

# *Universidad De Las Palmas De Gran Canaria*

*Facultad de Ingeniería Informática*

## *Informe sobre del trabajo final*

*App PAMN Museum*

*Autores:*

*David Barcón Niebla, Back-end Developer*

*Romen Adama Caetano Ramírez, Front-end Developer*

# *Índice del informe de la aplicación PAMN Museum*

## ***Tabla de contenido***

<b><i>Resumen y palabras clave</i></b> .....	<b>4</b>
<b><i>Informe Final: PAMN Museum</i></b> .....	<b>5</b>
<b>Introducción</b> .....	<b>5</b>
<b><i>Metodología</i></b> .....	<b>7</b>
Romen Adama Caetano Ramírez .....	7
David Barcón Niebla.....	8
<b><i>Resultados</i></b> .....	<b>9</b>
<b>Desarrollo del Front-End</b> .....	<b>9</b>
Romen Adama Caetano Ramírez .....	9
<b>Login</b> .....	<b>9</b>
<b>Register</b> .....	<b>10</b>
<b>Reset</b> .....	<b>11</b>
<b>Home</b> .....	<b>12</b>
<b>QRlector</b> .....	<b>13</b>
<b>Ticket</b> .....	<b>14</b>
<b>Profile</b> .....	<b>15</b>
<b>Desarrollo del Back-end</b> .....	<b>16</b>
David Barcón Niebla.....	16
<b>Register</b> .....	<b>16</b>
<b>Login</b> .....	<b>18</b>
<b>Reset</b> .....	<b>20</b>

## Referencias:

- [KorGE](#)
- [Advance Wars](#)
- [HOJA DE RUTA](#)
- [GitHub](#)

<b>Home .....</b>	<b>22</b>
<b>Profile .....</b>	<b>25</b>
<b>Errores y percances .....</b>	<b>28</b>
Romen Adama Caetano Ramírez .....	28
Semana 1 .....	28
Semana 2 .....	28
Semana 3 .....	28
Semana 4 .....	28
David Barcón Niebla.....	29
Semana 1 .....	29
Semana 2 .....	29
Semana 3 .....	29
Semana 4 .....	29
<b>Referencias Bibliográficas.....</b>	<b>30</b>
Romen Adama Caetano Ramírez .....	30
David Barcón Niebla.....	30

- [KorGE](#)
- [Advance Wars](#)
- [HOJA DE RUTA](#)
- [GitHub](#)

## ***Resumen y palabras clave***

En la asignatura **40983 – Programación de Aplicaciones Móviles Nativas**, la docente **María Dolores Afonso Suarez** nos propuso como una práctica entregable realizar el prediseño de una aplicación móvil en la cual el objetivo y funcionalidad principal era la lectura de códigos QR para stands fijos en un museo o centros educativos, donde se muestre la información del objeto en cuestión.

Se realizó la tarea y se documentó un informe detallado de la realización en cuestión y subido al Moodle con la terminación de: *“Informe2\_PAMN\_Romen\_David”*. Esta tarea estaba prevista para facilitar la realización del trabajo final de la aplicación móvil en el lenguaje Kotlin, que consistiría en realizar el prediseño y transformarlo en una aplicación Android nativa mediante la herramienta o IDE Android Studio.

Nuestro equipo, en vez de realizar la tarea prevista solicitamos realizar un juego, pero tras una investigación y varios intentos de desarrollo, concluimos que era una tarea muy complicada de realizar utilizando el lenguaje de Android nativo. Por tanto, desde que fue propuesta esta tarea, Jueves 20 de Octubre, hasta determinar que era una tarea imposible pasaron un total de 5 semanas, hasta el día Jueves 17 de Noviembre. Durante este periodo el equipo fue realizando las prácticas para estudiar Codelabs además de investigar sobre los motores externos y librerías disponibles para realizar el intento de juego.

Posterior a esto, el equipo determinó dar un paso atrás e intentar realizar mediante la metodología Spring la realización de la aplicación del Museo. Nuestro objetivo era realizar un producto mínimo viable con todos los aspectos esenciales propuestos en el prediseño.

Los aspectos mínimos deseados consistían en tener una aplicación conectada con un servicio de bases de datos, para manejar la información del museo y los datos de los usuarios, además de un lector de QR conjunto a un generador de tickets.

Con esto, y teniendo en cuenta el resultado final, determinamos que hemos concluido con lo propuesto, y desarrollado nuestras capacidades y habilidades para el desarrollo de aplicaciones móviles en Android.

### Referencias:

- [KorGE](#)
- [Advance Wars](#)
- [HOJA DE RUTA](#)
- [GitHub](#)

# *Informe Final: PAMN Museum*

## Introducción

En nuestro caso en particular debemos empezar documentando nuestros errores antes de empezar a desarrollar el procedimiento del desarrollo del proyecto final.

Debemos comenzar comentando que nuestro objetivo era realizar un juego de guerra turnos, con vista isométrica con movimiento por casillas hexagonales, buscando un resultado parecido al de la saga *Advance Wars*.

Estas características suponen unas complejidades tanto de perspectiva como de movimiento, ya que actualmente, a nivel nativo, Kotlin no tiene un motor nativo que soporte estas características, ya que el sistema de importar elementos tridimensionales supone una carga excesiva para el proceso de post-procesado en los procesadores ARM de la mayoría de los dispositivos.

Para intentar desarrollar estos elementos propuestos utilizamos diversas herramientas, como importar librerías tanto como utilizar motores ajenos al propio del lenguaje. Entre ellos quisiéramos destacar el único que cuenta con un apoyo actualizado y los desarrolladores proporciona suficiente información como para intentar utilizar la herramienta, siendo este *KorGE*.

KorGE Game Engine es un motor de juegos moderno de código abierto creado en Kotlin diseñado para ser extremadamente portátil y realmente agradable de usar.

Funciona en Escritorio, Web y Móvil. Y es totalmente asíncrono, por lo que también es agradable para la web. Puedes ver una pequeña presentación de KorGE aquí: <https://korge.org/>

Ante esta situación, el equipo decidió pivotar y volver un paso atrás y recuperar el trabajo realizado en el prediseño de la App propuesta.

Con esto presente, y viendo el tiempo disponible, se tomó la decisión de hacer funcional las funciones imprescindibles de manera mínima. Utilizando la metodología Scrum, ya que buscamos agilizar los procesos mediante sprints, con ciclos semanales para realizar el proyecto en un periodo de un mes.

Nos tomamos la primera semana del proyecto, para definir la estructura para el MVP del proyecto, estableciendo que la estructura consistirá en una aplicación con autenticación, la cual tendrá tres funcionalidades principales.

- QR Scan.
- Home feed estilo Instagram.
- Tickets de validación de actividades con muestreo de información.

Referencias:

- *KorGE*
- *Advance Wars*
- HOJA DE RUTA
- GitHub

Las siguientes semanas, gracias a la realización de los Codelabs, pudimos tomar ventaja de los conocimientos y directamente pudimos empezar a desarrollar las distintas screens de la app.

La primera semana consistió en formalizar la estructura y navegación de las distintas screens dentro de la aplicación, implementando la screen Home y QRscan.

