ILERNA

CFGS DESARROLLO DE APLICACIONES MULTIPLATAFORMA
Proyecto Final de Ciclo

ChatoChat

Autor: Pablo Ortiz Muñoz

Tutor: Javier Martin Rivero

Fecha de entrega: 31/3/25

Convocatoria: 2024 2025

Motivación

Objetivo General

En un mundo cada vez más interconectado, las aplicaciones de mensajería instantánea se han convertido en una de las herramientas digitales más utilizadas para la comunicación. Plataformas como WhatsApp han transformado la forma en que las personas interactúan, permitiendo conversaciones fluidas y en tiempo real. Sin embargo, muchas de estas aplicaciones están centradas únicamente en la comunicación entre contactos conocidos, dejando de lado la posibilidad de conectar con nuevas personas de manera sencilla y segura.

Este proyecto surge con la intención de desarrollar una aplicación de mensajería para Android que combine la funcionalidad clásica de los chats privados con una característica innovadora: la posibilidad de entablar conversaciones aleatorias con otros usuarios. Esta función busca ofrecer una alternativa de interacción más dinámica y espontánea, ideal para aquellos que desean ampliar su círculo social o simplemente conversar con alguien de manera casual.

Objetivos Específicos

1. Diseñar y desarrollar una interfaz intuitiva y atractiva para la experiencia de usuario (UX/UI), asegurando una navegación fluida en la aplicación.
2. Implementar un sistema de mensajería en tiempo real utilizando Firebase Realtime Database, para garantizar la comunicación instantánea.
3. Desarrollar una funcionalidad de emparejamiento aleatorio que permita a los usuarios iniciar conversaciones con personas desconocidas bajo ciertos criterios de anonimato y seguridad.
4. Garantizar la privacidad y seguridad de los usuarios mediante cifrado de mensajes, autenticación segura y mecanismos de moderación para evitar el mal uso de la plataforma.
5. Aplicar metodologías ágiles para la gestión y desarrollo del proyecto.

Metodología Utilizada

Para el desarrollo del proyecto, se ha optado por la metodología ágil KANBAN, ya que permite una gestión visual del flujo de trabajo, facilitando la organización de tareas y la optimización del proceso de desarrollo. Esta metodología se basa en la mejora continua, permitiendo adaptarse a cambios sin necesidad de ciclos de desarrollo estrictos.

KANBAN ayuda a mantener un flujo de trabajo constante mediante el uso de un tablero visual, donde las tareas se dividen en columnas representando diferentes estados, como "Pendiente", "En proceso" y "Finalizado". Esto permite un control eficiente del progreso y ayuda a identificar posibles bloqueos en el desarrollo de la aplicación.

Backlog de Tareas

1. Diseño de la Aplicación

- Creación de la interfaz en Figma.
- Documentación (Requisitos funcionales y no funcionales, diagrama de clases , etc ...)

2. Desarrollo del Chat

- Implementación del sistema de autenticación de usuarios
- Configuración de bases de datos en tiempo real
- Desarrollo del envío y recepción de mensajes

3. Funcionalidad de Chat Aleatorio

- Creación del algoritmo de emparejamiento aleatorio
- Implementación de filtros y criterios de selección

4. Seguridad y Privacidad

- Corrección de errores y optimización de código

Tecnologías y herramientas utilizadas en el proyecto:

Para el desarrollo de este proyecto, se han seleccionado herramientas y tecnologías que permiten la creación de una aplicación Android eficiente, escalable y con una buena experiencia de usuario. A continuación, se detallan las principales tecnologías utilizadas y los motivos de su elección:

1. Entorno de Desarrollo

Android Studio: Es el entorno de desarrollo integrado (IDE) oficial para la creación de aplicaciones Android. Se eligió por su compatibilidad con el SDK de Android, sus herramientas avanzadas de depuración y su integración con emuladores y dispositivos físicos.

2. Lenguaje de Programación

Kotlin: Se ha optado por Kotlin debido a sus ventajas sobre Java, como una sintaxis más limpia, mayor seguridad contra errores de nulabilidad y una mejor integración con Firebase.

3. Diseño de Interfaz y Experiencia de Usuario (UX/UI)

Figma: Utilizado para la creación de prototipos y el diseño de la interfaz de usuario. Se eligió por su facilidad de uso y compatibilidad con herramientas de desarrollo.

RemoveBG: Herramienta para la eliminación automática de fondos en imágenes, facilitando la creación de elementos gráficos sin necesidad de software más complejo.

