Síntesis de lo que hace el código

Para la **Actividad 1** se nos pidió un programa encargado de la creación de usuario. Viendo que las siguientes actividades describen una aplicación hecha para comunicarse, entendemos que es un registro de usuario. Y esta actividad en particular es la creación de una interfaz para el registro de ese usuario.

Para esto hemos hecho 2 actividades: **MainActivity** y **RegisterActivity**.

1. **MainActivity**: Esta es la pantalla inicial de la aplicación. Su interfaz muestra un mensaje de bienvenida y un botón que redirige al usuario a la pantalla de registro. La navegación entre actividades se realiza mediante el uso de **Intents**, una herramienta de Android que permite iniciar nuevas pantallas o componentes dentro de la aplicación.

2. **RegisterActivity**: En esta pantalla, el usuario puede registrarse proporcionando su nombre completo, correo electrónico y contraseña. Los datos ingresados se validan para asegurar que cumplan con los requisitos mínimos, como que la contraseña tenga al menos 6 caracteres. Una vez validados, los datos se envían a **Firebase**, que es una plataforma de desarrollo de aplicaciones que ofrece servicios en la nube.
3. **Firebase Authentication**: Se utiliza para autenticar al usuario mediante correo electrónico y contraseña. Este servicio se encarga de crear una cuenta segura para el usuario y gestionar su acceso a la aplicación (Cosa que usaremos después en la “Actividad 2”.
4. **Firebase Realtime Database**: Además de la autenticación, se almacena información adicional del usuario, como su nombre completo y la fecha de registro.

Pudimos elegir otros servicios de almacenamiento y gestión de datos pero elegimos Firebase como backend por su facilidad de integración con Android y su conjunto de herramientas completas para el desarrollo de aplicaciones móviles.

Durante el proceso de registro, la aplicación muestra una barra de progreso para indicar al usuario que se está realizando una operación en segundo plano. Además, se implementan mensajes de retroalimentación (mediante **Toast**) para informar al usuario sobre el éxito o fallo del registro, así como para indicar errores específicos, como un correo electrónico mal formateado o una contraseña demasiado corta.

A continuación, explicaremos paso por paso el código.

Explicación del código: Archivos Kotlin

MainActivity.kt

En este archivo no hay más que una pantalla que da un mensaje de bienvenida y un botón que activa a la Activity que sí contiene el registro. Así que es básicamente, una pantalla de bienvenida y ya.

```
import android.content.Intent
import android.os.Bundle
import androidx.appcompat.app.AppCompatActivity
import com.google.android.material.button.MaterialButton
```

Comenzamos importando las librerías básicas de configuración para una pantalla sencilla.

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

Y definiendo esta actividad con los parámetros de estilos de la aplicación principal. En este caso, el *layout* de setContentView(R.layout.*activity_main*)

```
findViewById<MaterialButton>(R.id.btnIrARegistro).setOnClickListener {
    // Inicia RegisterActivity
    startActivity(Intent(this, RegisterActivity::class.java))
}
      )
          setContentView(R.layout.activity_main)
```

1. Configuramos un *listener* para el botón "**Ir a Registro**". Cuando el usuario hace clic en este botón, iniciamos *RegisterActivity* usando un *Intent*.

RegisterActivity.kt

1.- Declaramos variables iniciales y dependencias.

```
private lateinit var etNombreCompleto: TextInputEditText
private lateinit var etCorreo: TextInputEditText
private lateinit var etContrasena: TextInputEditText
private lateinit var btnRegistrar: MaterialButton
private lateinit var progressBar: View

private lateinit var auth: FirebaseAuth
private lateinit var database: FirebaseDatabase
```

Declaramos variables para los elementos de la interfaz de usuario (como campos de texto y botones) y para las instancias de Firebase que usaremos en la actividad. Utilizamos lateinit para indicar que estas variables se inicializarán más adelante. En este caso, no utilizaremos valores val, sólo variables.

