INSTITUCIÓN: INSTITUTO TECNOLÓGICO SUPERIOR DE SAN PEDRO DE LAS COLONIAS

UNIDAD 3˚: SMBD MÓVILES

ACTIVIDAD: APLICACIÓN GUSHOP

ALUMNO: SERGIO SERVANDO PRADO LOZANO, ALVARO SANCHEZ GONZALEZ

SEMESTRE: 7˚ “A”

DOCENTE: EDUARDO ZAMORA QUEZADA

FECHA DE ENTREGA: 09/12/2025

APLICACIÓN GUSHOP

1. Dependencias del Proyecto (build.gradle.kts)

El archivo build.gradle.kts contiene todas las librerías externas que el proyecto utiliza. Cada una cumple una función específica dentro de tu aplicación.

Firebase BOM

```
// Firebase BOM
implementation(platform( notation = "com.google.firebase:firebase-bom:33.5.1"))
```

Permite manejar versiones compatibles de todos los servicios de Firebase desde un solo punto.

Firebase services

```
// Firebase services
implementation("com.google.firebase:firebase-auth-ktx")
implementation("com.google.firebase:firebase-analytics-ktx")
implementation("com.google.firebase:firebase-database-ktx")
```

Firebase Aunthentication:

Sirve para autenticar usuarios mediante correo y contraseña.

Se usa directamente en:

- LoginActivity
- RegisterActivity
- ChatActivity (para obtener el email del usuario que envía mensajes)

Firebase Realtime Database:

Este módulo permite guardar y leer mensajes en tiempo real dentro del chat grupal.

Se usa principalmente en:

- ChatActivity
- ChatAdapter

Firebase Analytics

Registra métricas del uso de la app.

AndroidX (UI básica)

```
// AndroidX
implementation("androidx.core:core-ktx:1.12.0")
implementation("androidx.appcompat:appcompat:1.6.1")
implementation("com.google.android.material:material:1.12.0")
implementation("androidx.activity:activity-ktx:1.9.0")
implementation("androidx.constraintlayout:constraintlayout:2.1.4")
```

Proveen componentes visuales modernos, temas Material Design y compatibilidad.

RecyclerView

```
// RecyclerView
implementation("androidx.recyclerview:recyclerview:1.3.1")
```

Permite mostrar listas dinámicas.

Se usa en:

- Lista de productos (API FakeStore)
- Lista de mensajes del chat grupal

Retrofit + Gson

```
// Retrofit + Gson
implementation("com.squareup.retrofit2:retrofit:2.9.0")
implementation("com.squareup.retrofit2:converter-gson:2.9.0")
```

Retrofit permite hacer peticiones HTTP a APIs externas.

Gson convierte JSON a objetos Kotlin automáticamente.

En este proyecto Retrofit se usa para:

- Obtener productos desde fakestoreapi.com
- Obtener un usuario random desde randomuser.me

Clases donde se usa:

- ApiService
- RetrofitFactory
- Productos
- Perfil

Glide

```
// Glide
implementation("com.github.bumptech.glide:glide:4.16.0")
```

Librería para cargar imágenes desde URLs.

Se usa en:

- ApiAdapter (imagen del producto)
- Perfil (foto del usuario random)

2. RetrofitFactory

```kotlin
package com.example.proyectofinal

import retrofit2.Retrofit
import retrofit2.converter.gson.GsonConverterFactory

2 Usages
object RetrofitFactory {
    val api: ApiService by lazy {
        Retrofit.Builder()
            .baseUrl( baseUrl = "https://fakestoreapi.com/")
            .addConverterFactory( factory = GsonConverterFactory.create())
            .build()
            .create(ApiService::class.java)
    }

    1 Usage
    val instance: ApiService by lazy {
        Retrofit.Builder()
            .baseUrl( baseUrl = "https://randomuser.me/")
            .addConverterFactory( factory = GsonConverterFactory.create())
            .build()
            .create(ApiService::class.java)
    }
}
```

- La clase es un Singleton (solo existe una instancia).
- Se crean dos clientes HTTP diferentes:
- api es para obtener productos.
- instance es para obtener usuarios de randomuser.me.
- GsonConverterFactory transforma JSON a objetos Kotlin.
- by lazy inicializa Retrofit solo cuando se usa, optimizando recursos.

