**EXPLICACION TAREA2**

**1. Estructura del Proyecto**

El proyecto se organiza en los siguientes archivos:

📂 **tarea2App.swift** (Punto de entrada de la aplicación)  
📂 **Views/** (Interfaz de Usuario - SwiftUI)  
📂 **ViewModels/** (Lógica de Negocio)  
📂 **Models/** (Estructuras de Datos)

Cada carpeta tiene una función específica en el patrón **MVVM**:

* **Models (Modelos de Datos)** → Representa la estructura de la información obtenida desde la API.
* **ViewModels (Lógica de Negocio)** → Maneja la obtención y procesamiento de los datos.
* **Views (Interfaz de Usuario)** → Muestra los datos en pantalla y permite la interacción del usuario.

**🔹 2. Descripción de los Archivos**

A continuación, te explico cada parte del código con su función dentro del proyecto.

**📌 2.1 tarea2App.swift (Punto de Entrada de la App)**

📂 **tarea2App.swift**  
🔹 Es el archivo principal que inicia la aplicación.  
🔹 Configura FilmListModel() como @StateObject, permitiendo que la app obtenga datos desde la API.

🔹 **Código clave**:

swift

CopiarEditar

@main

struct tarea2App: App {

var body: some Scene {

WindowGroup {

ContentView() // Carga la vista principal `ContentView.swift`

}

}

}

**📌 2.2 Models/ (Estructuras de Datos)**

📂 **Models/**  
Los modelos definen la estructura de los datos obtenidos desde la API.

**📌 🔹 Film.swift (Modelo para las películas)**

* Representa la respuesta JSON de la API.
* Film contiene una lista de películas (results).
* Result representa los datos de una **película individual**.

swift

CopiarEditar

struct Film: Codable {

let page: Int

let results: [Result]

let total\_pages, total\_results: Int

}

struct Result: Codable {

let id: Int

let title: String

let overview: String

let genre\_ids: [Int]

let poster\_path: String

let release\_date: String

let vote\_average: Double

}

✅ Permite **decodificar** los datos JSON en objetos Film.

**📌 🔹 Genero.swift (Modelo de Géneros de Películas)**

* Contiene una lista estática de **géneros de películas con su ID**.
* Sirve para convertir los genre\_ids en nombres de géneros legibles.

swift

CopiarEditar

struct Genero {

var id: Int

var nombre: String

}

extension Genero {

static var generos = [

Genero(id: 28, nombre: "Acción"),

Genero(id: 12, nombre: "Aventura"),

Genero(id: 16, nombre: "Animación"),

...

]

}

✅ Se usa en DetailFilmView.swift para mostrar el **nombre** del género en lugar del ID.

**📌 2.3 ViewModels/ (Lógica de Negocio)**

📂 **ViewModels/**  
Los **ViewModels** manejan la obtención y actualización de datos en la aplicación.

**📌 🔹 FilmListModel.swift (Manejo de Películas)**

* **Hace una petición HTTP GET a la API de TMDb** para obtener las películas mejor valoradas.
* **Decodifica los datos JSON** y los guarda en @Published var films.
* **Maneja errores de red y decodificación JSON**.

swift

CopiarEditar

class FilmListModel: ObservableObject {

@Published var films: [Result] = []

init() {

obtenerDatos()

}

func obtenerDatos() {

let request = NSMutableURLRequest(url: NSURL(string: "https://api.themoviedb.org/3/movie/top\_rated?language=es-ES&page=1")! as URL)

request.httpMethod = "GET"

URLSession.shared.dataTask(with: request as URLRequest) { data, response, \_ in

guard let data = data else { return }

do {

let datos = try JSONDecoder().decode(Film.self, from: data)

DispatchQueue.main.async {

self.films = datos.results

}

} catch {

print("Error al decodificar JSON")

}

}.resume()

}

}

✅ **Se usa en ContentView.swift para mostrar las películas en la UI**.

**📌 2.4 Views/ (Interfaz de Usuario)**

📂 **Views/**  
Las vistas se encargan de la presentación de la información y la interacción del usuario.

**📌 🔹 ContentView.swift (Pantalla Principal)**

* **Muestra la lista de películas** en un **grid de 2 columnas**.
* **Incluye una barra de búsqueda** para filtrar películas por título.
* **Carga los datos desde FilmListModel()** (ViewModel).
* **Permite navegar a DetailFilmView** cuando se selecciona una película.

swift

CopiarEditar

struct ContentView: View {

@StateObject var data = FilmListModel()

@State private var stringBuscar: String = ""

var body: some View {

NavigationStack {

VStack {

TextField("Buscar película", text: $stringBuscar)

ScrollView {

LazyVGrid(columns: [GridItem(.flexible()), GridItem(.flexible())]) {

ForEach(data.films.filter { film in

stringBuscar.isEmpty || film.title.lowercased().contains(stringBuscar.lowercased())

}, id:\.id) { film in

Item(film: film)

}

}

}

}

}

}

}

✅ Muestra **todas las películas** y permite filtrarlas por nombre.

**📌 🔹 DetailFilmView.swift (Vista de Detalles de Película)**

* **Muestra el póster, título, sinopsis y géneros** de la película seleccionada.
* **Usa AsyncImage para cargar la imagen** desde la API.
* **Muestra los nombres de los géneros usando Genero.swift**.

swift

CopiarEditar

struct DetailFilmView: View {

let film: Result

var body: some View {

VStack(alignment: .leading){

AsyncImage(url: URL(string: "https://image.tmdb.org/t/p/original/"+film.poster\_path))

Text(film.title)

.font(.title)

Text(film.release\_date)

.font(.subheadline)

.foregroundColor(.redCarpet)

Text(film.overview)

.font(.caption)

.multilineTextAlignment(.center)

HStack {

ForEach(film.genre\_ids, id: \.self) { id in

Text(Genero.generos.first(where: { $0.id == id })?.nombre ?? "Desconocido")

.font(.caption2)

.padding(.all, 5)

.background(.gray.opacity(0.2))

.cornerRadius(10)

}

}

}

}

}

✅ Se abre cuando el usuario selecciona una película en ContentView.swift.

**🔹 3. Flujo de la Aplicación**

1️⃣ **ContentView.swift** carga y muestra las películas en una **lista de tarjetas**.  
2️⃣ **El usuario puede buscar películas** con la barra de búsqueda.  
3️⃣ **Al seleccionar una película**, se abre DetailFilmView.swift.  
4️⃣ **Se muestra el póster, sinopsis y géneros** de la película.

**🚀 Conclusión**

✔ **Bien estructurado en MVVM**.  
✔ **Carga de datos en tiempo real desde una API**.  
✔ **Uso eficiente de SwiftUI con LazyVGrid y NavigationStack**.

📌 **¿Quieres agregar alguna funcionalidad o mejorar la UI?** 🚀🎬