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

Sobrescribimos el método **onCreate**, que es el primer método que se ejecuta cuando se crea la actividad. Aquí, establecemos el diseño de la interfaz de usuario usando el archivo *XML activity_register.xml*. Por ahora seguiremos explicando la parte lógica, y con la parte de diseño nos enfocaremos más adelante para mostrar el contenido de dicho archivo de diseño.

```
etNombreCompleto = findViewById(R.id.etNombreCompleto)
etCorreo = findViewById(R.id.etCorreo)
etContrasena = findViewById(R.id.etContrasena)
btnRegistrar = findViewById(R.id.btnRegistrar)
progressBar = findViewById(R.id.progressBar)
```

Vinculamos las variables declaradas anteriormente con los elementos de la interfaz de usuario definidos en el archivo XML. Esto nos permite interactuar con estos elementos desde el código. Señalando cada uno de los elementos a un nombre de variable como:
Elemento = findViewById(R.idCorrespondiente) donde "*FindViewById*" es la función para seleccionar un elemento a través de su "Id". Y "R", es el acrónimo del archivo de recursos predeterminado del proyecto.

```
auth = FirebaseAuth.getInstance()
database = FirebaseDatabase.getInstance()
```

Inicializamos las instancias de Firebase Auth y Firebase Realtime Database. Estas instancias nos permiten interactuar con los servicios de autenticación y base de datos de Firebase.

```
btnRegistrar.setOnClickListener {
        registrarUsuario()
    }
}
```

Configuramos un listener para el botón "Registrar". Cuando el usuario hace clic en este botón, se ejecuta la función **registrarUsuario().** Un “listener” es un objeto *evento* que se actualiza constantemente para comprobar el estado de otro objeto.

```
private fun registrarUsuario() {
    val nombre = etNombreCompleto.text?.toString()?.trim() ?: ""
    val correo = etCorreo.text?.toString()?.trim() ?: ""
    val contrasena = etContrasena.text?.toString()?.trim() ?: ""
```

Obtenemos el valor (En este caso de tipo “String” porque es una cadena de texto), ingresado por el usuario en cada uno de los campos de texto (nombre, correo y contraseña).
Usamos trim() para eliminar espacios en blanco al inicio y al final.

Aquí comienzan las validaciones de cada uno de los valores ingresados por el usuario en los campos.

```
if (nombre.isEmpty() || correo.isEmpty() || contrasena.isEmpty()) {
    mostrarMensaje("Todos los campos son obligatorios")
    return
}
```

Validamos que todos los campos estén completos. Si algún campo está vacío, mostramos un mensaje de error y detenemos la ejecución de la función.

```
if (contrasena.length < 6) {
    mostrarMensaje("La contraseña debe tener al menos 6 caracteres")
    return
}
```

Validamos que la contraseña tenga al menos 6 caracteres. Si no cumple con este requisito, mostramos un mensaje de error y detenemos la función.

```
mostrarProgreso(true)
```

Mandamos a llamar a la función de **mostrarProgreso( )** con el argumento de “true”. Esta función está al final del programa por motivos de semántica pero el contenido se lee en este momento de la ejecución:

```
private fun mostrarProgreso(mostrar: Boolean) {
    progressBar.visibility = if (mostrar) View.VISIBLE else View.GONE
    btnRegistrar.isEnabled = !mostrar
}
```

Mostramos u ocultamos la barra de progreso y deshabilitamos el botón "Registrar" mientras se realiza una operación en segundo plano.

```
    private fun mostrarMensaje(mensaje: String) {
        Toast.makeText(this, mensaje, Toast.LENGTH_SHORT).show()
    }
}
```

Mostramos un mensaje temporal (`Toast`) en la pantalla para informar al usuario sobre el resultado de una operación.

```
auth.createUserWithEmailAndPassword(correo, contrasena)
    .addOnCompleteListener { task ->
        if (task.isSuccessful) {
            val userId = auth.currentUser?.uid
            if (userId != null) {
                guardarDatosUsuario(userId, nombre)
            } else {
                mostrarProgreso(false)
                mostrarMensaje("Error al obtener el ID de usuario")
            }
        } else {
            mostrarProgreso(false)
            mostrarMensaje(obtenerMensajeError(task.exception))
        }
    }
```

