MOV02 - Laboratorio 8

Juan Pablo Arias Mora

Febrero 2021

1 Introducción

En algún momento, habrá creado o creará un **API** para su aplicación. El **API** permite que su aplicación se comunique con otros sistemas. Uno de los puntos más importantes a tener en cuenta es que la **API** es solo una interfaz entre su aplicación y el servidor / base de datos. No es una base de datos ni un servidor en sí. Simplemente permite que todas estas entidades se comuniquen entre sí. La finalidad del laboratorio es ejemplificar cómo se puede obtener datos en **SwiftUI** desde una **API**.

Para referencias futuras el código fuente final del mismo esta disponible en:
https://github.com/pabloariasmora/MOV02-Cenfotec-Demo-Labs.git

1.1 Versiones

- MacOs 10.15.7 - Inglés
- Xcode 12.4
- Simulador iPhone 12 Pro Max

2 Instrucciones

"Hay empresas que prefieren reducir plantilla cuando las cosas van mal. Nosotros pensamos lo contrario. Invertimos más para desarrollar mejores productos y que los clientes estén dispuestos a dejarse el dinero por ellos." Steve Jobs

2.1 Creación de aplicación base

Paso 1: Cree un nuevo proyecto en Xcode, utilizando el template App dentro de iOS, en la ventana de opciones para la configuración especifica de la nueva aplicación.

Product Name : api
Organization Identifier : com.cenfotec.mov02.08
Interface : SwiftUI
Life Cycle : SwiftUI App
Languge : Swift
Use Core Data: Sin Seleccionar
Include Test: Sin Seleccionar

Paso 2: Abrir el siguiente link https://cat-fact.herokuapp.com/facts en navegador. De esta manera vamos a acceder al Método **GET** del **API** gratuito con un par de datos interesantes sobre Gatos.

Paso 3: Si esta utilizando Chrome como navegador es recomendable instalar la siguiente extensión https://chrome.google.com/webstore/detail/json-viewer/gbmdgpbipfallnflgajpaliibnhdgobh y luego recargar el url. Vemos como se carga una lista de objetos similares al siguiente:

```
{
    "status": {
        "verified": true,
        "sentCount": 1
    },
```

```
    "type": "cat",
    "deleted": false,
    "_id": "58e008800aac31001185ed07",
    "user": "58e007480aac31001185ecef",
    "text": "Wikipedia has a recording of a cat meowing, because why not?",
    "__v": 0,
    "source": "user",
    "updatedAt": "2020-08-23T20:20:01.611Z",
    "createdAt": "2018-03-06T21:20:03.505Z",
    "used": false
}
```

Paso 4: El laboratorio solamente se centraliza sobre los **keys**: **_id** y **text**, pero es importante saber que existen muchos más presentes.

Paso 5: Abrir el archivo **ContentView.swift**. (El código en el archivo debe ser similar al que se muestra a continuación)

```
import SwiftUI

struct ContentView: View {
    var body: some View {
        Text("Hello, world!")
            .padding()
    }
}

struct ContentView_Previews: PreviewProvider {
    static var previews: some View {
        ContentView()
    }
}
```

Paso 6: Por lo general, las API devuelven un JSON y necesitaría decodificar y codificar usando JSONEncoder y JSONDecoder, por lo tanto, debemos asegurarnos de que nuestro modelo se pueda codificar y decodificar. Por lo tanto, hemos agregado **Codable** para hacer que el **Model** sea codificable. Cree un nuevo archivo llamado **CatFact.swift** y dentro del mismo agregue el siguiente código:

```
import Foundation
struct CatFact: Codable  {
    let _id: String
    let text: String
}
```

En este punto utilizando Codable mapeamos los los **keys**: **_id** y **text** con sus respectivos tipos.

