PonyApp iOS

Tabla de contenido

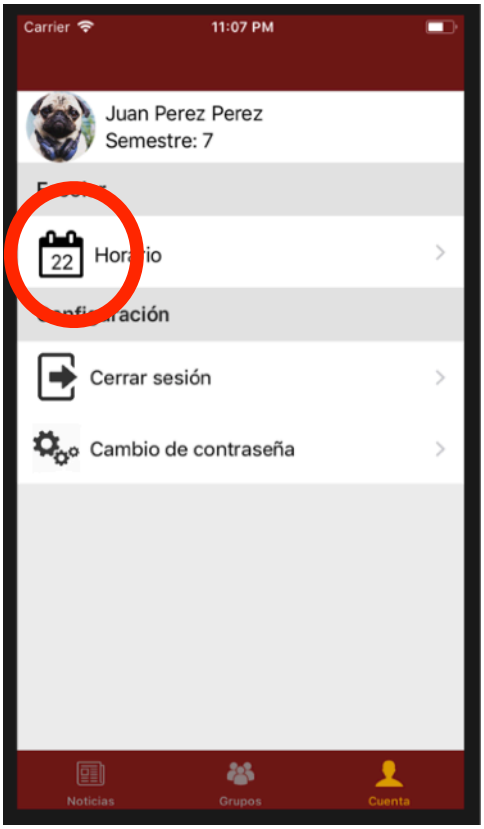
SettingTableViewController.swift 3
ScheduleViewControler.swift 3
GroupViewController.swift 6
GroupsCollectionViewCell.swift 6
CreatePostViewController.swift 7
ExpandableHeaderView 7
    Section.swift 7
    ExpandableHeaderView 7
LoginViewContoller.swift 7
NewsFeedtableViewController.swift 8
    Sección 2 9
    Sección 4 9
Sección 5 10
GroupsFeedViewContoller 10
    Sección 1 10
ShowPostViewController.swift 11
PostCell 11
CommentCell.swift 12
GroupPostCell 12
MembersCollectionViewCell 12

SettingTableViewController.swift

Clase para crear una tabla estática para el menú de opciones de la aplicación, cuenta solo con dos funciones las cuales son:
Horario (esta función no esta escrita en la clase, es solo un segue que nos lleva a la interfaz de Schedule y esta transición se hizo desde el main.storyboard) y Cerrar sesión (se muestra en el siguiente if):

```
if (indexPath.section == 2 && indexPath.row == 0) {
            UserDefaults.standard.set(false,forKey:"isUserLoggedIn");
            UserDefaults.standard.synchronize();
            self.tabBarController?.selectedIndex = 0
        }
```

En el viewDidLoad se carga la fecha actual para mostrarla en el icono de calendario que se tiene en la interfaz y se le da formato a la imagen del usuario. Se obtiene la imagen del usuario del SGE y el nombre completo del usuario así como su semestre actual.



Se opto por una tabla estática ya que no se tienen muchas opciones y todos los usuarios verán el mismo menu.

ScheduleViewControler.swift

Se tiene una estructura “subject” la cual representa a una materia, tiene como atributos nombre, profesor y un arreglo de horario para almacenar hora y salón. Se creó una extensión de la clase

Date para poder obtener el día de la semana (lunes, martes, etc.) y poder desplegar el horario del día actual.

