

## Programación en Móviles Avanzado

### LABORATORIO 14

### RESTFul JSON

CODIGO DEL CURSO:



<b>Alumno(s):</b>	<i>Andrade Chura Mary Carmen</i>					<b>Nota</b>	
<b>Grupo:</b>	<i>C-24 B</i>					<b>Ciclo: V</b>	
<b>Criterio de Evaluación</b>	<i>Excelente (4pts)</i>	<i>Bueno (3pts)</i>	<i>Regular (2pts)</i>	<i>Requiere mejora (1pts)</i>	<i>No acept. (0pts)</i>	<b>Puntaje Logrado</b>	
Instala y configura JSONPlaceHolder para crear un servidor Json Web Service						<i>3</i>	
Utiliza comandos HTTP para el consumo de Web Service (GET, POST, PUT, DELETE)						<i>3</i>	
Desarrolla adecuadamente los ejercicios propuestos						<i>6</i>	
Realiza observaciones y conclusiones que aporten un opinión crítica y técnica						<i>3</i>	
Es puntual y redacta el informe adecuadamente sin copias de otros autores						<i>2</i>	
Evidencia avance en laboratorio						<i>3</i>	

**I.- OBJETIVOS:**

- Creación y Consumo de Servicios Web

**II.- SEGURIDAD:****Advertencia:**

**En este laboratorio está prohibida la manipulación del hardware, conexiones eléctricas o de red; así como la ingestión de alimentos o bebidas.**

**III.- FUNDAMENTO TEÓRICO:**

Revise sus diapositivas del tema antes del desarrollo del laboratorio.

**IV.- NORMAS EMPLEADAS:**

No aplica

**V.- RECURSOS:**

- OS. En este laboratorio cada alumno trará con un equipo con MAC OS.

**VI.- METODOLOGÍA PARA EL DESARROLLO DE LA TAREA:**

- El desarrollo del laboratorio es individual.

**VII.- PROCEDIMIENTO:****ACTIVIDADES:****OPCIONAL: CREACION DE JSON-SERVER**

1. Intentaremos consumir los recursos brindados por un JSON, el cual estará lleno por valores ya obtenidos de una base de datos X.
2. El servicio a configurar y/o utilizar será **JSONPlaceHolder**
3. Instale el gestor de paquetes **Homebrew**. En su pagina principal indica el comando a ejecutar para su instalación



4. Apertura un terminal de digite el siguiente comando  
`/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"`

```
https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
```

5. Espere hasta que termine el proceso de instalación

```
==> Cleaning up /Library/Caches/Homebrew...
==> Migrating /Library/Caches/Homebrew to /Users/dennisapaza/Library/Caches/Homebrew...
==> Deleting /Library/Caches/Homebrew...
Already up-to-date.
==> Installation successful!

==> Homebrew has enabled anonymous aggregate user behaviour analytics.
Read the analytics documentation (and how to opt-out) here:
https://docs.brew.sh/Analytics.html

==> Next steps:
- Run `brew help` to get started
- Further documentation:
  https://docs.brew.sh
denniss-Mac:~ dennisapaza$
```

6. Ahora utilice el paquete **grew** descargado para instalar **NODE**, con el siguiente comando  
**brew install npm**

7. Espere a que termine la descarga e instalación del paquete NODE

```
==> Summary
🍺 /usr/local/Cellar/icu4c/61.1: 249 files, 67.2MB
==> Installing node
==> Downloading https://homebrew.bintray.com/bottles/node-10.4.0.high_sierra.bottle.tar.gz
#####
100.0%
==> Pouring node-10.4.0.high_sierra.bottle.tar.gz
==> Caveats
Bash completion has been installed to:
  /usr/local/etc/bash_completion.d
==> Summary
🍺 /usr/local/Cellar/node/10.4.0: 6,793 files, 59.6MB
denniss-Mac:~ dennisapaza$
```

8. Ahora proceda a instalar la aplicación JSON-SERVER que permitirá implementar un REST API JSON.
9. En la pagina oficial de oficial de dicha aplicación <https://github.com/typicode/json-server> se muestran los pasos a seguir para su configuración.
10. En su terminal digite el siguiente comando para empezar con la descarga e instalacion del mismo  
**npm install -g json-server**

11. Espere a que termine de descargar



```
denniss-Mac:~ dennisapaza$ npm install -g json-server
/usr/local/bin/json-server -> /usr/local/lib/node_modules/json-server/bin/index.js
+ json-server@0.14.0
added 229 packages from 130 contributors in 35.853s
denniss-Mac:~ dennisapaza$
```

12. Proceda a crear el archivo **db.json**. Para esto invoque el siguiente comando  
**json-server --watch db.json**

```
[MacBook-Pro-de-Macintosh:~ macintosh$ json-server --watch db.json
\{^_^\}/ hi!
Loading db.json
Oops, db.json doesn't seem to exist
Creating db.json with some default data

Done

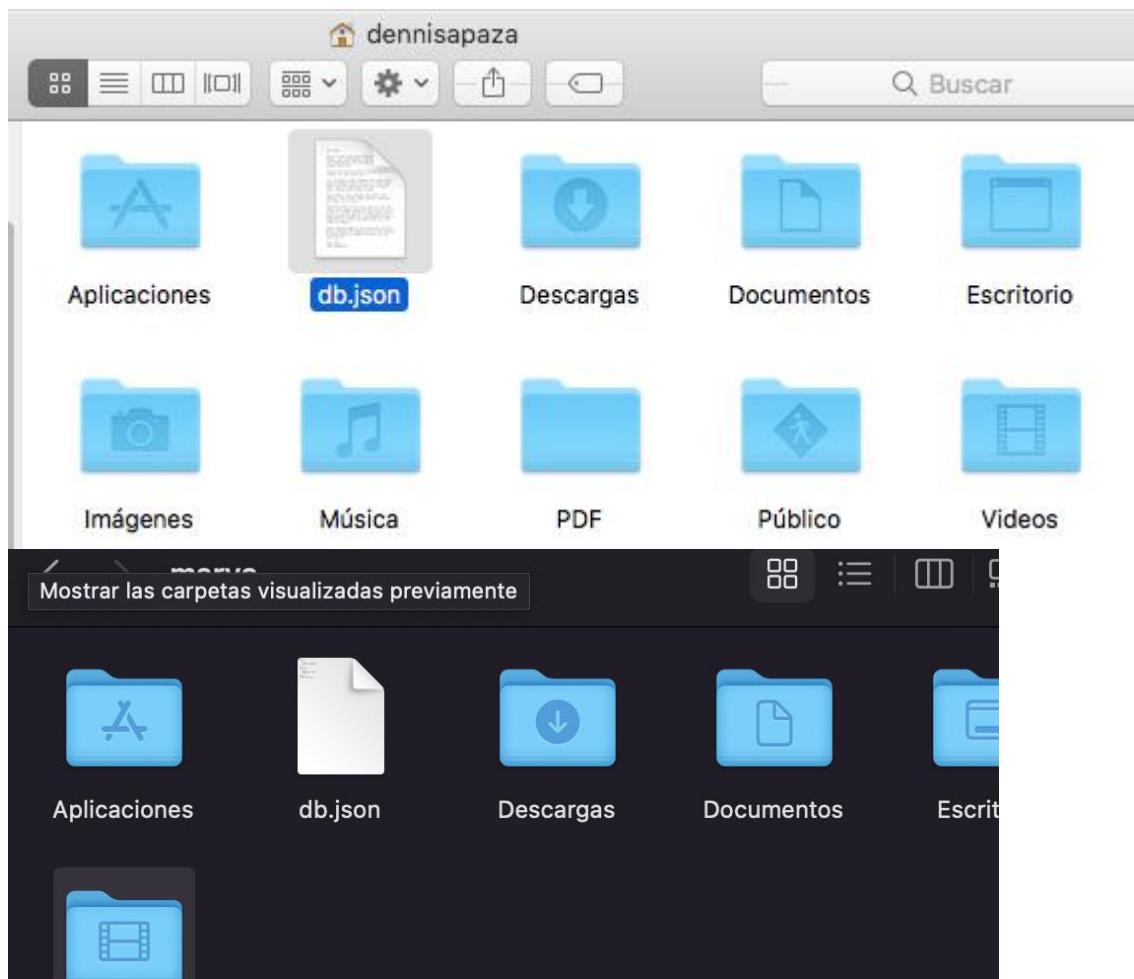
Resources
http://localhost:3000/posts
http://localhost:3000/comments
http://localhost:3000/profile

Home
http://localhost:3000

Type s + enter at any time to create a snapshot of the database
Watching...
```

Note que se le indica en que puerto se publico el servicio y la lista de rutas de demostracion creadas.

13. Presione **Control + C** para terminar el proceso del servidor
14. Acceda a la carpeta general de su usuario (en mi caso **dennisdeah**) y verifique que se creo un archivo **db.json**



15. Abra este **archivo db.json** en un editor de texto y reemplace su contenido por el que se muestra a continuación

Prof. Dennis A.

```
{  
    "usuarios": [  
        { "id": 1, "nombre": "dennis", "clave": "12345678", "email":  
"dennis2deah@gmail.com" },  
        { "id": 2, "nombre": "usuario1", "clave": "usuario1234", "email":  
"usuario1@empresa.com" },  
        { "id": 3, "nombre": "usuario2", "clave": "usuario1234", "email":  
"usuario2@empresa.com" }  
    ],  
    "peliculas": [  
        { "usuarioId": 1, "id": 1, "nombre": "Los vengadores", "genero": "Accion" ,  
"duracion" : "120" },  
        { "usuarioId": 1, "id": 2, "nombre": "Armagedon", "genero": "Aventura" ,  
"duracion" : "150" },  
        { "usuarioId": 1, "id": 3, "nombre": "Son como ninos 1", "genero": "Comedia" ,  
"duracion" : "140" },  
        { "usuarioId": 2, "id": 4, "nombre": "Son como ninos 2", "genero": "Comedia" ,  
"duracion" : "120" },  
        { "usuarioId": 2, "id": 5, "nombre": "Piratas del caribe", "genero":  
"Aventura" , "duracion" : "100" },  
        { "usuarioId": 2, "id": 6, "nombre": "Cadena perpetua", "genero": "Suspeno" ,  
"duracion" : "90" },  
        { "usuarioId": 3, "id": 7, "nombre": "De mendigo a millonario", "genero":  
"Comedia" , "duracion" : "80" },  
        { "usuarioId": 3, "id": 8, "nombre": "El aro", "genero": "Terror" ,  
"duracion" : "110" },  
        { "usuarioId": 3, "id": 9, "nombre": "Toy story", "genero": "Comedia" ,  
"duracion" : "90" }  
    ],  
    "comentarios": [  
        { "peliculaId": 1 , "id": 1, "comentario": "Pelicula recien estrenada este  
mes" },  
        { "peliculaId": 1 , "id": 2, "comentario": "Buena trama , pero un poco  
predecible el final" },  
        { "peliculaId": 2 , "id": 3, "comentario": "Pelicula con un triste final" },  
        { "peliculaId": 3 , "id": 4, "comentario": "Buena pelicula para verla en  
familia" }  
    ],  
    "profile": {  
        "name": "typicode"  
    }  
}
```

16. Guarde los cambios efectuados y diríjase a su terminal y vuelva digitar el comando **json-server --watch db.json** para inicializar su servicio JSON-SERVER

```
[MacBook-Pro-de-Macintosh:~ macintosh$ json-server --watch db.json

\{^_^\} hi!

Loading db.json
Done

Resources
http://localhost:3000/usuarios
http://localhost:3000/peliculas
http://localhost:3000/comentarios
http://localhost:3000/profile

Home
http://localhost:3000

Type s + enter at any time to create a snapshot of the database
Watching...

Resources
http://localhost:3000/usuarios
http://localhost:3000/peliculas
http://localhost:3000/comentarios
http://localhost:3000/profile

Home
http://localhost:3000

Type s + enter at any time to create a snapshot of the database
Watching...
```

17. Ahora si , desde su navegador acceda a <http://localhost:3000/usuarios> para ver el listado usuarios definidos

```
[
  {
    "id": 1,
    "nombre": "dennis",
    "clave": "12345678",
    "email": "dennis2deah@gmail.com"
  },
  {
    "id": 2,
    "nombre": "usuario1",
    "clave": "usuario1234",
    "email": "usuario1@tecsup.edu.pe"
  },
  {
    "id": 3,
    "nombre": "usuario2",
    "clave": "usuario1234",
    "email": "usuario2@tecsup.edu.pe"
  }
]
```

18. Verifique que al realizar cualquier consulta , esta es detectada por el servidor **JSON**

```
[MacBook-Pro-de-Macintosh:~ macintosh$ json-server --watch db.json

\{{^_^\}/ hi!

Loading db.json
Done

Resources
http://localhost:3000/usuarios
http://localhost:3000/peliculas
http://localhost:3000/comentarios
http://localhost:3000/profile

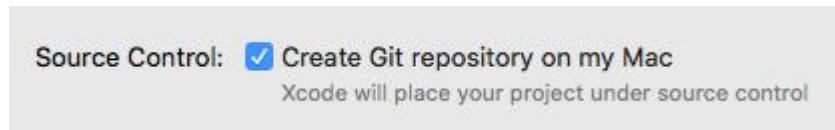
Home
http://localhost:3000

Type s + enter at any time to create a snapshot of the database
Watching...

GET /usuarios 200 10.394 ms - 334
```