4. Backend y Gestión de Datos

Firebase Realtime Database: Base de datos NoSQL utilizada para el almacenamiento y sincronización de datos en tiempo real. Se eligió por su facilidad de integración con Android y su bajo tiempo de respuesta para aplicaciones de mensajería.

Firebase Cloud Messaging (FCM): Servicio utilizado para el envío de notificaciones push y la gestión de mensajes en tiempo real entre los usuarios. Es una solución eficiente y gratuita dentro del ecosistema Firebase.

5. Gestión de Proyecto

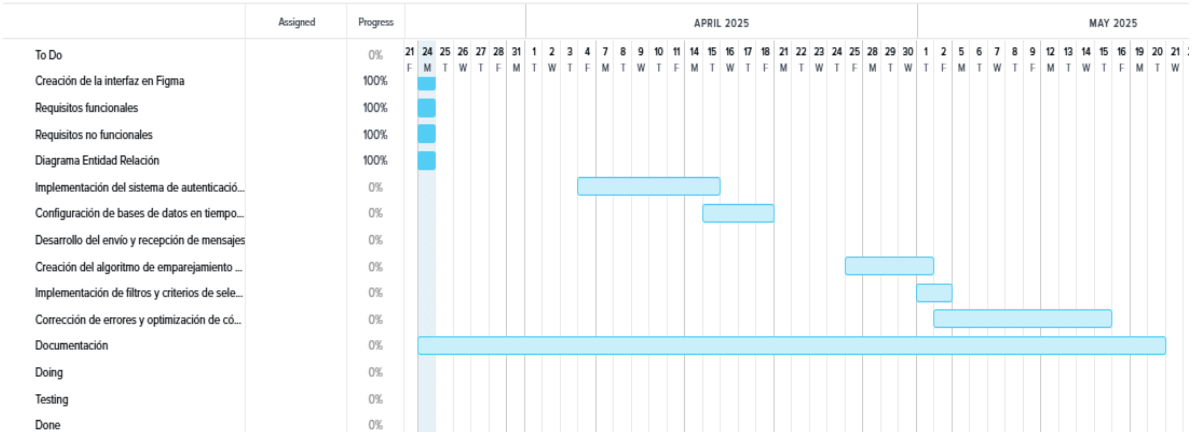
Trello: Plataforma para la gestión ágil del proyecto basada en metodología SCRUM. Se eligió por su flexibilidad en la organización de tareas y facilidad de uso en equipos de desarrollo.

Justificación de la Elección de las Tecnologías

El ecosistema de Android ofrece múltiples herramientas y tecnologías para el desarrollo de aplicaciones móviles. En este caso, la elección de Firebase como backend se debe a su facilidad de uso y su integración nativa con Android. Kotlin ha sido seleccionado por su eficiencia y popularidad en el desarrollo moderno de Android. Además, herramientas como Figma y Trello facilitan el diseño y la gestión del proyecto.

En conjunto, estas tecnologías permiten desarrollar una aplicación de mensajería rápida, segura y con una experiencia de usuario optimizada.

Estimación de recursos y planificación:



Requisitos funcionales:

Chat Privado

- Los usuarios podrán enviar y recibir mensajes en tiempo real.
- Se podrá compartir imágenes dentro del chat.
- Los mensajes se almacenarán en Firebase Realtime Database.
- Se implementará la funcionalidad de eliminar mensajes enviados.
- Se permitirá ver el estado en línea de otros usuarios.

Chat Aleatorio

- El usuario podrá conectarse con una persona aleatoria mediante la funcionalidad de chat aleatorio.
- El usuario podrá abandonar la conversación en cualquier momento.
- Se podrá reportar o bloquear usuarios en caso de mal uso.

Notificaciones

- Se enviarán notificaciones push cuando el usuario reciba un nuevo mensaje.
- El usuario podrá configurar las preferencias de notificación (activar/desactivar).

Seguridad y Privacidad

- Se implementará cifrado de mensajes para proteger la privacidad de las conversaciones.
- El usuario podrá bloquear o reportar a otros usuarios.
- Se implementará un sistema de moderación para detectar contenido inapropiado.

Requisitos No funcionales:

Rendimiento y escalabilidad

- La aplicación debe ser capaz de manejar múltiples conversaciones simultáneamente sin afectar el rendimiento.
- El tiempo de respuesta del chat en tiempo real no debe superar los 2 segundos.
- La aplicación debe ser compatible con dispositivos Android desde la versión 8.0 (Oreo) en adelante.