En la funciono viewDidLoad se hacen modificaciones en la interfaz para que tenga el titulo y color adecuado la interfaz

```
self.navigationItem.title = "Horario"
        self.navigationController?.navigationBar.tintColor = UIColor(hexString:
"#dc9c03")
```

Se obtiene el JSON de las materias en la siguiente sentencia try-catch y se transforma a un arreglo de materias las cuales están ordenadas por día, siendo la localidad 0 del JSON el día lunes, localidad 2 día martes y así sucesivamente.

```
 let path = Bundle.main.path(forResource: "subjecttest", ofType: "json")
        let url = URL(fileURLWithPath: path!)
        do{
            let data = try Data(contentsOf: url)
            subjects = try JSONDecoder().decode([subject].self, from: data)
        }
        catch { }
```

El JSON deberá tener la siguiente estructura:

```
 [
  {
  "name": "Calculo Integral",
  "professor": "Juan Perez Perez",
  "schedule": [
                "7:00-8:00 am AG5",
                "7:00-8:00 am AG5",
                "7:00-8:00 am AG5",
                "7:00-8:00 am AG5",
                "7:00-8:00 am F4"
                ]
  },
  {
  "name": "Programación orientada a objetos",
  "professor": "Pedro Sanchez Perez",
  "schedule": [
                "8:00-9:00 am AG5",
                "8:00-9:00 am AG5",
                "8:00-9:00 am AG5",
                "8:00-9:00 am AG5",
                "sin horario asignado"
                ]
  },
  {
  "name": "Matemáticas Discretas",
  "professor": "Roberto Juarez Lopez",
  "schedule": [
                "9:00-10:00 am AG5",
                "9:00-10:00 am AG5",
                "9:00-10:00 am AG5",
                "9:00-10:00 am AG5",
                "sin horario asignado"
                ]
  }
  ]
```

En caso de no tener clase un día, se deberá escribir “sin horario asignado” como se muestra en el ejemplo de tal manera que la aplicación no mostrara la materia que contenga sin horario asignado en ese día. Todas las materias deberán tener en el arreglo schedule 5 strings y es deseable que se manden las materias en orden ascendente por horario como en el ejemplo para que la aplicación muestre el horario de esa manera ya que para generarlo toma de manera ordenada las materias como están en el json siendo la materia 0 la primera en aparecerá en todos los días (excepto si tiene “sin horario asignado”) sin importar el horario.

Para crear un día en la sección se debe declarar de la siguiente manera

```
var lunes = Section(day: "Lunes", subjects: daySchedule(day: 0), expanded: false )
```

Y después agregar al arreglo de sections

```
sections = [lunes, martes, miercoles, jueves, viernes]
```

El siguiente switch es para desplegar el día actual de manera automática, obteniendo el número del día (Domingo= 1, Lunes = 2, Martes= 3, Miércoles = 4, Jueves = 5, Viernes = 6, sábado = 7) enviando en el constructor “true” para que esté desplegado desde el inicio:

```
let day = Date().dayNumberOfWeek()!
        switch(day){
        case 2: //Monday
            lunes = Section(day: "Lunes", subjects: daySchedule(day: 0), expanded: true
)
            break;
        case 3: //Tuesday
            martes = Section(day: "Martes", subjects: daySchedule(day: 1), expanded:
true )
            break;
        case 4: //Wednesday
            miercoles = Section(day: "Miércoles", subjects: daySchedule(day: 2),
expanded: true )
            break;
        case 5: //Thursday
            jueves = Section(day: "Jueves", subjects: daySchedule(day: 3), expanded:
true )
            break;
        case 6: //Friday
            viernes = Section(day: "Viernes", subjects: daySchedule(day: 4), expanded:
true )
            break;
        default:
            break;
        }
```

La función:

```
func daySchedule(day : Int) -> [String]
```

Recibe un entero que representa el día que se quiere obtener (0 = lunes; 4 = viernes), y regresa una cadena el nombre, profesor y horario de todas las materias de cada día. En caso de tener “sin horario asignado”no será agregado a la cadena del día y no se mostrara en la tabla de horario.

