

Programación en Móviles Avanzado

LABORATORIO 14

RESTFuI JSON

CODIGO DEL CURSO:



Alumno(s):	Rojas Huayhua Yesica Nancy					Nota	
Grupo:	A			Ciclo: V			
Criterio de Evaluación	Excelente (4pts)	Bueno (3pts)	Regular (2pts)	Requiere mejora (1pts)	No acept. (0pts)	Puntaje Logrado	
Instala y configura JSONPlaceholder para crear un servidor Json Web Service						3	
Utiliza comandos HTTP para el consumo de Web Service (GET, POST, PUT, DELETE)						3	
Desarrolla adecuadamente los ejercicios propuestos						6	
Realiza observaciones y conclusiones que aporten un opinión crítica y técnica						3	
Es puntual y redacta el informe adecuadamente sin copias de otros autores						2	
Evidencia avance en laboratorio						3	

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:

- En este laboratorio cada alumno trabajará 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/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 RESTApi JSON.
9. En la pagina 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

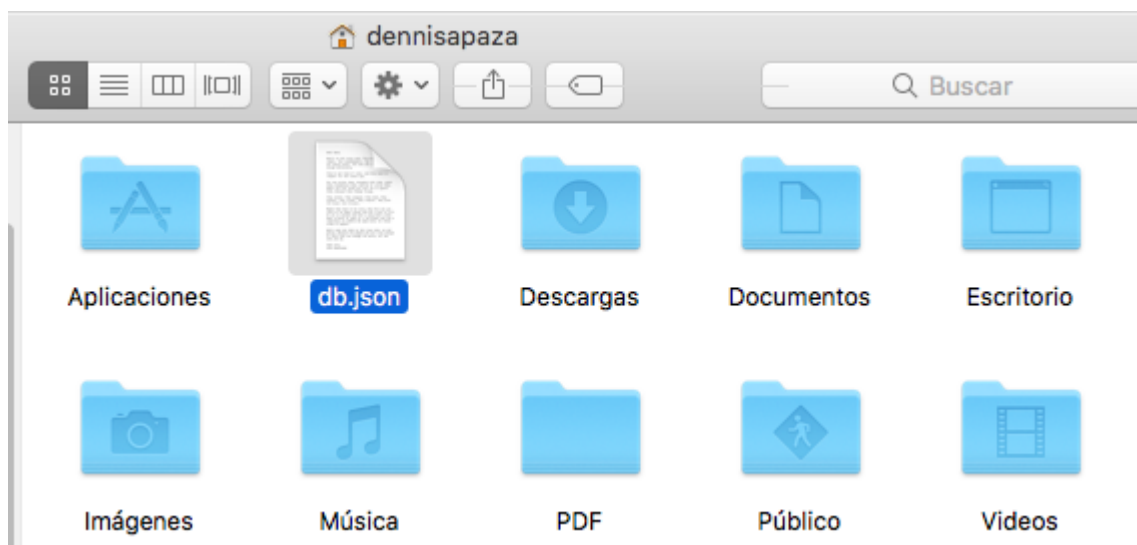
Resources
http://localhost:3000/posts
http://localhost:3000/comments
http://localhost:3000/profile

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

```
{
  "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": "Suspenso" ,
      "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 recién 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 pelicla 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...
```

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": "usuariol",
    "clave": "usuariol234",
    "email": "usuariol@tecsup.edu.pe"
  },
  {
    "id": 3,
    "nombre": "usuario2",
    "clave": "usuariol234",
    "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**

Source Control: ☒ Create Git repository on my Mac
Xcode will place your project under source control

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



Usuario:

Contraseña:

Iniciar Sesión

22. Para la parte de autenticación usaremos el recurso **/usuarios** provisto por la API ya mencionada.
23. Para la pantalla de logeo que se realizara, nos autenticaremos 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(**txtContraseña**). Y Defina una instancia de la estructura **Users** creada en el paso anterior

```
class ViewController: UIViewController {

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

26. Cree la función **validarUsuario()** que solicitara como parámetro la URL de destino en al cual se realizara 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"
  }
]
```

