

Proyecto de Prácticas

App red social - Tuister



Universidad de Jaén

Manuel Antonio Dueñas Casas
Roberto Puentes Marchal
Madc0004 - 26509590C
Rpm00052 - 26509004D

Índice

Aclaraciones iniciales	4
Usuarios	5
Acceso y registro de usuarios en la base de datos	6
Tuists	8
TablonVC.swift	8
TuistTableViewController.swift	9
BusquedaTableViewController	11
PerfilTableViewController	12
Resultado final	14

1. Aclaraciones iniciales

La aplicación que hemos implementado ha sido una red social de tablón, al estilo de Twitter, para ello hemos implementado funcionalidad de almacenamiento en la nube mediante una base de datos creada utilizando Google Firebase. En la base de datos mencionada almacenamos los usuarios creados, así como los mensajes que estos escriben en el tablón de la aplicación.

Accedemos a la base de datos tanto para enviar información como para recibirla:

- Enviamos:
 - Los credenciales de acceso de un determinado usuario una vez este se registra en el servidor.
 - Peticiones para efectuar el login con unos determinados credenciales de acceso.
 - Los mensajes escritos en el tablón por los usuarios, asignados al identificador del usuario que los escribe.
 - Modificaciones a la contraseña o correo de un determinado usuario.
- Recibimos:
 - El login del usuario al tratar de entrar en la aplicación.
 - Los mensajes enviados por otros usuarios y por nosotros mismos, que se muestran en el tablón.
 - Datos de usuarios, como el id público.

La aplicación posee una vista de búsqueda que permite filtrar los mensajes escritos por usuarios en el tablón, en función de si contienen una cadena de caracteres determinada, o están escritos por un usuario determinado. Además, en la vista de perfil de un usuario determinado se muestran todos los mensajes que han sido escritos y enviados por este, esta última vista permite también modificar los credenciales de acceso de usuario, cambiando su correo o su contraseña.

2. Usuarios

Como hemos nombrado en los conceptos iniciales nuestra aplicación se basa en la conexión a través de la nube gracias a Google Firebase, implementamos un sistema de autenticación Firebase para acceder a una sesión a través de este servicio por medio de un correo y una contraseña.

Para la realización de las pruebas hemos creado 3 usuarios de ejemplo:

Usuario: p1@t1.com

Contraseña: pruebapass123*

Usuario: p2@t.com

Contraseña: pruebapass123*

Usuario: manu@t.com

Contraseña: pruebapass123*

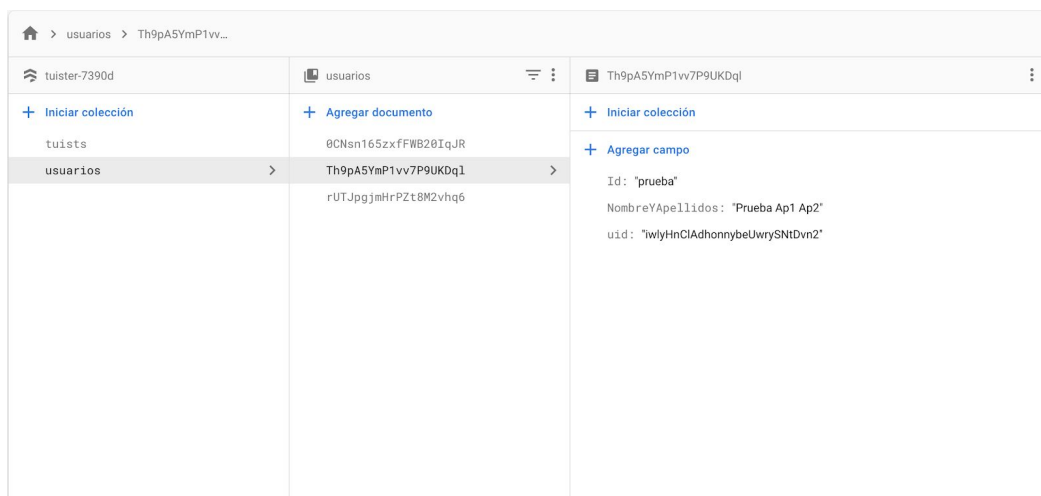
En la base de datos:



A screenshot of the Firebase console's 'Users' section. It shows a table with columns: Identificador, Proveedores, Creado, Accediste a tu cuenta, and UID de usuario. There are three users listed: p2@t.com, manu@t.com, and p1@t1.com. All were created on May 15, 2020, except for p1@t1.com which was created on May 14, 2020. All have access to their accounts as of May 15, 2020. The table has a search bar at the top and a 'Agregar usuario' button. At the bottom, it shows 'Filas por página: 50' and '1 a 3 de 3'.