1. Mostramos una barra de progreso para indicar que se está realizando una operación en segundo plano.
2. Intentamos crear un usuario en Firebase Auth usando el correo y la contraseña proporcionados.
3. Si el registro es exitoso, obtenemos el ID del usuario y guardamos sus datos adicionales en Firebase Realtime Database.
4. Si hay algún error, lo manejamos y mostramos un mensaje adecuado al usuario.

```
private fun guardarDatosUsuario(userId: String, nombre: String) {
    val userData = hashMapOf(
        "nombre" to nombre,
        "fechaRegistro" to System.currentTimeMillis()
    )
```

Creamos un mapa (*HashMap*) con los datos del usuario que queremos guardar en la base de datos, incluyendo su nombre y la fecha de registro. Que aunque no se las pedimos en el registro, se calculan y generan como registro con la función *Systen.currentTimeMillis()*

```
database.reference.child("usuarios").child(userId).setValue(userData)
    .addOnSuccessListener {
        mostrarProgreso(false)
        mostrarMensaje("Usuario registrado exitosamente")
        finish()
    }
```

1. Guardamos los datos del usuario en **Firebase Realtime Database** bajo la ruta usuarios/userId.
2. Si la operación es exitosa, mostramos un mensaje de éxito y cerramos la actividad.

```
.addOnFailureListener { e ->
    mostrarProgreso(false)
    mostrarMensaje("Error al guardar datos: ${e.message}")
}

  }
```

Y al ocurrir un error, identificado como "e", omitimos que se muestre la barra de progreso, en su lugar, mostramos un mensaje con el error utilizando la función "**mostrarMensaje()**". Que definiremos más adelante como:

```
private fun mostrarMensaje(mensaje: String) {
    Toast.makeText(this, mensaje, Toast.LENGTH_SHORT).show()
}

  }
```

Lo que hacemos aquí es mostrar un mensaje temporal (*Toast*) en la pantalla para informar al usuario sobre el resultado de una operación.

Continuando con el flujo principal de trabajo para el registro.

```
private fun obtenerMensajeError(exception: Exception?): String {
    return when (exception?.message) {
        "The email address is badly formatted." -> "El formato del correo electrónico no es válido"
        "The email address is already in use by another account." -> "Este correo ya está registrado"
        "A network error (such as timeout, interrupted connection or unreachable host) has occurred." ->
            "Error de conexión. Verifica tu internet"
        else -> exception?.message ?: "Error desconocido"
    }
}

  }
```

Definimos una función para obtener los mensajes de error, que retorna, justamente en una cadena de texto el mensaje de error.

Obtiene el mensaje de error de ese mensaje, y prácticamente lo traduce. Esta traducción la escribimos manualmente nosotros.

Y es así como termina la parte funcional de ese registro.

Explicación del código: Archivos XML

activity_main.xml

```
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="16dp">
```

Utilizamos un ConstraintLayout como contenedor principal de la actividad. Este tipo de layout nos permite posicionar los elementos de la interfaz de manera flexible y eficiente.

Definimos el ancho y alto del layout como match_parent, lo que significa que ocupará todo el espacio disponible en la pantalla.

Añadimos un padding de 16dp para evitar que los elementos de la interfaz estén pegados a los bordes de la pantalla.

```
<TextView
    android:id="@+id/tvBienvenida"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Bienvenido a la App"
```

Creamos un TextView para mostrar un mensaje de bienvenida en la pantalla.

Le asignamos un ID (tvBienvenida) para poder referenciarlo desde el código Kotlin si es necesario.

Configuramos el ancho y alto como wrap_content, lo que significa que el tamaño del texto determinará el tamaño del TextView.

```
android:textSize="24sp"
    android:textStyle="bold"
    app:layout_constraintBottom_toTopOf="@+id/btnIrARegistro"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintVertical_chainStyle="packed" />
```

Establecemos el texto como "**Bienvenido a la App**", con un tamaño de fuente de 24sp y en negrita (bold).

