APPS

Methodologies Scrum, TDD, BDD

Scrum

Vision, Requerimientos (Product Backlog), planificacion inicial, Desarrollo (spring), Revision diaria (Daily)

Revision de spring (Spring review), revision retrospectiva (sprint retrospectiva)

TDD ((Test Driven Development, Desarrollo guiado por pruebas)

Proceso TDD

Test- first: Las pruebas son primero antes del. Código.

Acciones

- Se verifica que el test falla
- Se escribe el código para pasar el test
- Se verifica que el test pasa
- Se refctoriza el código para eliminar la duplicación

Repetir el proceso hasta completar los tests

BDD (Desarrollo guiado por comportamiento)

Dado: los pasos necesarios para poner al sistema en el estado que se desea probar

Cuando: La interacción del usuario que acciona la funcionalidad que deseamos testear.

Entonces: En este paso vamos a observar los cambios en el sistema y ver si son los deseados.

Diseño Human Interface Guide

Pruebas Stubs, Mocks, XCTest

Programacion Mejores practicas

SOLID

S (SRP) – Principio de responsabilidad única (Single responsibility principle)

este principio dice que un objeto/clase debería únicamente tener una responsabilidad completamente encapsulada por la clase.

O (OCP) – Principio de abierto/cerrado (Open/closed principle)

“Entidades de Software (classes, módulos, funciones, etc.) han de

estar abiertas para extensiones, pero cerradas para modificaciones”
Aquí la idea es que una entidad permite que comportamiento se extienda pero nunca modificando el código de fuente. Cualquier clase (o cualquier cosa que escribas) debe de estar escrito de un manera que puede utilizarse por lo que es. Puede ser extensible, si se necesita, pero nunca modificado.

L (LSP) – Principio de sustitución de Liskov (Liskov substitution principle)
“Subtypes deben ser sustituidos por su tipo de base”
La idea aquí es que los objetos deberían ser reemplazados por ejemplos de su subtipo, y ello sin que la funcionalidad del sistema desde el punto de vista de los clientes se vea afectada. Básicamente, en vez de utilizar la implementación actual, deberías ser capaz de utilizar una clase base y obtener el resultado esperado. A menudo, cuando queremos representar un objeto, modulamos nuestra clase de bases en sus propiedades y en vez de eso, deberíamos poner más énfasis en su comportamiento.

I (ISP) – Principio de segregación de la interfaz (Interface segregation principle)
“Classes implementan interfaces, no deberían ser forzadas a implementar los métodos no utilizados”
Aquí, se basa en cómo escribir interfaces. Entonces, qué significa? Básicamente, una vez la interfaz se convierte en larga, se necesita absolutamente de separarla en pequeñas partes más específicas. Una interfaz será definida por el cliente que lo utilice, lo que significa que el sera el unico que tenga conocimientos de los métodos relacionados con ellos.

D (DIP) – Principio de inversión de la dependencia (Dependency inversion principle)
“Módulos de altos modelos no deberían depender de niveles de modulos bajos, sino de abstracciones. Abstracciones no deberían depender de detalles; sino los detalles de la abstracción”
Este principio está principalmente basado en reducir las dependencias entre los módulos del código. Básicamente, este principio será de gran ayuda para entender cómo atar correctamente sistemas juntos.

Use sangria consistente
Siga el principio DRY (Don’t repeat yourself) evite repetir códigos
Evite animaciones profundas
Limite la longitud de la linea
Estructure en archivos y carpetas
Convenciones y nomenglaturas
Mantenga código simple

Para pruebas
FIRST

F(Fast)	Rapido
I (Isolated)	Aislado independiente
R (Repetible)	Repetible
S (self-validating)	autoevaludo
T (Timely)	Oportuno

Uso de patrones de arquitectura

- MVC (Modelo, vista, controlador)
- MVP (Modelo, vista, Presentador)
- MVVM (Modelo Vista Vista Modelo)
- VIPER (vista, interactor, presentador, entidad, enrutamiento)

Para swift programación orientada a protocolos, No empieces con una clase, comienza con un protocolo

Colaboracion ,JIRA Zeplin
Control de versiones, Github, Bitbucket
SaaS (Software como servicios), Firebase, Google, AWS Movil hub, Amplify

Conceptos Básicos

Cocoa

Cocoa es un entorno de aplicación tanto para el sistema operativo

Mac OS X como para iOS,
el sistema operativo utilizado en dispositivos multitáctiles como iPhone, iPad y iPod touch.
Consiste en un conjunto de bibliotecas de software orientadas a objetos, un tiempo de ejecución y un entorno de desarrollo integrado.

Foundation y UIKit cocoa touch

Estructuras (Structs)

Las estructuras son tipos de valor. Esto significa que se asigna una copia de su valor cada vez que se pasa a variable, constante o función.
Todos los tipos básicos o primitivos en Swift son estructuras. Esto incluye enteros, dobles, flotadores y cuerdas.

