Universidad Don Bosco.

Campus Virtual

Asignatura:

Desarrollo de Software para Móviles DSM941

Grupo Teórico:

01

Foro 2

Integrantes:

Alas Linares Alejandro Antonio AL192188

De Paz Velásquez Vicente Daniel DV192307

Duran Meléndez Gilberto Emmanuel DM192201

Gutiérrez Borja Rafael Armando GB192205

Docente:

Alexander Alberto Sigüenza Campos

Fecha: domingo 24 de noviembre del 2024

Contenido

Introducción

La autenticación de usuarios es un elemento fundamental en el desarrollo de aplicaciones móviles, ya que permite proteger los datos sensibles del usuario y garantizar el acceso seguro a los servicios y funcionalidades del aplicativo. Firebase a lo largo de los años, se ha convertido en una plataforma integral que ofrece múltiples opciones de autenticación, muy completos y fáciles de implementar en las aplicaciones móviles. En este trabajo se describirá principalmente la integración de la autenticacion con Firebase en aplicaciones desarrolladas en Kotlin.

El objetivo principal de este estudio es explorar y documentar las diferentes opciones de autenticación proporcionadas por Firebase, como la autenticación mediante correo electrónico y contraseña, y la autenticación con Google. Estas alternativas son ampliamente utilizadas debido a su simplicidad y a la familiaridad que los usuarios tienen con ellas, lo que contribuye a mejorar la experiencia de usuario sin comprometer la seguridad.

Para alcanzar este objetivo, se comienza con una investigación exhaustiva sobre las características, ventajas y desventajas de cada opción de autenticación. A partir de los hallazgos obtenidos, se realiza la implementación de un sistema de autenticación por correo electrónico utilizando las herramientas que ofrece Firebase. Este proceso se detalla minuciosamente, acompañándose de imágenes y descripciones paso a paso que buscan facilitar la comprensión y replicación por parte de otros desarrolladores.

Además, se desarrolla una pantalla de inicio de sesión que incluye tanto la funcionalidad de autenticación por correo electrónico como la autenticación con Google. Esta pantalla no solo ejemplifica la integración de múltiples métodos de autenticación en una misma aplicación, sino que también sigue un diseño centrado en el usuario, asegurando una interacción fluida y amigable. Como parte del desarrollo, se presenta el código fuente con una explicación detallada de su lógica

y estructura, cubriendo los principales componentes del proyecto, como los archivos Validations.kt y MainActivity.kt.

En definitiva, este trabajo busca proporcionar una guía práctica sobre como integrar sistemas de autenticación seguros y eficientes en las aplicaciones móviles. Además, resaltar la importancia de utilizar herramientas modernas como Firebase y Kotlin para optimizar el proceso de desarrollo y garantizar una experiencia de usuario segura y confiable.

Objetivos

Objetivo General:

- Analizar los distintos tipos de autenticación, conociendo sus principales características, para proporcionar una visión clara que facilite la elección del mejor método

Objetivos Específicos:

- Realizar un ejemplo práctico, por medio correo electrónico
- Implementar y evaluar un sistema de evaluación para datos de usuario, como el formato de correo electrónico y la robutez de contraseñas

Investigación sobre opciones de autenticación

Firebase ofrece varios métodos de autenticación que pueden utilizarse según las necesidades del proyecto, los más importantes son:

1. **Autenticación con correo electrónico y contraseñas:** este modo de autenticación es el más simple de implementar, permite a los usuarios identificarse mediante un correo electrónico y una contraseña personalizada, esto hace que sea un servicio altamente personalizable y flexible, ideal para aplicaciones que requieren un nivel de control sobre el proceso de registro.

   Cabe aclarar que sí bien este modelo de autenticación es fácil de implementar puede conllevar un alto desarrollo en políticas de seguridad dentro del sistema para no comprometer las credenciales de los usuarios, una revisión de la autenticidad de los correos electrónicos y contraseñas fuertes.