## CREACION DE PROYECTO

19. Cree un proyecto Xcode de tipo **Single View App**, con las siguientes características:
  - a. Nombre: **JSONRESTful**.
20. Active la casilla de **Create Git repository on my Mac** para poder utilizar el cliente de repositorios **SouceTree**



21. En **main.storyboard** en el **viewController** creado por defecto diseñe una interfaz de logeo



**Usuario:**

**Contraseña:**

**Iniciar Sesión**

22. Para la parte de autentificación usaremos el recurso **/usuarios** provisto por la API ya mencionada.
23. Para la pantalla de logeo que se realizara, nos autentificaremos con **nombre**(como usuario) y **clave**(como contraseña)
24. Proceda a definir dicha estructura de datos. Para esto cree un nuevo archivo **Swift** denominado **Users.swift** y coloque el siguiente código

```
import Foundation
struct Users:Decodable{
    let id:Int
    let nombre:String
    let clave:String
    let email:String
}
```

25. En **ViewController.swift** defina los **outlets** respectivos para cada **textField** insertado, como se indica : **usuario(txtUsuario)**, **contraseña(txtContrasena)**. Y Defina una instancia de la estructura **Users** creada en el paso anterior

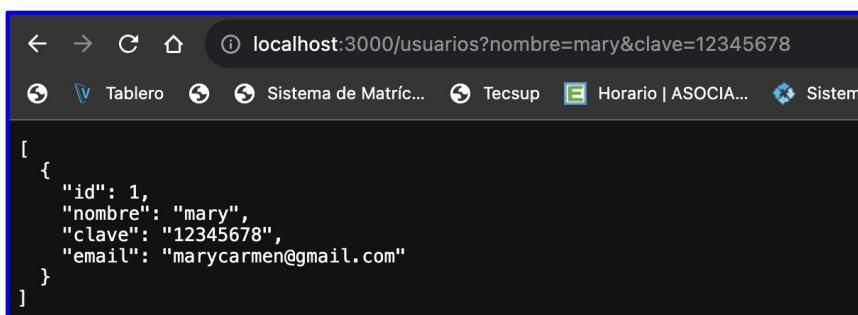
```
class ViewController: UIViewController {

    @IBOutlet weak var txtUsuario: UITextField!
    @IBOutlet weak var txtContrasena: UITextField!
    var users = [Users]()
```

26. Cree la función **validarUsuario()** que solicitará como parámetro la URL de destino en al cual se realizará la consulta del usuario y contraseña proporcionado. Si la consulta no devuelve ningún dato, retornará un array vacío, en caso contrario el resultado obtenido se reflejará en la estructura **Users.swift** creada previamente

```
func validarUsuario(ruta:String, completed: @escaping () -> ()) {
    let url = URL(string: ruta)
    URLSession.shared.dataTask(with: url!) { (data, response,
        error) in
        if error == nil{
            do{
                self.users = try JSONDecoder().decode([Users].self,
                    from: data!)
                DispatchQueue.main.async {
                    completed()
                }
            }catch{
                print("Error en JSON")
            }
        }
    }.resume()
}
```

27. Para prueba vamos a buscar un usuario específico para ver el resultado obtenido:  
<http://localhost:3000/usuarios?nombre=dennis&clave=12345678>

```
[  
 {  
     "id": 1,  
     "nombre": "dennis",  
     "clave": "12345678",  
     "email": "dennis2deah@gmail.com"  
 }  
]  


The screenshot shows a browser window with the URL localhost:3000/usuarios?nombre=dennis&clave=12345678. The page content displays a single JSON object in an array:

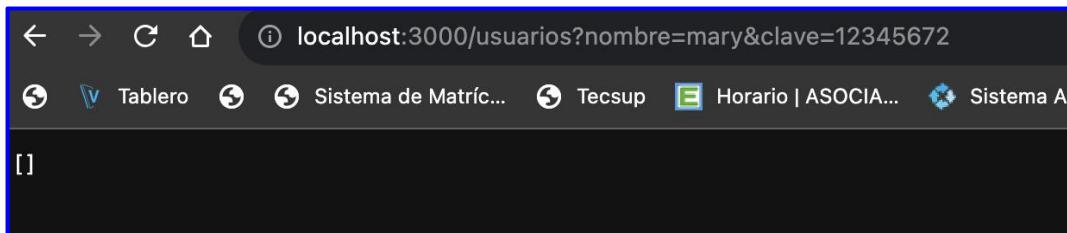


```
[  
 {  
     "id": 1,  
     "nombre": "dennis",  
     "clave": "12345678",  
     "email": "dennis2deah@gmail.com"  
 }  
]
```


```

28. Ahora pruebe colocar un dato incorrecto que no exista y verifique que se devuelve un array vacío, por ejemplo: <http://localhost:3000/usuarios?nombre=otro&clave=12345678>





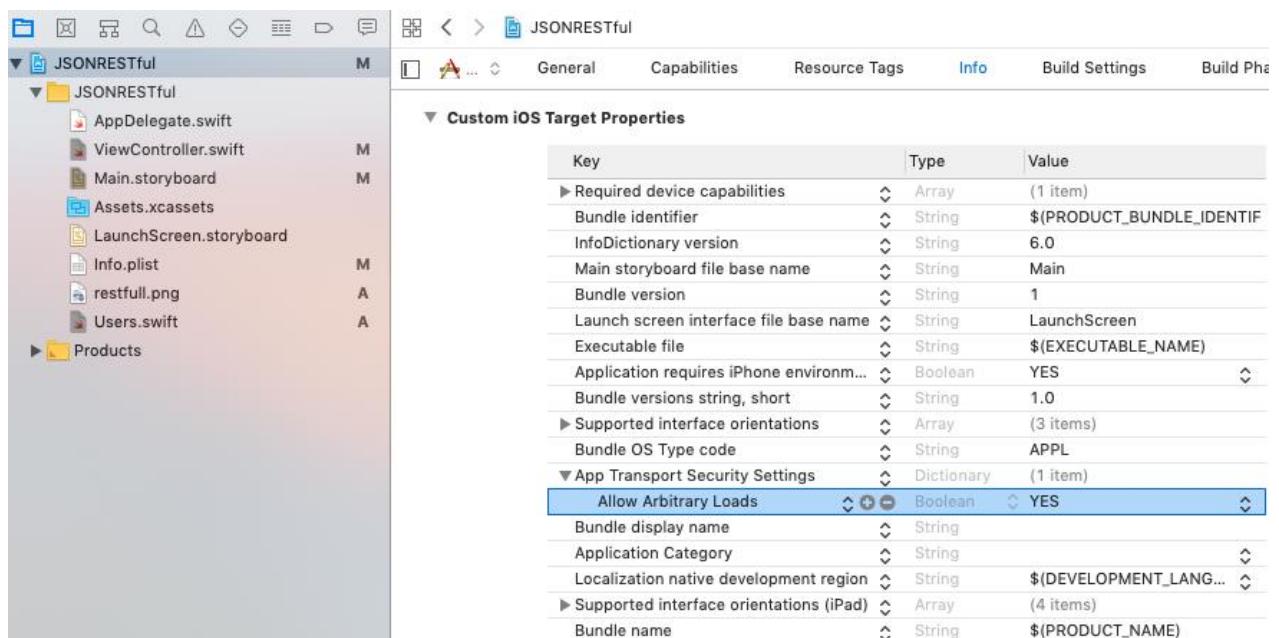
29. Cree un **Action** para el botón de “**Iniciar Sesión**” denominado **logear** que nos permita invocar a la función previamente creada y validar los datos de un usuario. Coloque el siguiente código

```
@IBAction func logear(_ sender: Any) {
    let ruta = "http://localhost:3000/usuarios?"
    let usuario = txtUsuario.text!
    let contrasena = txtContrasena.text!
    let url = ruta + "nombre=\(usuario)&clave=\(contrasena)"
    let crearURL = url.replacingOccurrences(of: " ", with: "%20")
    validarUsuario(ruta: crearURL) {
        if self.users.count <= 0{
            print("Nombre de usuario y/o contraseña es incorrecto")
        }else{
            print("Logeo Exitoso")

            for data in self.users{
                print("id:\(data.id), nombre:\(data.nombre), nombre:\(data.email)")
            }
        }
    }
}
```

30. Antes de ejecutar la aplicación conceda los permisos para poder usar el protocolo **http**.

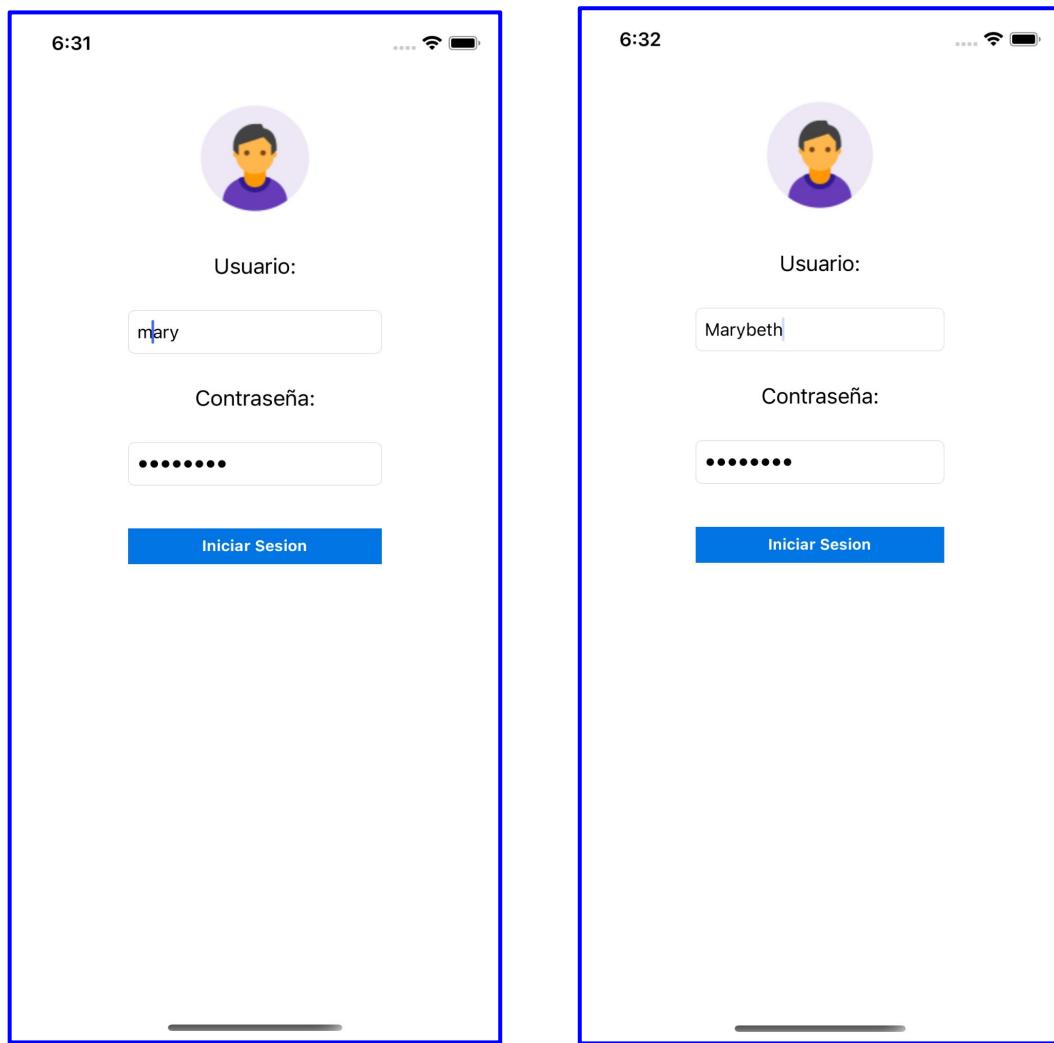
31. Haga clic en la carpeta general de su proyecto, luego haga clic en **TARGET(JSONRestful)**, diríjase a la pestaña de **INFO**, en la lista mostrada haga clic derecho sobre **Bundle OS Type code** y elija **Add Row**, en la lista mostrada busque el elemento **App Transport Security Setting**. Notara que al lado derecho de este elemento insertado aparece un botón de **+**, haga clic sobre este y en la lista mostrada seleccione la opción de **Allow Arbitrary Loads** con valor de **YES**. Debe quedar como se muestra



32. Ejecute la aplicación y pruebe logearse con un usuario existente en la base de datos y luego con datos incorrectos y verifique el resultado por consola