Usamos restricciones (***constraints***) para posicionar el TextView:

1. Lo centramos horizontalmente (layout_constraintStart_toStartOf y layout_constraintEnd_toEndOf).
2. Lo colocamos en la parte superior de la pantalla (layout_constraintTop_toTopOf).
3. Lo vinculamos al botón "**Ir a Registro**" (layout_constraintBottom_toTopOf), asegurando que el texto esté justo encima del botón.
4. Usamos layout_constraintVertical_chainStyle="packed" para agrupar el TextView y el botón en una cadena vertical, lo que ayuda a mantenerlos juntos y bien alineados.

```
<com.google.android.material.button.MaterialButton
    android:id="@+id/btnIrARegistro"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginTop="24dp"
```

Creamos un botón utilizando **MaterialButton**, que es un componente de la biblioteca de **Material Design** para Android. Este botón tiene un estilo moderno y personalizable.

Le asignamos un ID (**btnIrARegistro**) para poder referenciarlo desde el código Kotlin.

Configuramos el ancho como match_parent, lo que significa que el botón ocupará todo el ancho disponible en la pantalla. El alto se ajusta al contenido (wrap_content).

Añadimos un margen superior de 24dp para separar el botón del TextView anterior.

```
        android:padding="12dp"
        android:text="Ir a Registro"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/tvBienvenida" />
```

Establecemos un *padding* interno de 12dp para que el texto del botón no esté pegado a los bordes.

Definimos el texto del botón como "Ir a Registro".

Usamos restricciones (constraints) para posicionar el botón:

- Lo colocamos debajo del **TextView** (layout_constraintTop_toBottomOf).
- Lo anclamos a la parte inferior de la pantalla (layout_constraintBottom_toBottomOf), lo que asegura que el botón esté siempre en la parte inferior, independientemente del tamaño de la pantalla.

```
</androidx.constraintlayout.widget.ConstraintLayout>
```

Cerramos el *ConstraintLayout*, indicando que aquí termina la definición de la interfaz de la pantalla principal.

activity_register.xml

```
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="16dp">
```

- Utilizamos un ConstraintLayout como contenedor principal de la actividad. Este tipo de layout nos permite posicionar los elementos de la interfaz de manera flexible y eficiente.
- Definimos el ancho y alto del layout como match_parent, lo que significa que ocupará todo el espacio disponible en la pantalla.
- Añadimos un padding de 16dp para evitar que los elementos de la interfaz estén pegados a los bordes de la pantalla.

```
<com.google.android.material.textfield.TextInputLayout
    android:id="@+id/tilNombreCompleto"
```

- Creamos un campo de texto para que el usuario ingrese su nombre completo. Utilizamos TextInputLayout y TextInputEditText, componentes de Material Design que mejoran la apariencia y funcionalidad de los campos de texto.
- Le asignamos un ID (tilNombreCompleto) para poder referenciarlo desde el código Kotlin.
- Aplicamos un estilo de caja con borde (OutlinedBox) para darle un aspecto moderno.

```
style="@style/Widget.MaterialComponents.TextInputLayout.OutlinedBox"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:layout_marginTop="32dp"
android:hint="Nombre completo"
app:layout_constraintTop_toTopOf="parent">
```

- Configuramos el ancho como match_parent (ocupa todo el ancho disponible) y el alto como wrap_content (se ajusta al contenido).
- Añadimos un margen superior de 32dp para separar este campo del borde superior de la pantalla.

```
    <com.google.android.material.textfield.TextInputEditText
        android:id="@+id/etNombreCompleto"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:inputType="textPersonName" />
</com.google.android.material.textfield.TextInputLayout>
```

- Establecemos un texto de sugerencia (hint) que dice "Nombre completo".

