

MOV02 - Laboratorio 9

Juan Pablo Arias Mora

Febrero 2021

1 Introducción

Como hemos conversado en algunas clases la mejor forma de utilizar SwiftUI es contra el patrón MMVM

Para referencias futuras el código fuente final del mismo esta disponible en:

<https://github.com/pabloariasora/MOV02-Cenfotec-Demo-Labs.git>

1.1 Versiones

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

2 Instrucciones

”Si quieres vivir una vida creativa y artística, no debes mirar demasiado hacia atrás. Debes estar dispuesto a arrojar a la basura cualquier cosa que hiciste.” 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 : mmvmapi
Organization Identifier : com.cenfotec.mov02.09
Interface : SwiftUI
Life Cycle : SwiftUI App
Language : Swift
Use Core Data: Sin Seleccionar
Include Test: Sin Seleccionar

Paso 2: Abrir el siguiente link <http://api.letsbuildthatapp.com/static/courses.json> en navegador. De esta manera vamos a acceder al Método **GET** del **API** gratuito con un par de datos interesantes sobre cursos en venta.

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

```
{
  "bannerUrl": "https://letsbuildthatapp-videos.s3-us-west-2.amazonaws.com/d845f2df-4b77-4d61-858b-c",
  "name": "Fullstack Social iOS NodeJS REST",
  "price": 90
}
```

Paso 4: 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 5: Como primer vamos a remplazar el código dentro de **ContentView.swift**, con el código a continuación.

```
import SwiftUI

struct ContentView: View {
    var body: some View {
        NavigationView {
            ScrollView {
                Text("Hello World!!!")
            }.navigationBarTitle("Courses")
            .navigationBarItems(
                trailing: Button(
                    action: {
                        print("Fetching json data")
                    },
                    label: {
                        Text("Fetch Courses")
                    })
            )
        }
    }
}

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

Paso 6: Compilamos y ejecutamos el código, una vez el simulador contenga la aplicación corriendo presionamos el botón **Fetch Courses**, y veremos como aparece un mensaje dentro del área de **Debug**, con el mensaje **"Fetching json data"**. (ver imagen 1)

lab_8_img/mvvmapi1.png

Figure 1: Código Inicial y Ejecución Inicial

Paso 7: Si bien es cierto, para aplicaciones de gran escala debemos segmentar en Grupos los diferentes **Model**, **View**, **ModelView** y **Servicios**, para este ejemplo mantendremos todos dentro del mismo archivo inicial, esto para facilitar su análisis. Partiendo de esto creamos una clase en la parte superior de **ContentView.swift**, justo sobre la definición del **ContentView** la cual incluya el siguiente código.

```
class CoursesViewModel: ObservableObject {
    @Published var messages = "Message inside the observable object"
}
```

Paso 8: Luego dentro de **ContentView** justo antes del body, definiremos una variable de la siguiente manera.

```
var coursesVM = CoursesViewModel()
```

Paso 9: Reemplazaremos el código del primer **Text** el que contiene **"Hello World!!!"** con el siguiente código:

```
Text(coursesVM.messages)
```

Paso 10: Compilamos y ejecutamos, el resultado debería ser similar a la figura 2.

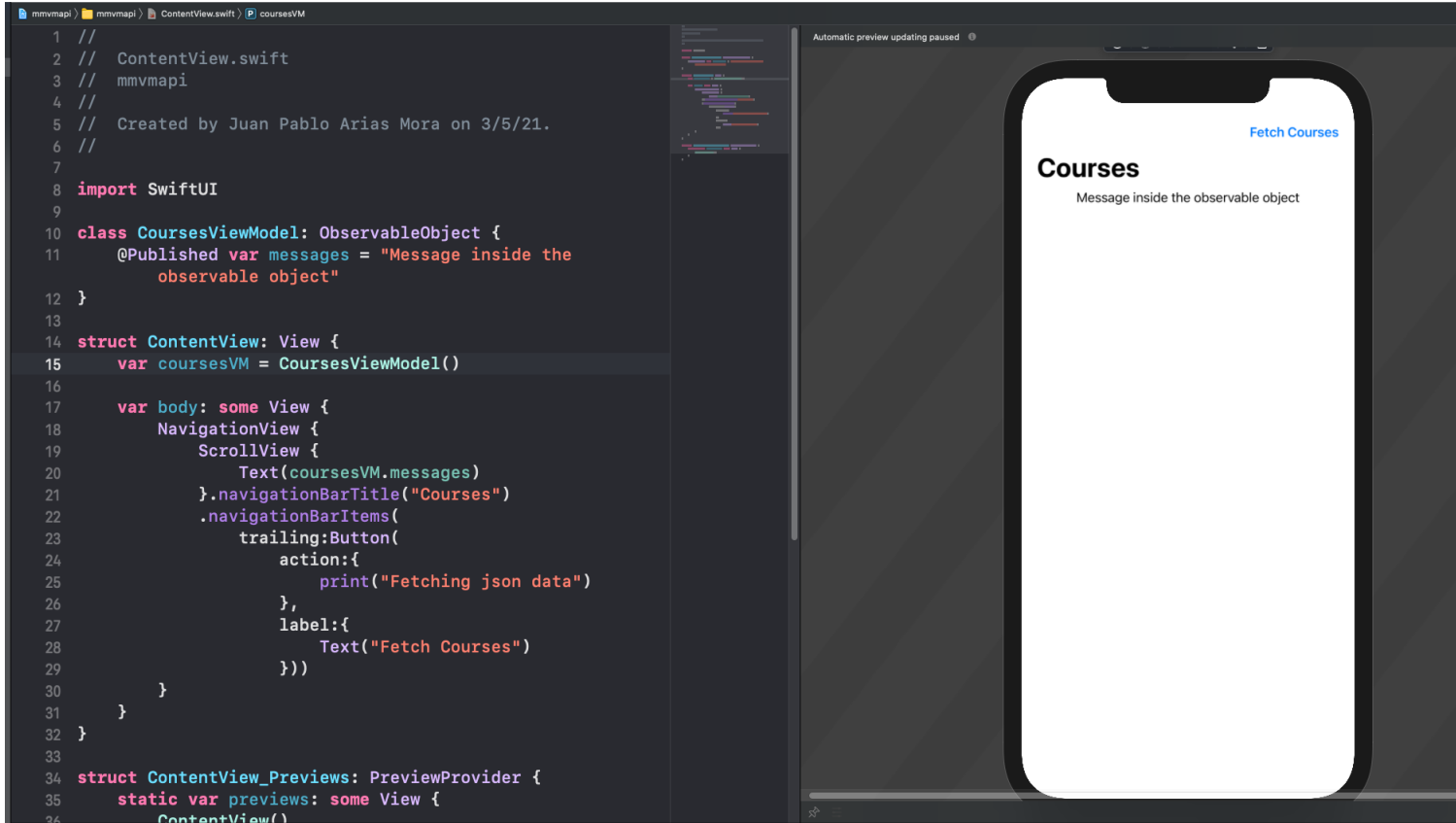


Figure 2: Observable en Ejecución

Paso 11: La finalidad del laboratorio es ligar la acción del botón a que nos descargue la lista de cursos y nos lo despliegue en pantalla. Para eso agregamos la siguiente línea luego del mensaje que imprimimos en la consola de **Debug**.