3. ApiService

```
package com.example.proyectofinal

import retrofit2.http.GET

4 Usages
interface ApiService {
    1 Usage
    @GET( value = "products")
    suspend fun getProducts(): List<Producto>

    1 Usage
    @GET( value = "api/")
    suspend fun getRandomUser(): RandomUserResponse
}
```

- Es una interfaz que define las rutas exactas de la API.
- @GET indica que se hace una petición HTTP GET.
- suspend permite ejecutar estas funciones dentro de corrutinas (asincronía sin bloquear la UI).
- Los datos JSON se convierten automáticamente a:
    - List<Producto> para fake store
    - RandomUserResponse para randomuser.me

4. Productos Activity

```kotlin
package com.example.proyectofinal

import android.os.Bundle
import androidx.activity.enableEdgeToEdge
import androidx.appcompat.app.AppCompatActivity
import androidx.core.view.ViewCompat
import androidx.core.view.WindowInsetsCompat
import androidx.lifecycle.lifecycleScope
import androidx.recyclerview.widget.LinearLayoutManager
import com.example.proyectofinal.databinding.ActivityProductosBinding
import kotlinx.coroutines.launch

3 Usages
class Productos : AppCompatActivity() {
    4 Usages
    private lateinit var binding: ActivityProductosBinding
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        enableEdgeToEdge()
        binding = ActivityProductosBinding.inflate(layoutInflater)
        setContentView(binding.root)

        lifecycleScope.launch {
            val products = RetrofitFactory.api.getProducts()
            binding.recyclerProductos.adapter = ApiAdapter( lista = products)
        }

        binding.recyclerProductos.layoutManager = LinearLayoutManager( context = this)
    }
```

- lifecycleScope.launch ejecuta código asincrónico respetando el ciclo de vida.
- Se obtiene una lista de productos desde la API.
- Se pasa a ApiAdapter, el cual se encarga de inflar cada item visual.
- LinearLayoutManager muestra la lista verticalmente.

5. ApiAdapter

```
override fun onBindViewHolder(holder: ViewHolder, position: Int) {
    val item = lista[position]

    holder.binding.txtTitle.text = "Titulo: " + item.title
    holder.binding.txtDescription.text = "Descripción: " + item.description
    holder.binding.txtPrice.text = "$${item.price}"


    Glide.with( context = holder.itemView.context)
        .load( string = item.image)
        .into( view = holder.binding.imgProduct)
}
```

- onBindViewHolder() coloca los valores reales en cada elemento.
- Se usa Glide para descargar y mostrar imágenes desde URLs.
- El adapter toma la lista de productos y la transforma en vistas.

6. ChatActivity

Inicialización de Firebase

```
// Inicializar Firebase Realtime Database
database = FirebaseDatabase.getInstance().getReference( path = "messages")
```

Esto referencia la ruta /messages donde se guardan todos los mensajes del chat grupal.

Lectura en tiempo real

```
private fun listenForMessages() {
    database.addValueEventListener( listener = object : ValueEventListener {
        override fun onDataChange(snapshot: DataSnapshot) {
            messagesList.clear()

            for (item in snapshot.children) {
                val msg = item.getValue( valueType = Message::class.java)
                if (msg != null) messagesList.add(msg)
            }

            adapter.notifyDataSetChanged()
            binding.recyclerMessages.scrollToPosition( position = messagesList.size - 1)
        }

        override fun onCancelled(error: DatabaseError) {}
    })
}
```

Cada cambio en la base de datos provoca que:

- Se actualice la lista messagesList
- Se refresque el RecyclerView
- Se haga scroll automático al último mensaje

Enviar mensaje

```
private fun sendMessage() {
    val text = binding.editMessage.text.toString().trim()
    if (text.isEmpty()) return

    val user = FirebaseAuth.getInstance().currentUser?.email ?: "Anónimo"
    val msg = Message(user, text, timestamp = System.currentTimeMillis())

    // Enviar mensaje
    val id = database.push().key!!
    database.child( pathString = id).setValue(msg)

    binding.editMessage.setText("")
}
```

- Se obtiene el email real del usuario autenticado.
- Se crea un objeto Message (usuario, texto, timestamp).
- Se guarda en Firebase generando un ID único.