**Logeo Exitoso**  
**id:1,nombre:dennis,nombre:dennis2deah@gmail.com**

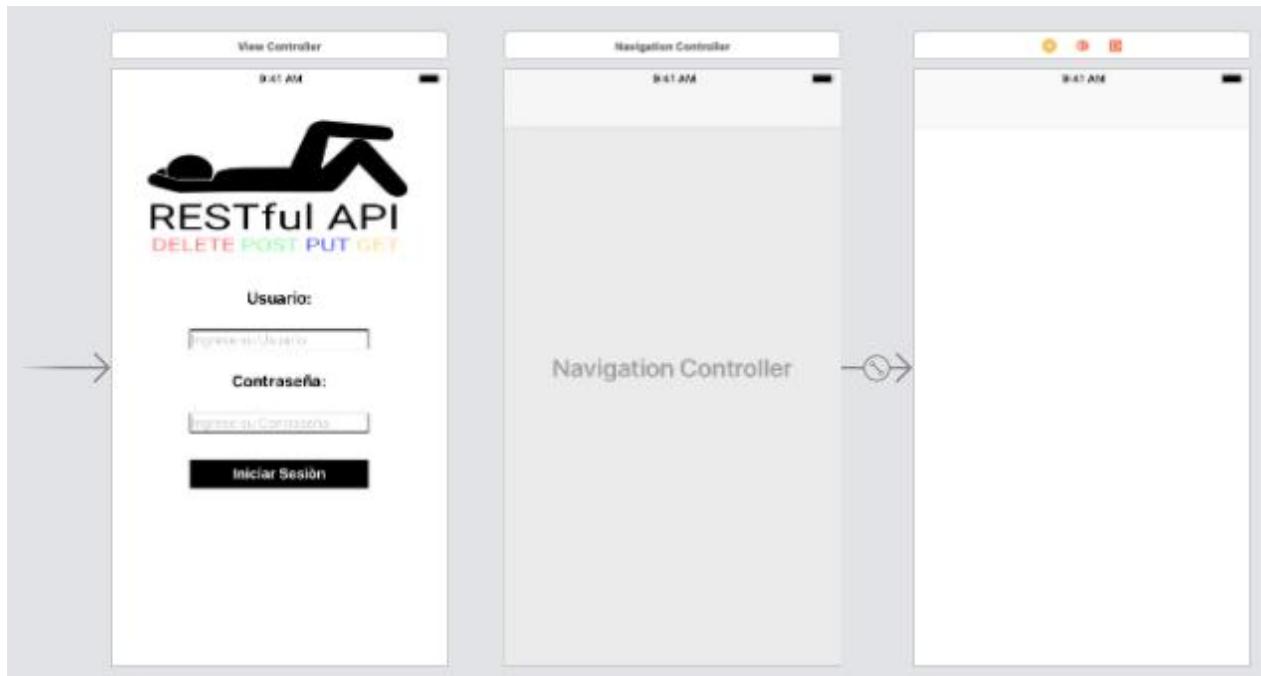
**Nombre de usuario y/o contraseña es incorrecto**



```
JSONRESTful Line: 21
t:], failed to fetch device property for senderID
(778835616971358211) use primary keyboard info instead.
Logeo exitoso
id:1,nombre:mary,nombre:marycarmen@gmail.com
Nombre de usuario y/o contraseña es incorrecto
```

Como podemos ver primero inicie sesion con un usuario ya existente y este se logeo de forma exitosa, pero cuando inicie sesion son un usuario que no tengo en mi JSON no se peude ingresar y nos envia en la termina un mensaje de que el usuario o contrsseña es incorrecta.

33. En **Main.storyboard**, agregue un nuevo **viewController** y agregue antes de este un **NavigationController**



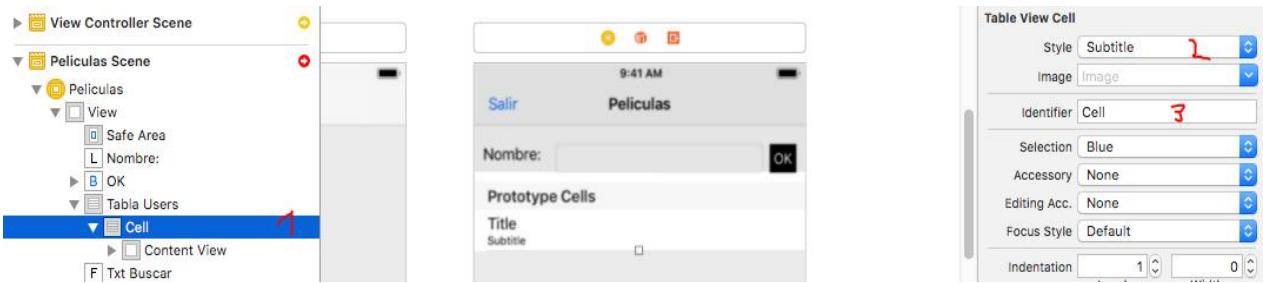
34. Agregue un título al nuevo **viewController** como se muestra



35. Diseñe el **viewController** como se muestra (**Bar button Item**, **Label**, **textField**, **Button**, **tableView**)



36. En el explorador de escenas seleccione el **tableView Cell** e ingrese al **Inspector de Atributos** y cambie el atributo **Style** a: **"Subtitle"**, y coloque como **Identifier** a: **Cell**



37. Cree un archivo **Cocoa Touch** denominado **viewControllerBuscar.swift** y asígnelo al nuevo **viewController** creado  
 38. Cree un nuevo archivo **Swift** denominado **Peliculas.swift** el cual servirá de estructura para cargar la lista de películas. Defina la siguiente estructura

```
import Foundation
struct Peliculas:Decodable{
    let usuarioId:Int
    let id:Int
    let nombre:String
    let genero:String
    let duracion:String
}
```

39. Cree una instancia de la estructura **Peliculas.swift** y defina un **outlet** para el **textField**(como **txtBuscar**) y un **Action** para el **button(OK** como **btnBuscar**) como se muestra

```
var peliculas = [Peliculas]()

@IBOutlet weak var txtBuscar: UITextField!

override func viewDidLoad() {
    super.viewDidLoad()
}

@IBAction func btnBuscar(_ sender: Any) {
```

40. En **viewControllerBuscar.swift** cree la función **cargarPelículas** que permitirá mostrar la lista de películas publicadas

```
func cargarPelículas(ruta:String, completed: @escaping () -> ()) {
    let url = URL(string: ruta)
    URLSession.shared.dataTask(with: url!) { (data, response, error) in
        if error == nil{
            do{
                self.películas = try JSONDecoder().decode([Películas].self, from: data!)
                DispatchQueue.main.async {
                    completed()
                }
            }catch{
                print("Error en JSON")
            }
        }
    }.resume()
}
```

41. Implemente los protocolos para manejo del **tableView**

```
class ViewControllerBuscar: UIViewController,
    UITableViewDelegate, UITableViewDataSource {
```

42. Implemente las funciones **numberOfRowsInSection** y **cellForRowAt** pertenecientes al llenado del **tableView**, como se muestra

```
func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) ->
    Int {
    return películas.count
}

func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) ->
    UITableViewCell {
    let cell = tableView.dequeueReusableCell(withIdentifier: "Cell", for:
        indexPath)
    cell.textLabel?.text = "\(películas[indexPath.row].nombre)"
    cell.detailTextLabel?.text = "Genero:\(películas[indexPath.row].genero)
        Duracion:\(películas[indexPath.row].duracion)"
    return cell
}
```

43. Cree un outlet para el **tableView** denominado: **tablaPelículas**

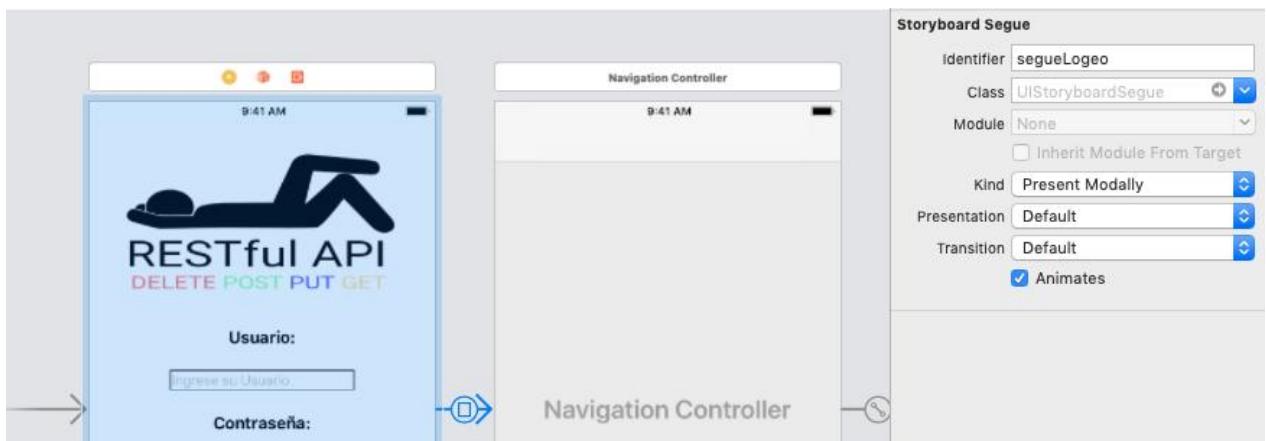
44. Implemente los **delegados** respectivos, y llame a la función **buscarPelícula** para llenar el **tableView** con la data recibida. Inserte estos datos en **viewDidLoad**

```
@IBOutlet weak var txtBuscar: UITextField!
@IBOutlet weak var tablaPeliculas: UITableView!

override func viewDidLoad() {
    super.viewDidLoad()
    tablaPeliculas.delegate = self
    tablaPeliculas.dataSource = self

    let ruta = "http://localhost:3000/peliculas/"
    cargarPeliculas(ruta: ruta) {
        self.tablaPeliculas.reloadData()
    }
}
```

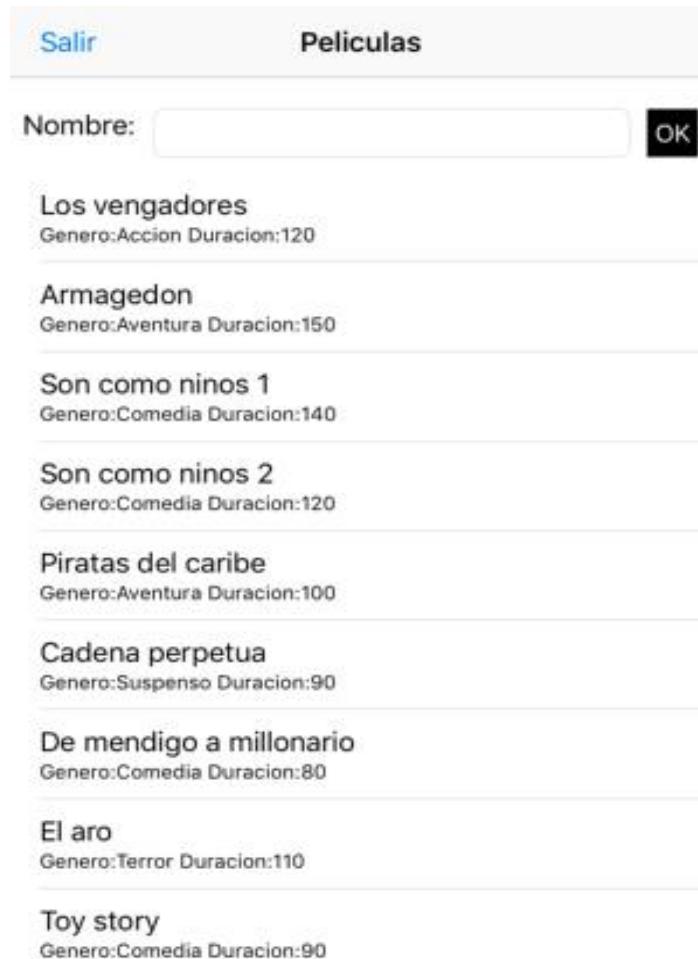
45. En **Main.storyboard**, cree un **segue** tipo **Present Modally** del **viewController** de logeo al **navigationController**. Y coloque como identificador del segue “**segueLogeo**”

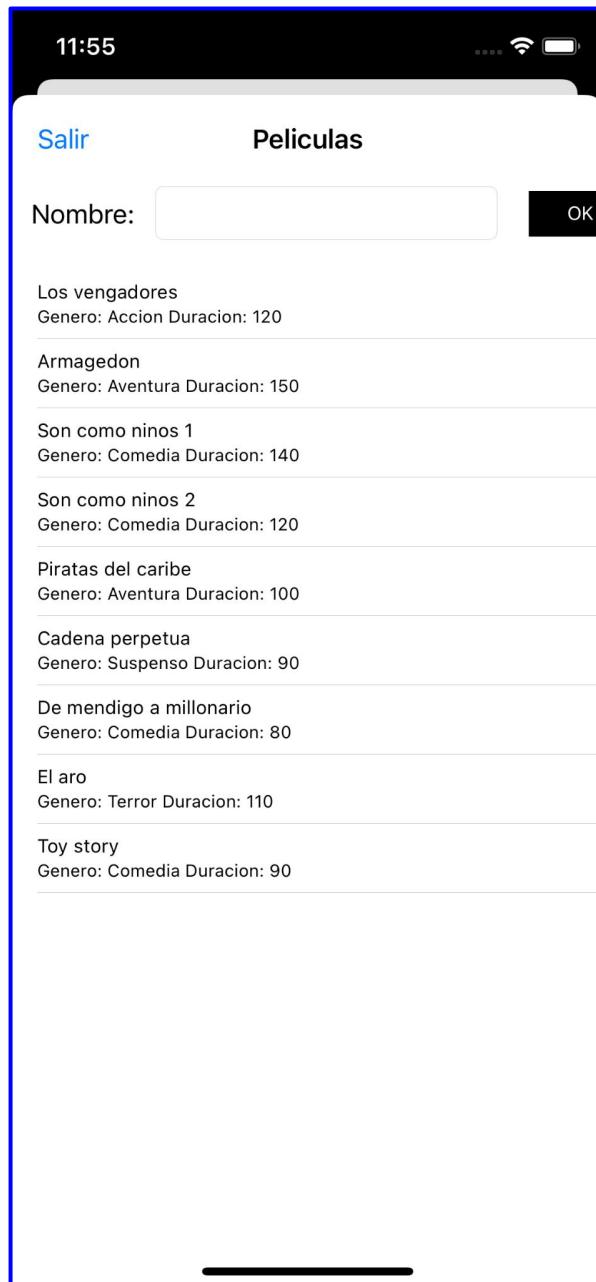


46. Modifique la función **logear** del **viewController** de logeo y agregue la siguiente línea para invocar al siguiente **viewController** en caso la autenticación sea correcta

```
@IBAction func logear(_ sender: Any) {
    let ruta = "http://localhost:3000/usuarios?"
    let usuario = txtUsuario.text!
    let contraseña = txtContrasena.text!
    let url = ruta + "nombre=\(usuario)&clave=\(contraseña)"
    let crearURL = url.replacingOccurrences(of: " ", with: "%20")
    validarUsuario(ruta: crearURL) {
        if self.users.count <= 0{
            print("Nombre de usuario y/o contraseña es incorrecto")
        }else{
            print("Logeo Exitoso")
            self.performSegue(withIdentifier: "segueLogeo", sender: nil)
            for data in self.users{
                print("id:\(data.id), nombre:\(data.nombre), nombre:\(data.email)")
            }
        }
    }
}
```

47. Ejecute la aplicación, autentifíquese con datos de un usuario y verifique que se muestra la lista de usuarios





48. Indique los detalles más importantes del código implementado hasta esta sección

En Peliculas.swift estamos estructurando los datos de película a decodificar desde JSON y en viewControllerBuscar.swift declaramos un array películas para almacenar las películas y conecta outlets para la interfaz de usuario (txtBuscar y tablaPelículas). En el método viewDidLoad, configura la tabla (tablaPelículas) para delegar y gestionar los datos, luego carga películas desde una URL local (<http://localhost:3000/peliculas/>). Despues realiza una solicitud a la URL proporcionada para obtener datos JSON de películas. Una vez obtenidos, decodifica los datos en un array de objetos Peliculas y actualiza la tabla.