Identificador	Proveedores	Creado	Accediste a tu cuenta	UID de usuario ↑
p2@t.com	✉	15 may. 2020	15 may. 2020	Xgio1WYjKSZwoiXMN0kgwFYzUs...
manu@t.com	✉	15 may. 2020	15 may. 2020	fGrNGapmOAMdz81HSjQ1ISbU5W...
p1@t1.com	✉	14 may. 2020	15 may. 2020	iwlyHnCIAdhonybeUwrySntDvn2

Además de cada usuario guardamos una serie de datos como un id público o un nombre y apellidos:



A screenshot of the Firebase Realtime Database. The path is 'usuarios > Th9pA5YmP1vv7P9UKDq1'. The selected node contains the following data: Id: 'prueba', NombreYAPELLIDOS: 'Prueba Ap1 Ap2', and uid: 'iwlyHnCIAdhonybeUwrySntDvn2'.

usuarios	Th9pA5YmP1vv7P9UKDq1
	<p>Id: "prueba"</p> <p>NombreYAPELLIDOS: "Prueba Ap1 Ap2"</p> <p>uid: "iwlyHnCIAdhonybeUwrySntDvn2"</p>

3. Acceso y registro de usuarios en la base de datos

Para loguear o registrar usuarios utilizamos la vista controlador login o registro, primero en el appDelegate realizamos la conexión con Firebase:

```
import Firebase

@UIApplicationMain
class AppDelegate: UIResponder, UIApplicationDelegate {

    func application(_ application: UIApplication, didFinishLaunchingWithOptions launchOptions: [UIApplication.LaunchOptionsKey: Any]?) -> Bool {
        // Override point for customization after application launch.
        FirebaseApp.configure()
        return true
    }
}
```

Una vez tenemos la conexión funcionando en nuestras vistas implementamos lo siguiente:

```
//acceder al usuario
Auth.auth().signIn(withEmail: stringCorreo, password: stringContraseña) { (result, error) in

    if error != nil {
        //no se pudo logear
        self.mostrarError("Error: usuario o contraseña inválidos")
    }else{
        //pasamos a tablon
        self.transicionPantallaInicio()
    }
}
```

Básicamente realiza la conexión con las credenciales de los textfield, además de unas comprobaciones previas.

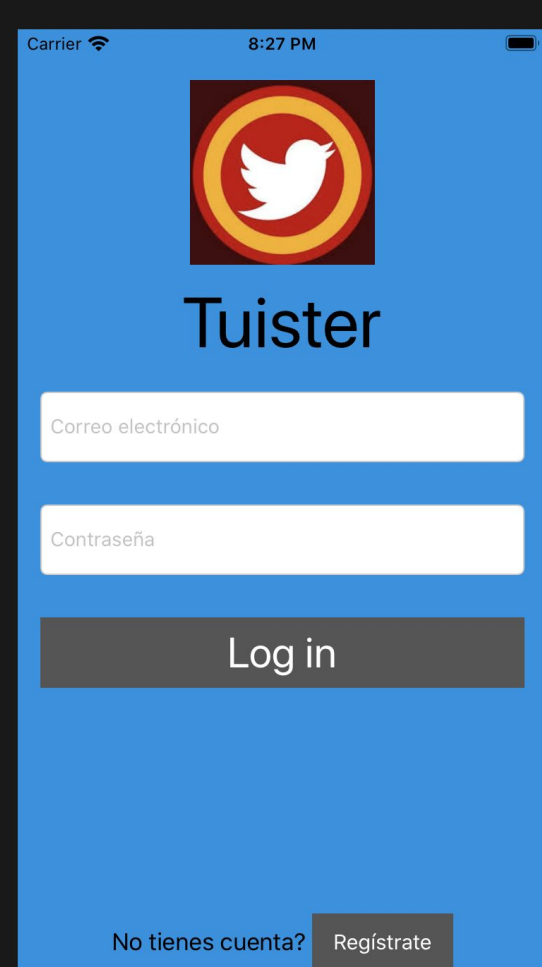
Para crear usuarios nuevos en la ventana de registro

```
//crear usuario
Auth.auth().createUser(withEmail: correo, password: contraseña) { (resultado, err) in
    //comprobar error
    if err != nil {
        //hubo un error
        self.mostrarError("Inserte un correo válido")
    }else{
        //el usuario se ha creado, vamos a guardar el Id y el nombre completo
        let db = Firestore.firestore()

        db.collection("usuarios").addDocument(data: ["Id":id, "NombreYApellidos":nombre,"uid":
            resultado!.user.uid ]) { (error) in
            if error != nil {
                self.mostrarError("error guardando datos de usuario en la base de datos")
            }
        }
        //llevar a página inicio
        self.transicionPantallaInicio()
    }
}
```