   Además, este modo requiere que los usuarios auto gestionen sus propias credenciales, lo que significa que se debería considerar un modelo de expiración de contraseñas para asegurar que los usuarios tengan sus credenciales actualizadas.

2. **Autenticación con proveedores de identidad federada:** este modo permite a los usuarios iniciar sesión utilizando sus cuentas existentes en otros servicios como Google, Facebook, X, Apple, etc.

   Simplifica bastante el proceso de registro y de inicio de sesión de los usuarios, pues pueden reutilizar sus credenciales y por lo general tiende a garantizar que más usuarios inicien sesión en la aplicación debido a la facilidad para hacerlo.

Si bien el proceso final para el usuario es mucho más fácil y simple requiere una gran configuración dentro de la consola de firebase, por lo que el proceso de implementación no es tan simple como otros servicios mencionados.

3. **Autenticación telefónica:** este método de autenticación es parecido al anterior, en el sentido que las credenciales del usuario pasan a ser algo que el usuario ya posee, en este escenario la autenticación se da mediante un código de verificación enviado vía SMS. Es una opción segura y fácil de usar, además garantiza que los usuarios tienen algo que los identifique y no es necesario verificar su autenticidad.

   Mediante este método usuarios que no poseen correo electrónico pueden acceder a las aplicaciones lo que hace la experiencia de usuario mucho más amigable. Aunque, la parte mala de este servicio, puede conllevar a un alto costo asociado a la cantidad de SMS que la aplicación debe enviar en bases de usuarios altas.

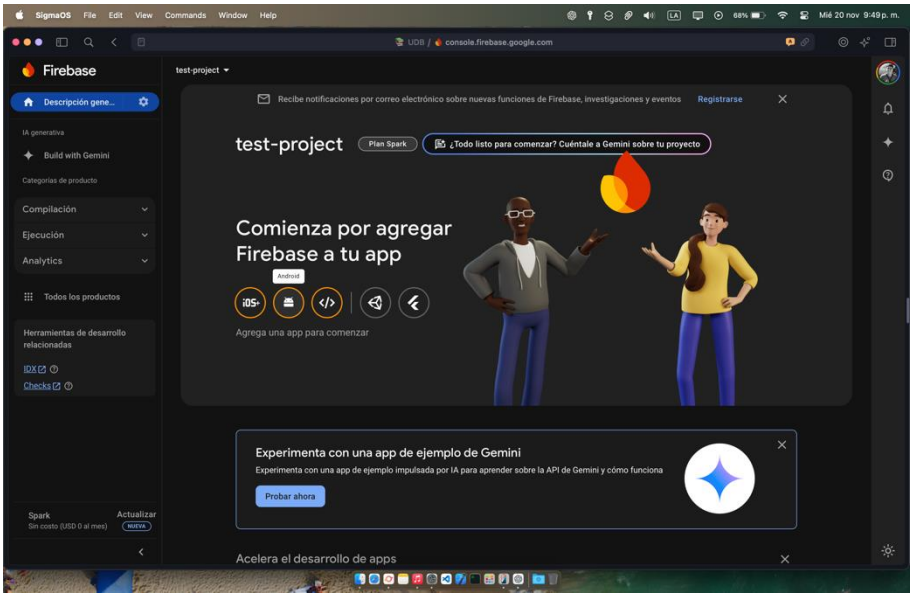
4. **Autenticación anónima: p**ermite a los usuarios acceder a la aplicación sin necesidad de registrarse, es Ideal para aplicaciones que desean permitir el uso básico sin comprometer la privacidad del usuario.

   Hay que tomar en cuenta de que este modo no proporciona una forma de identificar a los usuarios de forma única, lo que puede limitar las funcionalidades disponibles.