Y como resultado despues de logearnos nos muestra los datos de nuestra API loca en una tabla.

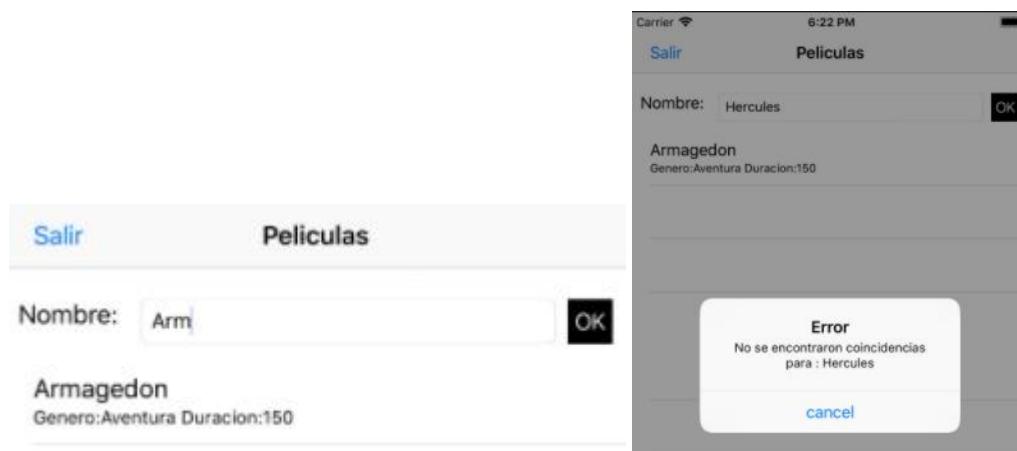
49. En **viewControllerBuscar.swift** implemente la función **mostrarAlerta()** que nos permitira mostrar si se encontro o no un elemento buscado

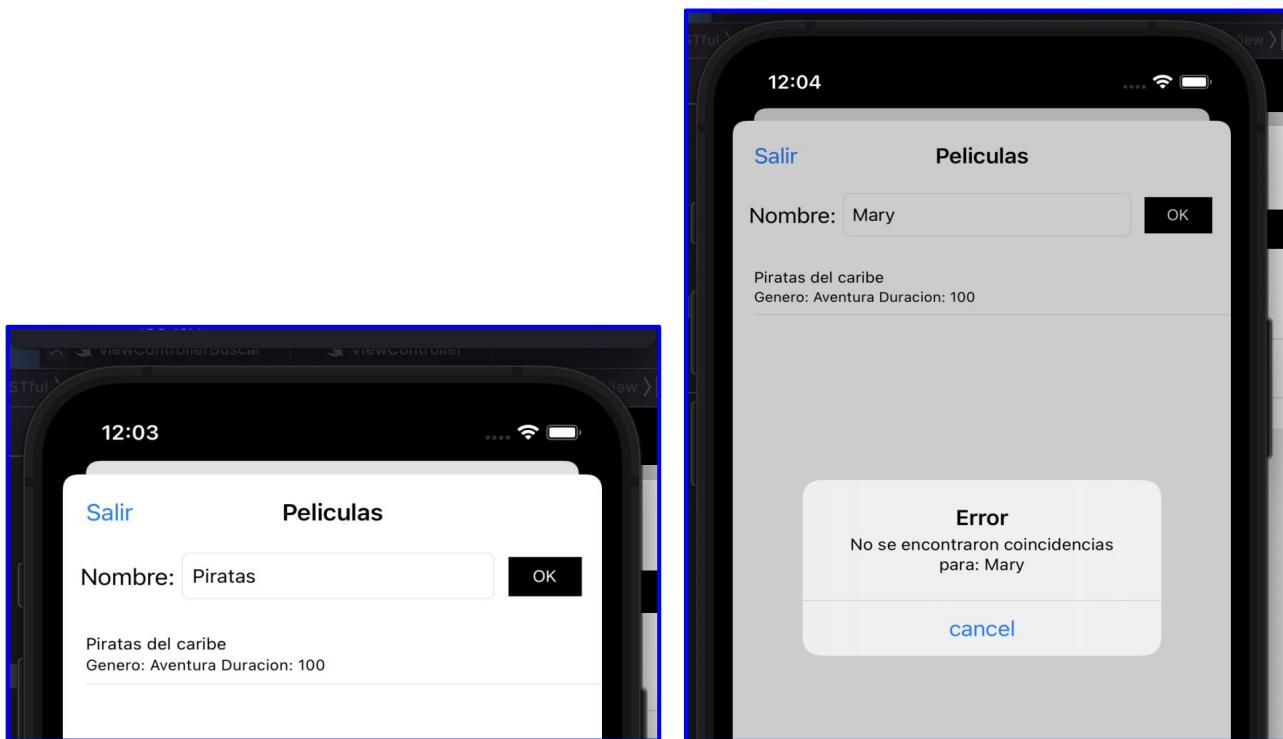
```
func mostrarAlerta(titulo: String, mensaje: String, accion: String) {  
    let alerta = UIAlertController(title: titulo, message: mensaje,  
        preferredStyle: .alert)  
    let btnOK = UIAlertAction(title: accion, style: .default, handler: nil)  
    alerta.addAction(btnOK)  
    present(alerta, animated: true, completion: nil)  
}
```

50. En la función **btnBuscar** implemente el código para buscar el nombre de un usuario, en caso se encuentre se mostrará en el **tableView**, en caso contrario se mostrará un mensaje indicando el error producido

```
@IBAction func btnBuscar(_ sender: Any) {  
    let ruta = "http://localhost:3000/peliculas?"  
    let nombre = txtBuscar.text!  
    let url = ruta + "nombre_like=\(nombre)"  
    let crearURL = url.replacingOccurrences(of: " ", with: "%20")  
  
    if nombre.isEmpty{  
        let ruta = "http://localhost:3000/peliculas/"  
        self.cargarPeliculas(ruta: ruta) {  
            self.tablaPeliculas.reloadData()  
        }  
    }else{  
        cargarPeliculas(ruta: crearURL) {  
            if self.peliculas.count <= 0{  
                self.mostrarAlerta(titulo: "Error", mensaje: "No se  
                encontraron coincidencias para : \(nombre)", accion:  
                "cancel")  
            }else{  
                self.tablaPeliculas.reloadData()  
            }  
        }  
    }  
}
```

51. Ejecute la aplicación y verifique el funcionamiento





52. Indique los detalles más importantes del código implementado hasta esta sección

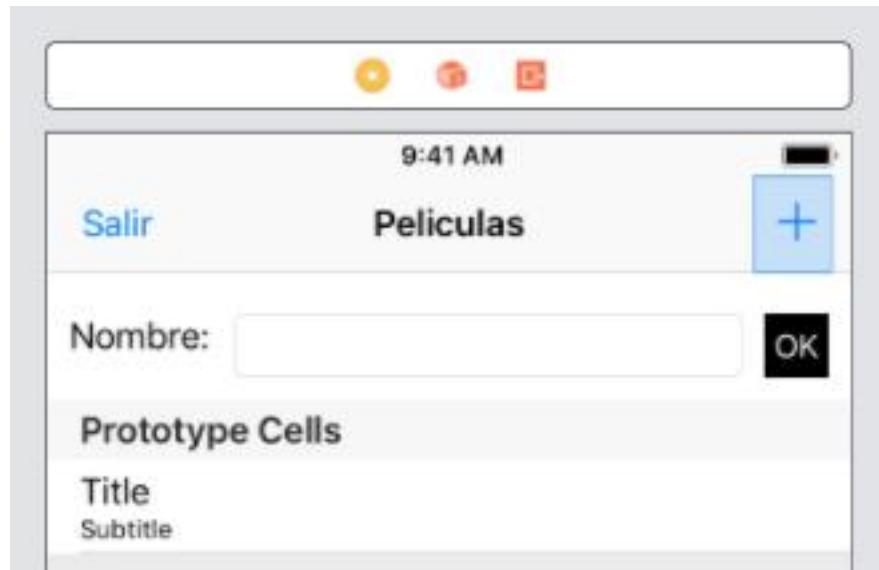
Lo que hicimos fue definir una función `btnBuscar` que se activa al presionar un botón de búsqueda. La función construye una URL según el texto ingresado en `txtBuscar`, consulta la API y actualiza la tabla de películas en consecuencia. Si no se encuentra ninguna coincidencia, se muestra una alerta utilizando la función `mostrarAlerta`. Esta última función crea y muestra una alerta con un título, un mensaje y un botón de acción para manejar la situación cuando no se encuentran coincidencias.

53. Cree un **Action** para el botón de Salir denominado: **btnSalir** e implemente el código para cerrar y volver al anterior **viewController**

```
@IBAction func btnSalir(_ sender: Any) {
    dismiss(animated: true, completion: nil)
}
```

**COMANDOS POST**

54. Se pasará a insertar elementos usando la **RESTApi** creada.  
55. Diríjase a **main.storyboard** y agregue un **bar button item** sobre el **viewController** que permite buscar un elemento



56. Agregue un nuevo **viewController** y diseñe el siguiente modelo

57. Cree un archivo **Cocoa Touch** denominado **viewControllerAgregar.swift** y asócielo al nuevo **viewController** creado.

58. Cree un **segue** tipo **Show** desde botón + del **viewControllerBuscar** hacia el nuevo **viewController** creado(**viewControllerAgregar**) que servirá para insertar nuevas películas. Coloque como nombre de **identificador** del segue creado: **segueAgregar**

59. Cree **Outlets** para los **textFields** insertados con el nombre que indica la siguiente tabla

Objeto	Nombre Outlet
textField Nombre	txtNombre
textField Genero	txtGenero
textField Duracion	txtDuracion

60. Cree un **Action** para el botón Guardar denominado **btnGuardar**

61. El código resultante debe ser como el que se muestra

```
import UIKit

class viewControllerAgregar: UIViewController {

    @IBOutlet weak var txtNombre: UITextField!
    @IBOutlet weak var txtGenero: UITextField!
    @IBOutlet weak var txtDuracion: UITextField!

    override func viewDidLoad() {
        super.viewDidLoad()

    }

    @IBAction func btnGuardar(_ sender: Any) {
    }

}
```

62. Cree la función **metodoPOST** que permitirá solicitar una **URL**(dirección del nodo en el cual se desea insertar) y **datos**(elemento a insertar). Coloque el siguiente código

```
func metodoPOST(ruta:String, datos:[String:Any])  {
    let url : URL = URL(string: ruta)!
    var request = URLRequest(url: url)
    let session = URLSession.shared
    request.httpMethod = "POST"
    // this is your input parameter dictionary
    let params = datos

    do{
        request.httpBody = try JSONSerialization.data(withJSONObject: params, options:
            JSONSerialization.WritingOptions.prettyPrinted)
    }
    catch
    {
        // catch any exception here
    }
    request.addValue("application/json", forHTTPHeaderField: "Content-Type")
    request.addValue("application/json", forHTTPHeaderField: "Accept")
    let task = session.dataTask(with: request, completionHandler:
        {(data,response,error) in
            if (data != nil)
            {
                do{
                    let dict = try JSONSerialization.jsonObject(with: data!, options:
                        JSONSerialization.ReadingOptions.mutableLeaves)
                    print(dict);
                }
                catch
                {
                    // catch any exception here
                }
            }
        })
    task.resume()
}
```