GroupViewController.swift

En esta interfaz se usa un collectionViewController para mostrar los grupos del usuario.
Se utiliza la estructura “group” para representar a una materia, la cual tiene un nombre, id y grupo.
En la función viewDidLoad se carga un json el cual deberá contener una estructura similar a la siguiente:

```
[
  {
    "id":11,
    "nombre_completo_materia":"FUNDAMENTOS DE PROGRAMACION",
    "grupo":"A"
  },
  {
    "id":12,
    "nombre_completo_materia":"TALLER DE DESARROLLO DE SOFTWARE",
    "grupo":"A"
 },
 {
 "id":13,
 "nombre_completo_materia":"CALCULO DIFERENCIAL",
 "grupo":"A"
 }
]
```

Para poder convertir el json recibido en un arreglo de grupo y mostrarlos en la interfaz.

La función randomNumber es para regresar un valor aleatorio y mostrar uno de los colores en el arreglo de colores “colors” para que cada grupo tenga un color distinto.

La siguiente función se encarga de crear y personalizar la celda de cada grupo para mostrar el nombre de la materia, la Inicial como icono y asignarle un color aleatorio.

```
func collectionView(_ collectionView: UICollectionView, cellForItemAt indexPath:
IndexPath) -> UICollectionViewCell
```

Esta función retorna el número total de grupos que se tendrán en el CollectionView los cuales son almacenados en la variable:

```
var groups:[group]?
func collectionView(_ collectionView: UICollectionView, numberOfItemsInSection section:
Int) -> Int
```

GroupsCollectionViewCell.swift

Esta clase solo contiene los componentes que se necesitan para mostrar los grupos en la interfaz de GroupViewController. Esta compuesta por 3 controladores los cuales son:

```
@IBOutlet weak var capitalLetter: UILabel!
    @IBOutlet weak var view: UIView!
    @IBOutlet weak var name: UILabel!
```

capitalLetter es un label que mostrará la primera letra del nombre de la materia.
Name es un label que mostrará el nombre completo de la materia.

View es un view para organizar el orden de los controladores.

CreatePostViewController.swift

Esta función permite importar imágenes desde la galería de fotos del dispositivo.

```
@IBAction func importImageFromLibrary(_ sender: UIBarButtonItem)
```

Esta función coloca la imagen en un imageView de la interfaz

```
func imagePickerController(_ picker: UIImagePickerController,
didFinishPickingMediaWithInfo info: [String : Any])
```

Esta función permite que la barra del toolbar con los botones de imagen y cámara se empuje hacia arriba al mostrar el teclado

```
@objc func keyboardWillShow(_ notification: Notification)
```

Esta función permite tomar fotografías y guardar la imagen en el dispositivo

```
@IBAction func importImageFromCamera(_ sender: UIBarButtonItem)
```

Esta función borra el texto escrito en el textView para iniciar un post

```
func textViewDidBeginEditing(_ textView: UITextView)
```

ExpandableHeaderView

Section.swift

Es una estructura para definir a las secciones que se muestran en el horario, las cuales son los dñias de la semana (lunes-viernes). La variable subjects representa las materias que se van a tener por día, day representa el día con un string y expanded indica si la sección está desplegada o no.

ExpandableHeaderView

Esta clase es usada para crear secciones expansibles en ScheduleViewControler, de forma que se puedan ocultar o mostrar las secciones de la tabla.

```
Esta función permite modificar la apariencia de las secciones.
override func layoutSubviews()
```

LoginViewContoller.swift

Esta es la clase usada por la interfaz de login en la que se realizara la consulta par determinar si se le concede acceso no al usuario consta de las siguientes funciones:

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

```
        let userNoControl = controlNoText.text
        let userPassword = passwordText.text
        if(validateUserAndPassword(User: userNoControl!,Password: userPassword!)){
            UserDefaults.standard.set(true,forKey:"isUserLoggedIn")
            UserDefaults.standard.set(userNoControl!,forKey:"loggedUser")
            UserDefaults.standard.synchronize()
            errorLabel.text = ""
            self.dismiss(animated: true, completion:nil)
        } else {
            errorLabel.text = "Error, favor de verificar los datos"
            AudioServicesPlayAlertSound(SystemSoundID(kSystemSoundID_Vibrate))
            LoginViewController.shake(view: logoImage)
            LoginViewController.shake(view: controlNoText)
            LoginViewController.shake(view: passwordText)
            LoginViewController.shake(view: logInButton)
            passwordText.text = ""
        }
    }
```