La segunda semana consistió en formalizar errores acumulados y fallos a la hora de implementar un menú burger conjunto al Bottom NAV. Desarrollo final de la Screen Home estilo Instagram e implementación del lector de QR usando la librería Zxing.

La tercera semana consistió en implementar Screens Login, Register y Reset y las funciones lógicas, conjunto a un segundo sistema de navegación.

La cuarta y última semana consistió en depurar códigos y conjuntar colores, estructura y realizar testeo de las funcionalidades.

Referencias:

- [KorGE](#)
- [Advance Wars](#)
- [HOJA DE RUTA](#)
- [GitHub](#)

# ***Metodología***

## ***Romen Adama Caetano Ramírez***

La tecnología para el desarrollo de la aplicación es Jetpack Compose, Kotlin.

Como desarrollador Front-end del equipo, mis tareas han sido organizadas utilizando la metodología Scrum, con el enfoque de agilizar las tareas planteadas, donde como núcleo central decidimos utilizar los sprints.

Como guía de las tareas propuestas y organizadas el equipo desarrollo un Excel donde semanalmente, donde los días de sesiones practicas se organizaban las tareas a realizar.

Mis tareas desde el comienzo del proyecto PAMN Museum, fueron las siguientes:

- **Semana 1**
  - Screens mainScreen, MainActivity, ScreenHome, BottonNav, Profile BottomScreen, DataClass, Ticket
- **Semana 2**
  - Screens QrLector, QrCodeAnalyzer y Profile
  - Depuracion Home, Ticket, mainScreen, MainActivity, Profile
- **Semana 3**
  - Screens Login, Register, Reset, TextFieldHelper BottonNav y BottomScreen
  - Depuracion Home, Ticket, mainScreen, MainActivity, Profile
- **Semana 4**
  - Screens Ticket y Depuracion del código para sincronizar colores y estructura visual de la App.
  - Depuracion Home, mainScreen, MainActivity, QrLector, QrCodeAnalyzer

En caso de que se quisiera consultar la hoja de ruta planteada por el equipo, se adjunta el link al libro de Excel utilizado: [HOJA DE RUTA](#).

Referencias:

- [KorGE](#)
- [Advance Wars](#)
- [HOJA DE RUTA](#)
- [GitHub](#)

## ***David Barcón Niebla***

La tecnología usada para el desarrollo de la aplicación fue Kotlin junto a la librería de Firebase firestore y auth.

Como desarrollador principalmente de back, me dedique a realizar tareas relacionadas con el correcto comportamiento de la app.

Mi metodología ha sido scrum, de forma que así se agiliza el proceso de desarrollo usando sprints.

Como guía de las tareas propuestas y organizadas el equipo desarrollo un Excel donde semanalmente, donde los días de sesiones practicas se organizaban las tareas a realizar.

Mis tareas desde el comienzo del proyecto PAMN Museum, fueron las siguientes:

- **Semana 1**
  - Investigación sobre el estudio del Back-end
  - Realización de Codelabs de Kotlin Firebase
- **Semana 2**
  - Back del Home
- **Semana 3**
  - Back de Login, Register y Reset
- **Semana 4**
  - Back de Profile
  - Diseño final de la Screen Profile, implementando la funcionalidad log-out.

En caso de que se quisiera consultar la hoja de ruta planteada por el equipo, se adjunta el link al libro de Excel utilizado: [HOJA DE RUTA](#).

### Referencias:

- [KorGE](#)
- [Advance Wars](#)
- [HOJA DE RUTA](#)
- [GitHub](#)



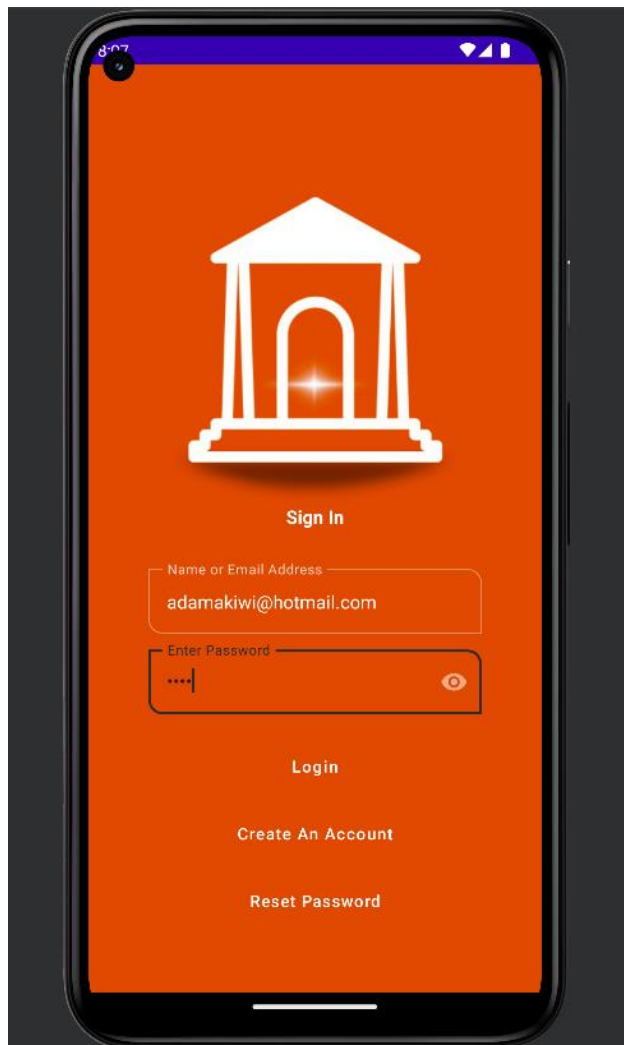
## ***Resultados***

### **Desarrollo del Front-End**

***Romen Adama Caetano Ramírez***

Empezaremos a mostrar lo desarrollado en orden ***no cronológico***, sino en orden de cómo se ejecuta la App para hacerlo más natural a la hora de experimentar. Los títulos de las vistas tienen asociado el código de cada pantalla. Link a la rama principal del [GitHub](#), donde dentro del README se encuentra un informe detallado de cada elemento que conforma la App, además de los comentarios internos dentro de cada fichero que conforma la aplicación.