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>



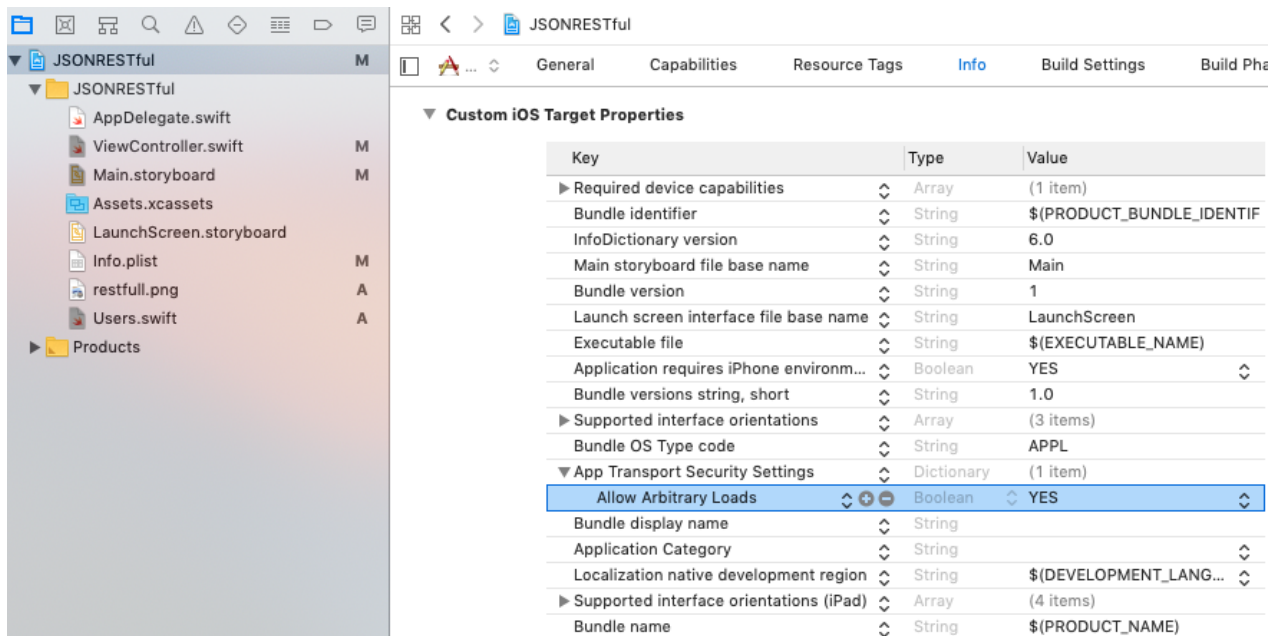
```
[]
```


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

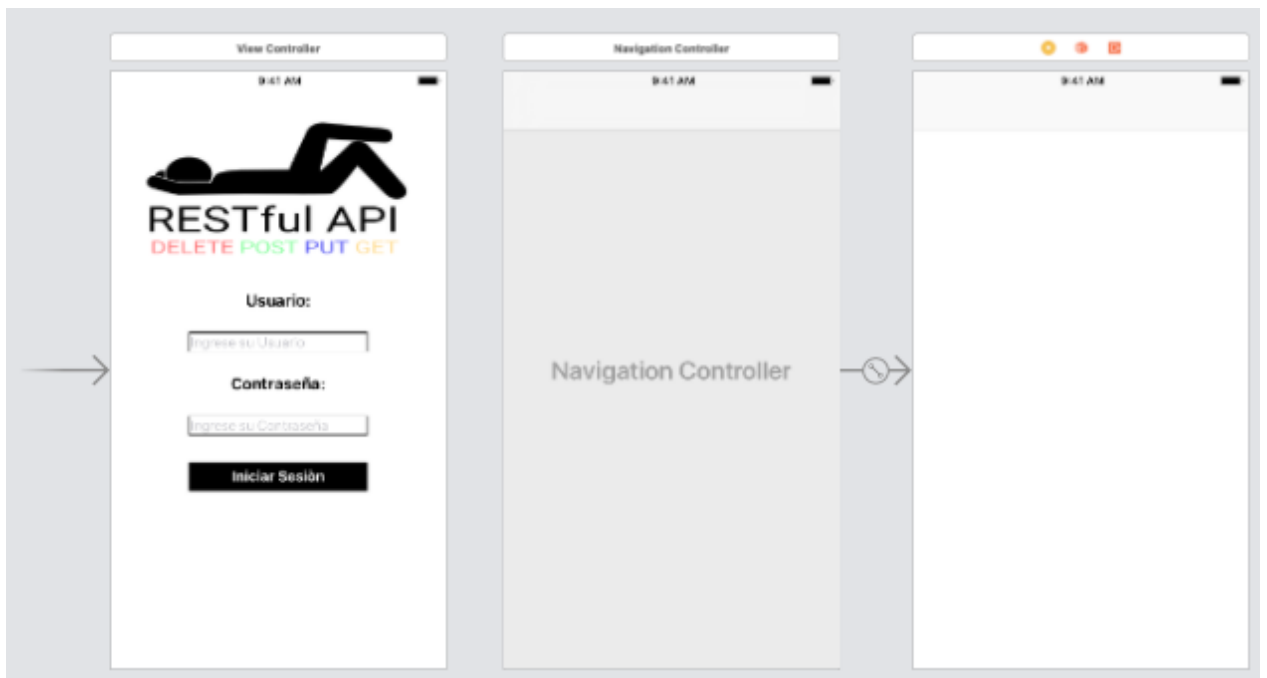


32. Ejecute la aplicación y pruebe logearse con un usuario existente en la base de datos y luego con datos incorrectos y verifique el resultado por consola

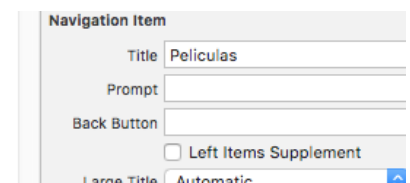
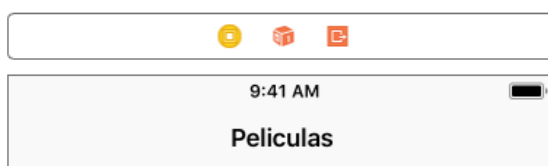
```
Logeo Exitoso
id:1,nombre:dennis,nombre:dennis2deah@gmail.com
```