Esta función es llamada al presionar el botón de la interfaz y determina que realizar dependiendo de si los datos fueron correctos o no en caso de serlo cambia dos variables globales una de tipo bool para conocer si se inicio sesión o no y el numero de control que inicio la sesión en una tipo string.

```
func validateUserAndPassword(User:String, Password:String)-> Bool{
        let users: [[String]] = [["13121005", "140995"], ["14121110", "123456"],
["14121167", "123456"],["", ""]]
        var flag = false
        for i in 1...users.count {
            if (User == users[i-1][0] && Password == users[i-1][1]){
                flag = true
                break
            }else {
                flag = false
            }
        }
        return flag
}
```

Actualmente esta función simula la existencia de usuarios y determina si la contraseña es la que corresponde a dicho usuario, esta función debe ser remplazada para que se valide la información del usuario.

La ultima función es únicamente la que se utiliza para hacer el efecto de Shake cuando la información es incorrecta.

NewsFeedtableViewController.swift

Esta es la clase usada por la interfaz de noticias la que se encuentra compuesta por cinco secciones, la primera son las funciones de ambos botones del NavigationView de la interfaz, el de buscar y el de crear post este ultimo solo manda llamar a dicha interfaz, la segunda sección son algunas de las estructuras utilizadas por la aplicación en general, la tercera los métodos normarles de la interfaz y la función prepare que envía el post a la interfaz que lo muestra, la cuarta es la sección de métodos de la barra de búsqueda instalada y la ultima los métodos utilizados por la tabla de noticias.

Sección 2

Esta es la estructura que tienen los objetos utilizados en la aplicación

```
struct Post: Decodable {
        var id_post: String = ""
        var createdBy: User? = nil
        var date: String = ""
        var group: String = ""
        var caption: String = ""
        var image: String? = nil
        var comments = [Comment]()
    }

    struct Comment: Decodable {
        var comment: String = ""
        var createdBy: User? = nil
    }

    struct User: Decodable {
        var username: String = ""
        var profileImage: String = ""
        var id_user: String = ""
    }
```

Por lo que el JSON que contiene las publicaciones tiene que ser de la siguiente forma

```
[
  {
  “id_post":"Identificador de la publicación",
  “date":"Fecha de la publicación",
  “group":"Grupo en el que se publico",
  “caption":"Texto de la publicación ",
  “image":"URL de la foto",
  "comments":[
              {
              “comment”:”texto del comentario",
              "createdBy": {
               "id_user": “Numero de control",
               "username": “Nombre del usuario",
               "profileImage": “URL donde esta almacenada la foto"
              }
              }
              ]
  "createdBy": {
  "id_user": "Numero de control",
  "username": "Nombre del usuario",
  "profileImage": "URL donde esta almacenada la foto"
  }
  }
]
```

Sección 4

En esta sección la función importante es la siguiente, actualmente se encuentra vacía pero en es en la que se recargara la interfaz filtrando los post de acuerdo a la búsqueda del usuario. Actualmente no realiza ninguna función lo que debe hacer es cambiar con base en lo que se

escriba el arreglo postsshowed que contiene los post que se mostraran y mandar llamar el método loadTable para que se actualice.

```
func searchBar(_ searchBar: UISearchBar, textDidChange searchText: String) { }
```