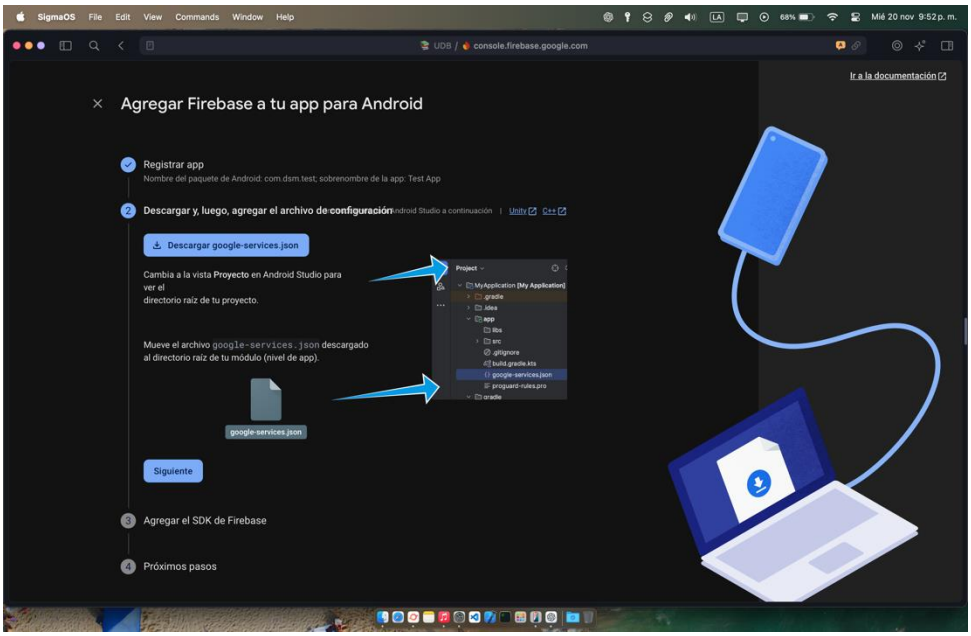
Implementación de autenticación por correo electrónico

Para implementar este método de autenticación se pueden seguir los siguientes pasos:

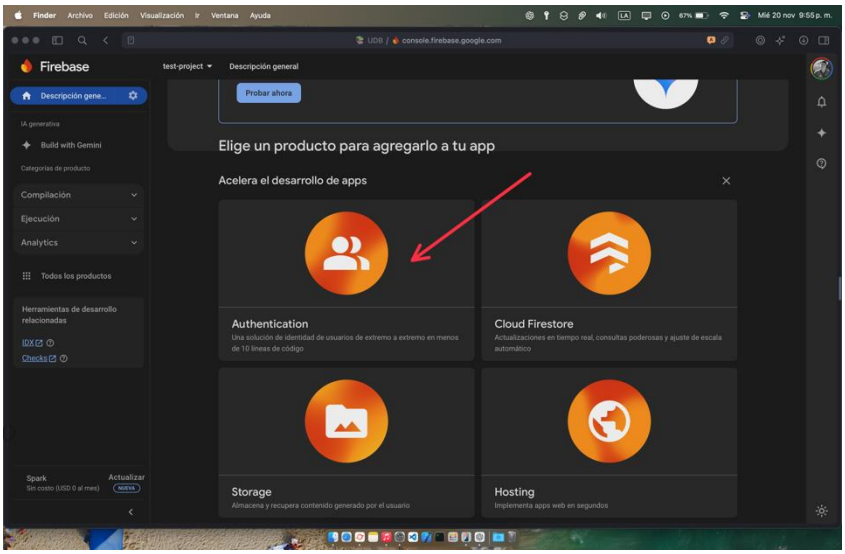
1. Crear un nuevo proyecto en firebase y agregar una aplicación android