7. Perfil

```
class Perfil : AppCompatActivity() {
    11 Usages
    private lateinit var binding: ActivityProfileBinding
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        enableEdgeToEdge()
        binding = ActivityProfileBinding.inflate(layoutInflater)
        setContentView(binding.root)

        lifecycleScope.launch {
            val response = RetrofitFactory.instance.getRandomUser()
            val user = response.results[0]

            binding.txtNombre.text = "Nombre: " + "${user.name.first} ${user.name.last}"
            binding.txtCorreo.text = "Email: " + user.email
            binding.txtTelefono.text = "Telefono: " + user.phone

            binding.txtLocation.text =
                "Locación: " + "${user.location.city}, ${user.location.state}, ${user.location.country}"

            binding.txtBirthday.text =
                "Fecha: ${user.dob.date.take( n= 10)}  (Edad: ${user.dob.age})"

            binding.txtRegistered.text =
                "Registro: ${user.registered.date.take( n= 10)}  (Hace ${user.registered.age} años)"

            binding.txtUUID.text = "UUID: " + user.login.uuid
            binding.txtUsername.text = "Usuario: " + user.login.username
```

- Se hace una petición HTTP al endpoint /api/.
- Se obtiene un usuario aleatorio.
- Se muestran todos los datos: nombre, email, teléfono, ciudad, estado, país, edad, fecha de nacimiento, fecha de registro, usuario, UUID.

```
            Glide.with( activity = this@Perfil)
                .load( string = user.picture.large)
                .into( view = binding.imgPerfil)
        }
```

Glide descarga y muestra la imagen del usuario random.

8. LoginActivity

```
auth.signInWithEmailAndPassword( p0 = email,  p1 = pass).addOnSuccessListener {

    // Guardar último correo
    shared.edit().putString( p0 = "last_email",  p1 = email).apply()

    startActivity(Intent( packageContext = this,  cls = MenuActivity::class.java))
    finish()

}.addOnFailureListener {
    Toast.makeText( context = this,  text = "Error: ${it.message}",  duration = Toast.LENGTH_LONG).show()
}
```

- El usuario ingresa email/contraseña.
- Si es correcto este lo lleva al menú.
- Guarda el último correo en SharedPreferences.

9. RegisterActivity

```
auth.createUserWithEmailAndPassword( p0 = email,  p1 = pass)
    .addOnSuccessListener {
        Toast.makeText( context = this,  text = "Cuenta creada",  duration = Toast.LENGTH_SHORT).show()
        finish()
    }
    .addOnFailureListener {
        Toast.makeText( context = this,  text = "Error: ${it.message}",  duration = Toast.LENGTH_LONG).show()
    }
```

- Crea nuevos usuarios en Firebase.
- No se usa base de datos propia, solo Firebase Auth.

10. MenuActivity

```kotlin
class MenuActivity : AppCompatActivity() {

    5 Usages
    private lateinit var binding: ActivityMenuBinding

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = ActivityMenuBinding.inflate(layoutInflater)
        setContentView(binding.root)

        binding.btnChat.setOnClickListener {
            startActivity(Intent( packageContext = this,  cls = ChatActivity::class.java))
        }

        binding.btnPerfil.setOnClickListener {
            startActivity(Intent( packageContext = this,  cls = Perfil::class.java))
        }
        binding.btnProductos.setOnClickListener {
            startActivity(Intent( packageContext = this,  cls = Productos::class.java))


        }
    }
}
```

Es un menú simple que dirige a

- Chat grupal
- Perfil (randomuser.me)
- Lista de productos (FakeStoreAPI)

PROBLEMAS ENCONTRADOS Y SOLUCIONES

1.- El RecyclerView no mostraba productos provenientes de FakeStoreAPI

Problema:

La Activity de productos cargaba la API, pero la lista aparecía vacía.

Causa técnica:

El error se debía a que el adaptador no se asignaba hasta después de llamar a Retrofit, por lo que la vista se inicializaba sin datos.

Solución:

```
lifecycleScope.launch {
    val products = RetrofitFactory.api.getProducts()
    binding.recyclerProductos.adapter = ApiAdapter(products)
}
```

Se movió el adapter dentro de la corrutina, garantizando que la lista estuviera llena.

2.- Glide no cargaba las imágenes de los productos

Problema:

Las imágenes no aparecían y no se lanzaba ningún error visible.

Causa técnica:

Faltaba el permiso de internet:

```
<uses-permission android:name="android.permission.INTERNET"/>
```

Sin este permiso, Glide no puede descargar imágenes desde URLs.

Solución:

Agregar el permiso en AndroidManifest.xml.