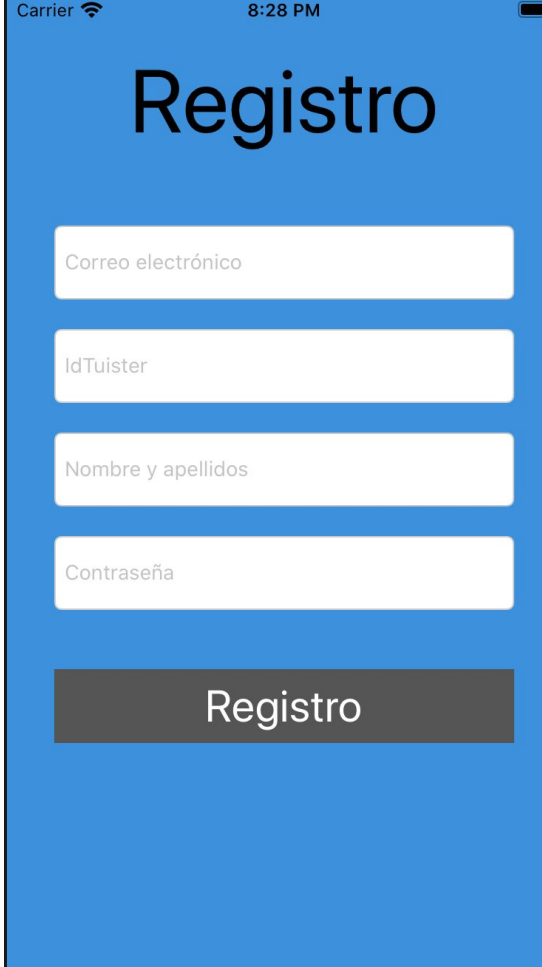
Al igual que el login tiene unas comprobaciones previas.

Escena Login



The login screen for 'Tuister' features a blue background. At the top, there is a status bar with 'Carrier', a signal icon, the time '8:27 PM', and a battery icon. Below this is the Tuister logo, which consists of a white bird inside a red and yellow circular frame. The word 'Tuister' is displayed in a large, black, sans-serif font. Underneath the logo, there are two white input fields: the first is labeled 'Correo electrónico' and the second is labeled 'Contraseña'. A dark gray button with the text 'Log in' is positioned below the input fields. At the bottom of the screen, there is a link 'No tienes cuenta?' followed by a dark gray button labeled 'Regístrate'.

Escena Registro



The registration screen for 'Tuister' has a blue background. The status bar at the top shows 'Carrier', a signal icon, the time '8:28 PM', and a battery icon. The word 'Registro' is prominently displayed at the top in a large, black, sans-serif font. Below the title, there are four white input fields: 'Correo electrónico', 'IdTuister', 'Nombre y apellidos', and 'Contraseña'. A dark gray button with the text 'Registro' is located at the bottom of the screen.

Hablaremos más detenidamente sobre los usuarios cuando lleguemos a las demás vistas.