63. Dentro de la función **btnGuardar** coloque el siguiente código para realizar un **POST** sobre la API publicada

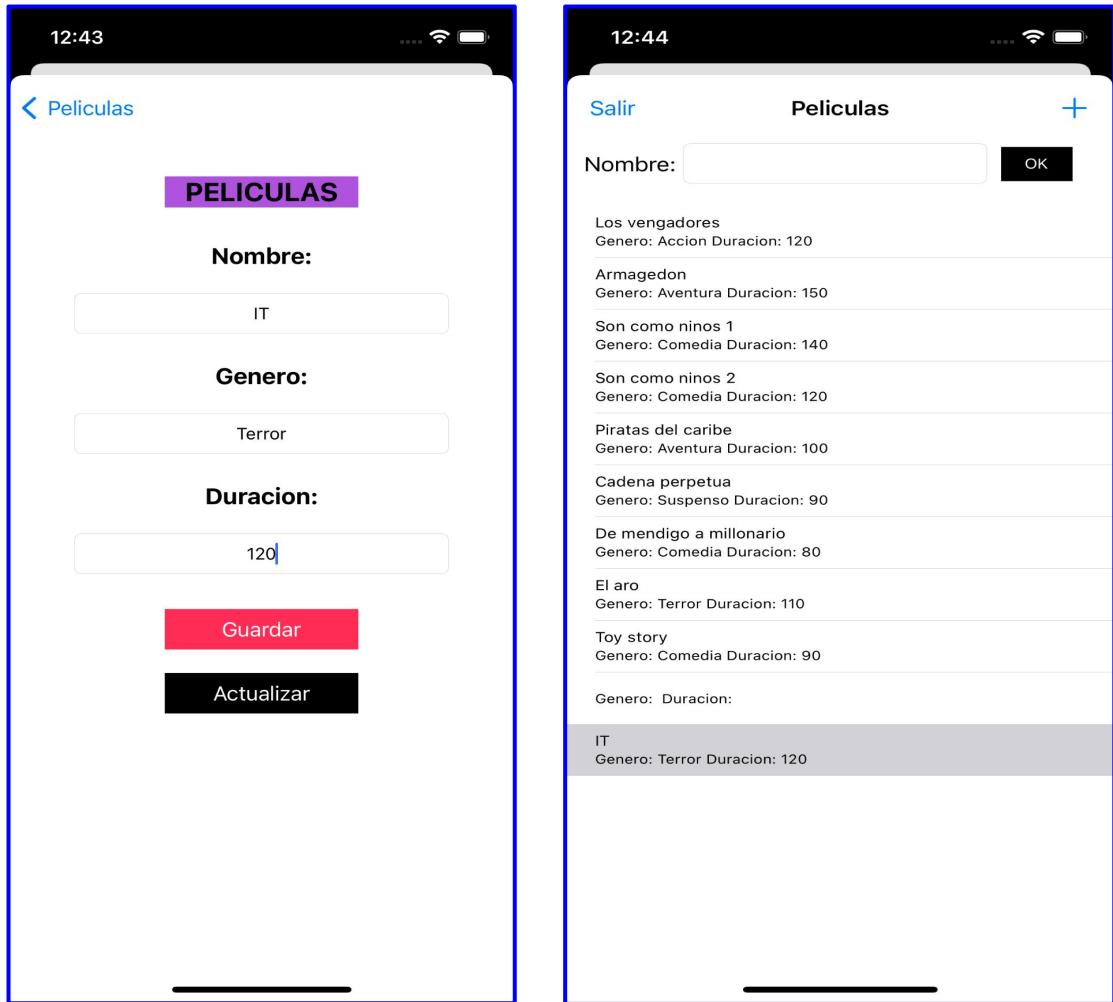
```
@IBAction func btnGuardar(_ sender: Any) {
    let nombre = txtNombre.text!
    let genero = txtGenero.text!
    let duracion = txtDuracion.text!
    let datos = ["userId": 1, "nombre": "\(nombre)", "genero": "\(genero)", "duracion": "\(duracion)"] as Dictionary<String, Any>
    let ruta = "http://localhost:3000/peliculas"
    metodoPOST(ruta: ruta, datos: datos)
    navigationController?.popViewController(animated: true)
}
```

64. En el archivo **viewControllerBuscar.swift** implemente el método `viewWillAppear` para que recarga la tabla de la vista cada vez que se agrege una nueva pelicula

```
override func viewWillAppear(_ animated: Bool) {
    let ruta = "http://localhost:3000/peliculas/"
    cargarPeliculas(ruta: ruta) {
        self.tablaPeliculas.reloadData()
    }
}
```

65. Ejecute la aplicación y verifique que puede insertar un elemento nuevo, y que el cambio se refleje en la lista principal





## METODO PUT

66. En **viewControllerAgregar.swift** cree un **Action** para el botón actualizar con el nombre **btnActualizar** y cree **outlets** para los botones **Guardar**(botonGuardar) y **Actualizar**(botonActualizar).
- Cree una variable denominada **pelicula** que sea del tipo de la clase **Peliculas**(Esta variable permitira almacenar la informacion de una pelicula seleccionada).
  - Modifique el código de la función **viewDidLoad** para determinar cuando debe permitirse usar el boton **Guardar**(cuando se crea una nueva pelicula) y el boton **Actualizar**(cuando se edita la informacion de una pelicula seleccionada).

```
@IBOutlet weak var botonGuardar: UIButton!
@IBOutlet weak var botonActualizar: UIButton!

var pelicula:Peliculas?

override func viewDidLoad() {
    super.viewDidLoad()
    if pelicula == nil{
        botonGuardar.isEnabled = true
        botonActualizar.isEnabled = false
    }else{
        botonGuardar.isEnabled = false
        botonActualizar.isEnabled = true
        txtNombre.text = pelicula!.nombre
        txtGenero.text = pelicula!.genero
        txtDuracion.text = pelicula!.duracion
    }
}

@IBAction func btnActualizar(_ sender: Any) {
```

67. Implemente la función **metodoPUT** el cual permitirá actualizar los datos de una película previamente seleccionada (**OJO**: esta función es la misma que el **metodoPOST** solo se cambió el tipo de consulta o metodo a realizar **PUT**)

```
func metodoPUT(ruta:String, datos:[String:Any]) {
    let url : URL = URL(string: ruta)!
    var request = URLRequest(url: url)
    let session = URLSession.shared
    request.httpMethod = "PUT"
    // this is your input parameter dictionary
    let params = datos

    do{
        request.httpBody = try JSONSerialization.data(withJSONObject: params, options:
            JSONSerialization.WritingOptions.prettyPrinted)
    }
    catch
    {
        // catch any exception here
    }
    request.addValue("application/json", forHTTPHeaderField: "Content-Type")
    request.addValue("application/json", forHTTPHeaderField: "Accept")
    let task = session.dataTask(with: request, completionHandler: {(data,response,error)
        in
        if (data != nil)
        {
            do{
                let dict = try JSONSerialization.jsonObject(with: data!, options:
                    JSONSerialization.ReadingOptions.mutableLeaves)
                print(dict);
            }
            catch
            {
                // catch any exception here
            }
        }
    })
    task.resume()
}
```

68. Implemente código para la función **btnActualizar** como se muestra

```
@IBAction func btnActualizar(_ sender: Any) {
    let nombre = txtNombre.text!
    let genero = txtGenero.text!
    let duracion = txtDuracion.text!
    let datos = ["usuarioId": 1, "nombre": "\(nombre)", "genero": "\(genero)", "duracion":
        "\(duracion)"] as Dictionary<String, Any>
    let ruta = "http://localhost:3000/peliculas/\(pelicula!.id)"
    metodoPUT(ruta: ruta, datos: datos)
    navigationController?.popViewController(animated: true)
}
```

69. Diríjase a **main.storyboard** y cree un **segue** desde el ícono amarillo del **viewControllerBuscar** hacia el **viewControllerAgregar** que sirve para **agregar/editar** películas, y coloque como identificador del segue creado: **segueEditar**

70. Vuelva al **viewControllerBuscar** en el que se listan las películas e implemente el método **didSelectRowAt**

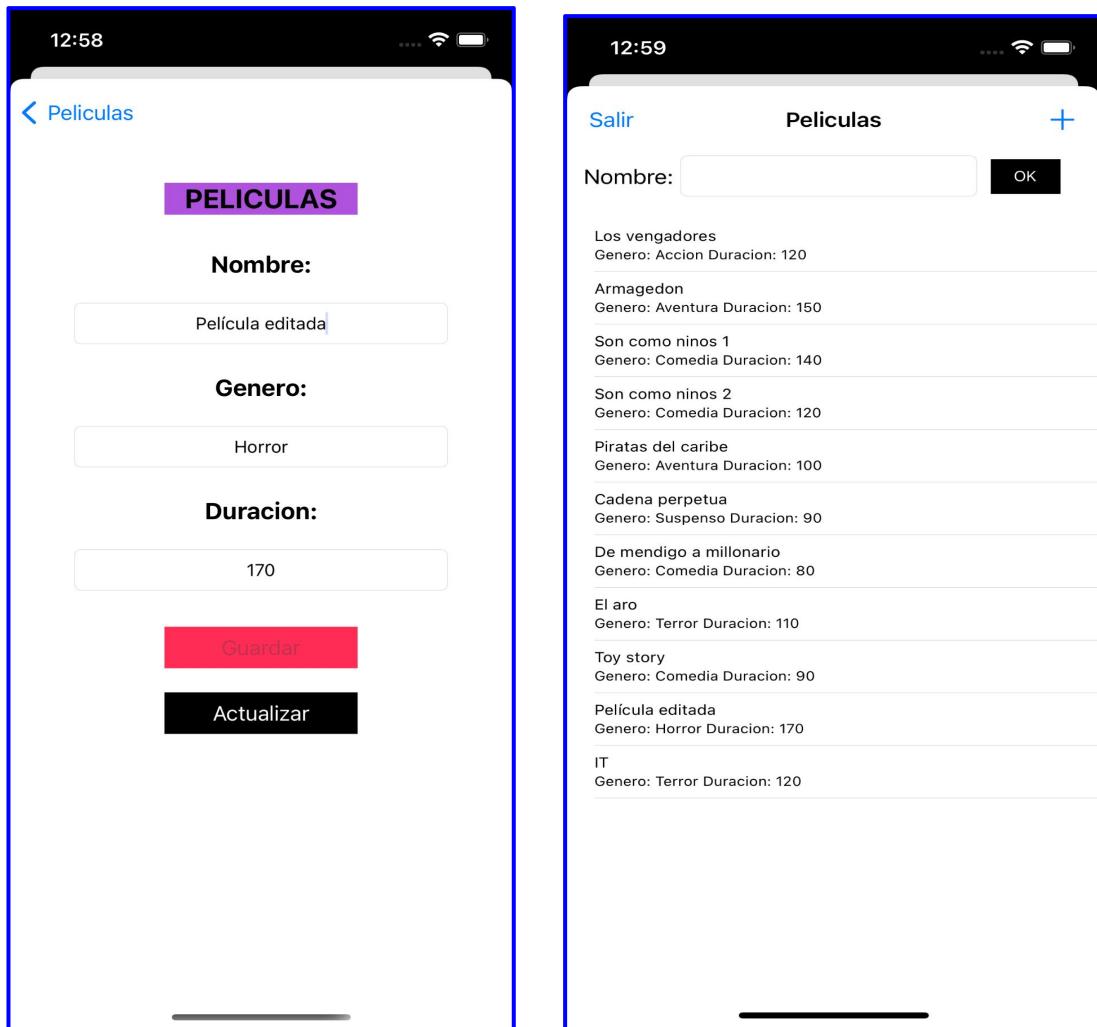
```
func tableView(_ tableView: UITableView, didSelectRowAt indexPath: IndexPath) {  
    let pelicula = peliculas[indexPath.row]  
    performSegue(withIdentifier: "segueEditar", sender: pelicula)  
}
```

71. Implemente el método **prepare for segue** para configurar el envío de la datos de la película seleccionada

```
override func prepare(for segue: UIStoryboardSegue, sender: Any?) {  
    if segue.identifier == "segueEditar" {  
        let siguienteVC = segue.destination as! ViewControllerAgregar  
        siguienteVC.pelicula = sender as? Peliculas  
    }  
}
```

72. Ejecute la aplicación y verifique su funcionamiento. Elija un elemento y pruebe que se puede actualizar la información de un elemento seleccionado

The screenshot displays a mobile application interface for editing movie details. At the top, a green header bar contains the title "Películas". Below it is a white form with three input fields and two buttons. The first field is labeled "Nombre" and contains the text "La Casa de Papel 2". The second field is labeled "Genero" and contains the text "Acción". The third field is labeled "Duracion" and contains the text "150". Below these fields are two buttons: a blue "Guardar" button and a grey "Actualizar" button. To the right of the form, a summary section displays the movie's name, genre, and duration. The summary text reads: "La Casa de Papel 2" followed by "Genero: Acción Duracion: 150".



### 73. Comente lo mas importante del código implementado

@IBAction func btnGuardar inicia el proceso de guardado de datos, metodoPOST se utiliza para crear nuevas entradas de película y metodoPUT para actualizar las existentes en una API mediante solicitudes HTTP POST y PUT respectivamente, manejando la serialización y la respuesta de datos JSON.

