



**TECNOLÓGICO
NACIONAL DE MÉXICO**

Proyecto Final 8vo Semestre

**Asignatura: Desarrollo de aplicaciones
Móviles**

Alumno: Andrés Iván Rodríguez Hernández

Docente: Rubén Miguel Riojas Rodríguez

Instituto Tecnológico Superior de Monclova

Turno Matutino

Semestre Febrero - Junio

Jueves 09 de Junio del 2022 Monclova, Coahuila



Proyecto final apps móviles

Componentes de código

AuthActivity.kt

Esta es la actividad que originalmente era la principal, la main, fue renombrada para hacer alusión a que es la que lleva a cabo los procesos de solicitar la autenticación, y desde luego es la primera que se ejecuta al iniciar la aplicación

Comenzando con las dependencias ocupamos las 2 de firebase, la primera es la de las analíticas de Firebase y la segunda la autenticación, después ya comienza la clase AuthActivity

```
package com.example.firebaseio_login
import android.content.Intent
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import androidx.appcompat.app.AlertDialog
import com.google.firebase.analytics.FirebaseAnalytics
import com.google.firebase.auth.FirebaseAuth
import kotlinx.android.synthetic.main.activity_main.*

class AuthActivity : AppCompatActivity() {
```

La primera función que se ocupa es la de onCreate, lo que hace es crear las instancias de firebase analytic

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)
```

```

    val analytics:FirebaseAnalytics = FirebaseAnalytics.getInstance(this)
    val bundle = Bundle()
    bundle.putString("message","Integración de Firebase completa")
    analytics.logEvent("InitScreen",bundle)

    //Setup
    setup()
}

```

La segunda función es la de setup, lo que hace es establecer el título de la activity y crea el evento que detecta cuando hacemos click en cada uno de los botones, el primero de estos verifica primero que no esté vacío campo alguno y después obtiene una instancia de firebase auth, crea usuario dados el email y contraseña y en caso de darse un error de tipeo por parte del usuario se invoca al método showAlert.

El botón que sigue es el de log in que igualmente obtiene una instancia de firebase auth y igual recibe los parámetros de email y contraseña por parte de los edit text con la diferencia de que ahora con parámetros para el método de sign in de firebase., después de eso tiene el mismo comportamiento que el otro botón

```

private fun setup(){
    title = "Autenticación"

    signUpButton.setOnClickListener {
        if (emailEditText.text.isNotEmpty() && passwordEditText.text.isNotEmpty()) {
            FirebaseAuth.getInstance().createUserWithEmailAndPassword(emailEditText.
                text.toString(),passwordEditText.text.toString()).addOnCompleteListener {
                if(it.isSuccessful) {
                    showHome(it.result?.user?.email ?: "", ProviderType.BASIC)
                } else {
                    showAlert()
                }
            }
        }
    }

    logInButton.setOnClickListener {
        if (emailEditText.text.isNotEmpty() && passwordEditText.text.isNotEmpty()) {
            FirebaseAuth.getInstance().signInWithEmailAndPassword(emailEditText.
                text.toString(),passwordEditText.text.toString()).addOnCompleteListener {
                if(it.isSuccessful) {
                    showHome(it.result?.user?.email ?: "", ProviderType.BASIC)
                } else {
                    showAlert()
                }
            }
        }
    }
}

```

La siguiente función es showAlert que despliega un prompt que alerta de un error, ya sea un error de tipo o que se halla presionado el botón equivocado, como intentar loguearse con un usuario inexistente o al contrario registrar un usuario ya registrado. Ocupa la clase AlertDialog y da el botón de aceptar, después de construir el cuadro de diálogo ya lo despliega.

Lo que hace el comando showHome es que nos llevará a la pantalla que verá el usuario después de loguearse exitosamente donde se verá el email y tipo de autenticación que ha ocupado el usuario, la función de este método es que la activity Home tenga la información que necesita para funcionar como lo es el email y el provider, almacenados en la constante homeIntent. Finalmente cerramos la clase AuthActivity

```
private fun showAlert () {
    val builder = AlertDialog.Builder(this)
    builder.setTitle("Error")
    builder.setMessage("Se ha producido un error autenticando al usuario")
    builder.setPositiveButton("Aceptar",null)
    val dialog: AlertDialog = builder.create()
    dialog.show()
}

private fun showHome(email: String, provider: ProviderType) {
    val homeIntent = Intent(this, HomeActivity::class.java).apply {
        putExtra("email",email)
        putExtra("provider",provider.name)
    }
    startActivity(homeIntent)
}

}
```

HomeActivity.kt

Comenzaremos con las dependencias específicas de esta aplicación que son las de autenticación de firebase al igual que la anterior actividad, pero en este caso ya no ocupamos la dependencia de analíticas de firebase sino que ocupa la de firestore que es el servicio de base de datos de nuestra tecnología. Después creamos una enum class cuya función es almacenar los distintos tipos de autenticación que tenemos, en este caso solo tenemos la autenticación básica. Iniciamos la clase HomeActivity