4. Tuists

Es la estructura que utilizamos para nuestra aplicación, son unas estructuras que guarda: el mensaje, el ID del usuario que lo escribió y la hora a la que se escribió

```
protocol DocumentSerializable {
    init?(diccionario:[String:Any])
}

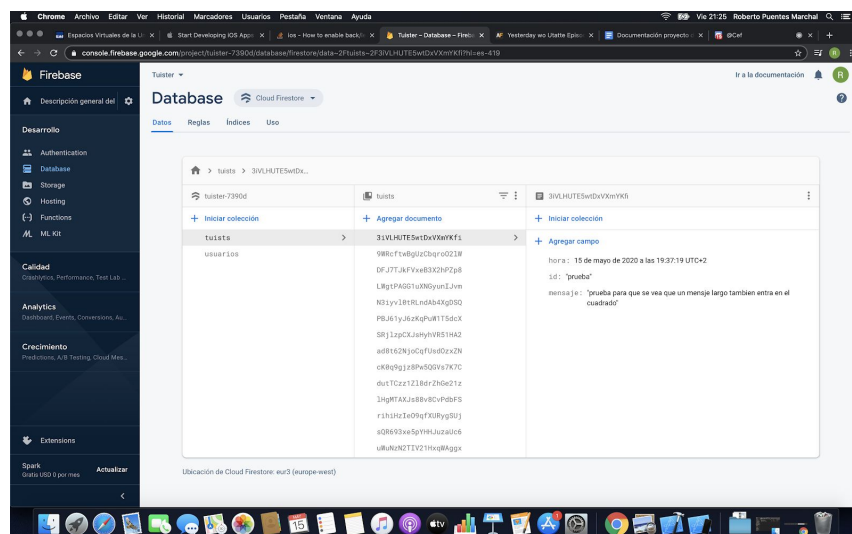
struct tuist{
    var id:String
    var mensaje:String
    var hora:Timestamp

    var diccionario:[String:Any] {
        return [
            "id":id,
            "mensaje":mensaje,
            "hora":hora
        ]
    }
}

extension tuist : DocumentSerializable{
    init?(diccionario:[String:Any]){
        guard let id = diccionario["id"] as? String,
              let mensaje = diccionario["mensaje"] as? String,
              let hora = diccionario["hora"] as? Timestamp else {return nil}

        self.init(id: id, mensaje: mensaje, hora: hora)
    }
}
```

en la base de datos:



Una vez sabemos que son los “Tuists” vamos a entrar a hablar de las vistas que implementa esta aplicación:

5. TablonVC.swift

Es la “homeScreen” o pantalla de inicio que vemos al iniciar sesión en nuestra app, se base en un tab bar controller en el que implementaremos las diferentes vistas dentro de la barra de abajo.

5.1. TuistTableViewCell.swift

En esta vista se muestran los mensajes de todos los usuarios ordenados por fecha de subida del último al primero. Además podemos utilizar el botón de arriba a la derecha para escribir nuestro “tuist”



Código a remarcar:

- Función para crear la conexión a la base de datos y encontrar el Id del usuario en base a su UID(código único creado por Firestore almacenado en la base de datos). Se ejecuta dentro de ViewDidLoad de la clase

```
db = Firestore.firestore()

let userId = Auth.auth().currentUser?.uid
print("UID USUARIO: ", userId!)
var Idabuscar: String?
self.db.collection("usuarios").getDocuments() {
    querySnapshot, error in

    if let error = error {
        print("\(error.localizedDescription)")
    }else{
        for usuario in querySnapshot!.documents {
            let uid = usuario.data()["uid"] as? String
            if uid! == userId!{
                print("Encontrado: ", uid!)
                Idabuscar = usuario.data()["Id"] as? String
                self.Id = Idabuscar!
                print("EncontradoID: ", self.Id)
            }
        }
    }
}
```


- Función cargar datos para almacenar los tuists de la base de datos dentro de nuestra app:

```
func cargarDatos(){
    db.collection("tuists").order(by: "hora", descending: true).getDocuments() {
        querySnapshot, error in
        if let error = error {
            print("\(error.localizedDescription)")
        }else{
            self.tuistArray = querySnapshot!.documents.compactMap({tuist(diccionario: $0.data())})
            DispatchQueue.main.async {
                self.tableView.reloadData()
            }
        }
    }
}
```