Clases (Class)

Las clases son tipos de referencia. Cuando pasa una instancia a una variable o constante, se asigna una referencia a la misma instancia.
Es posible que múltiples constantes y variables hagan referencia a la misma instancia de una clase. Esto se conoce como recuento de referencias.

Structs vs Classes

Use una estructura cuando:

Si tiene la intención de encapsular algunos tipos de datos relativamente simples, entonces debe usar estructuras.
Si tiene la intención de copiar los valores en lugar de hacer referencia a ellos, entonces debe usar estructuras.
Si las propiedades almacenadas por las estructuras son en sí mismas tipos de valor.
La estructura no necesita heredar propiedades o comportamientos de un tipo existent

Enumeración (enum)

Una enumeración define un tipo común para un grupo de valores relacionados y le permite trabajar con esos valores de forma segura dentro de su código.

Hay cuatro aspectos principales de OOP:

Encapsulación esto está ocultando el estado interno o los datos de los objetos.
Solo el objeto que 'posee' los datos puede cambiar su contenido.
Otros objetos solo pueden acceder o modificar estos datos enviando mensajes.
La abstracción está ocultando todos los métodos, menos los datos relevantes sobre un objeto.
Herencia esto significa que los objetos de una clase pueden derivar su comportamiento de otra clase y adaptar ese comportamiento a sus necesidades.
El polimorfismo permite que diferentes objetos que comparten una interfaz común se comporten a su manera.

Protocolos

Un protocolo define un plan de métodos, propiedades y otros requisitos que deben cumplir las clases, estructuras o enumeraciones.
Los protocolos no implementan ninguna funcionalidad, sino que se convierten en tipos completos cuando se usan.

Patrones de Arquitectura

MVC
MVP
MVVM
VIPER

Patrones

Comportamiento (Behavioral)
- Memento
- Observer

Creacional (Creational)
- Singleton

Estructural (Structural)
- Decorator
- Facade
- Composite
- Adapter

Comunicacion

Delegate

es un patrón de diseño que permite a una clase o estructura transferir (o delegar) algunas de sus responsabilidades a una instancia de otro tipo.
Este patrón de diseño se implementa mediante la definición de un protocolo que encapsula las responsabilidades delegadas, de manera que se garantiza que un tipo conforme (conocido como delegado) proporcione la funcionalidad que se ha delegado. La delegación se puede utilizar para responder a una acción en particular, o para recuperar datos de una fuente externa sin necesidad de saber el tipo subyacente de esa fuente.

Completion Handlers

Es un cierre ("un bloque de funcionalidad autónomo que se puede pasar y utilizar en su código"). Se pasa a una función como argumento y luego se llama cuando se realiza esa función.

NotificationCenter

Un mecanismo de envío de notificaciones que permite la transmisión de información a los observadores registrados.

Observer

es un patrón de diseño de comportamiento que permite que un objeto notifique a otros objetos sobre cambios en su estado.

CallBack

En la programación de iOS, existen dos formas dominantes de manejar el código asíncrono. 1. La forma de cierre 2. La forma de delegar.

https://www.bobthedeveloper.io/blog/the-delegate-and-callbacks-in-ios

KVO

Automatic Reference Counting

ARC and Retain Cycle

Cada vez que crea una nueva instancia de una clase, ARC asigna una porción de memoria para almacenar información sobre esa instancia.

Esta memoria contiene información sobre el tipo de la instancia, junto con los valores de las propiedades almacenadas asociadas con esa instancia.

El recuento de referencias se aplica solo a las instancias de la Class porque es un tipo de referencia.

Esto significa que varios objetos pueden referirse al mismo objeto. Donde Structuras, así como Enumerations son los tipos de valor.

Retain Cycle (ciclo de retención)

Podría ser un escenario en el que dos instancias de clase mantienen una fuerte referencia entre sí y no hay forma de que el sistema las desasigne.

Esto significa que el Recuento de retención de ambas clases de la clase nunca bajaría a 0. Esto se conoce como Strong Reference Cycle.

GCD (Grant Central Dispatcher)

Grand Central Dispatch o GCD es una API de bajo nivel para administrar operaciones concurrentes.
Hará que su aplicación sea suave y más sensible. También ayuda para mejorar el rendimiento de la aplicación.