```
package com.example.firebaseio_login

import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import com.google.firebase.auth.FirebaseAuth
import com.google.firebase.firestore.FirebaseFirestore
import kotlinx.android.synthetic.main.activity_home.*

enum class ProviderType {
    BASIC
```

```
}
```

```
class HomeActivity : AppCompatActivity() {
```

Primero declaramos una constante db que almacenará la instancia de firestore y después sobreescribimos la función onCreate para que nos despliegue la interfaz gráfica que verá el usuario y que está regida por archivos xml. Además se declara la constante bundle que guardará los valores email y provider, pero en esta ocasión será para mostrarlos, no para realizar una autenticación como en el caso anterior, por eso no desplegamos el campo contraseña también, además invocamos a la función setup y le damos los parámetros email y provider, cada uno seguido de `? : ""` para decir que en caso de que no exista(el valor puede ser nulo ya que se lo especificamos con ? porque de otra manera nos daría error) devuelva un string vacío(por eso las comillas)

```
private val db = Firebase Firestore.getInstance()
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_home)

    //setup
    val bundle = intent.extras
    val email = bundle?.getString("email")
    val provider = bundle?.getString("provider")
    setup(email ?: "", provider ?: "")
}
```

La clase HomeActivity también tiene su método setup que recibe el email y provider como parámetros, método que ya declaramos anteriormente y que al momento de ejecución recibirá en sus parámetros a los valores dados en la función onCreate. Después da el título que se mostrará en la pantalla y se despliegan los valores de email y provider en los text view. Tenemos el evento click del botón de log out que crea la instancia de firebase auth para llamar a su método de sign out y llamamos luego a onBackPressed que nos regresa a la anterior actividad. Configuramos el botón save que se encarga de guardar en la base de datos la información introducida por el usuario, en este caso dirección y teléfono que almacena en una colección hashMapOf que al igual que los diccionario de c# almacena la información en binas de datos de forma que cada uno tenga un par que describa que es lo que almacena y lo que guardará serán el valor de provider y de los textos que hallan en los edit text de address y phone.

```
private fun setup(email: String, provider: String) {
    title = "Inicio"
    emailTextView.text = email
    providerTextView.text = provider

    logOutButton.setOnClickListener{
        FirebaseAuth.getInstance().signOut()
```

```

        onBackPressed()
    }
    saveButton.setOnClickListener{
        db.collection("users").document(email).set(
            hashMapOf("provider" to provider,
            "address" to addressEditText.text.toString(),
            "phone" to phoneEditText.text.toString())
        )
    }
    getButton.setOnClickListener{
        db.collection("users").document(email).get().addOnSuccessListener {
            addressEditText.setText(it.get("address") as String?)
            phoneEditText.setText(it.get("phone") as String?)
        }
    }
    deleteButton.setOnClickListener{
        db.collection("users").document(email).delete()
    }
}
}

```

El getButton lo que hace es leer la base de datos “users” de firebase para buscar en donde están almacenadas la dirección y teléfono introducidas por el usuario(identificado por el document email) para mostrarlas en los mismos edit text donde el usuario los guardó en un comienzo, función útil para cuando el usuario cierra sesión y en vez de sobrescribir los datos solo quiere visualizarlos.

Finalmente tenemos el botón delete, a estas alturas document ya tiene un email guardado por lo que lo toma y busca el usuario en la colección users para eliminarlo junto con su dirección y teléfono, cuando el usuario vuelva a iniciar sesión se topará con que estos han sido eliminados.

Archivos build.gradle y .json

build.gradle a raíz de proyecto

Hizo falta modificar el archivo original para que las dependencias estén en orden, se debe comenzar con el bloque builscript que afectará a todos los proyectos derivados de este, después debe ir el bloque plugins que influye solo en el proyecto actual y el bloque task clean limpia archivos internos y debe ubicarse al final

```

// Top-level build file where you can add configuration options common to
// all sub-projects/modules.
buildscript {
    repositories {
        // Check that you have the following line (if not, add it):
        google() // Google's Maven repository
    }
}

```

```

        }
        dependencies {
            // Add this line
            classpath 'com.google.gms:google-services:4.3.10'
        }
    }

    plugins {
        id 'com.android.application' version '7.2.1' apply false
        id 'com.android.library' version '7.2.1' apply false
        id 'org.jetbrains.kotlin.android' version '1.6.21' apply false
    }

    task clean(type: Delete) {
        delete rootProject.buildDir
    }

```

build.gradle a nivel de aplicación

El bloque plugins estaba originalmente pero tuvo que comentarse para que los botones de la interfaz gráfica pudieran funcionar y del tercer al quinto plugin son necesarios para que el activity pueda reconocer los botones. En las dependencias de la parte inferior tuvieron que agregarse los firebase bom que administran las versiones de las dependencias para que estén actualizadas y para evitar conflictos entre ellas y también se implementan las de firebase auth, analytics y firestore.