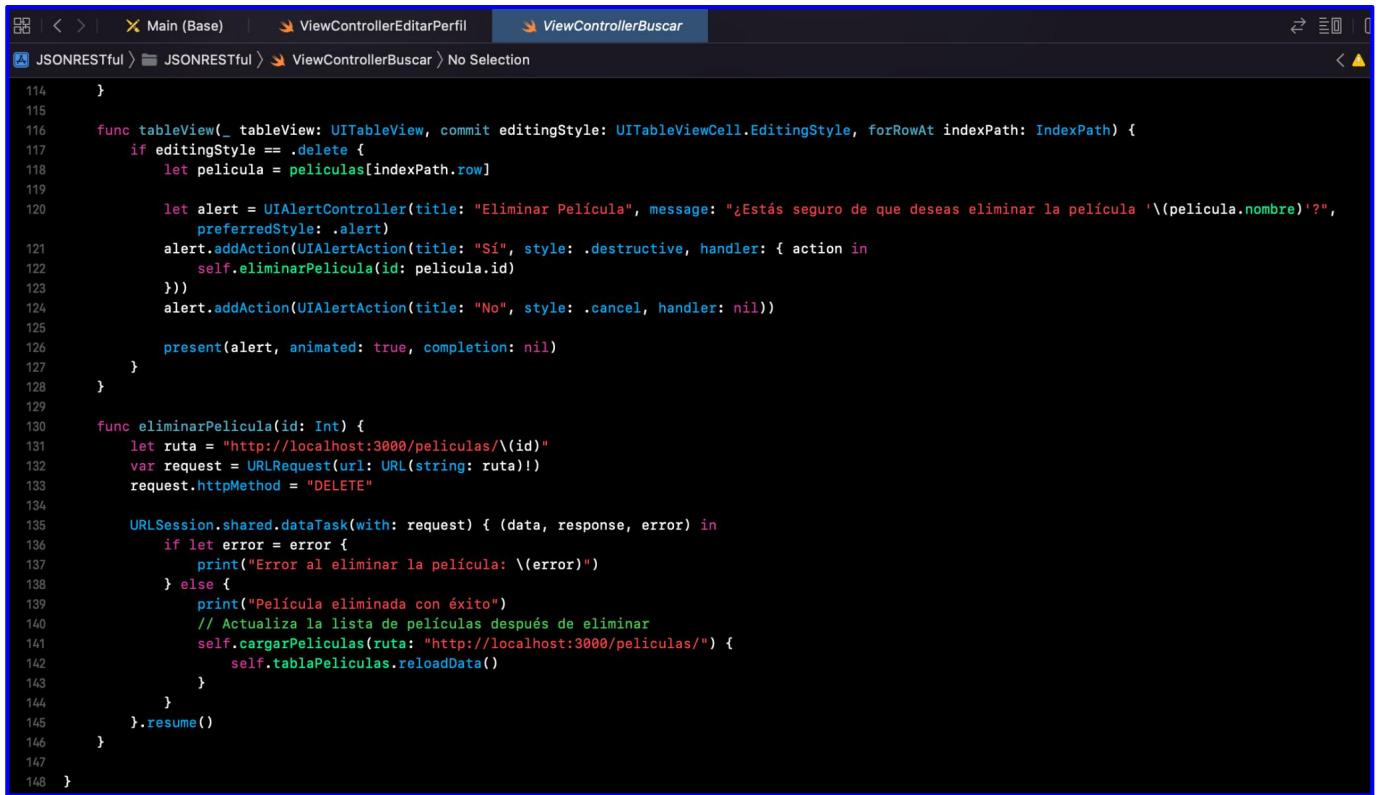
**TAREA:**

Investigue e implemente la funcionalidad de eliminar un elemento, en este caso una película  
Implemente esta funcionalidad con la opción de **editingStyle**, pero debe mostrar una alerta de si se desea  
eliminar la película(Sí o NO)

Implemente en **viewControllerBuscar** un **Bar Button Item** denominado “**Editar Perfil**” que permita editar en otro  
nuevo **ViewController** los datos del usuario logeado(nombre, clave, email)

**viewControllerBuscar.swift**

Hemos agregado la función de **eliminarPelícula** y la función **editingStyle** para poder eliminar una  
película y además mostrar una alerta para aceptar eliminar o no.

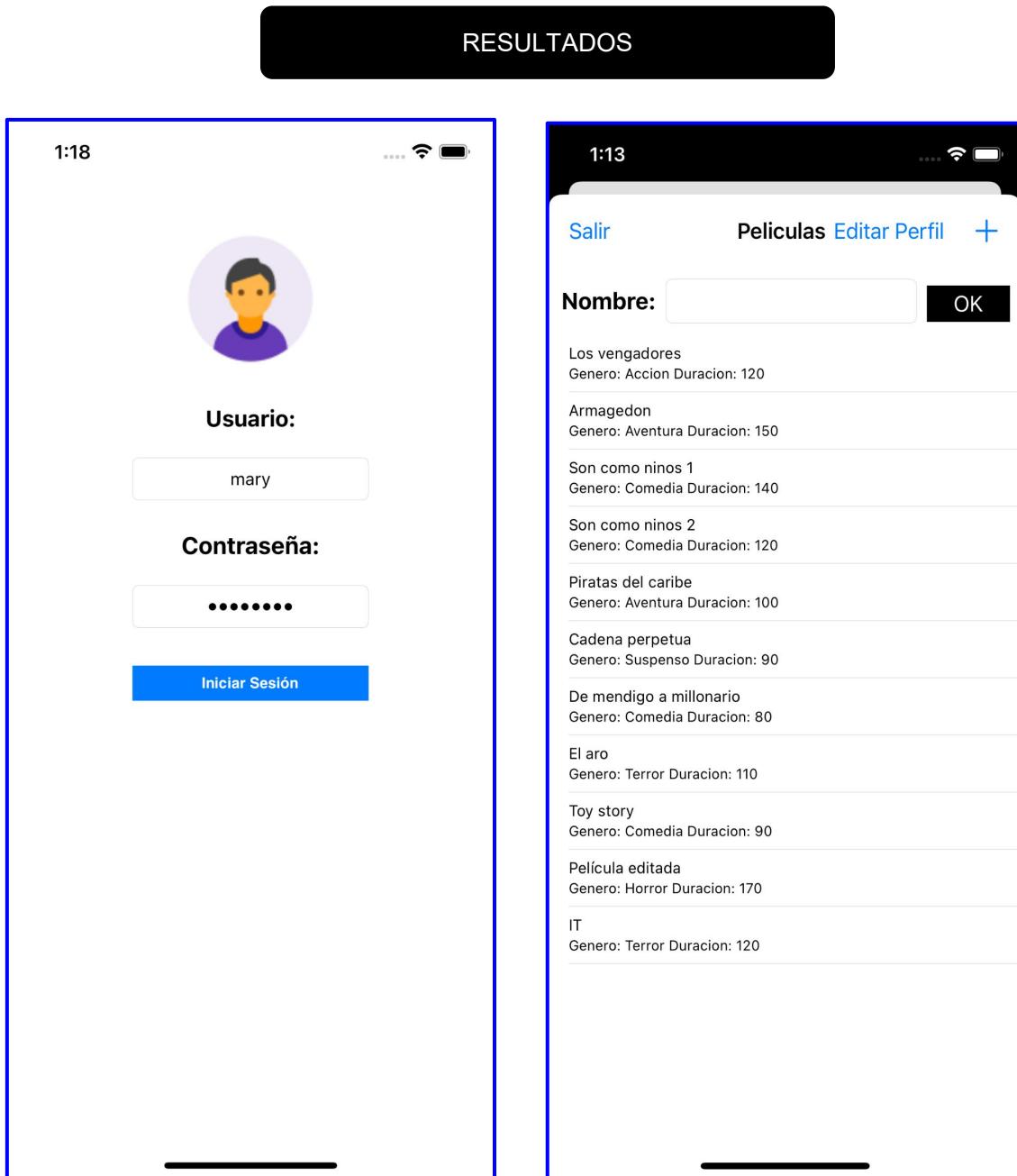


```
114 }
115
116 func tableView(_ tableView: UITableView, commit editingStyle: UITableViewCell.EditingStyle, forRowAt indexPath: IndexPath) {
117     if editingStyle == .delete {
118         let pelicula = peliculas[indexPath.row]
119
120         let alert = UIAlertController(title: "Eliminar Película", message: "¿Estás seguro de que deseas eliminar la película '\(pelicula.nombre)'?", preferredStyle: .alert)
121         alert.addAction(UIAlertAction(title: "Sí", style: .destructive, handler: { action in
122             self.eliminarPelícula(id: pelicula.id)
123         }))
124         alert.addAction(UIAlertAction(title: "No", style: .cancel, handler: nil))
125
126         present(alert, animated: true, completion: nil)
127     }
128 }
129
130 func eliminarPelícula(id: Int) {
131     let ruta = "http://localhost:3000/películas/\(id)"
132     var request = URLRequest(url: URL(string: ruta)!)
133     request.httpMethod = "DELETE"
134
135     URLSession.shared.dataTask(with: request) { (data, response, error) in
136         if let error = error {
137             print("Error al eliminar la película: \(error)")
138         } else {
139             print("Película eliminada con éxito")
140             // Actualiza la lista de películas después de eliminar
141             self.cargarPelículas(ruta: "http://localhost:3000/películas/")
142             self.tableView.reloadData()
143         }
144     }.resume()
145 }
146 }
147
148 }
```

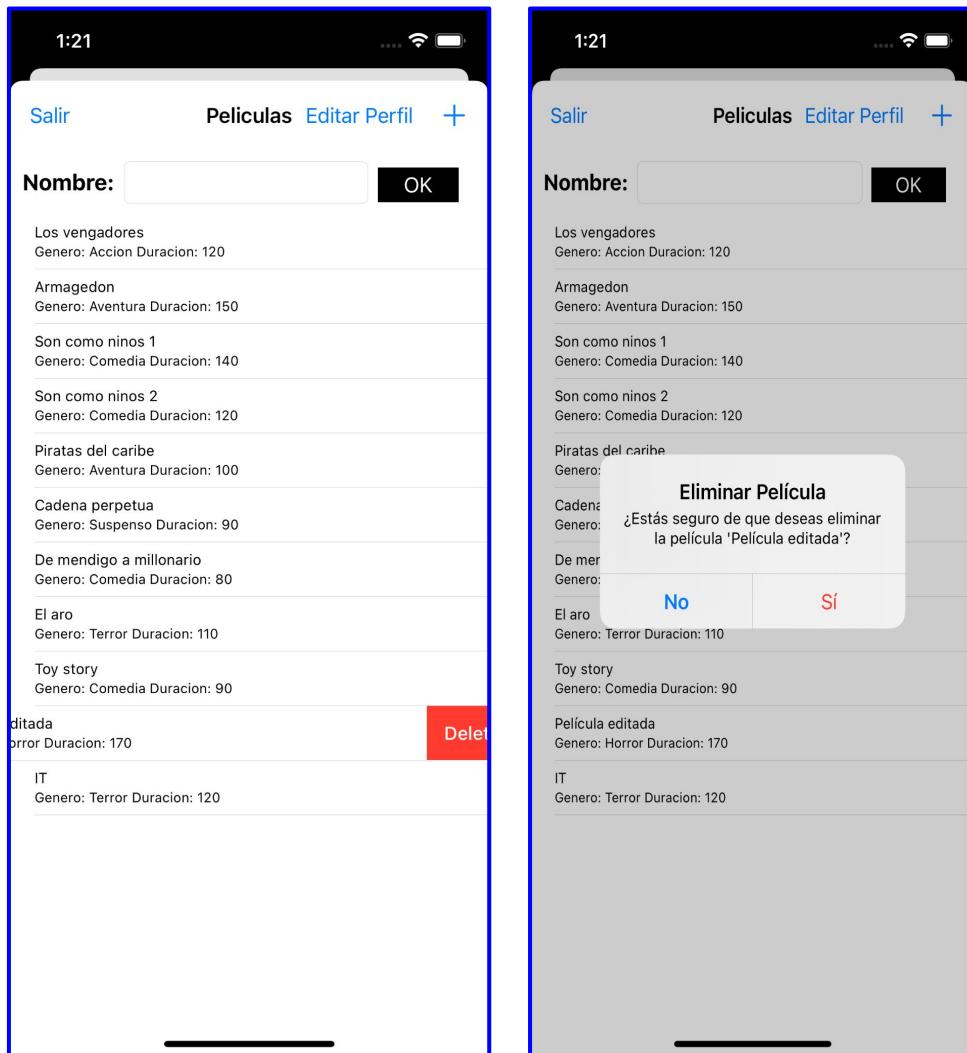
**viewControllerEditarPerfil.swift**

Se creó una nueva vista para poder editar los datos de un usuario que tengamos logeado, se hizo las respectivas  
conexiones, actions y outlets, también se crearon sus respectivas funciones.