Usabilidad y experiencia de usuario (UX/UI)

- La interfaz debe ser intuitiva y fácil de usar.
- El diseño debe adaptarse a diferentes tamaños de pantalla y resoluciones.
- La aplicación debe ofrecer un modo oscuro para mejorar la experiencia de uso.

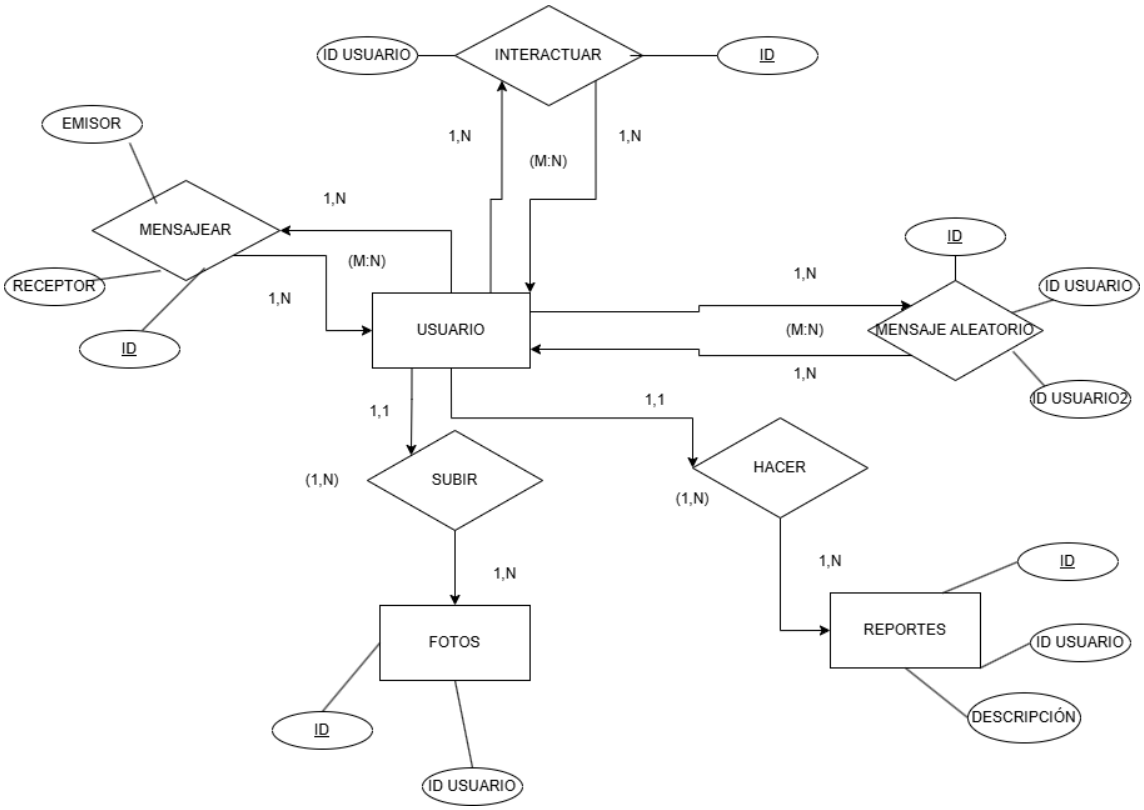
Seguridad y privacidad

- Los datos de usuario deben almacenarse de forma segura en Firebase con reglas de acceso adecuadas.
- La aplicación debe cumplir con normativas de protección de datos como GDPR o similares.
- Se debe evitar el acceso no autorizado a las conversaciones mediante autenticación segura.

Mantenimiento y actualización

- La aplicación debe permitir actualizaciones sin afectar la información almacenada de los usuarios.
- El código debe seguir buenas prácticas de desarrollo para facilitar su mantenimiento y escalabilidad.