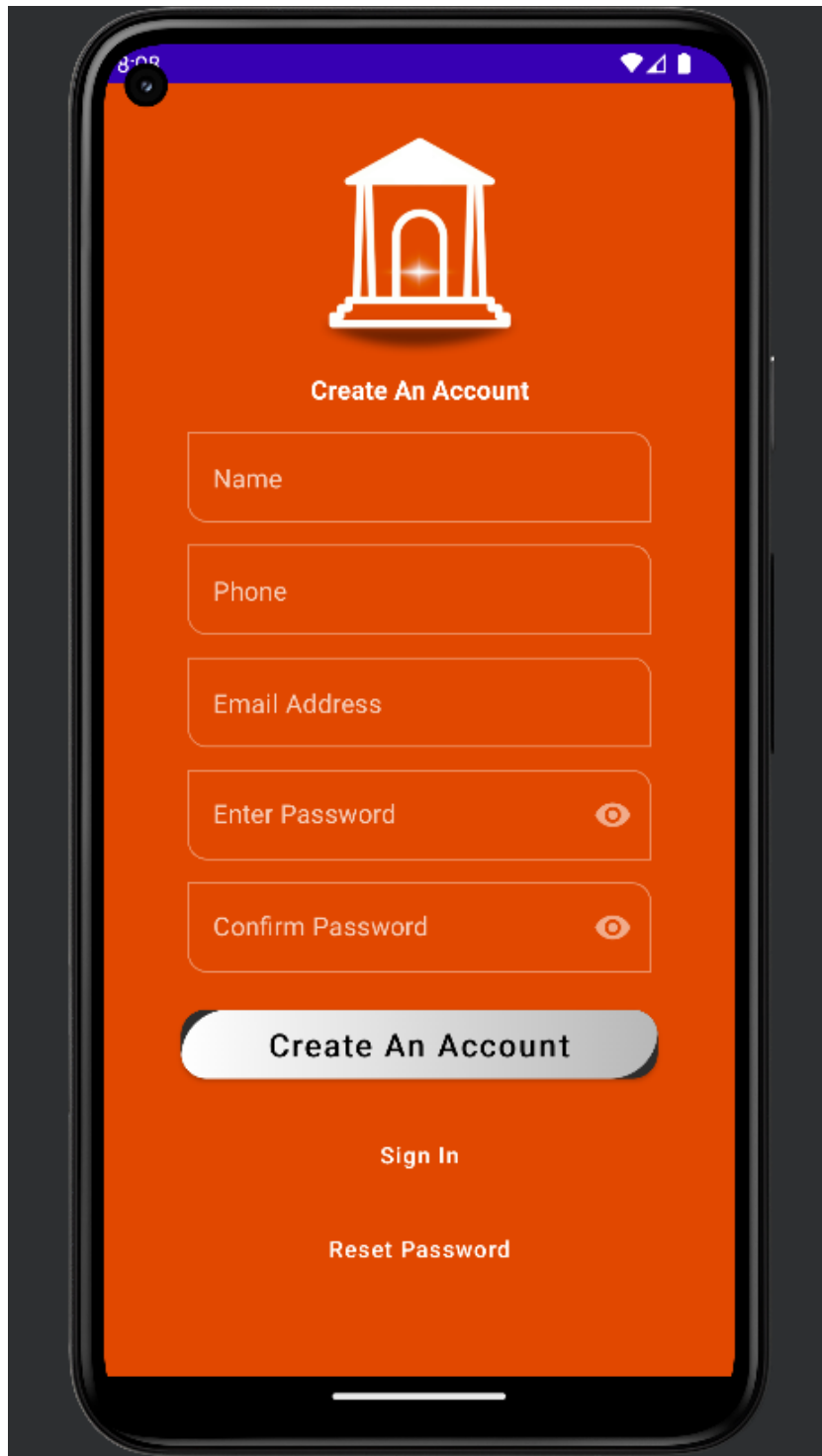
### **Login**



Referencias:

- [KorGE](#)
- [Advance Wars](#)
- [HOJA DE RUTA](#)
- [GitHub](#)

## Register

A mobile application registration screen with an orange background. At the top, there is a white icon of a classical building with columns. Below the icon, the text "Create An Account" is centered. The form consists of five input fields: "Name", "Phone", "Email Address", "Enter Password", and "Confirm Password". The "Enter Password" and "Confirm Password" fields have an eye icon to the right, indicating a password toggle. Below the input fields is a large, rounded button labeled "Create An Account". At the bottom of the screen, there are two links: "Sign In" and "Reset Password". The screen is framed by a black border representing a smartphone.

8:08

Create An Account

Name

Phone

Email Address

Enter Password

Confirm Password

Create An Account

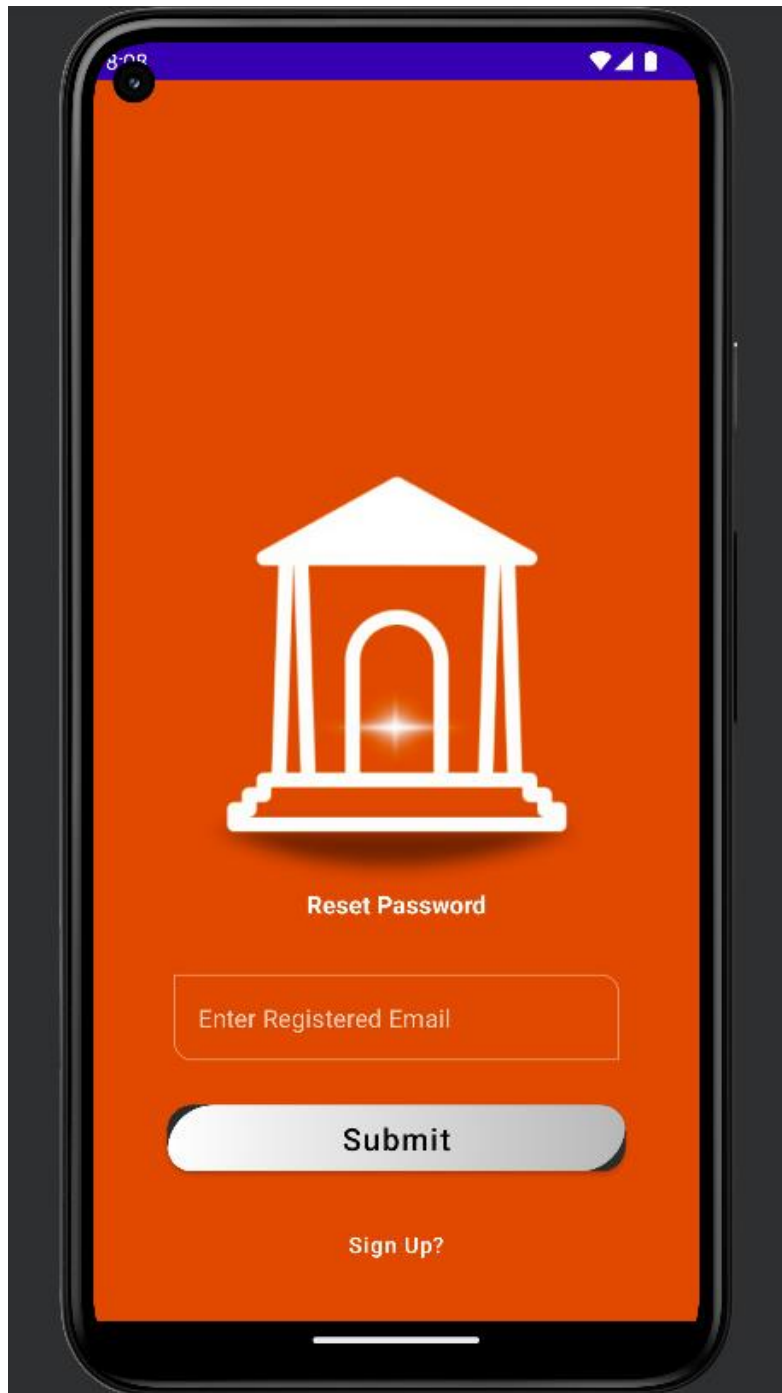
Sign In

Reset Password

Referencias:

- [KorGE](#)
- [Advance Wars](#)
- [HOJA DE RUTA](#)
- [GitHub](#)

## Reset



Referencias:

- [KorGE](#)
- [Advance Wars](#)
- [HOJA DE RUTA](#)
- [GitHub](#)

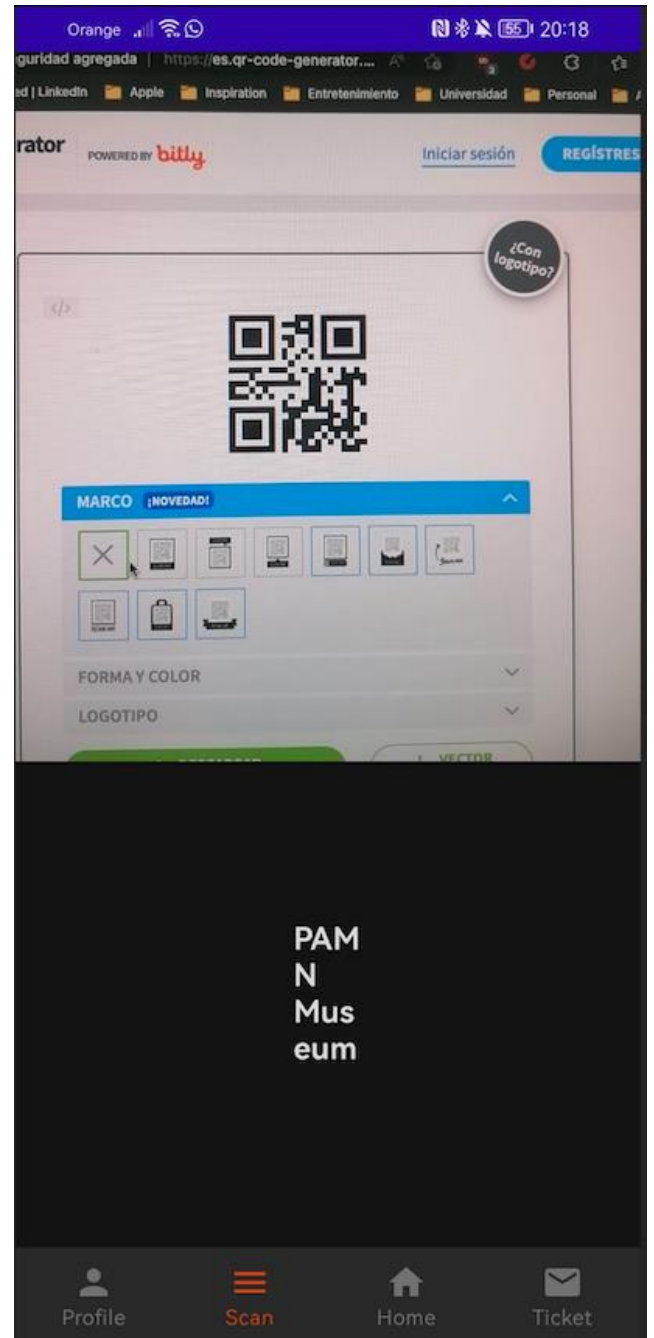
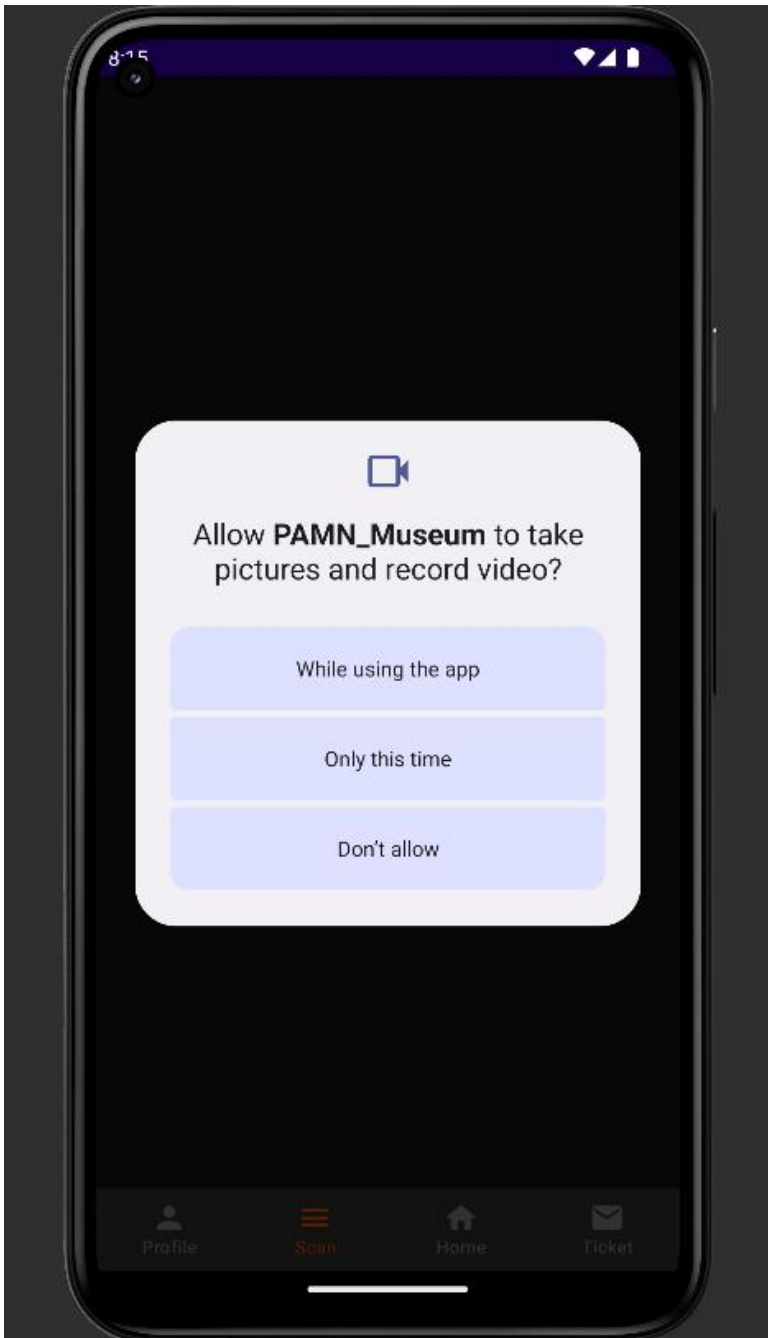
## Home



Referencias:

- KorGE
- Advance Wars
- HOJA DE RUTA
- GitHub

## ORLector



### Referencias:

- KorGE
- Advance Wars
- HOJA DE RUTA
- GitHub

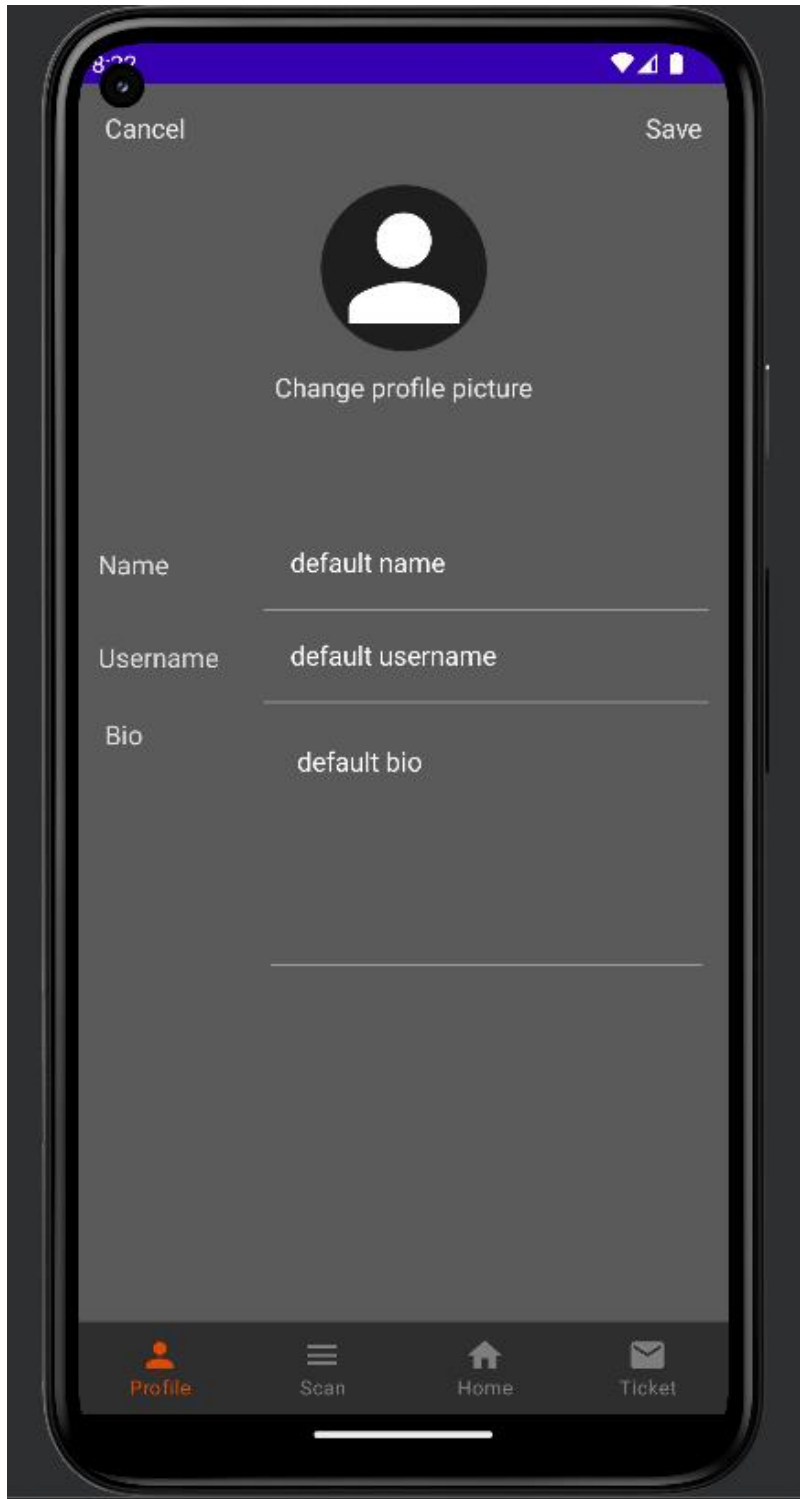
## *Ticket*



Referencias:

- [KorGE](#)
- [Advance Wars](#)
- [HOJA DE RUTA](#)
- [GitHub](#)

## Profile



Referencias:

- [KorGE](#)
- [Advance Wars](#)
- [HOJA DE RUTA](#)
- [GitHub](#)

## Desarrollo del Back-end

*David Barcón Niebla*

A continuación, explico cómo funciona el back en profundidad. El orden, al igual que la sección anterior, será el orden que vería el usuario si estuviera usando la aplicación.

No se asociará ningún link a GitHub ya que eso ya fue hecho en el apartado anterior.

### Register

Esta es la pantalla más importante de la parte de autenticación. Permite crear usuarios con los que iniciar sesión. Igual que en el login, se crea un DataClass para almacenar la información introducida en los campos.

```
data class regUser(  
    var email: String,  
    var password: String,  
    var passwordCheck: String,  
    var name: String,  
    var phone: String  
)  
  
@Composable  
fun RegisterPage(navController: NavController) {  
    var regUser = regUser( email: "", password: "", passwordCheck: "", name: "", phone: "")  
    Box(  
        contentAlignment = Alignment.Center,  
        modifier = Modifier.fillMaxSize()  
    ) {  
        Text("Register")  
    }  
}
```

Una vez esta información es introducida, se usará de 2 formas. En primer lugar, se usará para crear el usuario en la parte de autenticación de Firebase. En segundo lugar, se creará un perfil de usuario en firestore con el correo y el uid del usuario además del nombre y teléfono.

Referencias:

- [KorGE](#)
- [Advance Wars](#)
- [HOJA DE RUTA](#)
- [GitHub](#)



```

if(regUser.password == regUser.passwordCheck){
    Firebase.auth.createUserWithEmailAndPassword(regUser.email, regUser.password)
        .addOnCompleteListener{task ->
            if (task.isSuccessful) {
                Log.i( tag: "registro", msg: "register success")
                val user = hashMapOf(
                    "uid" to task.result.user?.uid,
                    "name" to regUser.name,
                    "email" to regUser.email,
                    "phone" to regUser.phone
                )
                task.result.user?.uid?.let { it: String
                    Firebase.firestore.collection( collectionPath: "users") CollectionReference
                        .document(it) DocumentReference
                            .set(user) Task<Void!>
                                .addOnSuccessListener { documentReference ->
                                    Log.d( tag: "registro", msg: "Usuario con id: ${documentReference} y nombre
                                }
                            }
                }
            } else {
                Log.e( tag: "registro", msg: "register error",task.exception)
            }
        }
}
}

```

Se realizan la comprobación para ver si las contraseñas introducidas son iguales, esto es una medida para asegurar que la contraseña introducida por el usuario no está mal escrita por equivocación. Si fueran a ser distintas, el proceso de registro se para y no sigue hasta que estén bien escritas y se pulse el botón de nuevo.