3.- El chat no actualizaba mensajes en tiempo real

Problema:

Los mensajes aparecían solo al reiniciar la app.

Causa técnica:

El Listener de Firebase se colocó en el lugar incorrecto.

Se estaba usando:

database.get().addOnSuccessListener { ... }

Esto solo obtiene los datos una vez.

Solución:

Cambiar a un listener en tiempo real:

database.addValueEventListener(object : ValueEventListener { ... })

Esto garantiza que cada cambio se refleje inmediatamente.

4.- El RecyclerView del chat no hacía scroll al último mensaje

Problema:

Los mensajes nuevos quedaban fuera de la pantalla.

Solución:

binding.recyclerChat.scrollToPosition(messagesList.size - 1)

Se colocó dentro de:

onDataChange

de Firebase, asegurando que cada actualización se reflejara en el UI.

5.- La aplicación crasheaba al girar el teléfono en la pantalla de productos

Problema:

Al rotar el dispositivo, la lista desaparecía.

Causa técnica:

La corutina se ejecutaba mientras la Activity estaba destruyéndose.

Solución:

Usar lifecycleScope en lugar de corutinas globales:

lifecycleScope.launch { ... }

Esto cancela la petición automáticamente al destruir el Activity.

FLUJO DE AUTENTICACIÓN EN LA APLICACIÓN

1.- El usuario abre LoginActivity

El usuario ingresa:

- Correo
- Contraseña

El botón de login ejecuta:

auth.signInWithEmailAndPassword(email, pass)

Firebase verifica:

- Si el usuario existe
- Si la contraseña es correcta
- Si la cuenta está activa

2.- Validación de campos

Antes de enviar datos a Firebase, se valida:

```
if(email.isEmpty() || pass.isEmpty()){
    Toast.makeText(this, "Llene todos los campos", Toast.LENGTH_SHORT).show()
    return@setOnClickListener
}
```

Esto evita errores como:

- Autenticación con campos vacíos
- Crashes de Firebase

3.- Inicio de sesión correcto

Si todo está bien, Firebase devuelve:

```
addOnSuccessListener {
    startActivity(Intent(this, MenuActivity::class.java))
}
```

Firebase genera un token interno y mantiene la sesión activa hasta que el usuario cierre sesión.

4. Guardado del correo (SharedPreferences)

Después del login, la app almacena el correo para mostrarlo después:

```
val sharedPreferences = getSharedPreferences("datos", MODE_PRIVATE)
sharedPreferences.edit().putString("correo", email).apply()
```

Esto permite:

- Mostrar el email en la pantalla principal
- Enviar mensajes al chat con nombre del usuario

5. Registro de nuevos usuarios

En RegisterActivity, la app usa:

```
auth.createUserWithEmailAndPassword(email, pass)
```

Firebase crea una nueva cuenta y la almacena en su sistema de usuarios.

Si el registro es exitoso:

```
startActivity(Intent(this, LoginActivity::class.java))
```

6. Autenticación persistente (Firebase mantiene la sesión abierta)

Cuando la app se abre otra vez, Firebase recuerda la sesión.

Esto permite que no se tenga que iniciar sesión cada vez.

Puedes comprobar si hay alguien conectado con:

FirebaseAuth.getInstance().currentUser

Esto se usa en:

- ChatActivity → para obtener el email del usuario actual
- MenuActivity → para mostrar “Bienvenido”