- función para actualizar datos en tiempo real por si algún usuario escribe un mensaje:

```
func comprobarActualizaciones() {
    db.collection("tuists").whereField("hora", isGreaterThan: Timestamp())
        .addSnapshotListener {
            querySnapshot, error in

            guard let snapshot = querySnapshot else {return}

            snapshot.documentChanges.forEach {
                diff in

                if diff.type == .added {
                    self.tuistArray.insert(tuist(diccionario: diff.document.data()!), at: 0)
                    DispatchQueue.main.async {
                        self.tableView.reloadData()
                    }
                }
            }
        }
}
```

- función de mostrar alerta para escribir un tuist:

```

@IBAction func componerTuist(_ sender: Any) {

    let composeAlert = UIAlertController(title: "Nuevo Tuist", message: "Introduce el mensaje", preferredStyle:
        .alert)

    composeAlert.addTextField { (textField:UITextField) in
        textField.placeholder = "mensaje"
    }

    composeAlert.addAction(UIAlertAction(title: "Cancelar", style: .cancel, handler: nil))

    composeAlert.addAction(UIAlertAction(title: "Publicar", style: .default, handler: { (action:UIAlertAction) in

        // INTERESTING PART

        if let mensaje = composeAlert.textFields?.first?.text{
            let nuevoTuist = tuist(id: self.Id,mensaje: mensaje, hora: Timestamp())

            var ref:DocumentReference? = nil
            ref = self.db.collection("tuists").addDocument(data: nuevoTuist.diccionario) {
                error in

                if let error = error {
                    print("Error añadiendo el tuist: \(error.localizedDescription)")
                }else{
                    print("Tuist añadido con ID: \(ref!.documentID)")
                }
            }
        }

    })

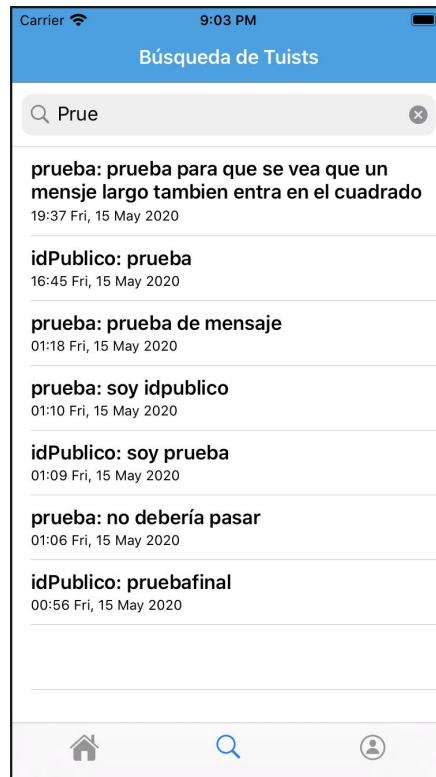
    self.present(composeAlert, animated: true, completion: nil)
}

```

- Otras funciones auxiliares y de dar formato a la fecha que no son importantes como para incluir en la documentación.

5.2. BusquedaTableViewController

Esta tableViewController parte de la anterior eliminando lo innecesario como el botón de escribir tuists.



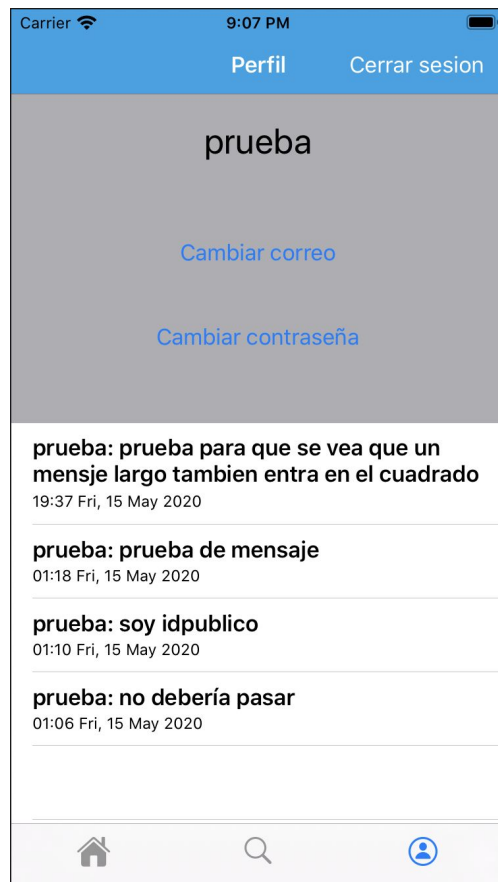
Implementa una searchbar con el siguiente método para realizar la búsqueda de tuists, tanto por id como por mensaje:

```
func searchBar(_ searchBar: UISearchBar, textDidChange searchText: String){
    guard !searchText.isEmpty else {
        currentTuistArray = tuistArray
        tableView.reloadData()
        return
    }

    currentTuistArray = tuistArray.filter({ tuist -> Bool in
        return tuist.mensaje.lowercased().contains(searchText.lowercased()) ||
            tuist.id.lowercased().contains(searchText.lowercased())
    })
    tableView.reloadData()
}
```