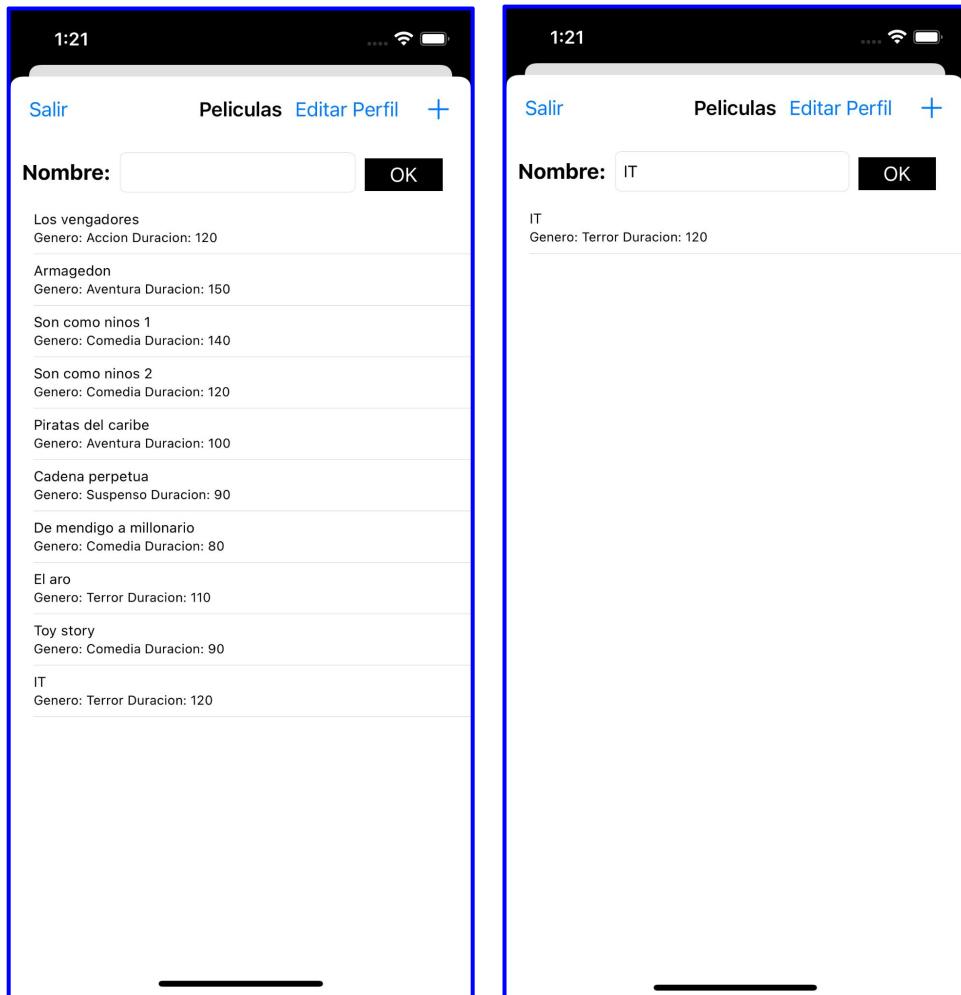
```
 4  //
 5 // Created by Andrade Chura Mary Carmen on 30/11/23.
 6 //
 7
 8 import UIKit
 9
10 class ViewControllerEditarPerfil: UIViewController {
11
12     @IBOutlet weak var txtNombre: UITextField!
13     @IBOutlet weak var txtClave: UITextField!
14     @IBOutlet weak var txtEmail: UITextField!
15     @IBOutlet weak var btnGuardarCambios: UIButton!
16
17     var usuario: Users?
18     var cambiosRealizados = false
19
20     override func viewDidLoad() {
21         super.viewDidLoad()
22
23         if let usuario = usuario {
24             txtNombre.text = usuario.nombre
25             txtClave.text = usuario.clave
26             txtEmail.text = usuario.email
27             txtClave.isSecureTextEntry = true
28         }
29
30         txtNombre.addTarget(self, action: #selector(textFieldDidChange), for: .editingChanged)
31         txtClave.addTarget(self, action: #selector(textFieldDidChange), for: .editingChanged)
32         txtEmail.addTarget(self, action: #selector(textFieldDidChange), for: .editingChanged)
33
34         btnGuardarCambios.isEnabled = false
35     }
36
37     @objc func textFieldDidChange() {
38         cambiosRealizados = true
39         btnGuardarCambios.isEnabled = true
40     }
41
42     @IBAction func guardarCambiosTapped(_ sender: UIButton) {
43         guard let usuario = usuario, cambiosRealizados else {
44             return
45         }
46
47         let ruta = "http://localhost:3000/usuarios/\(usuario.id)"
48         let datos = ["nombre": txtNombre.text!, "clave": txtClave.text!, "email": txtEmail.text!]
49         metodoPUT(ruta: ruta, datos: datos)
50
51         mostrarAlertaConCierreSesion(titulo: "Cambios Guardados", mensaje: "Vuelve a iniciar sesión con los nuevos datos.")
52     }
53
54     func metodoPUT(ruta: String, datos: [String: Any]) {
55         print("Llamada a metodoPUT en ViewControllerEditarPerfil")
56         let url: URL = URL(string: ruta)!
57         var request = URLRequest(url: url)
58         let session = URLSession.shared
59         request.httpMethod = "PUT"
60         let params = datos
61
62         do {
63             request.httpBody = try JSONSerialization.data(withJSONObject: params, options: JSONSerialization.WritingOptions.prettyPrinted)
64         } catch {
65             print("Error in JSON serialization")
66         }
67
68         request.addValue("application/json", forHTTPHeaderField: "Content-Type")
69         request.addValue("application/json", forHTTPHeaderField: "Accept")
70         let task = session.dataTask(with: request) { (data, response, error) in
71             if let data = data {
72                 do {
73                     let dict = try JSONSerialization.jsonObject(with: data, options: JSONSerialization.ReadingOptions.mutableLeaves)
74                     print(dict)
75                 } catch {
76                     print("Error in JSON deserialization")
77                 }
78             }
79         }
80
81         task.resume()
82     }
83     func mostrarAlertaConCierreSesion(titulo: String, mensaje: String) {
84         let alerta = UIAlertController(title: titulo, message: mensaje, preferredStyle: .alert)
85         let btnOK = UIAlertAction(title: "OK", style: .default) { [weak self] _ in
86             self?.dismiss(animated: true, completion: nil)
87         }
88         alerta.addAction(btnOK)
89         present(alerta, animated: true, completion: nil)
90     }
91 }
92 }
```



Inicie sesión de forma correcta y como podemos ver se muestran nuestra lista de películas.

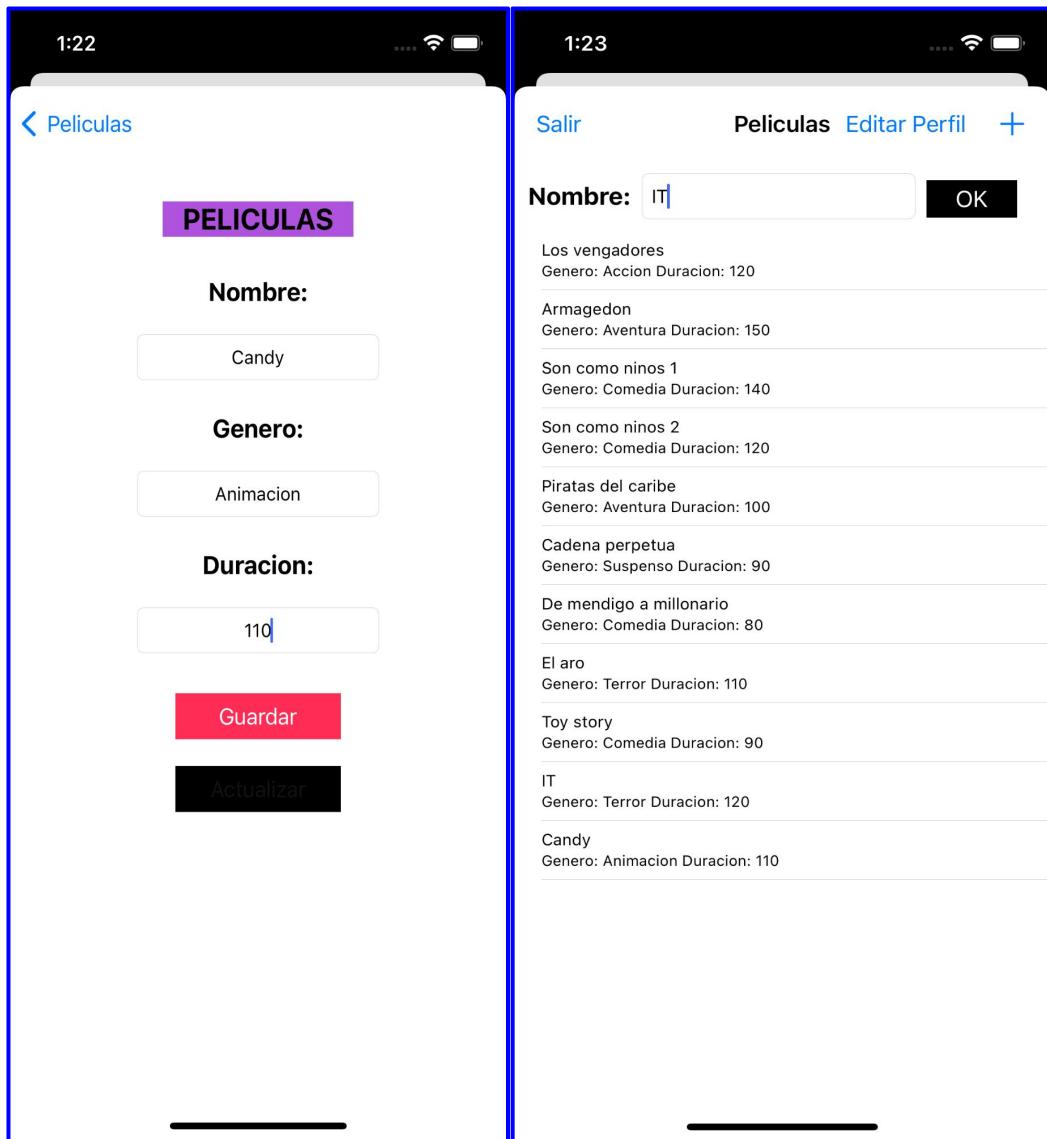


Se esta probando la función de eliminar, como podemos ver cuando deslizamos sale el Delete, y cuando lo deslizamos por completo nos sale una ventana de alerta, donde confirmamos si estamos seguros de eliminar la película o no, si pongo la opción de Si, se borará y si no se cancelara a operación.

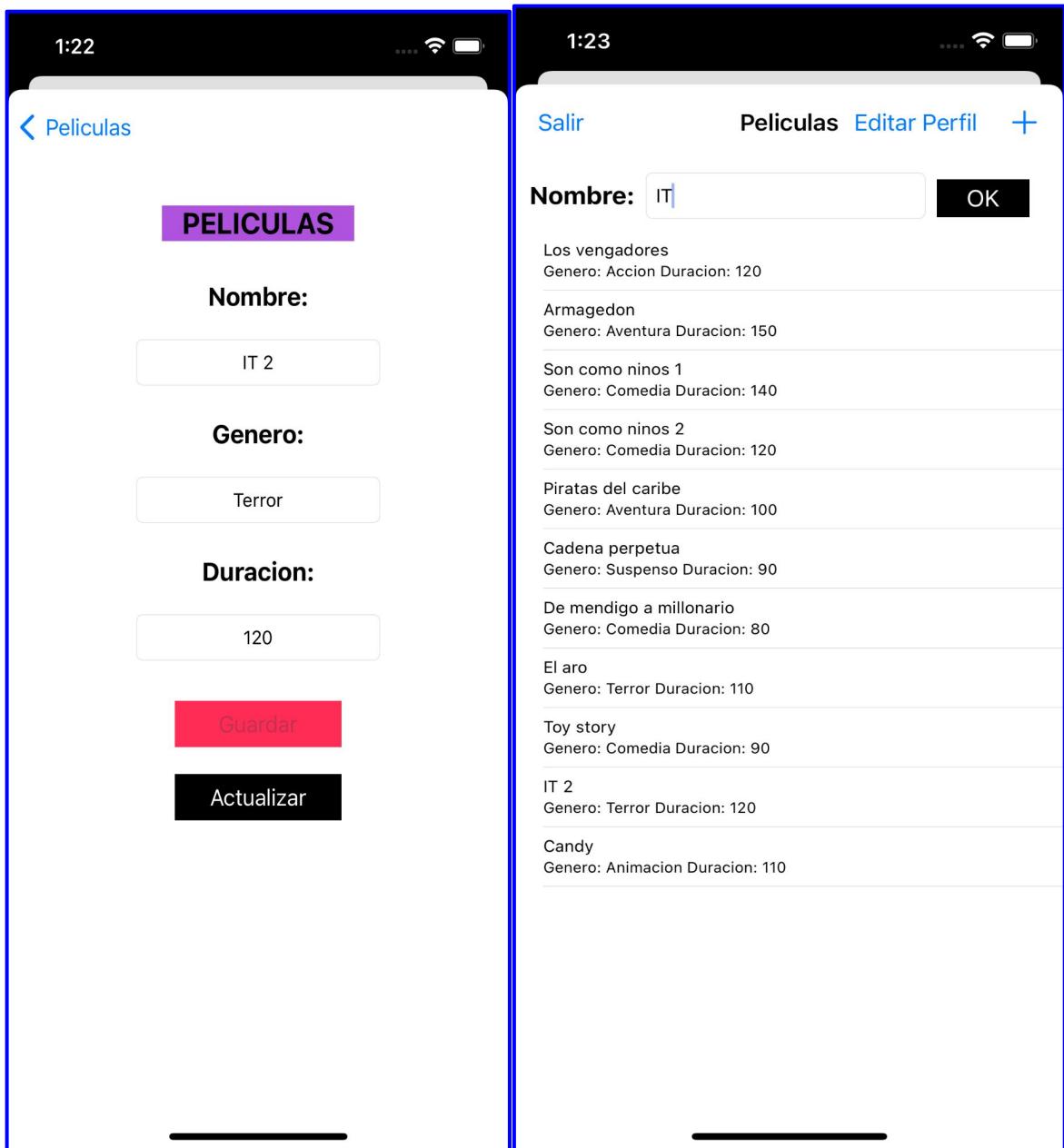


Aquí ya se puede ver que se elimino correctamente la película seleccionada.