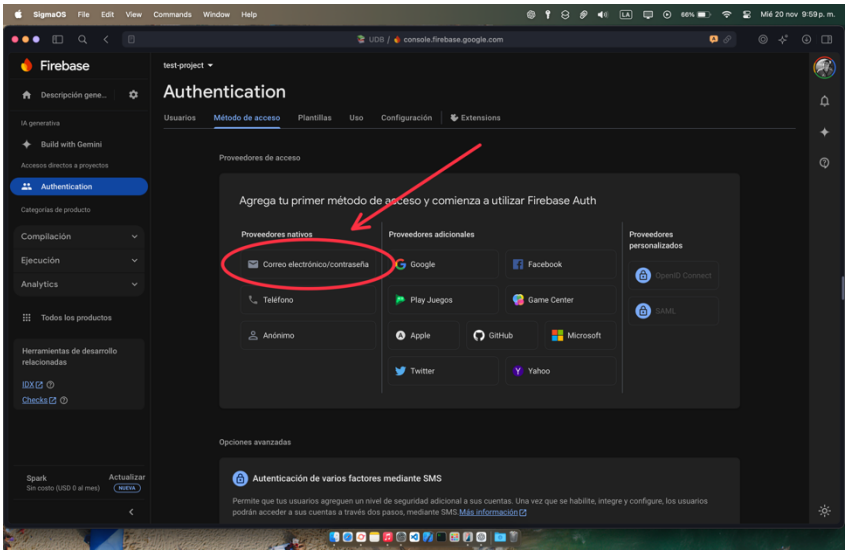
2. Una vez agregada la aplicación necesitaremos descargar el archivo google-services.json y agrégalo a la carpeta app/ de nuestro proyecto Android.



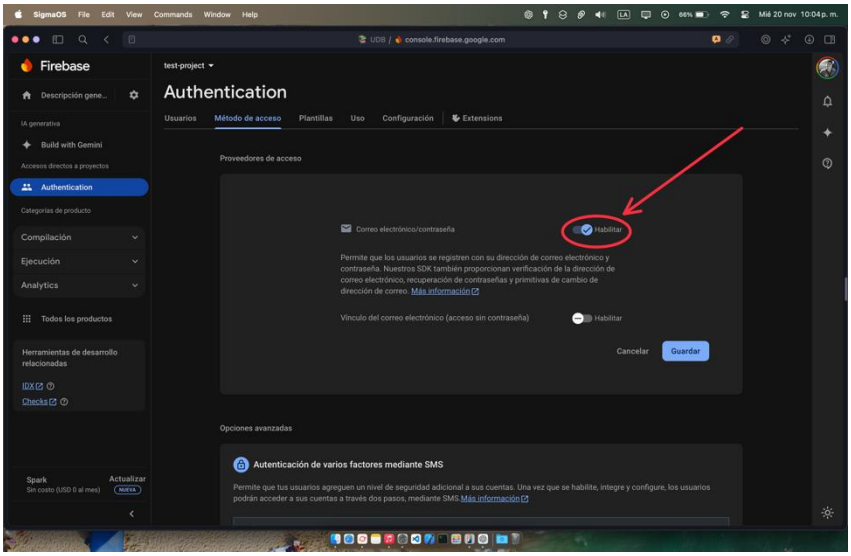
3. Ahora debemos configurar nuestro proyecto en firebase para hacer uso del método de autenticación por correo electrónico:
    a. Vamos a la consola de firebase y hacemos click en “Autenticación”



    b. Ahora damos click en el apartado de “Método de acceso” y seleccionamos “Correo electrónico/contraseña”.



c. En la nueva interfaz activamos el servicio deseado:



4. Ahora necesitamos hacer unos cambios para que nuestra aplicación Android funcione correctamente, implementando lo siguiente en el código Kotilin:
   a. Agregamos las dependencias necesarias en el archivo build.gradle: *En el nivel del módulo (app):

```
implementation 'com.google.firebase:firebase-auth-ktx'
implementation platform('com.google.firebase:firebase-bom:32.1.0')
```

   b. Aplicamos el plugin de Google Services en nuestro archivo build.gradle de nivel de proyecto:

```
classpath 'com.google.gms:google-services:4.4.0'
```

c. Agregar el siguiente código en el archivo build.gradle de nivel módulo:

```
apply plugin: 'com.google.gms.google-services'
```