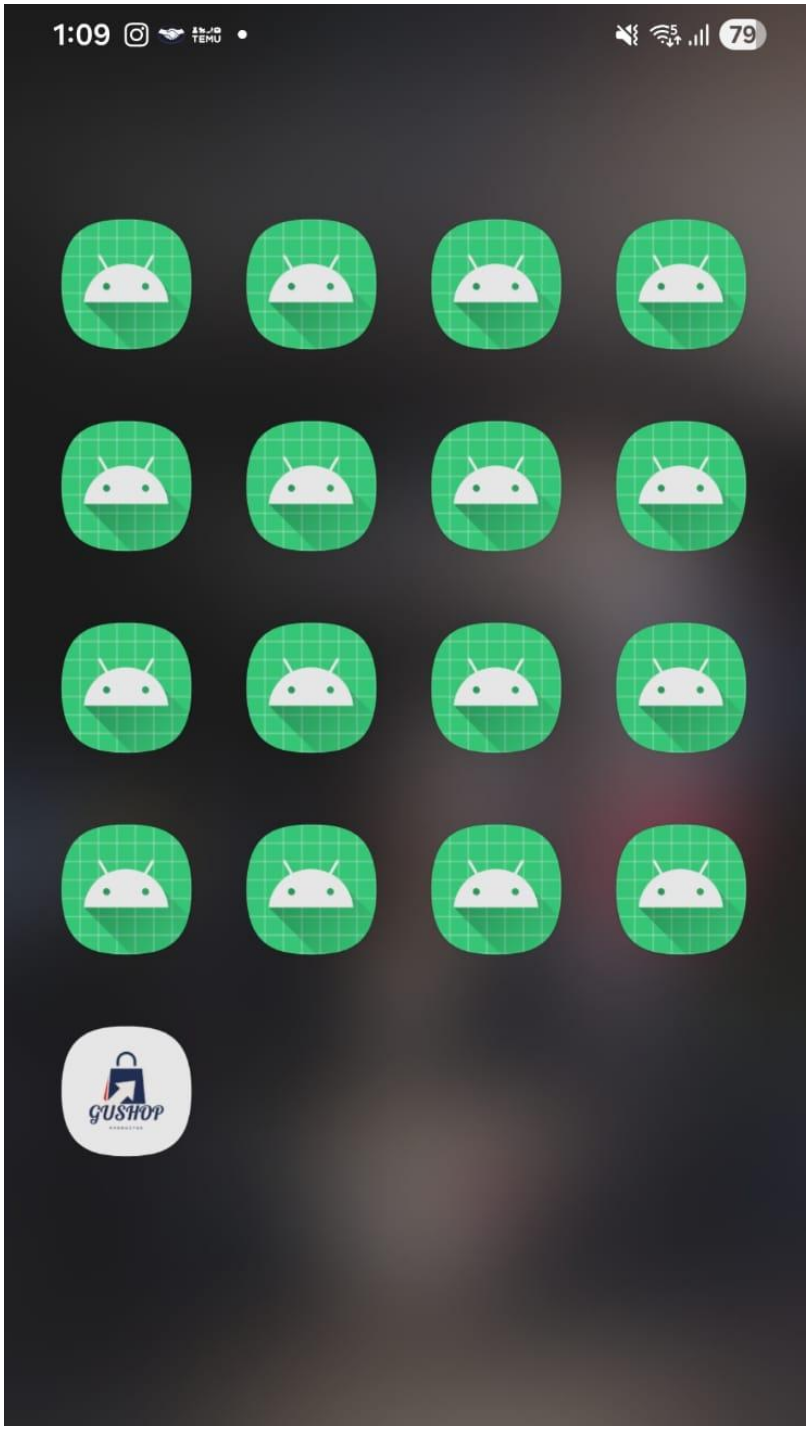
7. Cerrar sesión

Desde el menú, el usuario puede cerrar sesión:

FirebaseAuth.getInstance().signOut()

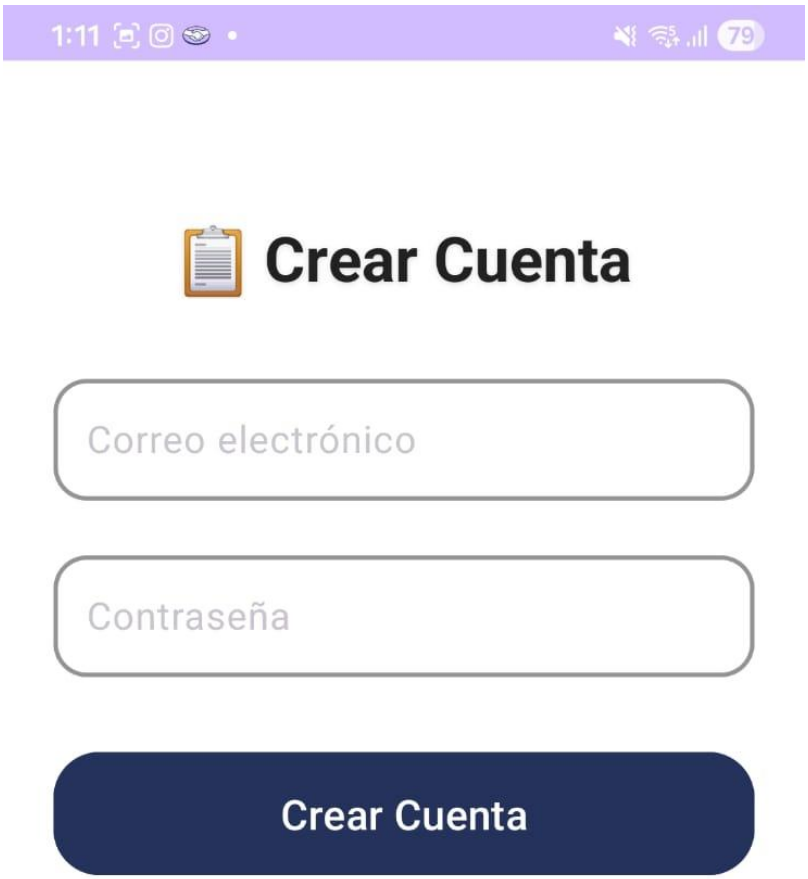
Esto invalida el token y regresa al LoginActivity.