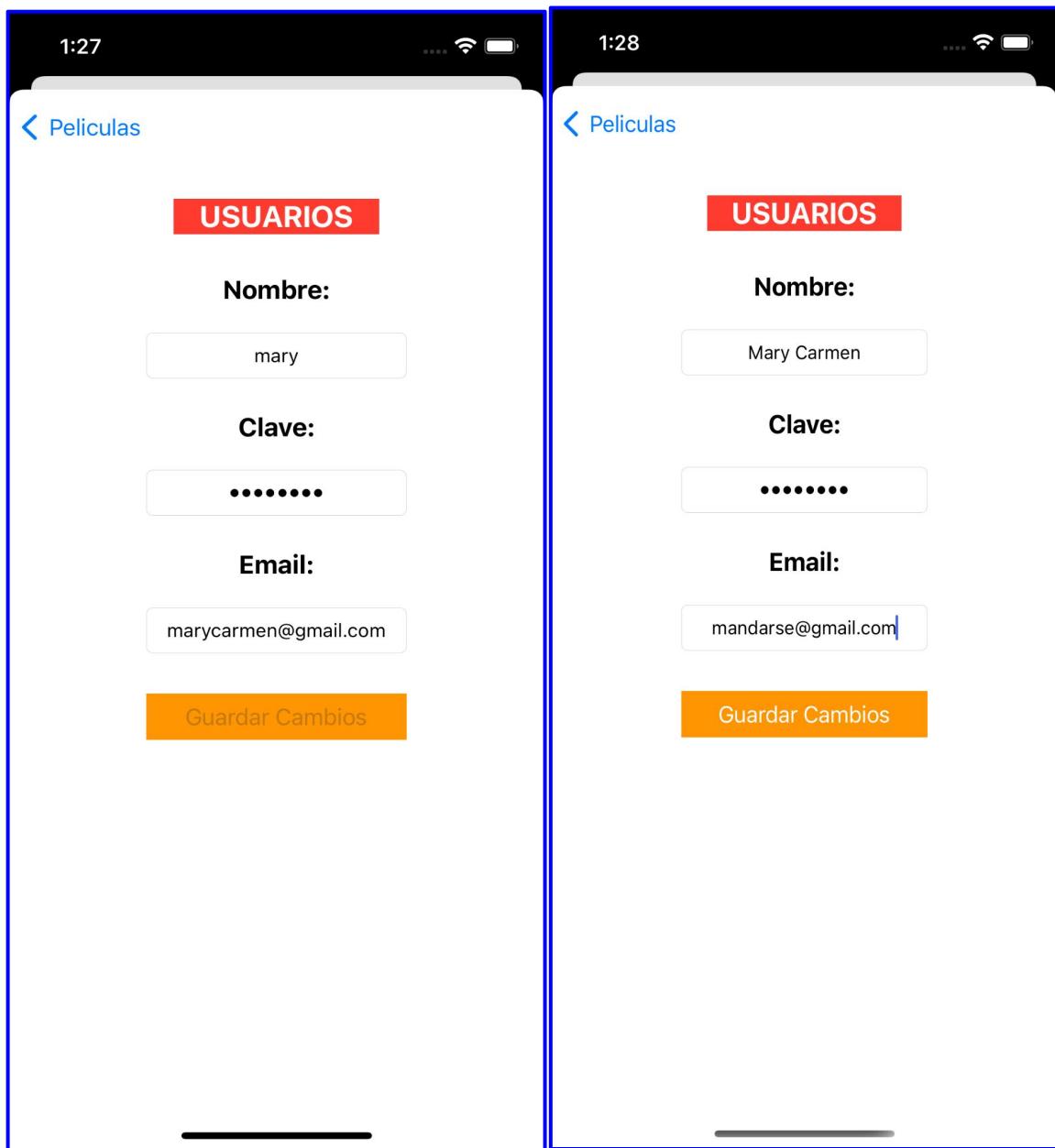
En la otra parte podemos ver que se puede buscar las películas, por ejemplo se busco la película IT y se busca de forma correcta.



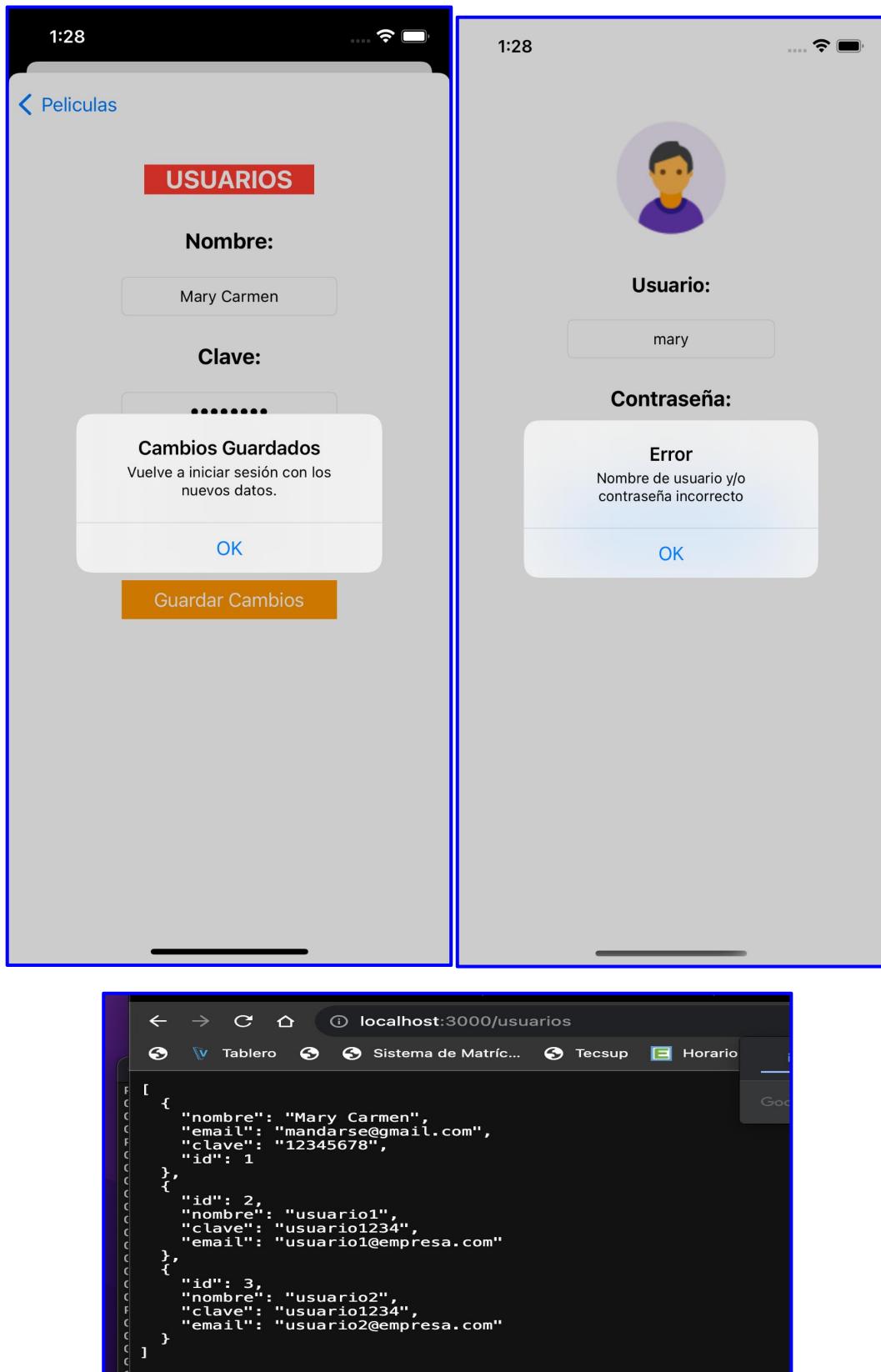
Aquí estamos Agregando una nueva película, llamada Candy, y como podemos ver se creó de forma correcta.



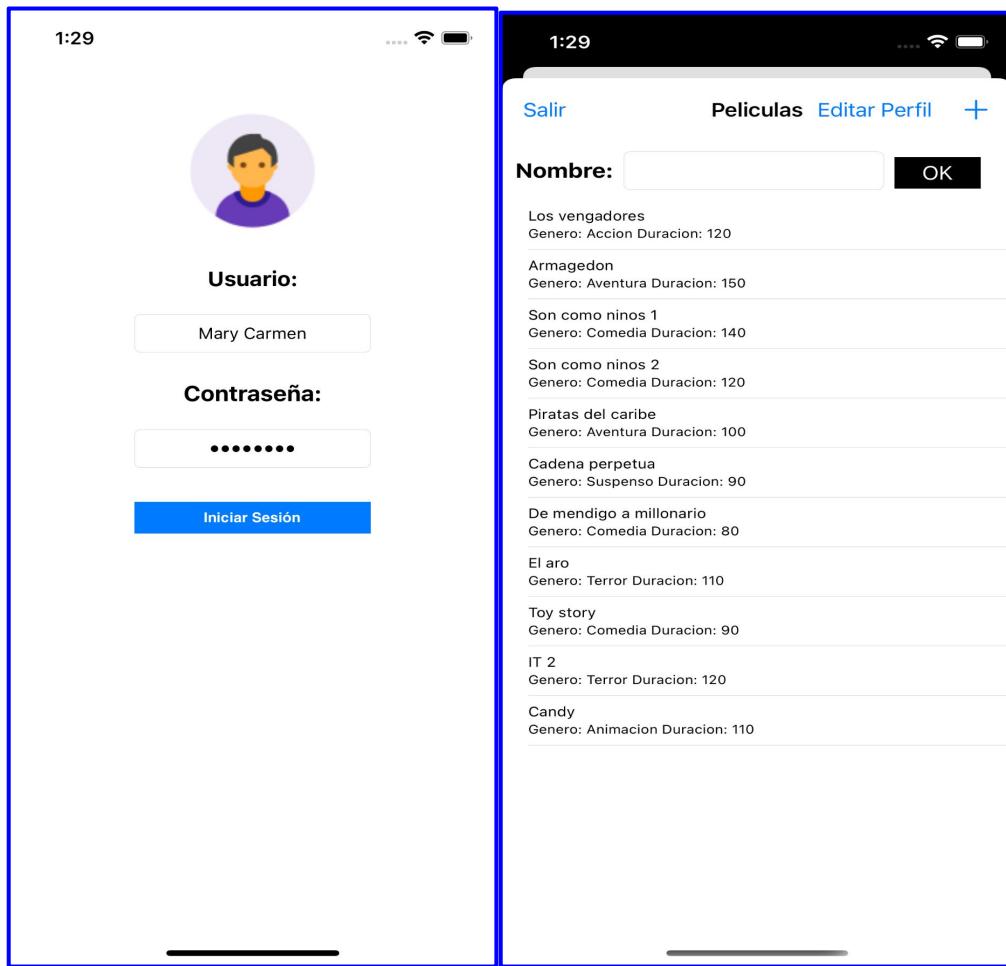
Aquí estamos Editando la película IT a IT 2 y este se actualizo de forma correcta.



Cuando le doy al botón de Editar Perfil, nos manda a una nueva vista donde podemos editar nuestro usuario logeado , en este caso mi usuario se llama mary y lo cambie a Mary Carmen, así como cambie el correo electrónico y lo guardamos.



Al guardar nos muestra una alerta para guardar los cambio, ahora podemos verificar si funciona, como podemos ver se quizo iniciar sesión con mi antiguo usuario, y no se puede sale una alerta donde nos dice que el usuario o contraseña es incorrecta. Ademas se puede ver en el localhost que mi usuario cambio, no es el mismo.



Podemos ver que con mi usuario ya modificado si se logra ingresar de forma correcta.

**OBSERVACIONES (5 mínimo):**

(Las observaciones son las notas aclaratorias, objeciones y problemas que se pudo presentar en el desarrollo del laboratorio)

- La configuración del servidor JSON local limitó la accesibilidad y portabilidad de la API, lo que podría afectar la escalabilidad y la disponibilidad en entornos de producción.
- Aunque se implementó la autenticación básica de usuarios, reconozco la necesidad de métodos más sólidos para proteger la información y evitar vulnerabilidades.
- Destaco la importancia de una interfaz de usuario intuitiva que permita acciones claras para la autenticación, búsqueda y manipulación de datos, facilitando la interacción del usuario.
- Encontré que la falta de comentarios y documentación detallada en el código podría dificultar la comprensión y mantenimiento a largo plazo.
- El énfasis en un entorno local proporcionó una excelente base para comprender los conceptos, pero es crucial considerar la escalabilidad y la adaptación a entornos más amplios.

**CONCLUSIONES (5 mínimo):**

(Las conclusiones son una opinión personal sobre tu trabajo, explicar como resolviste las dudas o problemas presentados en el laboratorio. Ademas de aportar una opinión crítica de lo realizado)

- Esta práctica me permitió adentrarme en la creación y consumo de servicios web utilizando RESTful y JSON, comprendiendo la importancia de los métodos HTTP para la comunicación entre servidores y clientes.
- Reconozco la importancia de fortalecer la seguridad en las aplicaciones, especialmente al manejar datos sensibles, y busco profundizar mis conocimientos en métodos avanzados de autenticación.
- Aprendí que, aunque los entornos locales son útiles para pruebas iniciales, es fundamental planificar la escalabilidad y la disponibilidad global al desarrollar aplicaciones para entornos de producción.
- Enfrenté desafíos al implementar validaciones de usuario y diseño de interfaz, lo que me enseñó la importancia de abordar problemas con múltiples enfoques y aprender de ellos.
- Este laboratorio me motivó a seguir aprendiendo sobre seguridad, mejores prácticas en el desarrollo de APIs y estrategias para mejorar la experiencia del usuario en aplicaciones basadas en servicios web.