5. Ahora solo nos quedaría diseñar una interfaz gráfica para registrar y permitir el acceso de los usuarios, una vez teniendo eso solo necesitamos crear una actividad para controlar las acciones de los usuarios, en este caso crearemos una actividad llamada “AuthActivity.kt”:

```
import android.os.Bundle
import android.widget.Toast
import androidx.appcompat.app.AppCompatActivity
import com.google.firebase.auth.FirebaseAuth
import kotlinx.android.synthetic.main.activity_auth.*

class AuthActivity : AppCompatActivity() {
    private lateinit var auth: FirebaseAuth

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

        auth = FirebaseAuth.getInstance()

        registerButton.setOnClickListener {
            val email = emailEditText.text.toString().trim()
            val password = passwordEditText.text.toString().trim()

            if (email.isNotEmpty() && password.isNotEmpty()) {
                auth.createUserWithEmailAndPassword(email, password)
                    .addOnCompleteListener { task ->
                        if (task.isSuccessful) {
                            Toast.makeText(this, "Registro exitoso", Toast.LENGTH_SHORT).show()
                        } else {
                            Toast.makeText(this, "Error: ${task.exception?.message}",
Toast.LENGTH_SHORT).show()
                        }
                    }
            } else {
                Toast.makeText(this, "Completa todos los campos", Toast.LENGTH_SHORT).show()
            }
        }

        loginButton.setOnClickListener {
            val email = emailEditText.text.toString().trim()
            val password = passwordEditText.text.toString().trim()

            if (email.isNotEmpty() && password.isNotEmpty()) {
                auth.signInWithEmailAndPassword(email, password)
                    .addOnCompleteListener { task ->
                        if (task.isSuccessful) {
                            Toast.makeText(this, "Inicio de sesión exitoso", Toast.LENGTH_SHORT).show()
                        } else {
                            Toast.makeText(this, "Error: ${task.exception?.message}",
Toast.LENGTH_SHORT).show()
                        }
                    }
            } else {
                Toast.makeText(this, "Completa todos los campos", Toast.LENGTH_SHORT).show()
            }
        }
    }
}
```