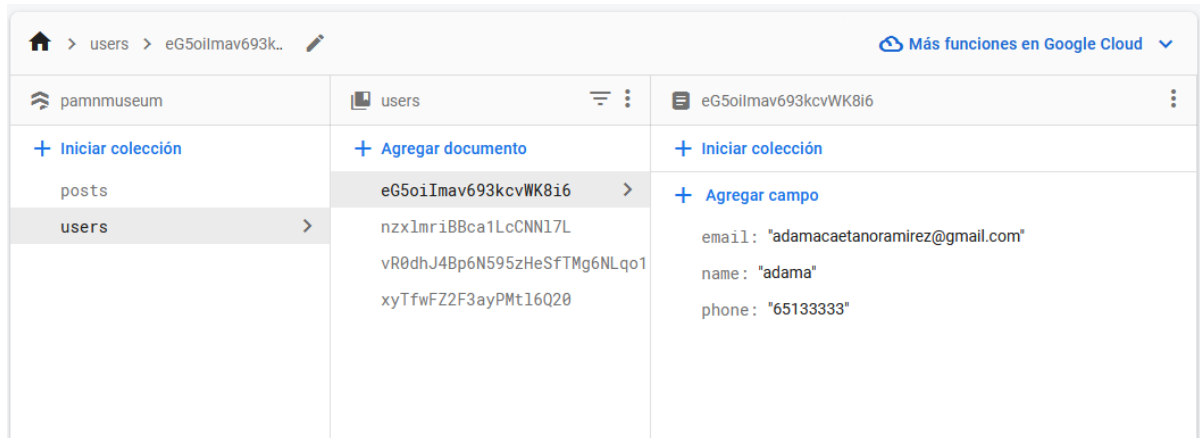
Después de eso, se usará una función proporcionada por la librería de Firebase para crear un usuario con el correo y la contraseña. Una vez creado el usuario, este se añadirá al Firebase si ha salido bien la operación.

<div> <input type="text" value="Buscar por dirección de correo electrónico, número de teléfono o UID de usuario"/> <div> <div>Agregar usuario</div> <div> <div></div> <div></div> </div> </div> </div>				
Identificador	Proveedores	Fecha de creación	Fecha de acceso	UID de usuario
adamacaetanoramirez@g...		18 dic 2022	18 dic 2022	U4bqPLoEeMcUKuJhGBtI4VZgux1
barconiebladavid@gmail.c...		18 dic 2022	18 dic 2022	vR0dhJ4Bp6N595zHeSfTMg6NLq...
<div> <div>Filas por página: 50</div> <div>1 - 2 of 2</div> <div> <div></div> <div></div> </div> </div>				

Referencias:

- [KorGE](#)
- [Advance Wars](#)
- [HOJA DE RUTA](#)
- [GitHub](#)

Una vez creado el usuario se usan el resto de datos para añadir un documento a la base de datos de firestore con la información del nombre, el correo y el teléfono. Cabe destacar que el id del documento es el mismo que el uid del usuario, esto es útil para realizar búsquedas rápidamente a la hora de buscar información del usuario si quisiera mostrarse en pantalla.



## Login

Esta es la primera pantalla que verá el usuario nada más entrar a la app. En esta pantalla, además de iniciar sesión se puede acceder a las pantallas de registro y recuperación de contraseña. En primer lugar, el usuario deberá introducir el correo y contraseña de una cuenta no existente, en el caso de no existir o ser incorrectos, no se iniciará sesión.

```
@Composable
fun LoginPage(navController: NavController, ) {
    var loginUser = LoginUser( email: "", password: "")
    if(FirebaseAuth.getInstance().currentUser != null){
        navController.navigate( route: "main_page"){ this: NavOptionsBuilder
            popUpTo(navController.graph.startDestinationId)
            launchSingleTop = true
        }
    }
}
```

Referencias:

- [KorGE](#)
- [Advance Wars](#)
- [HOJA DE RUTA](#)
- [GitHub](#)

Como comprobación inicial, se comprueba si ya hay una sesión iniciada, en cuyo caso, se omitirá el paso de iniciar sesión.

```
data class LoginUser(  
    var email: String,  
    var password: String  
)  
  
@Composable  
fun LoginPage(navController: NavController, ) {  
    var loginUser = LoginUser( email: "", password: "")  
    if(FirebaseAuth.getInstance().currentUser != null){
```

Para guardar el correo y contraseña de forma temporal se crea un objeto de un DataClass.

```
TextButton(onClick = {  
    Log.i( tag: "login", msg: "${loginUser.email} ${loginUser.password}")  
    FirebaseAuth.signInWithEmailAndPassword(loginUser.email, loginUser.password)  
        .addOnCompleteListener{task ->  
        if (task.isSuccessful) {  
            Log.i( tag: "login", msg: "login success")  
            navController.navigate( route: "main_page"){ this: NavOptionsBuilder  
                popUpTo(navController.graph.startDestinationId)  
                launchSingleTop = true  
            }  
        } else {  
            Log.e( tag: "login", msg: "login error",task.exception)  
        }  
    }  
}) { this: RowScope  
    Text(  
        text = "Login",  
        color = Color.White  
    )  
}
```

Referencias:

- [KorGE](#)
- [Advance Wars](#)
- [HOJA DE RUTA](#)
- [GitHub](#)

Cuando el usuario presiona el botón de login, se activará el Listener del botón. Esto provoca que se haga una llamada a una función de la librería de Firebase que permite iniciar sesión con credenciales que estén presentes en la base de datos. Una vez llamada a la función usando los datos guardados de los campos de correo y contraseña se hará una comprobación de si fue exitosa la tarea. En caso de fallar deberá ser que no existe un usuario con ese correo o que la contraseña está mal escrita.

## Reset

Esta pantalla fue posiblemente la más simple de implementar de todas. Esta pantalla sirve para que en el caso de que un usuario pierda o se olvide de su contraseña, puede reiniciarla. Esto se consigue introduciendo el correo del usuario y pulsando un botón.

Cundo se pulsa este botón, se hará una llamada a Firebase para que mande un correo de recuperación al usuario en cuestión.

```
fun ResetEmailID(email: email) {  
    val keyboardController = LocalSoftwareKeyboardController.current  
    var text by rememberSaveable { mutableStateOf( value: "") }  
  
    OutlinedTextField(  
        value = text,  
        onValueChange = { text = it;  
                           email.mail = it},  
        shape = RoundedCornerShape(topEnd =12.dp, bottomStart =12.dp),  
        label = {  
            Text( text: "Enter Registered Email",  
                ) },  
    )
```

Como se puede ver, el correo del campo de texto se guarda en una variable, que más tarde se mandara a la función de Firebase.