```
self.coursesVM.messages = "New Message"
```

Paso 12: Compilamos y ejecutamos. Presionamos el botón y podemos observar que como en el laboratorios anteriores no hubo cambio alguno del valor del **View**, esto se debe a que no se esta invalidando la vista una vez que se da el cambio de valor dentro de la variable, caso similar como el visto en el laboratorio de **@State**, en este caso como tenemos un **ObservableObject** y un **@Published** dentro de la clase **CoursesViewModel**, utilizaremos **@ObservableObject** en la definición de la variable **coursesVM**. Siguiendo la estructura a continuación:

```
@ObservableObject var coursesVM = CoursesViewModel()
```

Paso 13: Compilamos y ejecutamos nuevamente, y esta vez podemos observar como al presionar el botón, el mensaje en el **View** cambia, esto porque se esta invalidando el valor del mismo, en cada cambio de valor de la variable. (ver figura 3).

Paso 14: Ahora vamos a hacer unas mejoras a la clase **CoursesViewModel** para asemejarnos a una aplicación mas real de la misma, ya que no es costumbre acceder las propiedades directo de un **ViewModel** sin utilizar los métodos, para lo cual agregamos el siguiente método.

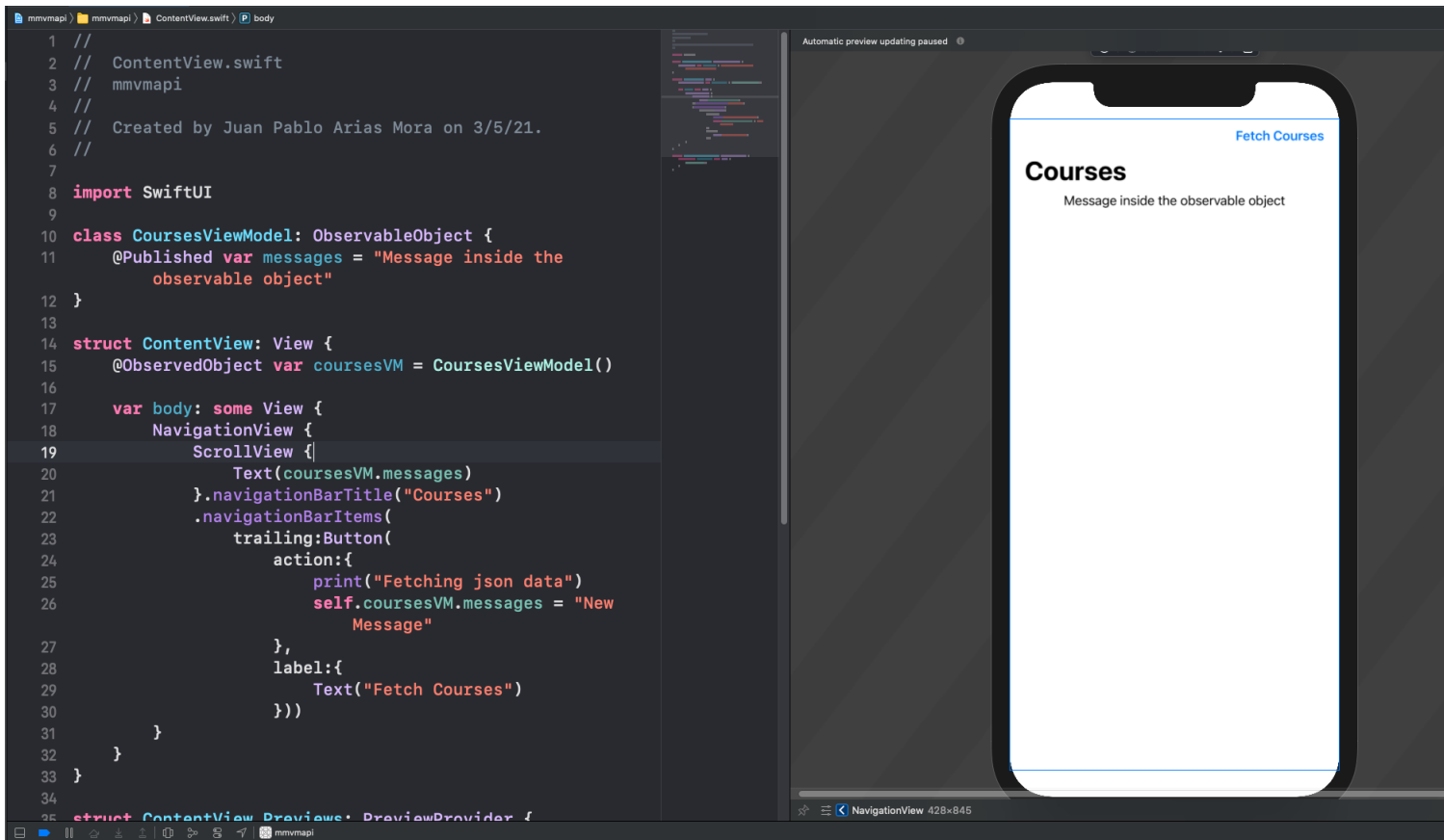


Figure 3: Observable en Ejecución invalidando View