ER



Modelo Relacional:

USUARIO (ID_USUARIO PK)

FOTOS (ID PK, ID_USUARIO FK → USUARIO(ID_USUARIO))

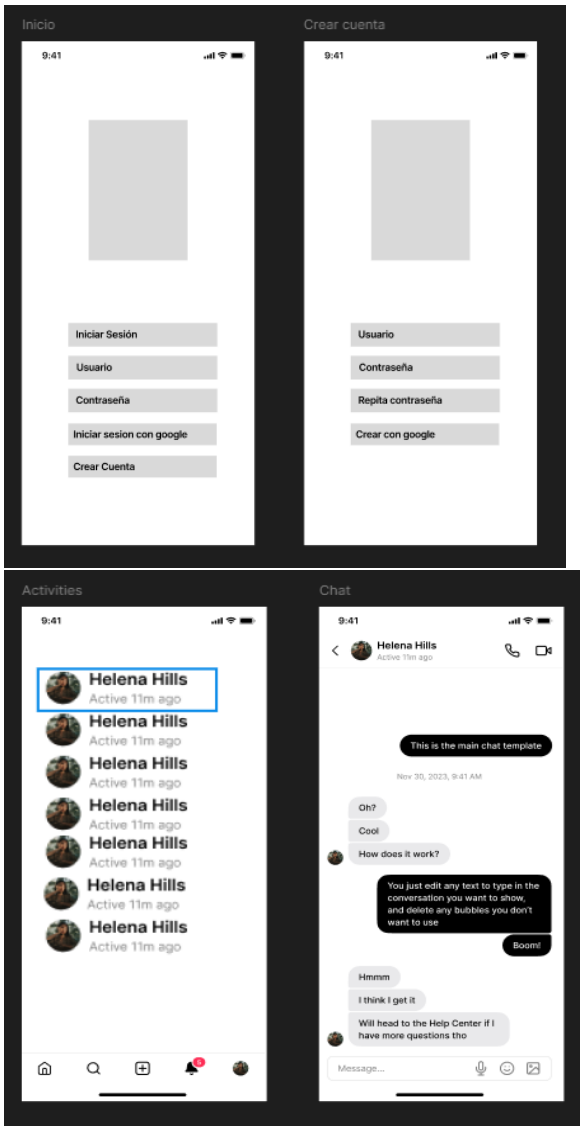
REPORTES (ID PK, ID_USUARIO FK → USUARIO(ID_USUARIO), DESCRIPCION)

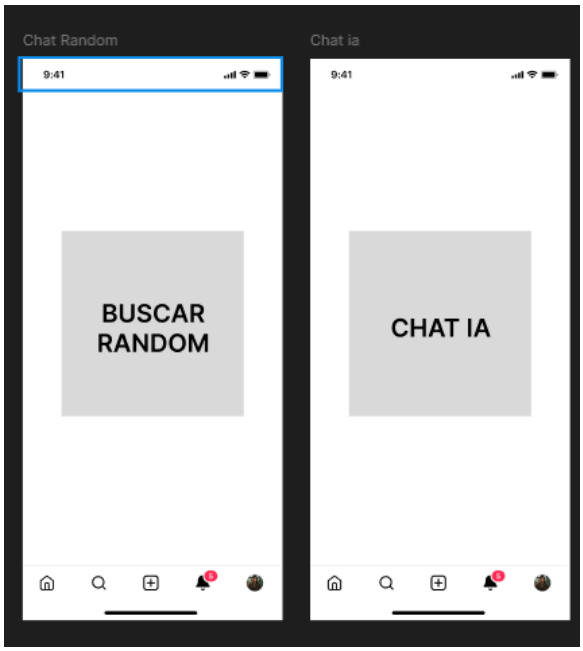
MENSAJEAR (ID PK, EMISOR_ID FK → USUARIO(ID_USUARIO), RECEPTOR_ID FK → USUARIO(ID_USUARIO))

INTERACTUAR (ID PK, ID_USUARIO1 FK → USUARIO(ID_USUARIO), ID_USUARIO2 FK → USUARIO(ID_USUARIO))

MENSAJE_ALEATORIO (ID PK, ID_USUARIO1 FK → USUARIO(ID_USUARIO), ID_USUARIO2 FK → USUARIO(ID_USUARIO))

Diseño:


Inicio
9:41
Iniciar Sesión
Usuario
Contraseña
Iniciar sesion con google
Crear Cuenta
Crear cuenta
9:41
Usuario
Contraseña
Repita contraseña
Crear con google
Activities
9:41
Helena Hills
Active 11m ago
Chat
9:41
Helena Hills
Active 11m ago
This is the main chat template
Nov 30, 2023, 9:41 AM
Oh?
Cool
How does it work?
You just edit any text to type in the conversation you want to show, and delete any bubbles you don't want to use
Boom!
Hmmm
I think I get it
Will head to the Help Center if I have more questions tho
Message...