- Usamos restricciones (constraints) para posicionar el campo en la parte superior de la pantalla (layout_constraintTop_toTopOf).
- Dentro del TextInputLayout, colocamos un TextInputEditText para que el usuario ingrese el texto. Configuramos el tipo de entrada (inputType) como textPersonName, lo que optimiza el teclado para nombres.
- Creamos un campo de texto para que el usuario ingrese su correo electrónico.

```
<com.google.android.material.textfield.TextInputLayout
    android:id="@+id/tilCorreo"

style="@style/Widget.MaterialComponents.TextInputLayout.OutlinedBox"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginTop="16dp"
    android:hint="Correo electrónico"
    app:layout_constraintTop_toBottomOf="@id/tilNombreCompleto">
```

- Le asignamos un ID (tilCorreo) para poder referenciarlo desde el código Kotlin.
- Aplicamos el mismo estilo de caja con borde (OutlinedBox).
- Configuramos el ancho como match_parent y el alto como wrap_content.
- Añadimos un margen superior de 16dp para separar este campo del campo anterior.
- Establecemos un texto de sugerencia (hint) que dice "Correo electrónico".
- Usamos restricciones (constraints) para posicionar el campo debajo del campo de nombre completo (layout_constraintTop_toBottomOf).

```
    <com.google.android.material.textfield.TextInputEditText
        android:id="@+id/etCorreo"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:inputType="textEmailAddress" />
</com.google.android.material.textfield.TextInputLayout>
```

- Dentro del TextInputLayout, colocamos un TextInputEditText con el tipo de entrada (inputType) como textEmailAddress, lo que optimiza el teclado para correos electrónicos.

```
<com.google.android.material.textfield.TextInputLayout
    android:id="@+id/tilContrasena"
```

- Creamos un campo de texto para que el usuario ingrese su contraseña.
- Le asignamos un ID (tilContrasena) para poder referenciarlo desde el código Kotlin.
- Aplicamos el mismo estilo de caja con borde (OutlinedBox).

```
style="@style/Widget.MaterialComponents.TextInputLayout.OutlinedBox"
android:layout_width="match_parent"
android:layout_height="wrap_content"
android:layout_marginTop="16dp"
android:hint="Contraseña (mín. 6 caracteres)"
app:layout_constraintTop_toBottomOf="@id/tilCorreo"
app:passwordToggleEnabled="true">
```

- Configuramos el ancho como match_parent y el alto como wrap_content.
- Añadimos un margen superior de 16dp para separar este campo del campo anterior.
- Establecemos un texto de sugerencia (hint) que dice "Contraseña (mín. 6 caracteres)".
- Usamos restricciones (constraints) para posicionar el campo debajo del campo de correo electrónico (layout_constraintTop_toBottomOf).
- Habilitamos un botón de alternancia (passwordToggleEnabled) que permite al usuario mostrar u ocultar la contraseña.

```
    <com.google.android.material.textfield.TextInputEditText
        android:id="@+id/etContrasena"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:inputType="textPassword" />
</com.google.android.material.textfield.TextInputLayout>
```

- Dentro del TextInputLayout, colocamos un TextInputEditText con el tipo de entrada (inputType) como textPassword, lo que oculta el texto ingresado.

```
<com.google.android.material.button.MaterialButton
    android:id="@+id/btnRegistrar"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginTop="24dp"
```

- Creamos un botón utilizando MaterialButton, que es un componente de la biblioteca de Material Design para Android.
- Le asignamos un ID (btnRegistrar) para poder referenciarlo desde el código Kotlin.
- Configuramos el ancho como match_parent (ocupa todo el ancho disponible) y el alto como wrap_content (se ajusta al contenido).
- Añadimos un margen superior de 24dp para separar el botón del campo de contraseña.

```
android:padding="12dp"
android:text="Registrar"
app:layout_constraintTop_toBottomOf="@id/tilContrasena" />
```

- Establecemos un padding interno de 12dp para que el texto del botón no esté pegado a los bordes.
- Definimos el texto del botón como "Registrar".
- Usamos restricciones (constraints) para posicionar el botón debajo del campo de contraseña (layout_constraintTop_toBottomOf).
- Creamos una ProgressBar para mostrar una animación de carga mientras se realiza una operación en segundo plano (como el registro).

```
<ProgressBar
    android:id="@+id/progressBar"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
```