### Referencias:

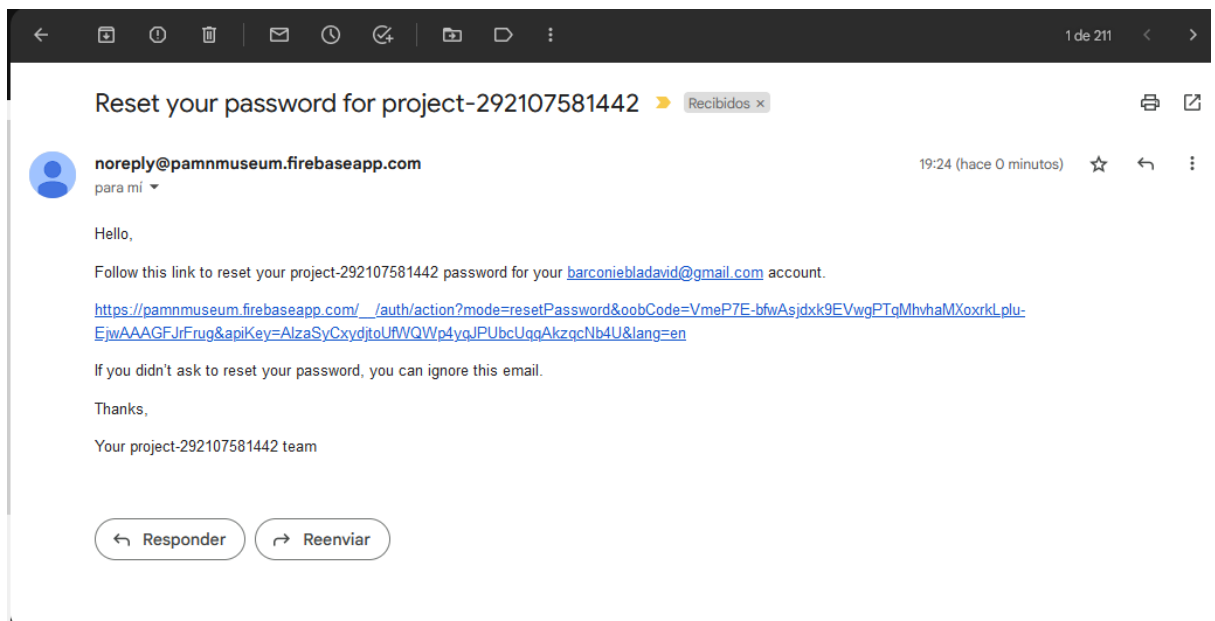
- [KorGE](#)
- [Advance Wars](#)
- [HOJA DE RUTA](#)
- [GitHub](#)

```

Button(
    modifier = Modifier
        .fillMaxWidth()
        .padding(start = 32.dp, end = 32.dp),
    onClick = {
        Firebase.auth.sendPasswordResetEmail(email.mail)
            .addOnCompleteListener {t ->
                Log.i( tag: "reset", msg: "correo mandado")
            }
    },
    contentPadding = PaddingValues(),
    colors = ButtonDefaults.buttonColors(
    ),
    shape = RoundedCornerShape(cornerRadius)
) {
    this RowScope
}

```

Una vez pulsado el botón, este usará el correo introducido anteriormente para mandar un correo con el cual se podrá recuperar la contraseña.

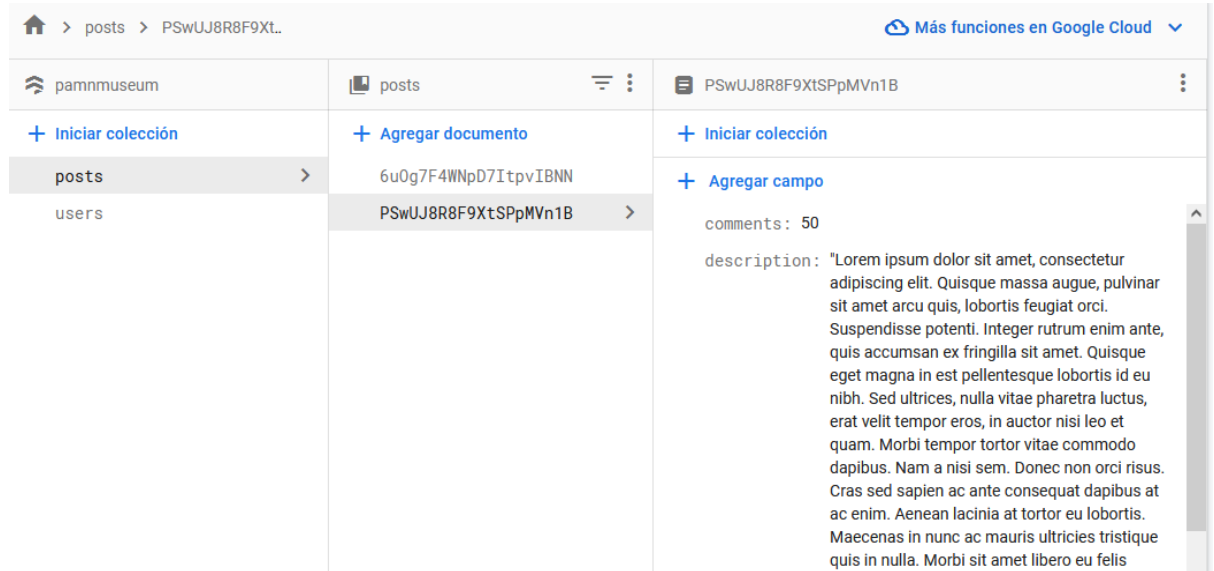


Referencias:

- KorGE
- Advance Wars
- HOJA DE RUTA
- GitHub

## Home

La primera pantalla que verá un usuario al iniciar sesión será el muro de publicaciones, o home. En esta pantalla aparecen las publicaciones que han sido guardadas anteriormente en la base de datos de Firebase.



Una publicación se compone de una descripción, la cantidad de favoritos y comentarios que tiene, (los cuales no son funcionales, solo decorativos por el momento) y la imagen. De la cual se hablará más adelante.

Para acceder a los documentos de las publicaciones almacenadas en la base de datos se usará una consulta de Firebase con la colección de “posts”, que es la que contiene las publicaciones.

Referencias:

- [KorGE](#)
- [Advance Wars](#)
- [HOJA DE RUTA](#)
- [GitHub](#)

```

val db = Firebase.firestore

fun loadPosts(): MutableList<MockPost> {
    var posts = mutableListOf<MockPost>()

    db.collection(collectionPath: "posts") CollectionReference
        .get() Task<QuerySnapshot>
        .addOnSuccessListener { result ->
            for (document in result){
                var description = document.data.get("description") as String
                var likes = (document.data.get("likes") as Long).toInt()
                var comments = (document.data.get("comments") as Long).toInt()
                var image = document.data.get("image") as String

                //Datos de usuario Random
                val uid = UUID.randomUUID().toString()
                val userImage = R.drawable.logo_white
                val username = " PAMN MUSEUM"
                var post= MockPost(uid, userImage, username, image, description, likes, comments)
                posts.add(post)

                Log.i( tag: "firebase", msg: "${post.description} ${post.image} ${post.likes} ${post.comments}")
            }
        }
}

```

Aquí como se puede ver se cargan las publicaciones en una lista que más adelante se usara para cargar los posts.

Cabe destacar que la imagen se carga como una string, ya que esta es la url para la misma.

Las imágenes están guardadas en Firebase storage para poder ser accedidas online y no de forma local.

## Storage

Files Rules Usage

Protege tus recursos de Storage contra los abusos, como fraudes de facturación o suplantación de identidad. [Configurar la Verificación de aplicaciones](#)

gs://pammuseum.appspot.com [Subir archivo](#)

<input type="checkbox"/>	Nombre	Tamaño	Tipo	Modificación más reciente
<input type="checkbox"/>	angel.jpg	4.2 MB	image/jpeg	15 dic 2022
<input type="checkbox"/>	cuevaverdes.jpeg	54.3 KB	image/jpeg	15 dic 2022
<input type="checkbox"/>	lanzarote.jpeg	234.33 KB	image/jpeg	15 dic 2022
<input type="checkbox"/>	monalisa.jpeg	5.24 KB	image/jpeg	15 dic 2022
<input type="checkbox"/>	nocheestrellada.jpeg	201.49 KB	image/jpeg	15 dic 2022

Referencias:

- [KorGE](#)
- [Advance Wars](#)
- [HOJA DE RUTA](#)
- [GitHub](#)

Para cargar la imagen simplemente se usa AsyncImage, que permite cargar imágenes desde la red.

```
AsyncImage(  
    model = post.image,  
    contentDescription = null,  
    modifier = Modifier.fillMaxWidth(),  
    contentScale = ContentScale.FillWidth  
)
```

El hecho de que las publicaciones estén completamente guardadas en la nube permitirá a los administradores actualizar las publicaciones sin necesidad de actualizar la app.