Paso 7: Dentro del archivo **ContentView.swift** debemos agregar la variable **results** utilizando el siguiente código:

```
@State var results = [CatFact]()
```

La variable **result** contendrá una lista de objetos del tipo **CatFact**

Paso 8: Dentro del archivo **ContentView.swift** debemos ahora vamos a agregar el código responsable de las consultas al **API** utilizando el siguiente código:

```
func loadData() {
    guard let url = URL(string: "") else {
        print("Your API end point is Invalid")
        return
    }
    let request = URLRequest(url: url)
```

```
    // The shared singleton session object.
    URLSession.shared.dataTask(with: request) { data, response, error in
        if let data = data {
            if let response = try? JSONDecoder().decode([CatFact].self, from: data) {
                DispatchQueue.main.async {
                    self.results = response
                }
                return
            }
        }
    }.resume()
}
```

El código a este punto con los conceptos vistos en clase permite observar que la función **loadData** cargará los datos en el **Struck**. Si en este punto (a nivel del **init()** del **Struck**), imprimimos los datos del **API**, deberíamos poder ver la respuesta del mismo.

Paso 9: En este punto dentro del archivo **ContentView.swift** debemos modificar el **View** actual y reemplazar el usual **Text** "Hello, world!" con el siguiente código:

```
List(results, id: \._id) { item in
    VStack(alignment: .leading) {
        Text(item.text)
    }
}.onAppear(perform: loadData)
```

Paso 10: Revise el código que agregamos con anterioridad (ver figura 1 para una referencia de como debe verse el código completo) y analice que es lo que se quiere lograr con el parámetro **perform**. Efectivamente es la llamada a la función que carga los datos desde el **API**.

```
import SwiftUI

struct ContentView: View {
    @State var results = [CatFact]()

    var body: some View {
        List(results, id: \._id) { item in
            VStack(alignment: .leading) {
                Text(item.text)
            }
        }.onAppear(perform: loadData)
    }


    func loadData() {
        guard let url = URL(string: "") else {
            print("Your API end point is Invalid")
            return
        }
        let request = URLRequest(url: url)

        // The shared singleton session object.
        URLSession.shared.dataTask(with: request) { data, response, error in
            if let data = data {
                if let response = try? JSONDecoder().decode([CatFact].self, from: data) {
                    DispatchQueue.main.async {
                        self.results = response
                    }
                    return
                }
            }
        }.resume()
    }

}
```

Figure 1: Código en ContentView

Paso 11: Compile y ejecute la aplicación. Que sucede la misma completa la carga en el simulador? Vemos como efectivamente el código no realiza la llamada al **API** ya que en ningún momento, le definimos al **parámetro string** del objeto **URL** el valor de nuestra **URL** para el **API**.

Paso 12: Compile y ejecute la aplicación nuevamente, el resultado debe ser similar a la figura 2

En este momento podemos comprobar como el servicio de consulta al **API** realiza la perfecta codificación/decodificación de los datos. Ahora el proceso de **rendering** utiliza el **modifier .onAppear** presente sobre el objeto **List**

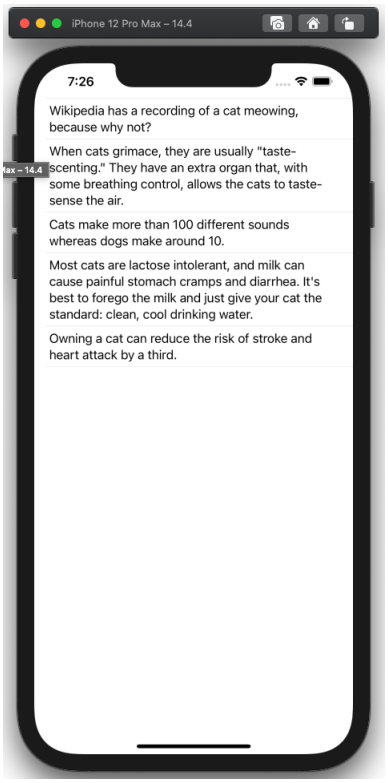

Figure 2: Resultado Final