```

/*plugins {
    id 'com.android.application'
    id 'org.jetbrains.kotlin.android'
}*/ //estas de arriba están comentadas para los botones

apply plugin: 'com.android.application'
apply plugin: 'com.google.gms.google-services'
//estas tres de abajo es para el en activity reconozca los botones
apply plugin: 'com.android.application'
apply plugin: 'kotlin-android'
apply plugin: 'kotlin-android-extensions'

android {
    compileSdk 32

    defaultConfig {
        applicationId "com.example.firebaseio"
        minSdk 23
        targetSdk 32
        versionCode 1
        versionName "1.0"

        testInstrumentationRunner "androidx.test.runner.AndroidJUnitRunner"
    }

    buildTypes {
        release {
            minifyEnabled false

```

```

        proguardFiles getDefaultProguardFile('proguard-android-optimize.txt'), 'proguard-rules.pro'
    }
}

compileOptions {
    sourceCompatibility JavaVersion.VERSION_1_8
    targetCompatibility JavaVersion.VERSION_1_8
}
kotlinOptions {
    jvmTarget = '1.8'
}
}

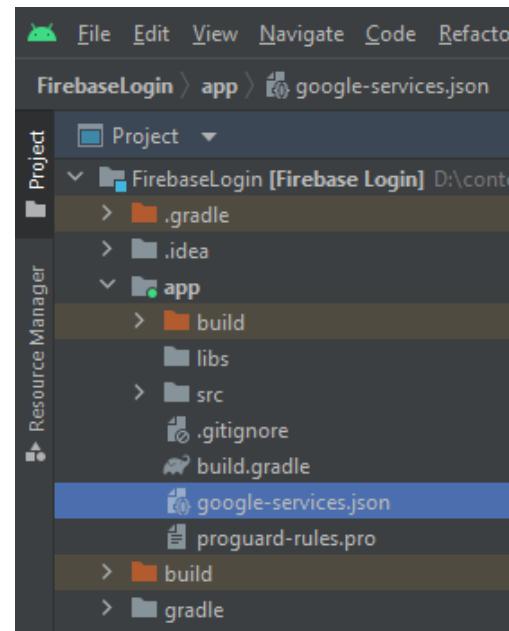
dependencies {

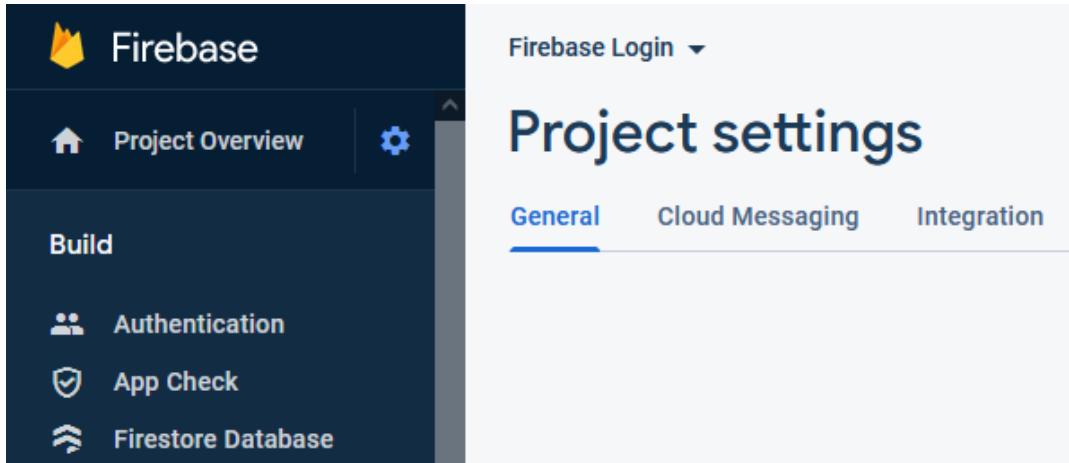
implementation 'androidx.core:core-ktx:1.7.0'
implementation 'androidx.appcompat:appcompat:1.4.1'
implementation 'com.google.android.material:material:1.6.0'
implementation 'androidx.constraintlayout:constraintlayout:2.1.4'
implementation platform('com.google.firebaseio:firebase-bom:30.1.0')
implementation 'com.google.firebase:firebase-analytics-ktx'
implementation platform('com.google.firebaseio:firebase-bom:30.0.0')
implementation 'com.google.firebase:firebase-analytics-ktx'
implementation platform('com.google.firebaseio:firebase-bom:29.3.1')
implementation platform('com.google.firebaseio:firebase-bom:30.0.0')
implementation 'com.google.firebase:firebase-firebase-ktx'
implementation 'com.google.firebase:firebase-auth-ktx'