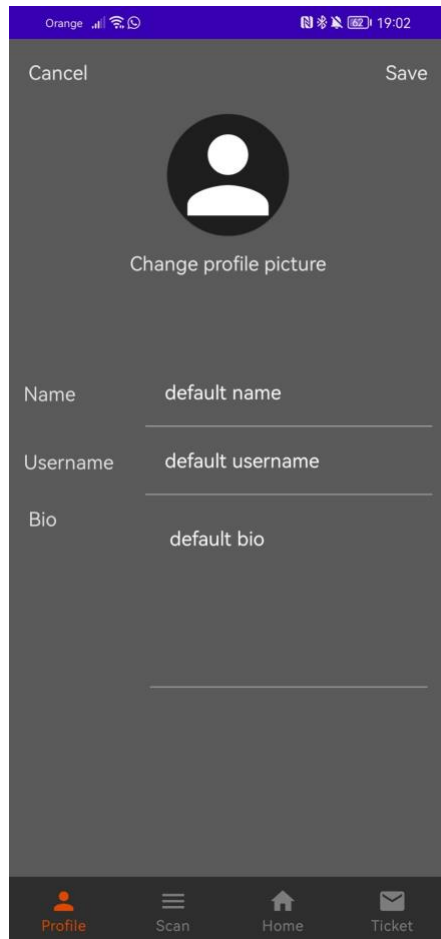
Referencias:

- [KorGE](#)
- [Advance Wars](#)
- [HOJA DE RUTA](#)
- [GitHub](#)



## Profile

La pantalla fue originalmente diseñada para poder editar los datos de usuario como el nombre y la bio, pero debido a limitaciones de tiempo se quedó en que solamente muestra la información del usuario.



Mas adelante la bio se sustituyó por el teléfono móvil, así que iba a tener el nombre, correo y teléfono del usuario mostrado en pantalla. Tras mucha investigación, resultó ser mucho más complejo de lo que esperábamos. Debido a que la información de usuario que no sea el correo y la contraseña debe ser guardada en un documento de firestore. Para sacar la información adicional del usuario hay que hacer una consulta a la base de datos, lo cual es realizado de manera asíncrona al resto del proceso.

Este problema provoca que haya que implementar una corrutina, que en Compose fue más difícil de hacer, así que por limitaciones de tiempo nos vimos forzados a que solo

Referencias:

- [KorGE](#)
- [Advance Wars](#)
- [HOJA DE RUTA](#)
- [GitHub](#)

se mostrara el correo, que esta guardado en la sesión, por lo que no requiere una consulta a la base de datos.

```
Row(  
    modifier = Modifier  
        .fillMaxWidth()  
        .padding(start = 4.dp, end = 4.dp),  
    verticalAlignment = Alignment.CenterVertically  
) { this: RowScope  
    Text(text = "Bienvenido/a ${FirebaseAuth.getInstance().currentUser?.email}" ,  
        modifier = Modifier  
            .fillMaxWidth()  
            .padding(8.dp),  
        color = White500)  
}
```

En la imagen anterior se puede ver justo eso. Se llama a la instancia de Firebase, que tiene almacenado el uid y el correo del usuario además de metadatos.

En esta pantalla también se puede apreciar un botón para cerrar sesión. Esto funciona simplemente lanzando un comando para que la sesión se cierre seguido del cierre de la app.



Referencias:

- [KorGE](#)
- [Advance Wars](#)
- [HOJA DE RUTA](#)
- [GitHub](#)

```

Row(
  modifier = Modifier
    .fillMaxWidth()
    .padding(8.dp),
  horizontalArrangement = Arrangement.SpaceBetween
) { this: RowScope
  Text(text = "Sign out",
    modifier = Modifier.clickable {
      FirebaseAuth.getInstance().signOut() ;

      exitProcess( status: -1)
    })
}

```

Referencias:

- [KorGE](#)
- [Advance Wars](#)
- [HOJA DE RUTA](#)
- [GitHub](#)

# Errores y percances

*Romen Adama Caetano Ramírez*

## *Semana 1*

- Pivote de la vista diseñada en Figma sobre TicketQR Info a un sistema animado de tickets donde se almacenan localmente y en un futuro los administradores podrían generar como actividad.

## *Semana 2*

- Marcar incidencia entre la incompatibilidad detectada entre aplicar un NavHost y un NavController.
- No posible la implementación del BottonNav y un BurguerNav
- Depuracion de los códigos

## *Semana 3*

- Navigation desarrollada incompatible con las pestanas Login, Register y Reset debido a que la diseñada está enfocada para una navegación por clics de iconos, por tanto, ha de realizarse una nueva navegación.

## *Semana 4*

- Depuracion del código presenta errores con el aspecto Profile al sincronizar los datos provenientes de la base de datos. Incompatibilidad que repercute en el diseño inicial.

Referencias:

- [KorGE](#)
- [Advance Wars](#)
- [HOJA DE RUTA](#)
- [GitHub](#)

## Errores y percances

*David Barcón Niebla*

### *Semana 1*

- Problemas de hardware que impedían usar un emulador, por lo que se tuvo que pasar a usar un dispositivo físico

### *Semana 2*

- Nada a destacar.

### *Semana 3*

- Problemas al integrar la funcionalidad de Firebase con Jetpack Compose. Esto provoco un retraso el desarrollo y provoco compromisos en el desarrollo.
- Retrasos debido a la navegación. La navegación tuvo que desarrollarse con un nuevo enfoque a aparte del diseñado en un inicio para las funcionalidades del login.

### *Semana 4*

- Depuracion del código presenta errores con el aspecto Profile al sincronizar los datos provenientes de la base de datos. Incompatibilidad que repercute en el diseño inicial.

Referencias:

- [KorGE](#)
- [Advance Wars](#)
- [HOJA DE RUTA](#)
- [GitHub](#)

## Referencias Bibliográficas

*Romen Adama Caetano Ramírez*

Lo utilizado para el desarrollo esta adjunto con su link dentro de cada fichero utilizado para facilitar la comprensión y distribución de los tutoriales respecto al código que se dispone a interpretar.

*David Barcón Niebla*

- [Codelabs de firebase](#)
- [firebase docs](#)
- [firebase firestore docs](#)
- [firebase authentication](#)
- [firebase storage](#)
- [android docs](#)
- [stack overflow](#)

Referencias:

- [KorGE](#)
- [Advance Wars](#)
- [HOJA DE RUTA](#)
- [GitHub](#)