Nombre de usuario y/o contraseña es incorrecto

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



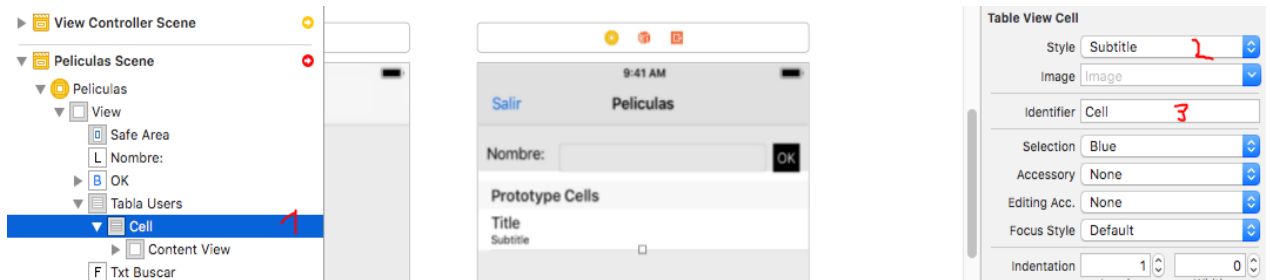
34. Agregue un título al nuevo **viewController** como se muestra



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**

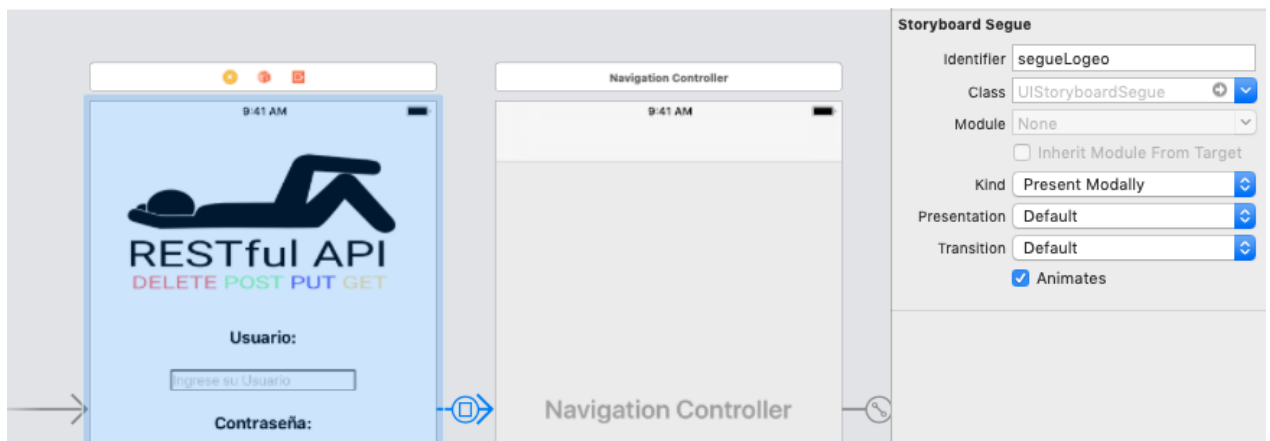
44. Implemente los **delegados** respectivos, y llame a la función **buscarPelicula** para llenar el **tableView** con la data recibida. Inserte estos datos en **viewDidLoad**

```
@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 login al **navigationController**. Y coloque como identificador del segue "**segueLogeo**"



46. Modifique la función **logear** del **viewController** de login 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 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")
            self.performSegue(withIdentifier: "segueLogeo", sender: nil)
            for data in self.users {
                print("id:\(data.id), nombre:\(data.nombre), email:\(data.email)")
            }
        }
    }
}
```

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

Salir

Películas

Nombre:

Los vengadores
 Genero:Accion Duracion:120

Armagedon
 Genero:Aventura Duracion:150

Son como ninos 1
 Genero:Comedia Duracion:140

Son como ninos 2
 Genero:Comedia Duracion:120

Piratas del caribe
 Genero:Aventura Duracion:100

Cadena perpetua
 Genero:Suspenso Duracion:90

De mendigo a millonario
 Genero:Comedia Duracion:80

El aro
 Genero:Terror Duracion:110

Toy story
 Genero:Comedia Duracion:90

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

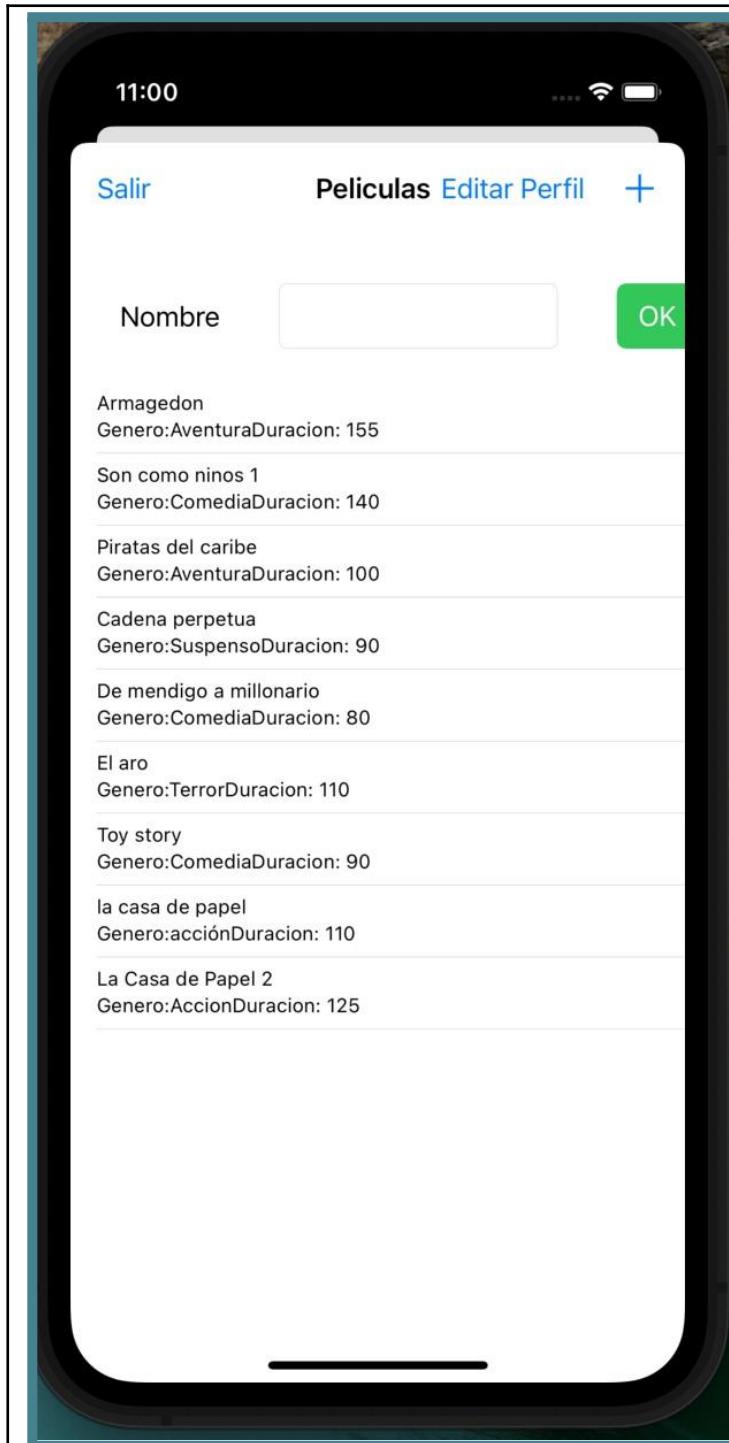
El código implementado interactúa con un servidor local (en la dirección <http://localhost:3000>) para validar usuarios y cargar datos de películas. Utiliza `URLSession` para hacer solicitudes HTTP y `JSONDecoder` para analizar las respuestas JSON en objetos utilizables en la aplicación.

Carga de Datos de Películas:

Similar al proceso de validación de usuarios, el código hace uso de `URLSession` para realizar una solicitud HTTP al servidor local.

En este caso, la solicitud se hace para obtener datos de películas almacenadas en el servidor.

Cuando el servidor responde con información sobre películas en formato JSON, el código nuevamente utiliza `JSONDecoder` para interpretar esta respuesta y convertirla en objetos utilizables en la aplicación.