- Le asignamos un ID (progressBar) para poder referenciarlo desde el código Kotlin.
- Configuramos el ancho y alto como wrap_content, lo que significa que la barra de progreso se ajustará a su tamaño predeterminado.

```
android:visibility="gone"
app:layout_constraintBottom_toBottomOf="parent"
app:layout_constraintEnd_toEndOf="parent"
app:layout_constraintStart_toStartOf="parent"
app:layout_constraintTop_toTopOf="parent" />
```

- Establecemos la visibilidad inicial como gone, lo que significa que la barra de progreso estará oculta hasta que se la muestre desde el código.
- Usamos restricciones (constraints) para centrar la barra de progreso en la pantalla, tanto horizontal como verticalmente.

```
</androidx.constraintlayout.widget.ConstraintLayout>
```

- Cerramos el ConstraintLayout, indicando que aquí termina la definición de la interfaz de la pantalla de registro.

Código completo MainActivity.kt

```
package com.example.proyecto_equipo_2

import android.content.Intent
import android.os.Bundle
import androidx.appcompat.app.AppCompatActivity
import com.google.android.material.button.MaterialButton

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

        // Encuentra el botón de registro y configura su click listener

findViewById<MaterialButton>(R.id.btnIrARegistro).setOnClickListener {
            // Inicia RegisterActivity
            startActivity(Intent(this, RegisterActivity::class.java))
        }
    }
}
```

Código completo RegisterActivity.kt

```
package com.example.proyecto_equipo_2

import android.os.Bundle
import android.view.View
import android.widget.Toast
import androidx.appcompat.app.AppCompatActivity
import com.google.android.material.textfield.TextInputEditText
import com.google.android.material.button.MaterialButton
import com.google.firebase.auth.FirebaseAuth
import com.google.firebase.database.FirebaseDatabase

class RegisterActivity : AppCompatActivity() {

    private lateinit var etNombreCompleto: TextInputEditText
    private lateinit var etCorreo: TextInputEditText
    private lateinit var etContrasena: TextInputEditText
    private lateinit var btnRegistrar: MaterialButton
    private lateinit var progressBar: View

    private lateinit var auth: FirebaseAuth
    private lateinit var database: FirebaseDatabase

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

        // Inicializar vistas
        etNombreCompleto = findViewById(R.id.etNombreCompleto)
        etCorreo = findViewById(R.id.etCorreo)
        etContrasena = findViewById(R.id.etContrasena)
        btnRegistrar = findViewById(R.id.btnRegistrar)
        progressBar = findViewById(R.id.progressBar)

        // Inicializar Firebase
        auth = FirebaseAuth.getInstance()
        database = FirebaseDatabase.getInstance()

        btnRegistrar.setOnClickListener {
            registrarUsuario()
        }
    }

    private fun registrarUsuario() {
        val nombre = etNombreCompleto.text?.toString()?.trim() ?: ""
        val correo = etCorreo.text?.toString()?.trim() ?: ""
        val contrasena = etContrasena.text?.toString()?.trim() ?: ""

        // Validaciones
        if (nombre.isEmpty() || correo.isEmpty() || contrasena.isEmpty())
{
            mostrarMensaje("Todos los campos son obligatorios")
            return
        }
```

```
        if (contrasena.length < 6) {
            mostrarMensaje("La contraseña debe tener al menos 6
caracteres")
            return
        }

        mostrarProgreso(true)

        // Registrar en Firebase Auth
        auth.createUserWithEmailAndPassword(correo, contrasena)
            .addOnCompleteListener { task ->
                if (task.isSuccessful) {
                    val userId = auth.currentUser?.uid
                    if (userId != null) {
                        guardarDatosUsuario(userId, nombre)
                    } else {
                        mostrarProgreso(false)
                        mostrarMensaje("Error al obtener el ID de
usuario")
                    }
                } else {
                    mostrarProgreso(false)
                    mostrarMensaje(obtenerMensajeError(task.exception))
                }
            }
    }

    private fun guardarDatosUsuario(userId: String, nombre: String) {
        val userData = hashMapOf(
            "nombre" to nombre,
            "fechaRegistro" to System.currentTimeMillis()
        )


database.reference.child("usuarios").child(userId).setValue(userData)
            .addOnSuccessListener {
                mostrarProgreso(false)
                mostrarMensaje("Usuario registrado exitosamente")
                finish()
            }
            .addOnFailureListener { e ->
                mostrarProgreso(false)
                mostrarMensaje("Error al guardar datos: ${e.message}")
            }
    }

    private fun obtenerMensajeError(exception: Exception?): String {
        return when (exception?.message) {
            "The email address is badly formatted." -> "El formato del
correo electrónico no es válido"
            "The email address is already in use by another account." ->
"Este correo ya está registrado"
            "A network error (such as timeout, interrupted connection or
unreachable host) has occurred." ->
                "Error de conexión. Verifica tu internet"
            else -> exception?.message ?: "Error desconocido"
```

```
            }
        }

        private fun mostrarProgreso(mostrar: Boolean) {
            progressBar.visibility = if (mostrar) View.VISIBLE else View.GONE
            btnRegistrar.isEnabled = !mostrar
        }

        private fun mostrarMensaje(mensaje: String) {
            Toast.makeText(this, mensaje, Toast.LENGTH_SHORT).show()
        }
    }
```

