pantalla.
22. Explique la diferencia entre `Stack View` y `Auto Layout`.

- Las vistas apiladas simplifican la disposición de vistas en una fila o columna, mientras que Auto Layout proporciona un control preciso sobre la posición.

23. ¿Cómo se usa `UIStackView` en iOS?
    - Agregue vistas a una vista apilada y configure su eje, distribución y alineación.
24. ¿Cuál es la diferencia entre `frame` y `bounds` en iOS?
    - `frame` define la posición y el tamaño de la vista en relación con su supervista, mientras que `bounds` define el propio sistema de coordenadas de la vista.
25. ¿Cómo se manejan diferentes tamaños de pantalla y orientaciones en iOS?
    - Use Auto Layout y clases de tamaño para adaptar la interfaz de usuario a varios dispositivos y orientaciones.
26. Explique cómo usar restricciones de `Auto Layout` en iOS.
    - Establezca restricciones entre vistas para definir sus relaciones y posiciones.
27. ¿Cuál es la diferencia entre `leading` y `trailing` en Auto Layout?
    - Leading y trailing se adaptan a la dirección del texto, mientras que left y right no lo hacen.
28. ¿Cómo se crea un diseño personalizado en iOS?
    - Subclase `UIView` y anule `layoutSubviews()` para posicionar subvistas manualmente.
29. Explique cómo usar `UIPinchGestureRecognizer` y `UIRotationGestureRecognizer`.
    - Adjunte reconocedores de gestos a vistas y maneje sus acciones en métodos delegados.
30. ¿Cómo se manejan los cambios de diseño para diferentes tipos de dispositivos (iPhone, iPad)?
    - Use clases de tamaño y diseños adaptables para ajustar la interfaz de usuario para diferentes dispositivos.

**Swift**

31. ¿Cuáles son las diferencias clave entre Swift y Objective-C?
    - Swift es más seguro, más conciso y admite características de lenguaje moderno como closures y generics.
32. Explique el concepto de opcionales en Swift.
    - Los opcionales representan valores que pueden ser `nil`, indicando la ausencia de un valor.
33. ¿Cuál es la diferencia entre `nil` y `optional`?
    - `nil` es la ausencia de un valor, mientras que un opcional puede contener un valor o ser `nil`.
34. ¿Cómo se manejan los errores en Swift?
    - Use bloques `do-catch` o propague errores usando `throw`.

35. Explique la diferencia entre `let` y `var`.
    - `let` declara constantes, mientras que `var` declara variables que se pueden modificar.
36. ¿Cuál es la diferencia entre una clase y una estructura en Swift?
    - Las clases admiten herencia y son tipos de referencia, mientras que las estructuras son tipos de valor.
37. ¿Cómo se crea un enum en Swift?
    - Defina un enum con la palabra clave `enum` y casos, que pueden tener valores asociados.
38. Explique el concepto de programación orientada a protocolos en Swift.
    - Los protocolos definen métodos, propiedades y requisitos que los tipos conformes deben implementar.
39. ¿Cuál es la diferencia entre un protocolo y un delegado?
    - Los protocolos definen métodos, mientras que los delegados implementan métodos de protocolo para interacciones específicas.
40. ¿Cómo se usan los genéricos en Swift?
    - Use tipos genéricos para escribir código flexible y reutilizable que funcione con cualquier tipo de datos.

**Redes**

41. ¿Cómo se manejan las solicitudes de red en iOS?
    - Use URLSession para tareas de red, o bibliotecas como Alamofire para abstracciones de nivel superior.
42. ¿Qué es URLSession?
    - URLSession maneja solicitudes de red, proporcionando tareas de datos, tareas de carga y tareas de descarga.
43. ¿Cómo se maneja el análisis JSON en Swift?
    - Use el protocolo `Codable` para decodificar datos JSON en estructuras o clases de Swift.
44. Explique la diferencia entre solicitudes sincrónicas y asincrónicas.
    - Las solicitudes sincrónicas bloquean el hilo de llamada, mientras que las solicitudes asincrónicas no lo hacen.
45. ¿Cómo se gestionan las solicitudes de red en un subproceso de fondo?
    - Use GCD o OperationQueue para realizar solicitudes fuera del hilo principal.
46. ¿Qué es Alamofire y cómo se diferencia de URLSession?

- Alamofire es una biblioteca de red de terceros que simplifica las solicitudes HTTP en comparación con URLSession.

47. ¿Cómo se manejan los errores de red y las repeticiones?
    - Implemente el manejo de errores en los controladores de finalización y considere mecanismos de repetición para errores transitorios.
48. Explique cómo usar métodos de `URLSessionDataDelegate`.
    - Implemente métodos delegados para manejar el progreso de la solicitud, la autenticación y más.
49. ¿Cuál es la diferencia entre solicitudes GET y POST?
    - GET recupera datos, mientras que POST envía datos a un servidor para crear o actualizar recursos.
50. ¿Cómo se aseguran las comunicaciones de red?
    - Use HTTPS para cifrar datos en tránsito y maneje certificados adecuadamente.

**Mejores Prácticas y Solución de Problemas**

51. ¿Cómo asegura la calidad del código en sus proyectos?
    - Use herramientas de linting, escriba pruebas unitarias y siga estándares de codificación.
52. Explique cómo depuraría una vista de SwiftUI.
    - Use las herramientas de depuración de Xcode, el lienzo de vista previa y las declaraciones de impresión para identificar problemas.
53. ¿Qué estrategias usa para optimizar el rendimiento de la aplicación?
    - Perfile la aplicación usando Instruments, optimice la recuperación de datos y reduzca las capas de la interfaz de usuario.
54. ¿Cómo maneja la gestión de memoria en Swift?
    - Use ARC (Automatic Reference Counting) y evite ciclos de retención.
55. Explique cómo abordaría la refactorización de código legado.
    - Identifique olores de código, escriba pruebas y refactorice de manera incremental.
56. ¿Cuál es su experiencia con pipelines CI/CD?
    - Configure pipelines usando herramientas como Jenkins, GitHub Actions o Fastlane para compilaciones y despliegues automatizados.
57. ¿Cómo se mantiene actualizado con los últimos desarrollos de iOS?
    - Siga los recursos de desarrolladores de Apple, asista a conferencias y participe en comunidades de desarrolladores.
58. Explique una vez que resolvió un error difícil en su proyecto.

- Describa el proceso de identificar, aislar y corregir el problema.

59. ¿Cuál es su enfoque hacia el control de versiones?
    - Use Git para ramificación, compromiso y colaboración efectiva.
60. ¿Cómo maneja plazos y presión en un proyecto?
    - Priorice tareas, comuníquese de manera efectiva y maneje el tiempo de manera eficiente.