```
func changeMessage(){
    self.messages = "New Message"
}
```

Paso 15: Luego reemplazaremos la asignación directa de `self.coursesVM.messages` por una llamada al método recién creado.

```
self.coursesVM.changeMessage()
```

Paso 16: Compilamos y ejecutamos nuevamente, solo para comprobar que el funcionamiento continua igual.

Paso 17: Agregamos el **struct Course**, para poder convertir los objetos **JSON** presentados al inicio en objetos dentro de nuestro código, algo interesante de notar es que los objetos no contienen un **id**, propiedad la cual nos facilita su presentación en el **View**, pero para ello podemos utilizar el **Keyword Identifiable**.

```
struct Course: Identifiable, Codable {
    let id = UUID()
    let name: String
    let bannerUrl: String
    let price: Int
}
```

Paso 18: Para fines de ejemplificar como inicializar valores dentro de objetos que provienen de un API, agreguemos el siguiente código justo después de la definición de la variable `messages` de **CoursesViewModel**

```
@Published var courses: [Course] = [
    .init(name:"Course1",
          bannerUrl:"http://localhost.com",
          price:30
    ),
    .init(name:"Course2",
```

```

        bannerUrl:"http://localhost2.com",
        price:50
    )
]

```

Paso 19: Identificamos la línea del código que contiene **Text(coursesVM.messages)** y reemplazamos su aparición con el siguiente código:

```

ForEach(self.coursesVM.courses){ course in
    VStack{
        HStack{
            Text(course.name)
            Text(String(course.price))
        }
    }
}

```

Paso 20: Podemos refrescar el **Text(Preview)** y veremos como el mismo contiene los valores que ante definimos sobre los nombres de curso y su valor. (ver figura 4).

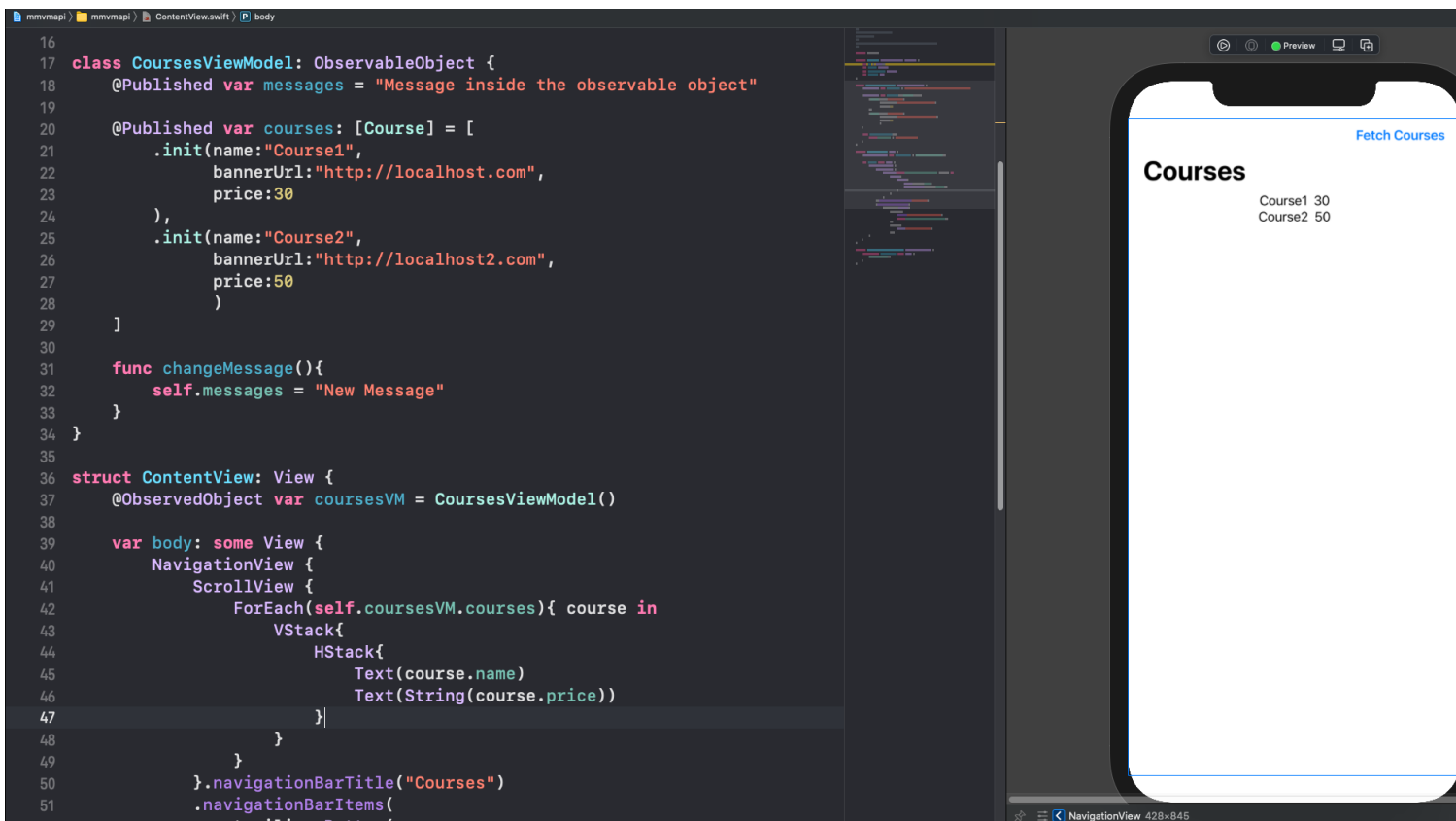


Figure 4: Default Values en Listas y Views

Paso 21: Ahora agregaremos la función usual para hacer llamados al **API** dentro de la clase **CoursesViewModel**

```

func fetchCourses(){
    guard let url = URL(string: "https://api.letsbuildthatapp.com/static/courses.json") 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([Course].self, from: data) {
                DispatchQueue.main.async {
                    self.courses = response
                }
                return
            }
        }
    }.resume()
}

```

Paso 22: Reemplazamos la aparición de `self.coursesVM.changeMessage()` por `self.coursesVM.fetchCourses()` (ver figura 5)

Paso 23: Compilamos nuevamente y ejecutamos, nuevamente presionaremos sobre el botón que trae los cursos, solamente que en esta ocasión la información de los mismos se desplegara en pantalla, similar a figura 6)

Paso 24: Para finalizar agregaremos una forma simple de mostrar imágenes en pantalla, aclarando que no es la que usualmente se pero suficiente para poder utilizar imágenes dentro del curso, esto debido a que para integrar una forma mas elegante tendríamos que introducir más temas dentro del curso, no se alarme si nota alguna redundancia dentro del mismo. Agregamos el siguiente **Extension**, el cual nos permite agregar una nueva propiedad al objeto **Image**. En su mayoría debería ser comprensible a alto nivel para lo visto en el curso.