Abstrac

In an increasingly interconnected world, instant messaging applications have become essential tools for digital communication. While platforms like WhatsApp have revolutionized how people stay in touch, they often focus solely on communication between known contacts. This leaves a gap for users who wish to connect with new people in a simple and secure way.

This project aims to develop an Android messaging app that merges the traditional functionality of private chats with a novel feature: random conversations with other users. This dynamic approach is designed to promote spontaneous interactions, ideal for those looking to expand their social circle or engage in casual conversations.

The main goal is to offer a user-friendly and secure platform that encourages both familiar and unexpected interactions. To achieve this, the application will include a modern, intuitive user interface (UI/UX), real-time messaging powered by Firebase Realtime Database, and a secure, anonymous matching system for random chats. Strong emphasis will be placed on user privacy and platform safety through encrypted communication, secure authentication, and moderation tools to prevent misuse.

The project will be developed using agile methodologies to ensure flexibility, iterative improvement, and a user-centered design process.

Diagrama de clases

Usuario

+UsuarioId: String
+nombre: String
+correo: String
+Imagen: String

+EnviarMensaje(to: nombre, mensaje: String): void
+MatchRandom(): nombre

Mensaje

+mensajeID: String +emisorId: String +receptorId: String

+timestamp: Long

+ contenido: String

+Hashmap(): Hashmap<String, Any>

Chat

+EnviarMensaje(emisorID: String, receptorId: String, contenido: String): void

+RecibirMensaje(UsuarioId: String, Mensaje : String): void

Auth

+signIn(email: String, password: String): Task

+signUp(email: String, password: String): Task

+signOut(): void

+getCurrentUser(): User

Match

+Random(EsteUsuarioId: String): User

+Excluidos(UsuarioId: String): List

Firebase

+guardarUsuario(usuario:Usuario): void

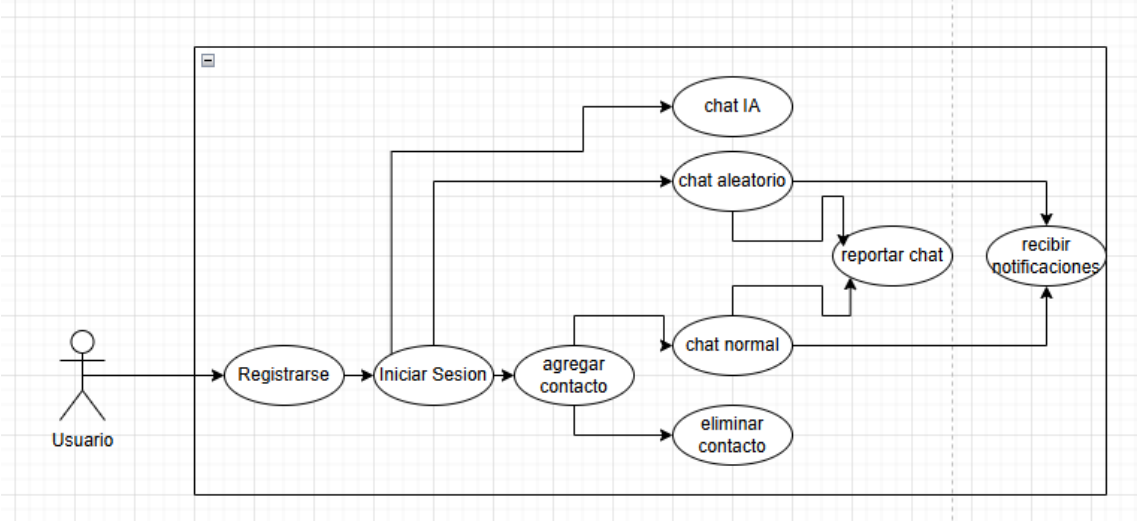
+getUsuarioById(UsuarioId: String): Task

+guardarMensaje(mensaje: Mensaje): void

Relaciones

- AuthService interactúa con FirebaseAuth.
- Match consulta a Firebase para emparejamientos.
- Usuario puede enviar mensajes a otro Usuario.
- Mensaje contiene referencias a Usuario mediante emisorID y ReceptorId.