6. Solo nos quedaría probar los cambios que hemos realizado y comprobar que la integración funciona como esperado.

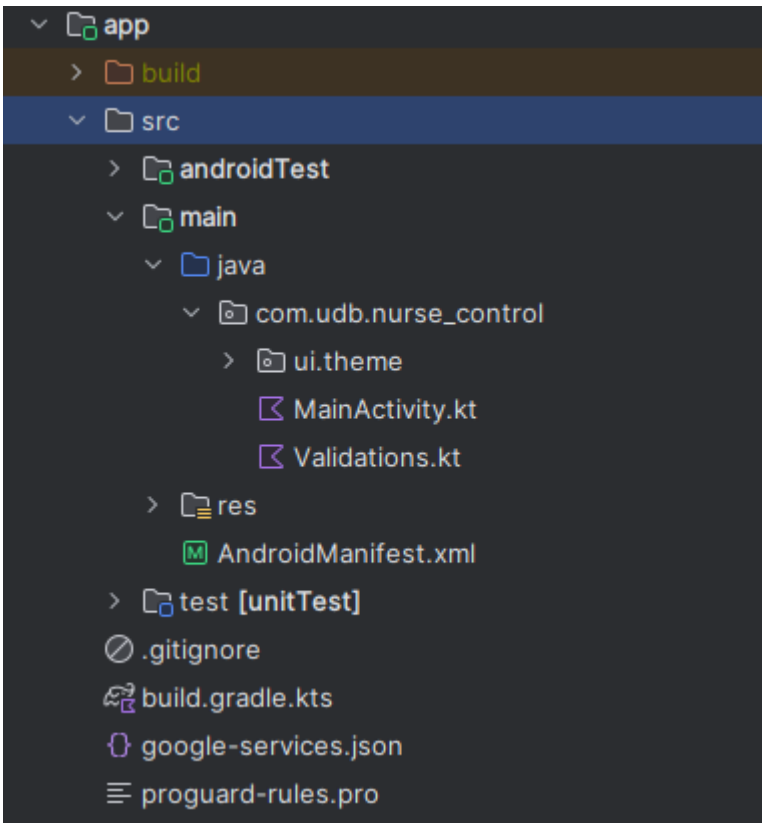
Desarrollo de pantalla de login

Código fuente

Link del repositorio: https://github.com/DanDPV/dsm-foro-2

Link de la video demostración: https://youtu.be/nahi4hPkHIY

Explicación del código



En la estructura de nuestro proyecto tenemos algunos archivos importantes:

- **google-services.json:** Archivo que nos permite acceder a los servicios de firebase.
- **Validations,kt:** En este archivo guardamos validaciones creadas para ser reutilizadas
- **MainActivity**.kt: En este archivo se encuentra la funcionalidad principal de la app.

Validations.kt

```kotlin
package com.udb.nurse_control

DanDPV
fun isValidEmail(email: String): Boolean {
    val emailRegex = "^[A-Za-z0-9+_.-]+@[A-Za-z0-9.-]+\\.[A-Za-z]{2,}$".toRegex()
    return emailRegex.matches(email)
}

```

De momento solo almacenamos la función para validar un email correcto.

MainActivity.kt

```kotlin
package com.udb.nurse_control

import ...

DanDPV
class MainActivity : ComponentActivity() {
    private lateinit var auth: FirebaseAuth
    private lateinit var googleSignInClient: GoogleSignInClient
    private var errorMessage by mutableStateOf<String?>( value: null)

    DanDPV
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        auth = FirebaseAuth.getInstance()

        // Configure Google Sign-In
        val gso = GoogleSignInOptions.Builder(GoogleSignInOptions.DEFAULT_SIGN_IN)
            .requestIdToken( serverClientId: "595030947142-rjatc4vfl333thmkkcil7rs5eqjeia2i.apps.googleusercontent.com")
            .requestEmail()
            .build()

        googleSignInClient = GoogleSignIn.getClient( activity: this, gso)
```

En esta parte del código declaramos las variables que vamos a necesitar para realizar la autenticación con firebase.

Obtenemos y guardamos la instancia de autenticación de firebase, además, configuramos el GoogleSignInClient para iniciar sesión con una cuenta de Google.

```
setContent {
    DSMForo2Theme {
        // A surface container using the 'background' color from the theme
        Surface(
            modifier = Modifier.fillMaxSize(), color = MaterialTheme.colorScheme.background
        ) {
            MyApp(
                auth = auth,
                googleSignInClient = googleSignInClient,
                onGoogleSignInClick = { signInWithGoogle() })
            // Show Error Dialog if errorMessage is not null
            errorMessage?.let { it: String
                AlertDialog(
                    onDismissRequest = { errorMessage = null },
                    title = { Text( text: "Error") },
                    text = { Text(it) },
                    confirmButton = {
                        Button(onClick = { errorMessage = null }) { this: RowScope
                            Text( text: "OK")
                        }
                    }
                )
            }
        }
    }
}
```