5.3. PerfilTableViewController

Esta última vista implementa toda la funcionalidad que queda en la app, se encarga de cambiar los datos de la cuenta logueada, ver los mensajes que se han escrito por el usuario y además se encuentra el botón de cerrar sesión:



Principales funciones:

- Botón Cerrar Sesión:

```
@IBAction func cerrarSesion(_ sender: Any) {
    do{
        try Auth.auth().signOut()
    }catch {}

    let login = storyboard?.instantiateViewController(identifier: Constantes.Storyboard.loginVC) as? Login
    view.window?.rootViewController = login
    view.window?.makeKeyAndVisible()
}
```

- Cambiar correo/contraseña, son equivalentes pero con su campo correspondiente:

```
@IBAction func botonCambiarCorreo(_ sender: Any) {
    let composeAlert = UIAlertController(title: "Cambiar correo", message: "Introduce el nuevo correo",
        preferredStyle: .alert)

    composeAlert.addTextField { (textField:UITextField) in
        textField.placeholder = "nuevo correo electrónico"
    }

    composeAlert.addAction(UIAlertAction(title: "Cancelar", style: .cancel, handler: nil))

    composeAlert.addAction(UIAlertAction(title: "Cambiar", style: .default, handler: { (action:UIAlertAction) in

        // INTERESTING PART

        if let mensaje = composeAlert.textFields?.first?.text{
            Auth.auth().currentUser?.updateEmail(to: mensaje, completion: { error in
                if error != nil{
                    let correoErroneo = UIAlertController(title: "Correo erroneo", message: "Error el correo es
                        erroneo", preferredStyle: .alert)

                    correoErroneo.addAction(UIAlertAction(title: "Entendido", style: .cancel, handler: nil))
                    self.present(correoErroneo, animated: true, completion: nil)
                }else{
                    let correoValido = UIAlertController(title: "Correo cambiado", message: "se ha cambiado el
                        correo correctamente", preferredStyle: .alert)

                    correoValido.addAction(UIAlertAction(title: "Entendido", style: .cancel, handler: nil))
                    self.present(correoValido, animated: true, completion: nil)
                }
            })
        }
    })
}
```

- Mostrar los mensajes escritos por el usuario, Parte de la func cargar datos cambiando un par de líneas para que se realice un filtrado por el Id de la sesión abierta. También se realiza el cambio a la función de comprobarActualizaciones:

```
db.collection("tuists").order(by: "hora", descending: true).getDocuments() {
    querySnapshot, error in
    if let error = error {
        print("\(error.localizedDescription)")
    }else{
        self.tuistArray = querySnapshot!.documents.compactMap({tuist(diccionario: $0.data())})
        self.tuistArray = self.tuistArray.filter({ tuist -> Bool in

            return tuist.id.contains(self.Id)
        })
        DispatchQueue.main.async {
```

6. Resultado final

Como vemos tenemos una app capaz de conectarse a una base de datos en la nube y a través de internet ser capaz de mantener los tuists y acceder a un sistema de credenciales:

Para el tema de la navegación de la aplicación sería:

Empezamos en la pantalla de login con la opción de logearnos o registrarnos, en la página de registro se permite volver atrás con el sistema de gestos. Una vez nos logeamos o creamos una cuenta nueva válida nos llevaría al tab bar controller, que por defecto nos aparece en la primera vista `tuistTableViewController`, desde aquí podemos navegar entre las pestañas del tab bar para acceder a la búsqueda o al perfil de usuario.