App Life Cycle

- Not running
- Inactive
- Active
- background
- suspended

func application(_ application: UIApplication,
didFinishLaunchingWithOptions launchOptions:
[UIApplicationLaunchOptionsKey: Any]?) -> Bool
func applicationWillResignActive(_ application: UIApplication)
func applicationDidEnterBackground(_ application: UIApplication)
func applicationWillEnterForeground(_ application: UIApplication)
func applicationDidBecomeActive(_ application: UIApplication)
func applicationWillTerminate(_ application: UIApplication)

View Controller Cycle Life

Lifecycle events order

init(coder:)
(void)loadView
(void)viewDidLoad
(void)viewWillAppear
(void)viewDidAppear
(void)didReceiveMemoryWarning
(void)viewWillDisappear
(void)viewDidDisappear

StoryBoard

Es una representación visual de la interfaz de usuario de una aplicaron de iOS, que muestra pantallas de contenido y las conexiones entre esas pantallas.
Se compone de una secuencia de escenas, cada una de las cuales representa un controlador de vista y sus vistas, las escenas están conectadas por objetos segue, que representan una transición entre dos controladores de vista.

Segue: Son flechas que unen los view controller con otros.
Un Segue representa 2 cosas 1) Una transición entre dos escenas de nuestro storyboard. 2) Una Relación entre 2 view controller de nuestro storyboard.

Tipos de Segue

Show (push)

Permite navegar hacia adelante y hacia atrás a través de una pila de vistas que se apilan una encima de otra.

Para habilitar la navegación debe iniciar con Navigation Controller

El Navigation Controller aparecerá como una barra en la parte superior

Es apropiado para presentar pantallas Secuenciales.

Show Detail (Replace)

Un show detail segue es un tipo especial de segue es util para un controlador de vistas dividas UISplitViewController

Si usas este segue entre 2 controladores de vista que no están en un controlador de vista dividida, se comporta como un segue modal.

Present Modally

Es el tipo de segue mas comun

Muestra un nuevo controlador de vista sobre el anterior

No es parte de ninguna jerarquía

El objetivo de segue modal es presentar a los usuarios información o hacer una pregunta de alcance limitado

El flujo segue modal es presentarse lograr el un objetivo

y volver al controlador inicial

Lo controladores de vista modal pueden aparecer y desaparecer utilizando varias animaciones.

Present Popover

Un segue emergente muestra el nuevo controlador de vista confinado dentro de una ventana emergente.

Solo tiene sentido en iPad en iPhone se comporta como una transición modal.

Custom.

Autolayout

El auto layout calcula dinámicamente el tamaño y la posición de todas las vistas en su jerarquía de vistas, según las restricciones puestas en esas vistas.

Interface Builder
Programaticamente (NSLayoutConstraint)

XCTest
Notificaciones

Resume de clases

Core Data

AppDelegate
NSManagedObjectContext
UIApplication.shared.delegate as! AppDelegate
appDelegate.persistentContainer.viewContext
NSManagedObject
NSFetchRequest

URLSession

URLSession.shared.dataTask
DispatchQueue.main.async

HTTPURLResponse

TableView

UITableViewDataSource
func numberOfSections(in tableView: UITableView) -> Int
func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int
func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell
func tableView(_ tableView: UITableView, heightForRowAt indexPath: IndexPath) -> CGFloat {
    return 100
}

UITableViewDelegate
func tableView(_ tableView: UITableView, didSelectRowAt indexPath: IndexPath)

self.tableView.indexPathForSelectedRow?.row

UITableViewCell

override func awakeFromNib()

UICollectionView

UICollectionViewFlowLayout
UICollectionViewDelegate,
UICollectionViewDataSource,
UICollectionViewDelegateFlowLayout
func numberOfSections(in collectionView: UICollectionView) -> Int
func collectionView(_ collectionView: UICollectionView, numberOfItemsInSection section: Int) -> Int
func collectionView(_ collectionView: UICollectionView, cellForItemAt indexPath: IndexPath) -> UICollectionViewCell

self.performSegue(withIdentifier: "segueShowRegister", sender: self)

```
 override func prepare(for segue: UIStoryboardSegue, sender: Any?)
```
UIActivityIndicatorView

JSON

```
protocol Networks {
    typealias response = (SuperHeroes)->()
    func getMarvelsHeroes(completion: @escaping response)
}
class MarvelUrlAPI: Networks {
    func getMarvelsHeroes(completion: @escaping (SuperHeroes) ->
()) {
        guard let url = URL(string: "https://api.myjson.com/bins/bvyob")
else{ return }
        URLSession.shared.dataTask(with: url) { (data, response, error)
in
            guard let httpURLResponse = response as?
HTTPURLResponse,
                (200...209).contains(httpURLResponse.statusCode),
                let mimetype = response?.mimeType, mimetype ==
"application/json",
                let data = data, error == nil else { return }

            do {
                //let json = try JSONSerialization.jsonObject(with: data,
options: [])
                //print(json)
                let heroes = try JSONDecoder().decode(SuperHeroes.self,
from: data)
                completion(heroes)
            } catch {
                print("Json Error \(error.localizedDescription)")
            }
        }.resume()
    }
}
```

Preguntas

https://medium.com/@chetan15aga/ios-interview-questions-part-1-differentiate-99e8f574a3f1