Esta función “setContent” es el punto de entrada donde renderizamos nuestros componentes usando jetpack compose, en nuestro caso tenemos un componente llamado “MyApp” que se encarga de renderizar la pantalla correcta dependiendo del estado de autenticación, además tenemos un pequeño alert dialog para mostrar mensajes de error de ser necesario.

```
// Google Sign-In result launcher
private val googleSignInLauncher =
    registerForActivityResult(ActivityResultContracts.StartActivityForResult()) { result ->
        val task = GoogleSignIn.getSignedInAccountFromIntent(result.data)
        handleGoogleSignInResult(task)
    }

// Start Google Sign-In
DanDPV
fun signInWithGoogle() {
    val signInIntent = googleSignInClient.signInIntent
    googleSignInLauncher.launch(signInIntent)
}

DanDPV
private fun showErrorDialog(message: String) {
    errorMessage = message
}
```

Luego siempre dentro del componente principal tenemos las funciones necesarias para iniciar sesión con Google.

Primero declaramos la variable googleSignInLauncher que usará un Intent para iniciar sesión con Google.

Luego declaramos la función signInWithGoogle que usará dicho launcher mandándole un intent preguntándole al usuario por la cuenta que desea usar.

```
DanDPV
private fun handleGoogleSignInResult(task: Task<GoogleSignInAccount>) {
    try {
        val account = task.getResult(ApiException::class.java)
        firebaseAuthWithGoogle(account,
            onSuccess = {
                // Navigate to HomeScreen or update UI to show HomeScreen
                setContent {
                    DSMForo2Theme {
                        // A surface container using the 'background' color from the theme
                        Surface(
                            modifier = Modifier.fillMaxSize(),
                            color = MaterialTheme.colorScheme.background
                        ) {
                            MyApp(
                                auth = auth,
                                googleSignInClient = googleSignInClient,
                                onGoogleSignInClick = { signInWithGoogle() })
                        }
                    }
                }
            },
            onError = { errorMessage ->
                // Show an error dialog with the provided error message
                showErrorDialog(errorMessage)
            }
        )
    } catch (e: ApiException) {
        if (e.statusCode == 12501) {
            showErrorDialog("Hubo un error al iniciar sesión con google.")
        }

        Log.d( tag: "TEST",  msg: "Google sign-in failed: ${e.statusCode}")
    }
}
```

En esta función hacemos el handling de la respuesta del intent, si el usuario selecciona una cuenta y no se presenta ningún error con firebase se ejecutará el callback “onSuccess”, dentro de ese callback modificamos la pantalla para que se actualice debido a que el estado de autenticación ha cambiado.

Si se presenta un error utilizamos el Dialog creado previamente mostrando un mensaje de error.

```
DanDPV
private fun firebaseAuthWithGoogle(
    account: GoogleSignInAccount?,
    onSuccess: () -> Unit,
    onError: (String) -> Unit
) {
    val credential = GoogleAuthProvider.getCredential(account?.idToken, null)
    FirebaseAuth.getInstance().signInWithCredential(credential)
        .addOnCompleteListener(this) { task ->
            if (task.isSuccessful) {
                // Sign-in successful, proceed to HomeScreen
                onSuccess()
            } else {
                // Handle failed sign-in
                val errorMessage =
                    task.exception?.localizedMessage ?: "Hubo un error en el inicio de sesión con google"
                onError(errorMessage)
            }
        }
}
```

Finalmente, en esta función utilizamos el firebase SDK para crear una credencial utilizando los datos de la cuenta de Google del usuario, luego iniciamos sesión con firebase utilizando dicha credencial.

```
DanDPV
@Composable
fun MyApp(
    auth: FirebaseAuth,
    googleSignInClient: GoogleSignInClient,
    onGoogleSignInClick: () -> Unit
) {
    var isLoggedIn by remember { mutableStateOf( value: false) }

    // Observe authentication state in a LaunchedEffect and DisposableEffect
    LaunchedEffect(auth) { this: CoroutineScope
        val currentUser = auth.currentUser
        isLoggedIn = currentUser != null
    }

    DisposableEffect(auth) { this: DisposableEffectScope
        // Add an auth state listener to update isLoggedIn on changes
        val authStateListener = FirebaseAuth.AuthStateListener { it: FirebaseAuth
            isLoggedIn = it.currentUser != null
            if (it.currentUser != null) {
                Log.d( tag: "TEST",  msg: "Email: ${it.currentUser?.email}")
            }
        }
        auth.addAuthStateListener(authStateListener)

        // Remove the listener when the composable is removed from composition
        onDispose {
            auth.removeAuthStateListener(authStateListener)
        } ^DisposableEffect
    }
```