```

extension Image {
    func data(url:URL) -> Self {
        if let data = try? Data(contentsOf: url) {
            guard let image = UIImage(data: data) else {
                return Image(systemName: "square.fill")
            }

            return Image(uiImage: image)
                .resizable()

        }
        return self
            .resizable()
    }
}

```

Paso 25: Luego dentro del **Foreach** agregamos el siguiente código a luego del **HStack** y dentro del **VStack**, compare el código con la figura 7 y 8

```

if let bannerURL = URL(string: course.bannerUrl) {
    Image(systemName: "square.fill").data(url: bannerURL)
        .frame(width: 200.0, height: 100.0)
}

```

Paso 26: Compilamos y ejecutamos, previo a la carga nos desplegará un corazón disponible dentro de la galería del sistema. Una vez presionamos el botón debería cargar las imágenes presentes en el **API**. (ver figura 9)

```

import SwiftUI

struct Course: Identifiable, Codable {
    let id = UUID()
    let name: String
    let bannerUrl: String
    let price: Int
}

class CoursesViewModel: ObservableObject {
    @Published var messages = "Message inside the observable object"

    @Published var courses: [Course] = [
        .init(name: "Course1",
              bannerUrl: "http://localhost.com",
              price: 30),
        .init(name: "Course2",
              bannerUrl: "http://localhost2.com",
              price: 50)
    ]

    func changeMessage(){
        self.messages = "New Message"
    }

    func fetchCourses(){
        guard let url = URL(string: "https://api.letsbuildthatapp.com/static/courses.json") 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([Course].self, from: data) {
                    DispatchQueue.main.async {
                        self.courses = response
                    }
                    return
                }
            }
        }.resume()
    }
}

struct ContentView: View {
    @ObservedObject var coursesVM = CoursesViewModel()

    var body: some View {
        NavigationView {
            ScrollView {
                ForEach(self.coursesVM.courses){ course in
                    VStack{
                        HStack{
                            Text(course.name)
                            Text(String(course.price))
                        }
                    }
                }
            }.navigationBarTitle("Courses")
            .navigationBarItems(
                trailing: Button(
                    action: {
                        print("Fetching json data")
                        self.coursesVM.fetchCourses()
                    },
                    label: {
                        Text("Fetch Courses")
                    })
            )
        }
    }
}

```

Figure 5: Código

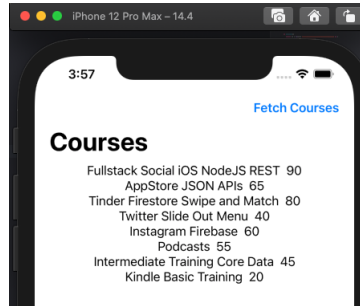


Figure 6: Código

```

17 extension Image {
18     func data(url:URL) -> Self {
19         if let data = try? Data(contentsOf: url) {
20             guard let image = UIImage(data: data) else {
21                 return Image(systemName: "square.fill")
22             }
23
24             return Image(uiImage: image)
25                 .resizable()
26
27         }
28         return self
29             .resizable()
30     }
31 }
32 }

```

Figure 7: Código 2

```

75 struct ContentView: View {
76     @ObservedObject var coursesVM = CoursesViewModel()
77
78     var body: some View {
79         NavigationView {
80             ScrollView {
81                 ForEach(self.coursesVM.courses){ course in
82                     VStack{
83                         HStack{
84                             Text(course.name)
85                             Text(String(course.price))
86                         }
87                         if let bannerURL = URL(string: course.bannerUrl) {
88                             Image(systemName: "square.fill").data(url: bannerURL)
89                                 .frame(width: 200.0, height: 100.0)
90                         }
91                     }
92                 }
93             }.navigationBarTitle("Courses")
94             .navigationBarItems(
95                 trailing:Button(
96                     action:{
97                         print("Fetching json data")
98                         self.coursesVM.fetchCourses()
99                     },
100                     label:{
101                         Text("Fetch Courses")
102                     })
103             )
104         }
105     }
106 }
107

```

Figure 8: Código 3

[Fetch Courses](#)

Courses

Fullstack Social iOS NodeJS REST 90



AppStore JSON APIs 65



Tinder Firestore Swipe and Match 80



Twitter Slide Out Menu 40



Instagram Firebase 60



Podcasts 55

Figure 9: Resultado Final