**SWIFTUI**

Ej. Lista cuyos elementos lo forman una imagen y unos textos a los que queremos añadirle un alineamiento y después cambiar el color.

Struct AlbumDetail: View { //Vista llamada AlbumDetail muestra una lista de canciones de un álbum. Es una estructura que conforma el protoco View, es decir representa una pantalla.

Var album: Album //Album es una propiedad de tipo Album, lo que significa que AlbumDetail necesita un objeto Album para funcionar.

Var body: some View { //Body es obligatorio en Swiftui y define el contenido de la vista.

List(album.songs) { song in. //List es un contenedor en SwiftUI que se usa para mostrar lista de elementos. Album.songs se asume que songs es un array de canciones dentro de albu,. {song in … } itera sobre cada canción y la muestra en la lista.

HStack { //Organiza sus elementos en horizontal.

Image(album.cover) // Muestra la imagen del álbum (probablemente un nombre de archivo o un UIImage).

VStack (alignment: .leading). { //Organiza los textos en vertical, alineados. A la izquierda

Text(song.title) //Muestra el título de la cancion

Text(song.artist.name) //Muestra el nombre del artista.

foreogroundStyle(.secondary) //le da un color secundario al texto, lo que normalmente significa que será mas tenue.

}

}

}

}

}

**ESTRUCTURA**

Dos archivos principals ‘(nombre)App.Swift’ y ‘ContentView.swift’

**ContentView**

* Componente principal (o vista) se añaden elementos a la interfaz de usuario.
* Dentro hay una variable llamada body que es la encargada de gestionar los elementos añadidos. (lienzo)
* Si queremos añadir variables adicionales u objetos, hay que definirlo fuera.
* Cada ContentView tiene un VStack global.

Image(systemName: “globe”) Utiliza un ícono del sistema proporcionado por SF Symbols.

.imageScale (.large) ajusta el tamaño del ícono a una escala grande

.foregroundStyle(.tint) Cambia el color de la imagen al color de acento predeterteminado de la app.

.padding () -> Se coloca fuera del contenedor, y lo que hace es agregar un margen espaciado alrededor separándolo de los bordes de la pantalla.

**Tipos contenedores**

VStack. Verticales(columnas), cada elemento se sitúa debajo del anterior.

HStack. Horizontales (filas) cada elemento se sitúa a la derecha del anterior.

ZStack. Profundidad, donde unos elementos se posicionen encima de otros. Estos son útiles cuando queremos añadir elementos de fondo como imágenes.

Si queremos añadir alguna característica de un elemento mediante código, debemos ir añadiendo personalizaciones justo después de la definición de cada elemento.

Hay que analizar los componentes de lo que queremos crear, cuantas filas, cuantas columnas etc.

Por ej. tenemos un icono y un texto, podemos maquetar esta estructura con un HStack.

Var body: some View {

VStack (alignment: .leading) //Alinea todo a la izquierda {

HStack {

Image (systemName: “person”)

.imageScale(.large)

Text (“Jorge Caro Poza”)

.font(.title2)

.bold()

.padding(.leading, 8) //Le estamos especificando que solo queremos aplicarlo al lado izquierdo y especificamos un tamaño inferior.

}

Text([correo@gmail.com](mailto:correo@gmail.com))

.accentColor(.black) //Se utiliza esta propiedad al tratarse de un correo, si no siempre es foregroundColor.

.padding(.top, 1)

Text (“Biografia”)

.foregroundColor(.gray)

padding(.top, 5)

Text (sgiowdniwjfljsojid)

.font(.system(size:13)))

padding(.top, 1)

}

.padding()

}

**App Icon y Launch Screen**

* Para añadir una pantalla de carga, tendremos que añadir la pantalla con un fichero de tipo Launch Screen. Esto creará un archivo de tipo StoryBoard, tendremos que referenciar este fichero en la configuración general del proyecto en la sección General>App Icons and Launch Screen hay que asignar el que hemos creado.
* Asignar logo aplicación: imagen cuadrada (1024x1024). Se añade en Assets, en la parte de AppIcon.

**CICLO DE VIDA SWIFTUI**

(nombre)App.swift, este archivo es el inicializador de la aplicación y donde se gestiona el ciclo de vida. Estructura:

Import SwiftUI

@main //Etiqueta que se utiliza para indicar que esta estructura es el punto de entrada en la ejecución de la aplicación

Struct a1App: App { //Se indica que la estructura utiliza el protocol App y que Lanza las vistas de la aplicación

Var body: some Scene { //Var body necesario para que la estructura pueda cumplir con el protocolo App. Cada Scene contiene una vista raíz de una jerarquía de vistas y tiene un ciclo de vida gestionada por el sistema.