APLICACIÓN EN FUNCIONAMIENTO



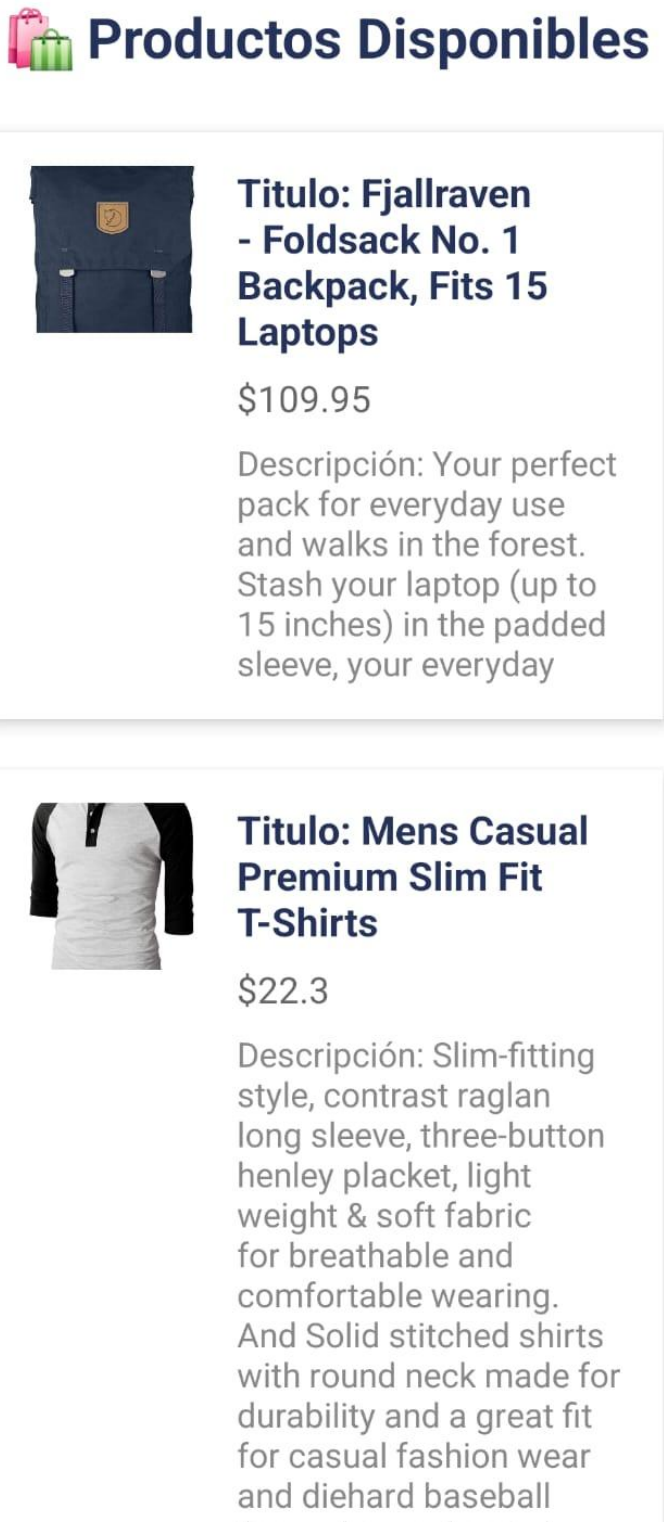
Aquí se puede observar la parte del icono o logotipo de la aplicación desde la perspectiva del usuario.