Código completo activity_main.xml

```
<?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"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="16dp">

    <TextView
        android:id="@+id/tvBienvenida"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Bienvenido a la App"
        android:textSize="24sp"
        android:textStyle="bold"
        app:layout_constraintBottom_toTopOf="@+id/btnIrARegistro"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_chainStyle="packed" />

    <com.google.android.material.button.MaterialButton
        android:id="@+id/btnIrARegistro"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="24dp"
        android:padding="12dp"
        android:text="Ir a Registro"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/tvBienvenida" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

Código completo activity_register.xml

```
<?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"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:padding="16dp">

    <com.google.android.material.textfield.TextInputLayout
        android:id="@+id/tilNombreCompleto"

style="@style/Widget.MaterialComponents.TextInputLayout.OutlinedBox"
        android:layout width="match parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="32dp"
        android:hint="Nombre completo"
        app:layout_constraintTop_toTopOf="parent">

        <com.google.android.material.textfield.TextInputEditText
            android:id="@+id/etNombreCompleto"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:inputType="textPersonName" />
    </com.google.android.material.textfield.TextInputLayout>

    <com.google.android.material.textfield.TextInputLayout
        android:id="@+id/tilCorreo"

style="@style/Widget.MaterialComponents.TextInputLayout.OutlinedBox"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="16dp"
        android:hint="Correo electrónico"
        app:layout_constraintTop_toBottomOf="@id/tilNombreCompleto">

        <com.google.android.material.textfield.TextInputEditText
            android:id="@+id/etCorreo"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:inputType="textEmailAddress" />
    </com.google.android.material.textfield.TextInputLayout>

    <com.google.android.material.textfield.TextInputLayout
        android:id="@+id/tilContrasena"

style="@style/Widget.MaterialComponents.TextInputLayout.OutlinedBox"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="16dp"
        android:hint="Contraseña (mín. 6 caracteres)"
        app:layout_constraintTop_toBottomOf="@id/tilCorreo"
        app:passwordToggleEnabled="true">
```

```
            <com.google.android.material.textfield.TextInputEditText
                android:id="@+id/etContrasena"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:inputType="textPassword" />
        </com.google.android.material.textfield.TextInputLayout>

        <com.google.android.material.button.MaterialButton
            android:id="@+id/btnRegistrar"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_marginTop="24dp"
            android:padding="12dp"
            android:text="Registrar"
            app:layout_constraintTop_toBottomOf="@id/tilContrasena" />

        <ProgressBar
            android:id="@+id/progressBar"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:visibility="gone"
            app:layout_constraintBottom_toBottomOf="parent"
            app:layout_constraintEnd_toEndOf="parent"
            app:layout_constraintStart_toStartOf="parent"
            app:layout_constraintTop_toTopOf="parent" />

    </androidx.constraintlayout.widget.ConstraintLayout>
```