WindowGroup { //Es una estructura multiplataforma que representa una escena de múltiples ventanas (Windows). Actúa como contenedor de la jerarquía de vistas de una aplicación.

ContentView()

}

}

}

* Init() Podemos añadir las primeras acciones que debe realizar la aplicación al inicializarse.

import SwiftUI

@main

struct a1App: App {

init(){. //inicializador de la estructura, llama al método startUp definido más abajo. Asegura que siempre se ejecute un proceso al inicializar la APP

startUp().

}

@Environment(\.scenePhase) private var scenePhase //@Enviroment permite acceder al entorno de ejecución de SwiftUI. \scenePhase monitorea el estado actual de la escena de la app (activa, segundo plano o inactiva) scenePhase es una propiedad que SwiftUI actualiza automáticamente según el estado de la aplicación.

var body: some Scene {

WindowGroup {

ContentView()

}

.onChange(of: scenePhase) { oldScenePhase, newScenePhase in

//OnChange (of: scenephase) es un modficador SwiftUI que detecta cambios en el valor de una variable

ScenePhase es una propiedad de entorno que proporciona el estado actual de la app

OldScenePhase (estado anterior de la app) newScenePhase(nuevo estado después del cambio)

switch newScenePhase{

case .active:

print("Active")

case .inactive:

print("Inactive")

case .background:

print("Background")

@unknown default:

print("Desconocido")

}

}

}

}

func startUp(){

print("Primera acción de la aplicación")

}

**SYNTATIC SUGAR**

-Con ? podemos acceder a propiedades o métodos de un objeto sin tener que desempaquetarlo ej.

If let Street = user?.adress?.street { Intenta obtener el valor de street solo si user no es nil, user.adress no es nil y user.address.street tiene un valor válido.

Print (street)

}

-?? Permite añadir valores por defecto en caso de que una variable opcional sea nula

Let orderStatus? Status ?? “Pending”

**VISTA MODAL.**  Se superpone la nueva vista.

Para añadirla hay que crear un nuevo archivo de tipo SwiftUI View, tendrá la misma estructura que el ContentView.

1º

Struct ModalView: View {

Var body: some View {

ZStack {

Color.red.edgesIgnoringSafearea(.all) // Color.red añadimos el color rojo para el fondo de esta vista y con la función edgesIgnoringSafeArea(.all) lo que conseguimos es asignar el color a toda la pantalla.a

Text (“Ventana modal”)

.font(.largeTitle)

.foregroundColor (.white)

}

}

}

2º Una vez temenos la vista definida, temenos que crear la conexión entre ambas, esta acción es gestionada por una variable

@state private var isVisible = false

@state se utiliza para declarar una propiedad que almacena un estado en una vista, se actualiza automáticamente cuando cambia el valor. El uso típico de “@state” es para almacenar datos que pueden cambiar durante el ciclo de vida de una vista, como variable de control de la interfaz de usuario, valores de entrada del usuario (inputs) o cualquier otro estado local que afecta a la representación de la interfaz de usuario. Un uso común de estos elementos es almacenar una variable booleana para controlar la visibilidad de ciertos elementos como ventanas modales, botones activados/desactivados, etc.

Strcut ContentView: View {

@State private var showModal = false

Varbody: some View {

VStack {

Button(“Abrir modal”) {

showModal.toggle() //Con .toggle() estamos cambiando el valor de la variable, pero también podriamos asignarle el valor true.

}

.sheet(isPresented:$showModal) { /.sheet se utiliza para mostrar una vissta modal desde otra vista, recibe un parámetro llamado isPresented al cual le asignaremos la variable de estado showModal, actua como escuchador esperando a que showModal cambie de estado a true para ejecutar el código que tiene en su interior.

ModalView()

}

}

}

}

**NAVIGATIONVIEW.** Una nueva vista en una pantalla diferente.

-Se envuelve la columna principal del contentView (VStack) con un elemento llamado ‘’NavigationStack”.

Strcut ContentView: View {

@State private var showModal = false

Varbody: some View {

NavigationStack { //Añadir un NavigationStack que envuelva nuestro VStack principal.