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

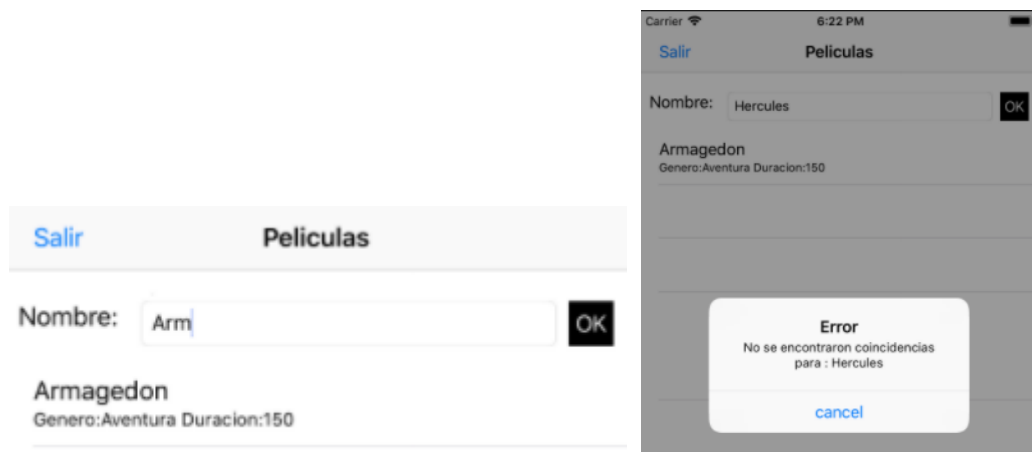
```
func mostrarAlerta(titulo: String, mensaje: String, accion: String) {  
    let alerta = UIAlertController(title: titulo, message: mensaje,  
        preferredStyle: .alert)  
    let btnOK = UIAlertAction(title: accion, style: .default, handler: nil)  
    alerta.addAction(btnOK)  
    present(alerta,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

Este código maneja la lógica de búsqueda y visualización de películas. Construye una URL de búsqueda en función del nombre ingresado, realiza la búsqueda y carga todas las películas si no se proporciona ningún nombre, y luego actualiza la interfaz de usuario con los resultados o muestra un mensaje si no se encuentran películas.

.Si el campo de nombre está vacío (`nombre.isEmpty`), se establece una nueva ruta base ("`http://10ca1host:3eee/pe1icu1as/`") para cargar todas las películas.

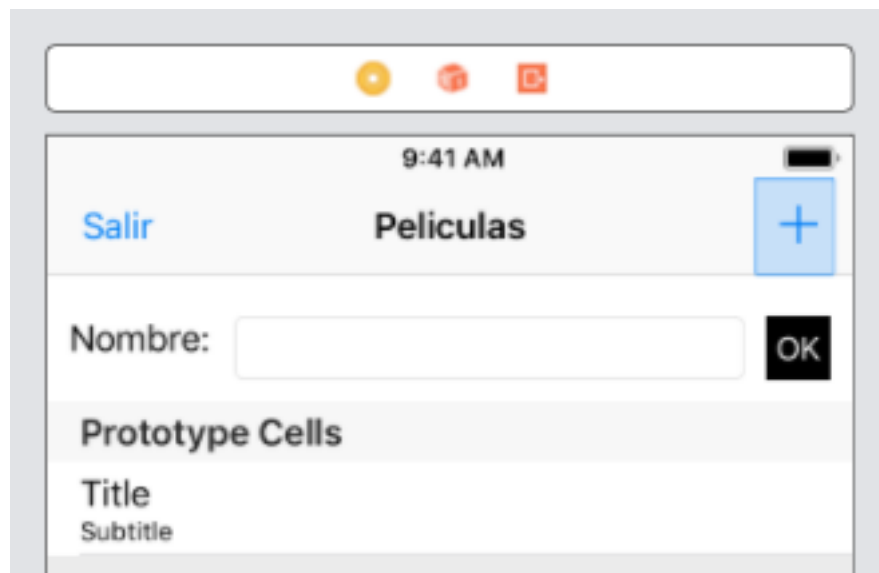
Llama a la función cargarpeliculas con la ruta correspondiente.
Si se encontraron películas, se recarga la tabla con la función `tablaPe1icu1as.reloadData()`.
Si no se encuentran películas, se muestra una alerta utilizando la función `mostrarA1erta`, indicando que no se encontraron coincidencias para el nombre proporcionado.

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

Peliculas

Nombre

Ingrese un nombre para la pelicula

Genero

Ingrese un genero

Duracion

Ingrse la duracion en minutos

Guardar

Actualizar

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 **textField**s 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 = ["usuarioId": 1, "nombre": "\(nombre)", "genero": "\(
        genero)", "duracion": "\(duracion)"] as Dictionary<String, Any>
    let ruta = "http://localhost:3000/peliculas"
    metodoPOST(ruta: ruta, datos: datos)
    navigationController?.pushViewController(animated: true)
}
```

64. En el archivo **viewControllerBuscar.swift** implemente el metodo **viewWillAppear** para que recarga la tabla de la vista cada vez que se agregue 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

Peliculas

Nombre

La Casa de Papel

Genero

Acción

Duracion

120

Guardar

Actualizar

Peliculas

Salir

Nombre: OK

Los vengadores
Genero:Accion Duracion:120

Armagedon
Genero:Aventura Duracion:150

Son como ninos 1
Genero:Comedia Duracion:140

Son como ninos 2
Genero:Comedia Duracion:120

Piratas del caribe
Genero:Aventura Duracion:100

Cadena perpetua
Genero:Suspenso Duracion:90

De mendigo a millonario
Genero:Comedia Duracion:80

El aro
Genero:Terror Duracion:110

Toy story
Genero:Comedia Duracion:90

La Casa de Papel
Genero:Acción Duracion:120

Este código administra el envío de datos al servidor mediante una solicitud POST, la carga de datos desde el servidor al iniciar la vista y la actualización de la interfaz de usuario con estos datos obtenidos del servidor.

La función metodo POST Crea una URL a partir de la ruta.
Establece la solicitud (URLRequest) utilizando la URL creada.
Especifica el método HTTP como POST.
Convierte el diccionario params en datos JSON y los asigna al cuerpo de la solicitud HTTP.
Establece los encabezados Content-Type y Accept como application/json.
Inicia una tarea de sesión (dataTask) con la solicitud creada, que envía la solicitud al servidor.
Maneja la respuesta del servidor:
Si se recibe datos, intenta convertirlos de JSON a un diccionario y los imprime.

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. Dirijase a **main.storyboard** y cree un **segue** desde el icono 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