testImplementation 'junit:junit:4.13.2'
androidTestImplementation 'androidx.test.ext:junit:1.1.3'
androidTestImplementation 'androidx.test.espresso:espresso-core:3.4.0'
}

```

Además se debe entrar en console.firebaseio.google.com e hacer click en el engrane de la esquina superior izquierda y seleccionar project settings y al bajar en la pestaña general debemos bajar y hacer click en download google-services.json para que nos descargue un archivo que contendrá los identificadores únicos de nuestra aplicación e instancias de las tecnologías que usamos como las apis. Una vez descargado el archivo debe acomodarse en la ubicación de la imagen de la derecha





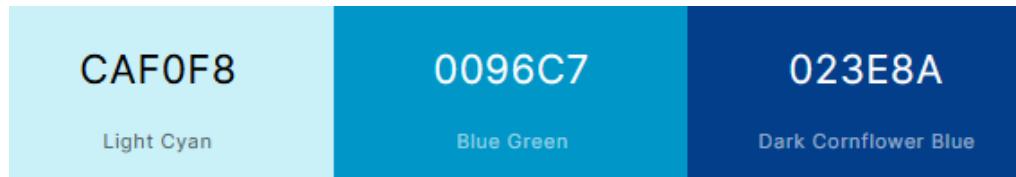
A screenshot of the 'Your apps' section within the Firebase Project Settings. It shows a list of Android apps. The first item is 'Firebase Tutorial Android' with the package name 'com.example.firebaseio'. The right-hand panel displays configuration details for this app, including the App ID (1:595083796823:android:8783b2b074ee46bba09e12), App nickname ('Firebase Tutorial Android'), Package name ('com.example.firebaseio'), SHA certificate fingerprints, and a 'Remove this app' button.

Archivos xml

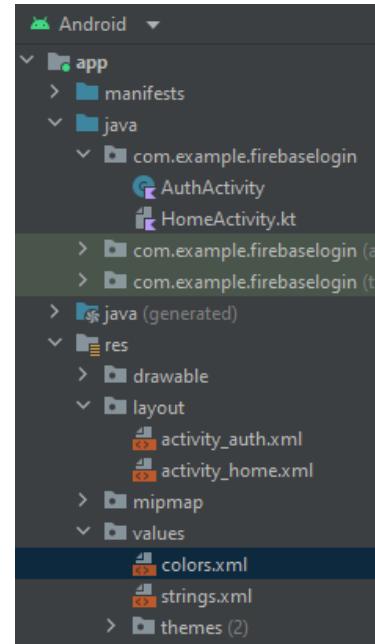
colors.xml

En este archivo xml personalicé algunos colores para asignarlos a los distintos elementos de acuerdo a la siguiente paleta de colores de forma que determinados botones y el fondo adopten

un colores relacionados entre sí



```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="purple_200">#FFBB86FC</color>
    <color name="purple_500">#FF6200EE</color>
    <color name="purple_700">#FF3700B3</color>
    <color name="teal_200">#FF03DAC5</color>
    <color name="teal_700">#FF018786</color>
    <color name="black">#FF000000</color>
    <color name="white">#FFFFFF</color>
    <color name="botónLogIn">#023e8a</color>
    <color name="botónLogOut">#0096c7</color>
    <color name="botónSignUp">#0096c7</color>
    <color name="fondo">#caf0f8</color>
</resources>
```



activity_auth.xml

Comenzaremos con el archivo que describe la estructura de la actividad auth, comenzando por la descripción de variables como altura, ancho y color de fondo personalizado, después el linear layout que es una estructura vertical compuesta de bloques acomodados horizontales uno encima de otro

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:
    android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@color/fondo"
    tools:context=".AuthActivity">

    <LinearLayout
        android:layout_width="0dp"
```

```
        android:layout_height="wrap_content"
        android:layout_marginStart="16dp"
        android:layout_marginTop="16dp"
        android:layout_marginEnd="16dp"
        android:orientation="vertical"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent">
```

Después tenemos los edit text donde decidimos el color del texto y del hint(texto que se muestra cuando el edit text está vacío), así como los valores de estos, además se tienen los espacios entre componentes

```
<EditText
    android:id="@+id/emailEditText"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:ems="10"
    android:hint="E-mail"
    android:inputType="textEmailAddress"
    android:textColor="#3A3131"
    android:textColorHint="#3A3131" />

<Space
    android:layout_width="match_parent"
    android:layout_height="8dp" />

<EditText
    android:id="@+id/passwordEditText"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:ems="10"
    android:hint="Contraseña"
    android:inputType="textPassword"
    android:textColor="#3A3131"
    android:textColorHint="#3A3131" />

<Space
    android:layout_width="match_parent"
    android:layout_height="8dp" />
```