VStack {

Button(“Abrir modal”) {

showModal.toggle()

}

.sheet(isPresented:$showModal) {

ModalView()

}

NavigationLink(destination: SecondView()}{ //En la nav modal es con un boton, aquí es con NavigationLink indicamos el destino de nuestra vista y definimos la visualización del elemento (lo que hay que pinchar para ir a la 2º vista

Text(“Navegar a la segunda vista”)

} .navigationTitle(“Home”).navigationBarTitleDisplayMode(.inline) //Con esto configuramos el título y la barra de navegación.

navigationBarTitleDisplayMode(.inline) -> .inline significa que el titulo aparecerá en la barra de navegación con un tamaño pequeño. Otras opciones disponibles:

.automatic (por defecto, ajusta según el contexto).

.large (título más grande, aparece como un encabezado).

.toolbar{ //Añadir en el ToolBar nos permite añadir más elementos en la parte superior derecha de la pantalla.

NavigationLink(destination: SecondView(){ //Se navega a SecondView cuando se presione el icono

Image(systemName: “plus”) //Muestra el signo de + de SF Symbols

}

}

}

**Envío de parámetros entre pantallas.**

* Hay que definir los elementos que se quieren enviar en la vista de destino y pasarlos como parámetros en la vista de origen.
* Vista destino

Struct ModalView: View {

Var name: String

Var body: some View {

ZStack {

Text(“Buenos días \(name)!!”)

.font(.title)

}

}

}

* Vista origen.

Struct ContentView: View {

Var body: some View {

TabView { //Es un contenedor que permite crear interfaces con pestañas, crea una barra de navegación en la parte inferior de la pantalla.

HomeView().tabItem { //tabitem configura el título y el icono de cada pestaña mediante Label. HomeView es la vista principal que se meustra cuando esta pestaña esta seleccionada.

Label( //muestra un título y un ícono en la pestaña. Y así todas.

title: { Text("Home") },

icon: { Image(systemName: "house") }

)

}

SearchView().tabItem {

Label(

title: { Text("Buscar") },

icon: { Image(systemName: "magnifyingglass") }

)

}

ProfileView().tabItem {

Label(

title: { Text("Perfil") },

icon: { Image(systemName: "person") }

)

}

}

}

}

**TabView.** Recomendable cuando tenemos entre 3-5 pantallas. Para añadirlo es necesario tener una vista que gestione el resto de las vistas que queremos mostrar. Esa vista controladora tendrá dentro del body un único elemento de tipo TabView.

Para añadir un elemento al menú de navegación basta con llamar al método tabItem y añadir el tipo de elemento visual que queremos utilizar para representar a la vista, en este caso estamos utilizando un label con un texto y un icono.

**LISTAS Y GRID.** Muestran una gran cantidad de datos con la misma estructura.

**Lista**

-Basta con crear un elemento List que funciona como contenedor de otros elementos al igual que ocurre con las filas y columnas (HStack y VStack). Los elementos de una lista pueden no ser del mismo tipo, alternando entre textos, filas, imágenes etc.

Struct ListView: View {

Var body: some View {

List {

Text(“Elemento 1”)

Text(“Elemento 1”)

Text(“Elemento 1”)

Text(“Elemento 1”)

}

}

}

**LISTA DINÁMICA**

import Foundation

struct Person: Identifiable { //El protocol Identifiable, lo que significa que cada instancia necesita un ID unico. Es importante implantarlo con las listas para que SwiftUI pueda diferenciar entre ellos.

let id = UUID() //Clase propia de SWIFT que asigna a cada elemento un identificador único

let name: String

let mail: String

}

let people: [Person] = [

Person(name: "Jorge", mail: "jorge@gmail.com"),

Person(name: "Pedro", mail: "pedro@gmail.com"),

Person(name: "Javier", mail: "javier@gmail.com"),

Person(name: "Laura", mail: "laura@gmail.com")

] //Se crea una lista (Array) de Person con 4 elementos.

Import SwiftUI

Struct HomeView: View { //Vista en SwiftUi que mostrará la lista de personas.

Var body: some View {

List (people) { item in

Text(item.name) //Recorre el array people y muestra cada persona en la lista. Con Text(item.name) muestra el nombre de cada persona en un Text.

}

}

}

@Preview {

HomeView()

}

Mejorar apariencia en el Código

import SwiftUI

struct HomeView: View {

var body: some View {

List(people) { item in

VStack(alignment: .leading) {

Text(item.name)

.bold()

Text(item.mail)

.fontWeight(.light)

}

}

}

}

#Preview {

HomeView()

}

**GRIDS**

Muy común para mostrar una galería de fotos u otros elementos audiovisuales. Mayor complejidad. Dificulta la interfaz en situaciones dinámicas y con una gran cantidad de información para mostrar.