Diagrama de casos de uso


chat IA
chat aleatorio
reportar chat
recibir notificaciones
Usuario
Registrarse
Iniciar Sesion
agregar contacto
chat normal
eliminar contacto

Código relevante

```
private fun RegistrarUsuario(nombre: String, correo: String, contraseña: String) {
    auth.createUserWithEmailAndPassword(correo, contraseña)
        .addOnCompleteListener(this) { task ->
            if (task.isSuccessful) {
                var uid: String = ""
                uid = auth.currentUser!!.uid
                reference =
                    FirebaseDatabase.getInstance().reference.child( pathString: "Usuarios").child(uid)

                val hashmap = HashMap<String, Any>()
                val h_nombre: String = RegistroNombre.text.toString()
                val h_correo: String = RegistroCorreo.text.toString()

                hashmap["uid"] = uid
                hashmap["nombre"] = h_nombre
                hashmap["correo"] = h_correo
                hashmap["imagen"] = ""
                hashmap["buscar"] = h_nombre.lowercase()

                reference.updateChildren(hashmap).addOnCompleteListener { task2 ->
                    if (task2.isSuccessful) {
                        Toast.makeText(
                            context: this,
                            text: "Usuario registrado correctamente",
                            Toast.LENGTH_SHORT
                        ).show()
                        val intent = Intent( packageContext: this@RegistroActivity, Inicio::class.java)
                        startActivity(intent)
                        finish()
                    }

                }.addOnFailureListener { e ->
                    Toast.makeText(
                        context: this,
                        text: "Error al registrar usuario: ${e.message}",
                        Toast.LENGTH_SHORT
                    ).show()
                }
```

Registro de usuario usando Firebase Authentication y Realtime Database

Al configurar el registro de usuarios, nos apoyamos en Firebase Authentication para dar de alta una cuenta nueva con un correo y una clave. A esto le añadimos Firebase Realtime Database, que es donde guardamos los detalles extra del usuario, como su nombre y su dirección de correo.

Todo esto se hace dentro de la función RegistrarUsuario(nombre: String, correo: String, contraseña: String). Primero, llamamos al método createUserWithEmailAndPassword que nos da Firebase Authentication para registrar al usuario. Si todo va bien, conseguimos el UID (un Identificador Único de Usuario), que Firebase da solo y asegura que cada usuario sea único en todo el sistema.

Después, entramos a la base de datos en tiempo real buscando el nodo Usuarios, usando el UID como el nombre del nuevo nodo. Creamos un HashMap con lo que queremos guardar: UID, nombre, correo electrónico, una imagen (que al principio está vacía) y un campo para buscar con el nombre en minúsculas, que sirve para las búsquedas en la app.

Estos datos se meten en la base de datos con el método updateChildren. Si todo sale bien, le avisamos al usuario con un mensaje (Toast) y lo mandamos directo a la pantalla principal de la app (Inicio). Si hay algún problema, atrapamos el error y le mostramos al usuario un mensaje explicando lo que pasó.

Pruebas de caja negra

```
// Clase auxiliar para lógica de validación
object ValidadorUsuario {
    fun validar(nombre: String, correo: String, contrasena: String, confirmar: String): ResultadoValidacion {
        if (nombre.isEmpty() || correo.isEmpty() || contrasena.isEmpty() || confirmar.isEmpty()) {
            return ResultadoValidacion.ERROR_CAMPOS_VACIOS
        }
        if (contrasena != confirmar) {
            return ResultadoValidacion.ERROR_CONTRASEÑAS_NO_COINCIDEN
        }
        return ResultadoValidacion.EXITO
    }

    enum class ResultadoValidacion {
        EXITO,
        ERROR_CAMPOS_VACIOS,
        ERROR_CONTRASEÑAS_NO_COINCIDEN
    }
}
```

```
import org.junit.Assert.*
import org.junit.Test

class ValidarDatosTest {

    @Test
    fun testCamposVacios() {
        val resultado = ValidadorUsuario.validar( nombre: "", correo: "correo@test.com", contrasena: "1234", confirmar: "1234")
        assertEquals(ValidadorUsuario.ResultadoValidacion.ERROR_CAMPOS_VACIOS, resultado)
    }

    @Test
    fun testContraseñasNoCoinciden() {
        val resultado = ValidadorUsuario.validar( nombre: "Juan", correo: "correo@test.com", contrasena: "1234", confirmar: "5678")
        assertEquals(ValidadorUsuario.ResultadoValidacion.ERROR_CONTRASEÑAS_NO_COINCIDEN, resultado)
    }

    @Test
    fun testValidacionExitosa() {
        val resultado = ValidadorUsuario.validar( nombre: "Juan", correo: "correo@test.com", contrasena: "1234", confirmar: "1234")
        assertEquals(ValidadorUsuario.ResultadoValidacion.EXITO, resultado)
    }
}
```