Ahora tenemos un linear layout horizontal que consiste en una barra horizontal compuesta de bloques verticales uno al lado del otro y adentro de esta barra se acomodan los botones, espacios, edit texts, etc. Finalmente se establecen los constraints para evitar que cuando se compile la app los elementos se desacomoden.

```
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="horizontal">

    <Button
```

```

        android:id="@+id/signUpButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:backgroundTint="@color/botónSignUp"
        android:text="Registrar" />

    <Space
        android:layout_width="8dp"
        android:layout_height="wrap_content"
        android:layout_weight="1" />

    <Button
        android:id="@+id/logInButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:backgroundTint="@color/botónLogIn"
        android:text="Acceder" />
    </LinearLayout>
</LinearLayout>
</androidx.constraintlayout.widget.ConstraintLayout>
```

activity_home.xml

Este archivo es muy similar al anterior por lo que mostraremos sus diferencias con el primero comenzando con el hecho de que cambia el contexto y el hecho de que aquí se usan los TextView que tienen su texto pero no se puede modificar manualmente como lo son los edit text, estos últimos son como los text box y los text view son como las labels.

```

<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android=
"http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@color/fondo"
    tools:context=".HomeActivity">

    <LinearLayout
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_marginStart="16dp"
        android:layout_marginTop="16dp"
        android:layout_marginEnd="16dp"
        android:orientation="vertical"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent">

        <TextView
```

```

        android:id="@+id/emailTextView"
        android:layout_width="match_parent"
        android:layout_height="40dp"
        android:text="Email"
        android:textColor="#3A3131"
        android:textColorHint="#3A3131" />

<Space
        android:layout_width="match_parent"
        android:layout_height="8dp" />

<TextView
        android:id="@+id/providerTextView"
        android:layout_width="match_parent"
        android:layout_height="40dp"
        android:text="Proveedor"
        android:textColor="#3A3131"
        android:textColorHint="#3A3131" />

<Space
        android:layout_width="match_parent"
        android:layout_height="8dp" />

```

El resto del archivo está compuesto por los mismos bloques que el anterior

```

<EditText
        android:id="@+id/addressEditText"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:ems="10"
        android:hint="Calle, número, piso"
        android:inputType="textPersonName"
        android:minHeight="48dp" />

<Space
        android:layout_width="match_parent"
        android:layout_height="8dp" />

<EditText
        android:id="@+id/phoneEditText"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:ems="10"
        android:hint="Teléfono"
        android:inputType="textPersonName"
        android:minHeight="48dp" />

<Space
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />

<LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="horizontal">

        <Button

```

```
        android:id="@+id/saveButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="Guardar" />
    </LinearLayout>

    <Space
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="horizontal">

        <Button
            android:id="@+id/getButton"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="Recuperar" />

    </LinearLayout>

    <Space
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="1" />

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_weight="1"
        android:orientation="horizontal">

        <Button
            android:id="@+id/deleteButton"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="Eliminar" />

    </LinearLayout>

    <Space
        android:layout_width="match_parent"
        android:layout_height="8dp" />

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="horizontal">

        <Button
            android:id="@+id/logOutButton"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:backgroundTint="@color/botónLogOut"
```

```

        android:text="Cerrar sesión" />

    </LinearLayout>

</LinearLayout>

</androidx.constraintlayout.widget.ConstraintLayout>

```

Empleo de la tecnología Firebase

Entrando en console.firebaseio.google.com en nuestra cuenta de Firebase podremos entrar en los paneles de las distintas tecnologías de las que disponemos, en este caso podemos ver con Authentication los usuarios que se han registrado en nuestra aplicación

The screenshot shows the Firebase Authentication console interface. On the left, there's a sidebar with various services: Build, Authentication (which is selected), App Check, Firestore Database, Realtime Database, Extensions, Storage, Hosting, Functions, and Machine Learning. Below the sidebar, it says "Spark No cost \$0/month" and "Upgrade". The main area is titled "Authentication" and has tabs for "Users", "Sign-in method", "Templates", and "Usage". Under the "Users" tab, there's a search bar and a button to "Add user". A table lists three users with their email addresses, provider (Email), creation date (Created), sign-in date (Signed in), and User UID. The table includes columns for Identifier, Providers, Created, Signed in, and User UID. At the bottom of the table, there are buttons for "Rows per page" (set to 50) and "1 – 3 of 3".

Identifier	Providers	Created	Signed in	User UID
redberry35@tutanota.com	Email	9 Jun 2022	9 Jun 2022	eFqvVx1wfRPkbBDHgdXpSPfRxo...
sudafrica66@hotmail.com	Email	1 Jun 2022	9 Jun 2022	EAzxtlUYYeYB8MgdMMrXULXhys...
andresivanrh@gmail.com	Email	29 May 2022	9 Jun 2022	mwsT43HJ0XhdnX9Gv8xsNJMCZ...