Sección 5

Esta sección contiene los métodos de la tabla, la ejecución del segue para mandar llamar la otra interfaz, el método para conocer la cantidad de celdas en la tabla, la función que construye las celdas, la que permite cargar por bloques, la que recibe el JSON de las publicaciones

```
    @objc func populateTableView() {
        // Obtain JSON with all the posts
        let path = Bundle.main.path(forResource: "allPosts", ofType: "json")
        let url = URL(fileURLWithPath: path!)
        do{
            let data = try Data(contentsOf: url)
            posts = try JSONDecoder().decode([Post].self, from: data)
        }
        catch {
        }
        totalEnteries = posts.count
        postsshowed.removeAll()
        var index = 0
        while index < limit {
            postsshowed.append(posts[index])
            index = index + 1
        }
        self.perform(#selector(loadTable), with: nil, afterDelay: 1.0)
        refresher.endRefreshing()
    }
```

Esta función deberá recibir el JSON de la API del SGE para cargar post reales almacenados en la base de datos, actualmente carga un archivo de ejemplo. Por ultimo en esta sección se encuentra el método loadTable que se encarga de recargar el TableView con los post que se encuentren en el arreglo postsshowed.

GroupsFeedViewContoller

Esta clase es la usada por aquella interfaz que muestra las publicaciones de un grupo, es muy parecida a la anterior, cuenta con tres secciones principales, la primera los métodos normales, su función prepare para enviar los datos de la publicación a la siguiente interfaz y una función para cargar tanto los posts como los miembros de dicho grupo, la segunda que contiene los métodos usados por la tabla y la tercera con los métodos utilizados por el CollectionView que muestra los miembros.

Sección 1

En esta sección el método importante es el siguiente:

```
 @objc func populateViews() {
        let pathmembers = Bundle.main.path(forResource: "members", ofType: "json")
```

```
        let urlmembers = URL(fileURLWithPath: pathmembers!)
        do{
            let datamem = try Data(contentsOf: urlmembers)
            members = try JSONDecoder().decode([NewsfeedTableViewController.User].self,
from: datamem)
        }
        catch { }
        let path = Bundle.main.path(forResource: "allPosts", ofType: "json")
        let url = URL(fileURLWithPath: path!)
        do{
            let data = try Data(contentsOf: url)
            posts = try JSONDecoder().decode([NewsfeedTableViewController.Post].self,
from: data)
        }
        catch { }
        totalEnteries = posts.count
        postsshowed.removeAll()
        var index = 0
        while index < limit {
            postsshowed.append(posts[index])
            index = index + 1
        }
        self.perform(#selector(loadTable), with: nil, afterDelay: 1.0)
        refresher.endRefreshing()
    }
```

Este método llena los arreglos que contienen los miembros del grupo y las publicaciones de dicho grupo, en este debe obtenerse el JSON de la API y sustituirlos por los que se cargan como ejemplo

ShowPostViewController.swift

Esta clase es utilizada por la interfaz que muestra los posts, esta recibe una publicación y la muestra cargando sus comentarios por lo que si se recibió de manera correcta el JSON anteriormente no se debe cargar nada nuevo, esta clase consta de varias secciones que realizan funciones como mostrar y ocultar el teclado, agrandar la imagen y el único cambio tiene que hacerse en la siguiente función.

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

}
```

Que es las función del botón para enviar le comentario en la que se debe cargar en la base de datos y con base en el id del post recargar ese objeto para que muestre los nuevos comentarios.

PostCell

Esta clase es la que conecta cada celda con su publicación en la interfaz de noticias, recibe un objeto tipo post y esta se encarga de cargarlo dinámicamente

CommentCell.swift

Esta clase es la que conecta cada comentario con su celda, recibe un objeto tipo comment y esa se encarga de cargarlo dinámicamente

GroupPostCell

Esta clase es la que conecta cada celda con su publicación en la interfaz de publicaciones de un grupo, recibe un objeto tipo post y esta se encarga de cargarlo dinámicamente

MembersCollectionViewCell

Este contiene los elementos de los miembros en la interfaz de publicaciones que es únicamente una imagen, esta no realiza la carga de dicha imagen solo sirve para conectarla.