Grid { // Crea un diseño en cuadrícula

GridRow { // Primera fila de la cuadrícula

Text("Row 1") // Texto que indica el nombre de la fila

ForEach(0..<2) { \_ in Color.red } // Agrega 2 elementos de color rojo

}

GridRow { // Segunda fila de la cuadrícula

Text("Row 2") // Texto que indica el nombre de la fila

ForEach(0..<5) { \_ in Color.green } // Agrega 5 elementos de color verde

}

GridRow { // Tercera fila de la cuadrícula

Text("Row 3") // Texto que indica el nombre de la fila

ForEach(0..<4) { \_ in Color.blue } // Agrega 4 elementos de color azul

}

}

struct ContentView: View {

let gridItem: [GridItem] = Array(repeating: .init(.flexible()), count: 3)

// Define un array de GridItem con 3 columnas flexibles en la cuadrícula

var body: some View {

ScrollView { // Permite desplazamiento vertical si hay muchos elementos

LazyVGrid(columns: gridItem, spacing: 30) {

// Crea una cuadrícula vertical con 3 columnas y un espaciado de 30 entre elementos

ForEach(people) { item in // Itera sobre la colección "people"

VStack { // Agrupa los elementos en una pila vertical

Image(systemName: item.icon) // Muestra un ícono de SF Symbols

.font(.title) // Ajusta el tamaño del ícono

Text(item.name) // Muestra el nombre del elemento

}

}

}

}

}

}

El elemento que define una grid es el llamado “LazyVGrid”. Este elemento recibe un parámetro principal que es una lista de “GridItem”. En este ejemplo estamos indicando que queremos crear celdas flexibles que se adapten al espacio disponible  (.flexible())y que haya tres columnas (3 elementos por fila)

Tenemos unos datos de categoría con nombre, imagen y descripción vamos a estructurarlos con GRID

struct ContentView: View {

let gridItem: [GridItem] = Array(repeating: .init(.flexible()), count: 1)

// Declara una variable que define cómo se organizarán los elementos en la cuadrícula.

// `.flexible()` permite que los elementos se ajusten automáticamente al tamaño disponible.

// `count: 1` significa que habrá solo una columna en la cuadrícula.

var body: some View {

ScrollView { // Permite desplazamiento vertical cuando hay más contenido del que cabe en pantalla

LazyVGrid(columns: gridItem, spacing: 20) {

// Crea una cuadrícula vertical con una sola columna (definida en gridItem) y un espaciado de 20 puntos entre los elementos.

ForEach(Category.sampleData) { item in

// Itera sobre `Category.sampleData`, que es una colección de datos de ejemplo.

// Cada elemento se almacena en `item` y se representará en la cuadrícula.

VStack {

Image(uiImage: UIImage(named: item.image)!)

// Carga una imagen desde los assets usando el nombre almacenado en `item.image`

// ¡OJO! El uso de `!` puede causar un crash si la imagen no existe.

.resizable()

.scaledToFill() // Escala la imagen para llenar completamente el espacio disponible

.frame(height: 220) // Define una altura fija de 220 puntos para la imagen

.clipShape(RoundedRectangle(cornerRadius: 10))

// Aplica un recorte a la imagen con bordes redondeados de 10 puntos

}

}

}

.padding()

}

}

}

struct DetailView: View {

let category: Category // Propiedad que almacena una categoría específica

var body: some View { // Cuerpo de la vista donde se define la interfaz gráfica

VStack(alignment: .leading) { // Organiza los elementos en una columna alineados a la izquierda

Text(category.name) // Muestra el nombre de la categoría

.font(.title) // Aplica un tamaño de fuente grande (título)

.bold() // Hace el texto en negrita

Image(uiImage: UIImage(named: category.image)!) // Carga una imagen desde los assets usando el nombre almacenado en la categoría

.resizable() // Permite que la imagen pueda redimensionarse

.scaledToFit() // Ajusta la imagen para que encaje sin distorsionarse

.clipShape(RoundedRectangle(cornerRadius: 10)) // Recorta la imagen con esquinas redondeadas de 10 píxeles

Text("Descripción") // Texto estático que sirve como encabezado de la descripción

.padding(.top, 10) // Agrega un margen superior de 10 puntos

.padding(.bottom, 5) // Agrega un margen inferior de 5 puntos

.foregroundStyle(.gray) // Cambia el color del texto a gris

Text(category.description) // Muestra la descripción de la categoría

} // Fin del VStack

.frame(maxHeight: .infinity, alignment: .top) // Expande la vista verticalmente hasta el máximo disponible y alinea el contenido en la parte superior

.padding() // Agrega un relleno alrededor de todo el contenido para espaciarlo de los bordes

}

}