Así como los métodos de inicio de sesión que tenemos, en este caso el de correo electrónico

The screenshot shows the Firebase Authentication console. On the left, the navigation bar includes 'Project Overview', 'Build' (with 'Authentication' selected), and other services like App Check, Firestore Database, Realtime Database, Extensions, Storage, Hosting, Functions, and Machine Learning. Under 'Build', there's a 'Spark' section with a 'No cost \$0/month' message and an 'Upgrade' button. The main content area is titled 'Authentication' and shows the 'Sign-in method' tab selected. It displays a table for 'Sign-in providers' with one row for 'Email/Password' which is 'Enabled'. Below this is a section for 'Authorised domains' with a table showing one row for 'frontends' which is 'Default'.

Además se cuentas con analíticas sobre los usuarios que están en la aplicación, el tiempo que permanecen en ella, las versiones que usan, etc.

The screenshot shows the Firebase Analytics dashboard. The left sidebar lists 'Analytics' (Dashboard, Realtime, Events, Conversions, Audiences, Custom Definitions, Latest Release, DebugView), 'Engage' (A/B Testing, Cloud Messaging, I...), and 'Spark' (No cost \$0/month). The main dashboard features several cards: 'User activity over time' (a line chart from May 15 to June 5 showing spikes in user activity), 'Users in last 30 minutes' (0 users per minute), 'Users by App version' (a line chart showing a peak around June 29), 'Latest app release overview' (APP: Firebase Tutorial Android, Version: 1.0, Status: Successful), 'App stability overview' (APP: Firebase...Android, Crash-Free Users: 100.0%), and 'Average engagement time' (10m 06s), 'Engaged sessions per user' (2.5), and 'Average engagement' (2m 31s).

En Firestore Database tenemos la base de datos que contiene la información que han ingresado los usuarios desde la aplicación

The screenshot shows the Firebase Cloud Firestore interface. On the left, the navigation sidebar includes options like Authentication, App Check, Firestore Database, Realtime Database, Extensions, Storage, Hosting, Functions, and Machine Learning. The main area is titled "Cloud Firestore" and has tabs for Data, Rules, Indexes, and Usage. Under the Data tab, a path "users > andresivanrh@gmail.com" is selected. The right pane displays a table with one row for "andresivanrh@gmail.com". The table has columns for "fir-login-b8250", "users", and the user's email. Below the table, there are buttons for "Start collection", "Add document", "Start collection", and "Add field". The "Add field" section shows fields: address: "Pera 33", phone: "555 555 55 55", and provider: "BASIC".

Puesto que este es un programa de desarrollo y no de producción las operaciones de lectura y escritura están permitidas.