Peliculas

Nombre

Genero

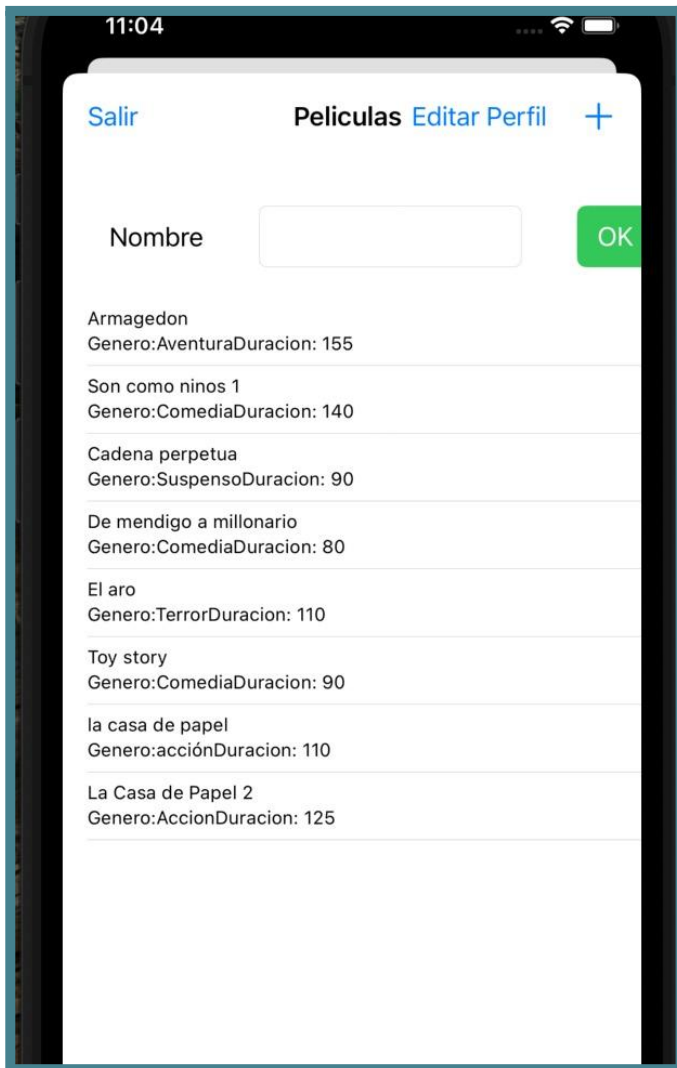
Duracion

La Casa de Papel 2
Genero:Acción Duracion:150

73. Comente lo mas importante del codigo implementado

Este código se encarga de actualizar los detalles de una película seleccionada en la interfaz de usuario y enviar los cambios al servidor mediante una solicitud PUT, utilizando los datos ingresados en los campos de texto.

```
func metodoPUT(ruta: String, datos: [String: Any])  
Esta función maneja la solicitud PUT para actualizar datos en el servidor.  
Crea una solicitud (URLRequest) con la URL proporcionada.  
Especifica el método HTTP como PUT.  
Convierte el diccionario params en datos JSON y los asigna al cuerpo de la solicitud HTTP.  
Establece los encabezados Content-Type y Accept como application/json.  
Inicia una tarea de sesión (dataTask) con la solicitud creada, que envía la solicitud al servidor.  
Si se recibe datos, intenta convertirlos de JSON a un diccionario e imprime el resultado.
```



11:04

Salir Películas Editar Perfil +

Nombre OK

Armagedon
Genero:AventuraDuracion: 155

Son como niños 1
Genero:ComediaDuracion: 140

Cadena perpetua
Genero:SuspenseDuracion: 90

De mendigo a millonario
Genero:ComediaDuracion: 80

El aro
Genero:TerrorDuracion: 110

Toy story
Genero:ComediaDuracion: 90

la casa de papel
Genero:acciónDuracion: 110

La Casa de Papel 2
Genero:AcciónDuracion: 125

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(SI 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)

Eliminar:

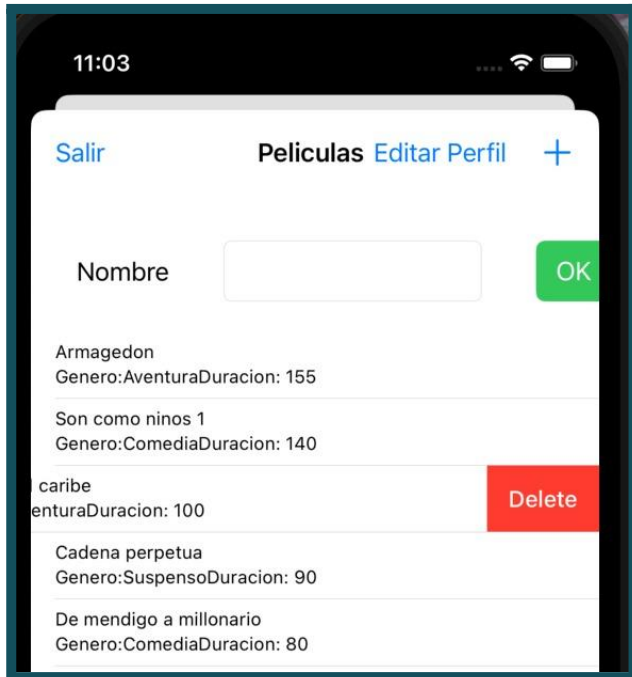
```
func eliminarPelicula(id: Int) {
    let ruta = "http://localhost:3000/peliculas/\"(id)\"
    var request = URLRequest(url: URL(string: ruta!))
    request.httpMethod = "DELETE"

    URLSession.shared.dataTask(with: request) { (data, response, error) in
        if let error = error {
            print("Error al eliminar la película: \"(error)\"")
        } else {
            print("Película eliminada con éxito")
            // Actualiza la lista de películas después de eliminar
            self.cargarPeliculas(ruta: "http://localhost:3000/peliculas/") {
                self.tablaPeliculas.reloadData()
            }
        }
    }.resume()
}
```

```
func tableView(_ tableView: UITableView, commit editingStyle:
UITableViewCellEditingStyle, forRowAt indexPath: IndexPath) {
    if editingStyle == .delete {
        let pelicula = peliculas[indexPath.row]

        let alert = UIAlertController(title: "Eliminar Película", message: "¿Estás
seguro de que deseas eliminar \"(pelicula.nombre)\"?", preferredStyle: .alert)
        alert.addAction(UIAlertAction(title: "Sí", style: .destructive, handler: {
action in
            self.eliminarPelicula(id: pelicula.id)
        })))
        alert.addAction(UIAlertAction(title: "No", style: .cancel, handler: nil))

        present(alert, animated: true, completion: nil)
    }
}
```



```
import UIKit

class ViewControllerEditarPerfil: UIViewController {

    @IBOutlet weak var txtNombre: UITextField!
    @IBOutlet weak var txtClave: UITextField!
    @IBOutlet weak var txtEmail: UITextField!
    @IBOutlet weak var btnGuardarCambios: UIButton!

    var usuario: Users?
    var cambiosRealizados = false

    override func viewDidLoad() {
        super.viewDidLoad()

        // Verificar si usuario está asignado
        if let usuario = usuario {
            // Configura los campos de texto con los datos del usuario
            txtNombre.text = usuario.nombre
            txtClave.text = usuario.clave
            txtEmail.text = usuario.email
        }
    }
}
```

```
        txtClave.isSecureTextEntry = true
    }

    // Observadores para detectar cambios en los campos de texto
    txtNombre.addTarget(self, action: #selector(textFieldDidChange), for:
.editingChanged)
    txtClave.addTarget(self, action: #selector(textFieldDidChange), for:
.editingChanged)
    txtEmail.addTarget(self, action: #selector(textFieldDidChange), for:
.editingChanged)

    // Deshabilita el botón al inicio
    btnGuardarCambios.isEnabled = false
}

@objc func textFieldDidChange() {
    // Habilita el botón si detecta cambios en los campos de texto
    cambiosRealizados = true
    btnGuardarCambios.isEnabled = true
}

@IBAction func guardarCambiosTapped(_ sender: UIButton) {
    // Verifica si hay un usuario logeado y cambios realizados
    guard let usuario = usuario, cambiosRealizados else {
        // Manejar el caso en el que no hay un usuario logeado o no hay cambios
realizados
        return
    }

    let ruta = "http://localhost:3000/usuarios/\(usuario.id)"
    let datos = ["nombre": txtNombre.text!, "clave": txtClave.text!, "email":
txtEmail.text!]
    metodoPUT(ruta: ruta, datos: datos)

    // Después de guardar los cambios, puedes realizar alguna acción como volver
atrás
    mostrarAlertaConCierreSesion(titulo: "Cambios Guardados", mensaje: "Vuelve a
iniciar sesión.")
}

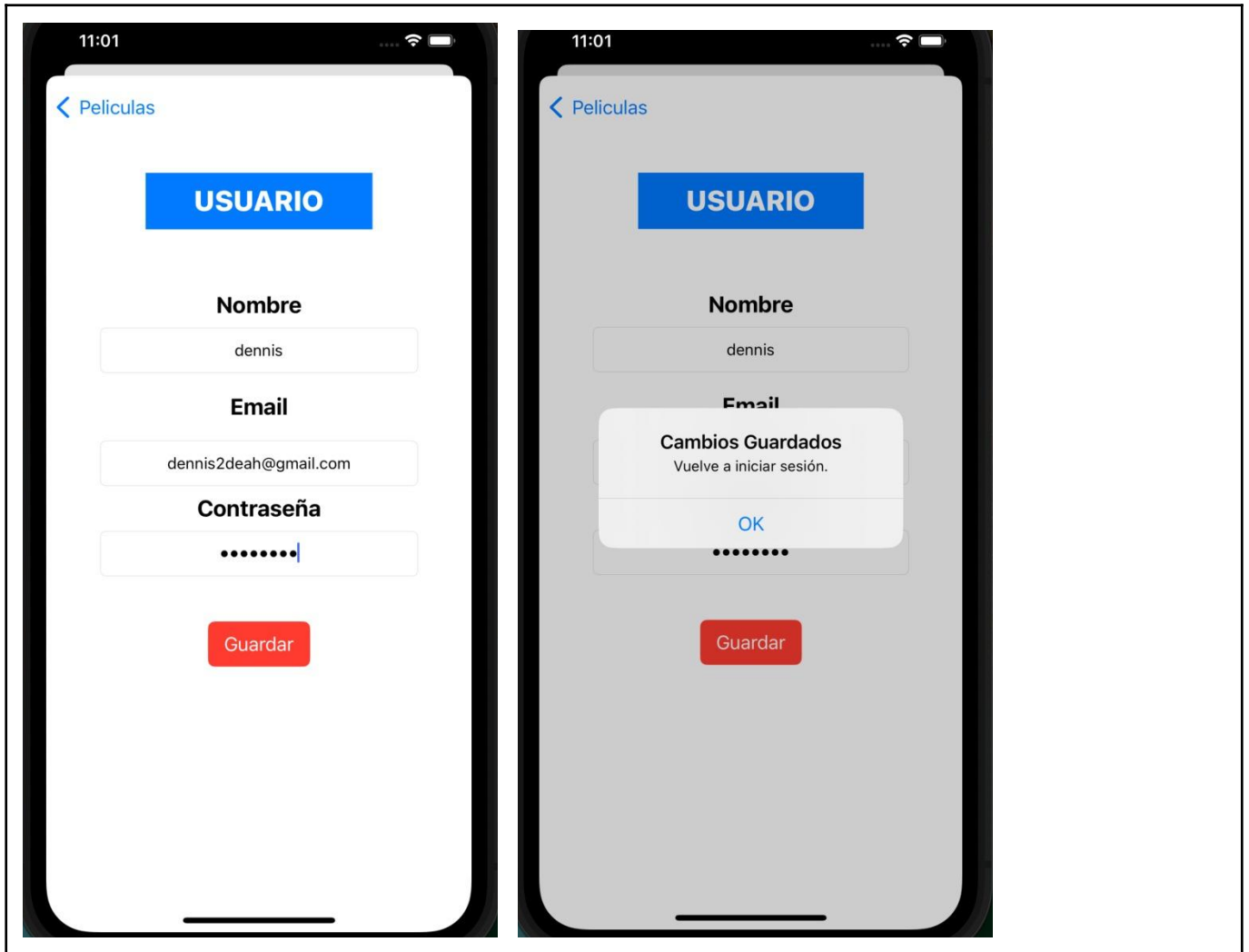
func metodoPUT(ruta: String, datos: [String: Any]) {
    print("Llamada a metodoPUT en ViewControllerEditarPerfil")
}
```

```
let url: URL = URL(string: ruta)!
var request = URLRequest(url: url)
let session = URLSession.shared
request.httpMethod = "PUT"
let params = datos

do {
    request.httpBody = try JSONSerialization.data(withJSONObject: params,
options: JSONSerialization.WritingOptions.prettyPrinted)
} catch {
    print("Error in JSON serialization")
}

request.addValue("application/json", forHTTPHeaderField: "Content-Type")
request.addValue("application/json", forHTTPHeaderField: "Accept")
let task = session.dataTask(with: request) { (data, response, error) in
    if let data = data {
        do {
            let dict = try JSONSerialization.jsonObject(with: data, options:
JSONSerialization.ReadingOptions.mutableLeaves)
            print(dict)
        } catch {
            print("Error in JSON deserialization")
        }
    }
}

task.resume()
}
```