Esta es la pestaña de el inicio de sesión de la aplicación y se puede observar que ya tiene un correo guardado, es el último que se ingresó.

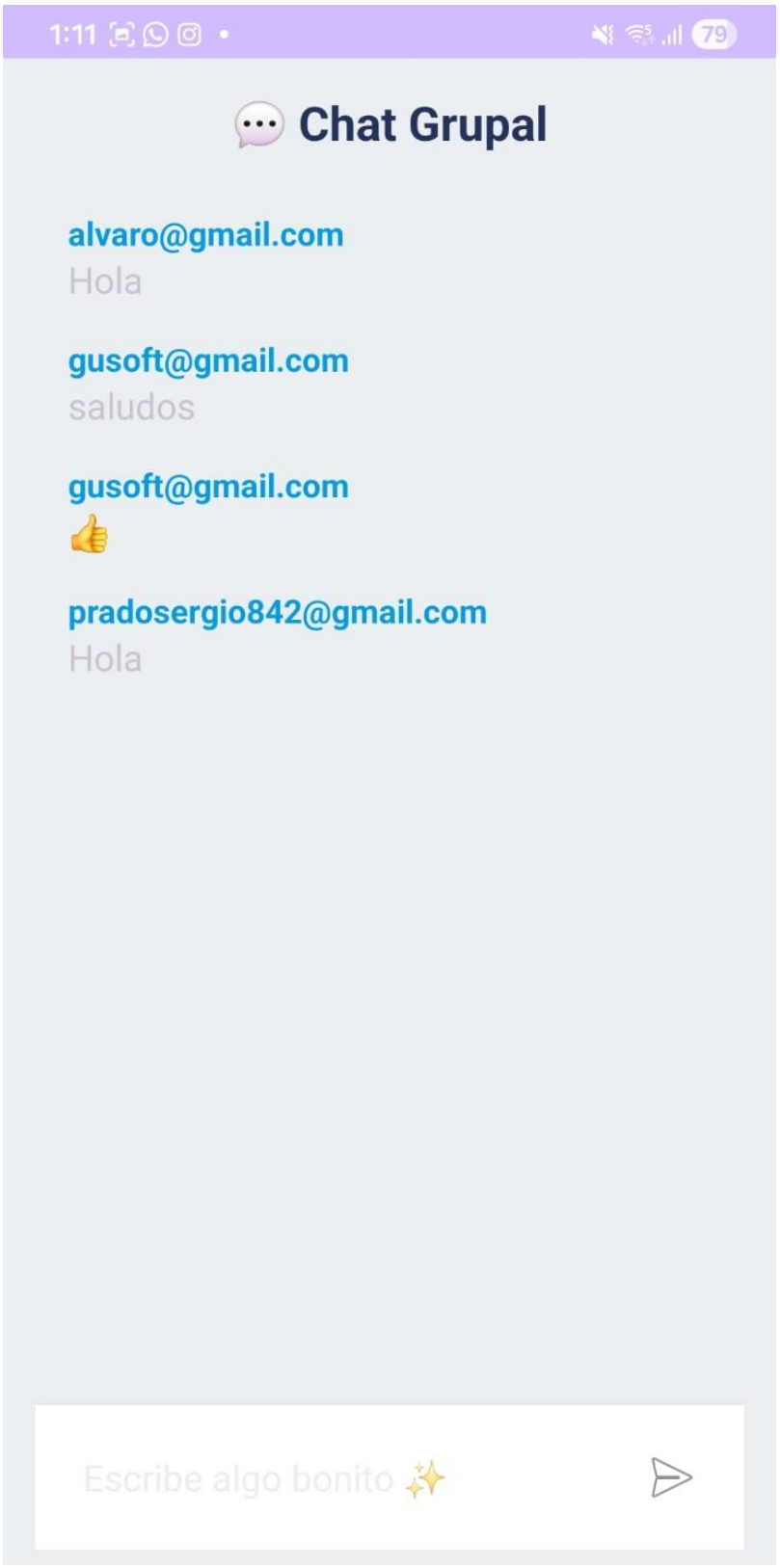


Esta es la pestaña de el registro de los usuarios, colocan su correo y contraseña y después los redirigirá a la pestaña de inicio de sesión.