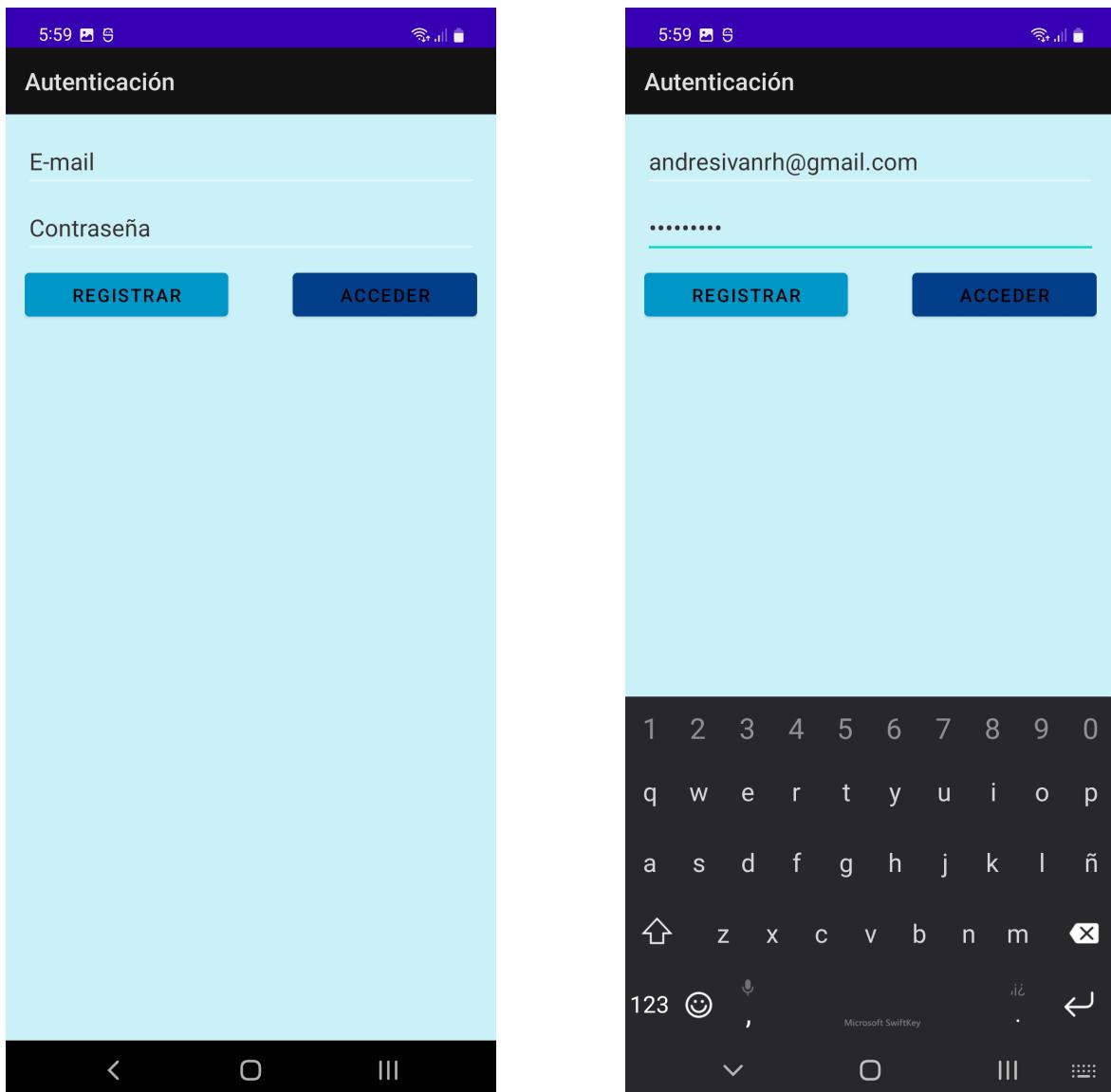
The screenshot shows the Firebase Cloud Firestore Rules tab. The left sidebar remains the same. The main area is titled "Cloud Firestore" and has tabs for Data, Rules, Indexes, and Usage. The "Rules" tab is selected, showing the title "Edit rules" and a "Monitor rules" button. Below this is a "Develop and Test" button. The central area contains a code editor with a preview of a dog icon. The code is as follows:

```
rules_version = '2';
service cloud.firestore {
  match /databases/{database}/documents {
    match /{document=**} {
      allow read, write: if
        request.time < timestamp.date(2022, 7, 8);
    }
  }
}
```

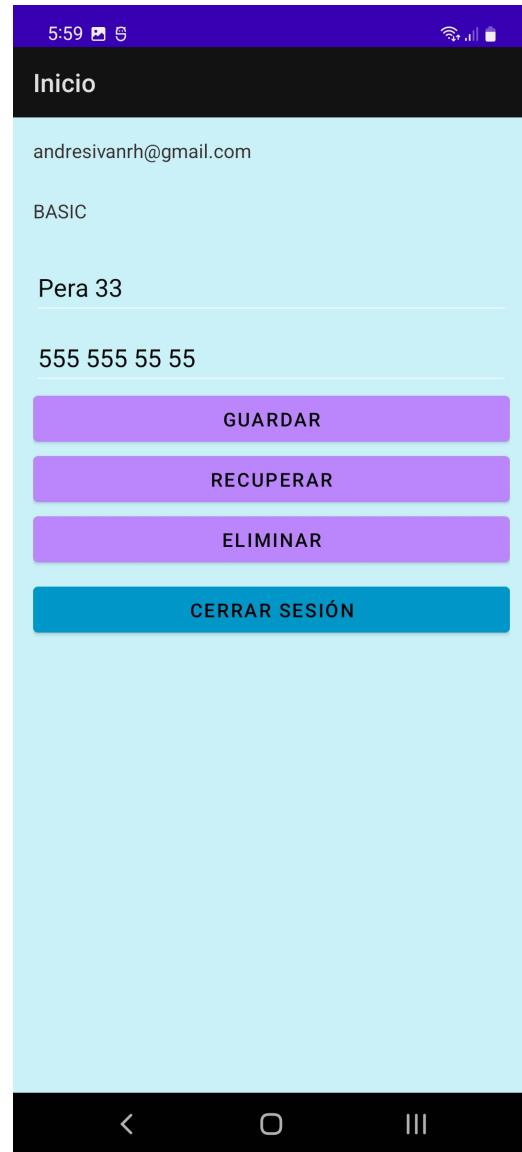
Below the code editor, there is a "Rules Playground" section with the sub-instruction "Experiment and explore with Security Rules".