OBSERVACIONES (5 mínimo):

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

usuarios y la obtención de películas desde un servidor cercano. Esto permite que los usuarios se identifiquen y vean información sobre películas almacenadas en el servidor local.

Usa URLSession para comunicarse: Utiliza URLSession para comunicarse con el servidor mediante solicitudes HTTP. Esto le permite validar usuarios y obtener datos de películas de manera eficiente y estructurada.

JSONDecoder para datos: Emplea JSONDecoder para interpretar las respuestas del servidor en formato JSON. Este proceso convierte esos datos en objetos que pueden ser fácilmente utilizados dentro de la aplicación.

viewDidLoad maneja botones: La función viewDidLoad dentro del ViewController controla la apariencia de los botones. Estos se habilitan o deshabilitan dependiendo de si hay una película seleccionada para editar. Esto asegura una interacción adecuada con la interfaz.

@IBAction func login valida usuarios: La función login verifica la identidad del usuario mediante una solicitud GET al servidor, utilizando el nombre de usuario y la contraseña proporcionados. Si la autenticación es exitosa, guarda la información del usuario para continuar en la siguiente vista.

@IBAction func btnBuscar busca películas: La función btnBuscar permite buscar películas por nombre. Al enviar una solicitud GET al servidor con el término de búsqueda, gestiona la visual

CONCLUSIONES (5 mínimo):

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

Los métodos HTTP, tales como GET, POST, PUT y DELETE, representan acciones específicas en la comunicación entre aplicaciones y servidores. Cada uno tiene un propósito claro: GET para obtener información, POST para crear nuevos datos, PUT para actualizarlos y DELETE para eliminarlos.

Los endpoints, expresados como URLs, actúan como puntos de acceso a los recursos en un servidor. En el código proporcionado, ejemplos como "http://localhost:3000/usuarios" y "http://localhost:3000/peliculas/" son ejemplos claros de endpoints para usuarios y películas respectivamente.

La serialización de datos es esencial para convertir la información en el formato adecuado para ser transmitida a través de la red. En este contexto, el uso de JSONSerialization.data() es un ejemplo práctico de cómo se transforman los datos a formato JSON para su envío.

Por otro lado, la deserialización de datos implica la habilidad de convertir la información JSON recibida desde la red en objetos utilitarios dentro de la aplicación. Ejemplos como JSONDecoder().decode() ilustran este proceso de transformación.

El uso de URLSession es crucial para establecer y manejar la comunicación con un servidor. Esta herramienta facilita las solicitudes HTTP y el manejo de las respuestas obtenidas, permitiendo intercambiar datos de manera efectiva y segura entre la aplicación y el servidor.

Los encabezados de solicitud, como Content-Type y Accept, juegan un papel fundamental al indicar al servidor qué tipo de datos se envían y qué formato se espera recibir. Son elementos esenciales para asegurar una comunicación precisa y efectiva entre la aplicación y el servidor.