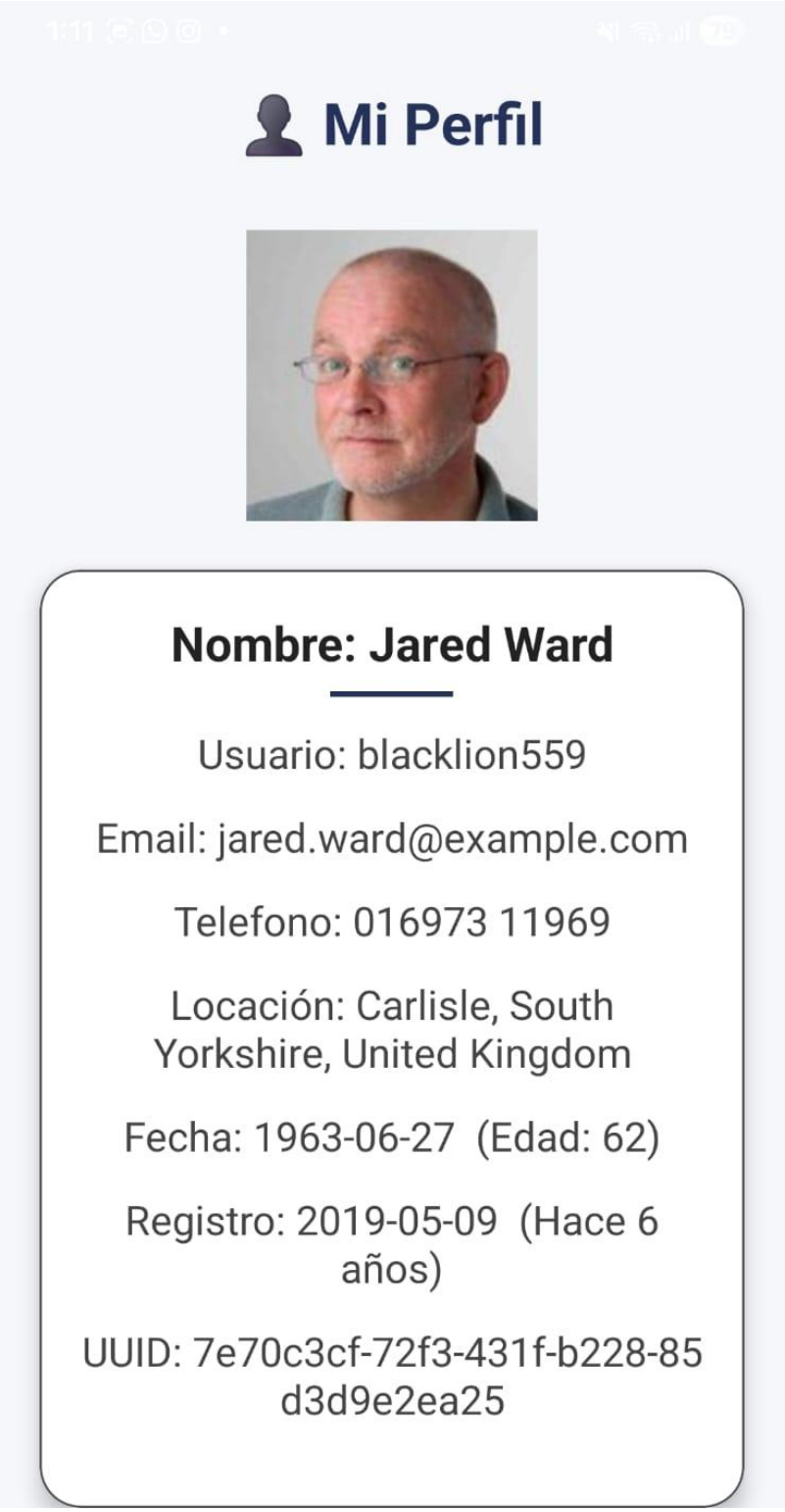
Esta es la pestaña de menú de inicio, aquí se encontrarán los botones para observar los productos, ir al chat grupal y mostrar nuestro perfil.



Esta es la pestaña donde se mostrarán todos los productos disponibles, solo se necesita darle click al botón de ver productos.



Al presionar chat grupal, se mostrará la pestaña para ver y enviar mensajes.



Al presionar ver perfil, nos redirigirá a esta pestaña, que nos mostrará los datos de nuestro perfil detalladamente.