Capturas de pantalla

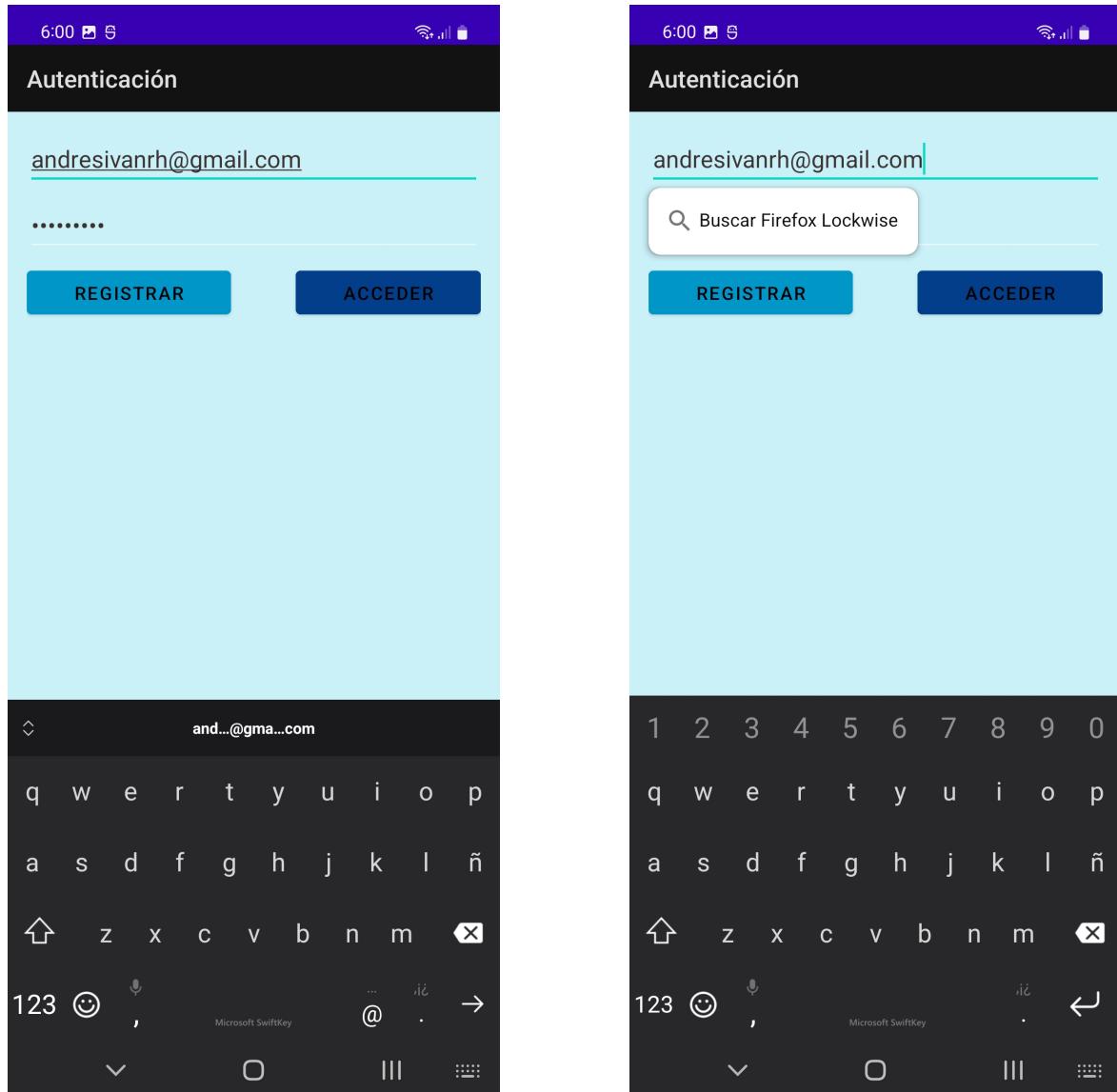
Al entrar en la aplicación se ve la pantalla de autenticación, aquí el usuario va a escribir su email y contraseña, si es un nuevo usuario va a presionar registrar, de lo contrario va a acceder, en caso de que no se pulse el botón correcto se desplegará un prompt que avise de que hubo un error y le dará al usuario la oportunidad de corregir su información.



Una vez el usuario ha accedido puede visualizar su email y tipo de inicio de sesión(básico) así como dos rectángulos para introducir su dirección y su teléfono, al presionar guardar se almacenarán los datos en la BD de Firestore, cuando el usuario presiona recuperar en los cuadros de texto aparecerá la información que el usuario ha guardado y el botón eliminar borra la información almacenada, en caso de que después se presione recuperar se mostrará el texto en blanco puesto que no hay información.



Al presionar cerrar sesión se regresará a la pantalla de autenticación y podrá iniciar sesión nuevamente el mismo usuario o bien otro. Incluso las cajas de texto son detectadas por el gestor de contraseñas Firefox Lockwise para en caso de que el usuario quiera guardar sus credenciales.



Vista de la aplicación cuando el usuario está escribiendo en la actividad o pantalla inicio. Cuando se comete un error al escribir o presionar botones figurará un prompt que dice “Error Se ha producido un error autenticando al usuario”