A continuación, mostramos nuestro componente “MyApp” que es el encargado principal de manejar qué pantalla se mostrará dependiendo del estado de autenticación.

Primero declaramos una variable para guardar el estado de autenticación.

Lugo usamos la función “LaunchedEffect” para estar pendientes del parámetro “auth” y ejecutar el código correspondiente si “auth” cambia de valor.

También usamos la función “DisposableEffect” para estar pendientes del parámetro “auth” y ejecutar otra parte de código cada que cambie de valor, este código es necesario ponerlo dentro de un “DisposableEffect” porque manejamos “listeners” y es necesario limpiar dichos listeners al salir de este componente para evitar problemas de rendimiento.

```
// Show the correct screen based on authentication state
if (isLoggedIn) {
    HomeScreen(auth = auth, onSignOut = {

        FirebaseAuth.getInstance().signOut()
        googleSignInClient.signOut() // Ensure Google client signs out
        isLoggedIn = false
    })
} else {
    Login(
        // Callback for successful login (update state if necessary)
        onLoginSuccess = { isLoggedIn = true },
        onGoogleSignInClick = onGoogleSignInClick
    )
}
```

En este último bloque de “MyApp” tenemos la lógica de mostrar la HomeScreen o LoginScreen dependiendo si el usuario ha iniciado sesión o no.

```
DanDPV
@Composable
fun HomeScreen(auth: FirebaseAuth, onSignOut: () -> Unit) {
    val currentUser = auth.currentUser
    // Home screen UI for logged-in users
    Column(modifier = Modifier.fillMaxSize(), verticalArrangement = Arrangement.Center) { this: ColumnScope
        Text(
            text: "Bienvenido",
            fontWeight = FontWeight.Bold,
            fontSize = 32.sp,
            textAlign = TextAlign.Center,
            modifier = Modifier.fillMaxWidth()
        )
        Text(
            text: "${currentUser?.email}", fontSize = 22.sp,
            textAlign = TextAlign.Center,
            modifier = Modifier.fillMaxWidth()
        )
        Button(
            onClick = {
                FirebaseAuth.getInstance().signOut()
                onSignOut()
            },
            modifier = Modifier
                .fillMaxWidth()
                .padding(horizontal = 20.dp)
                .padding(top = 20.dp),
        ) { this: RowScope
            Text( text: "Cerrar Sesión", modifier = Modifier.padding(vertical = 5.dp))
        }
    }
}
```

La pantalla HomeScreen es bastante sencilla, solo tenemos un texto centrado que muestra el correo del usuario que ha iniciado sesión y un botón para cerrar sesión.

```
DanDPV
@Composable
fun Login(onLoginSuccess: () -> Unit, onGoogleSignInClick: () -> Unit) {
    var email by remember { mutableStateOf( value: "") }
    var emailError by remember { mutableStateOf( value: false) }
    var password by rememberSaveable { mutableStateOf( value: "") }
    var passwordVisible by remember { mutableStateOf( value: false) }
    var passwordError by remember { mutableStateOf( value: false) }
    val openDialog = remember { mutableStateOf( value: false) }
    var errorMessage by remember { mutableStateOf( value: "Hubo un error al iniciar sesión") }

    fun validarEmail() {
        emailError = !isValidEmail(email)
    }

    fun validarPass() {
        passwordError = password.isEmpty()
    }
```

Finalmente, en nuestro componente “Login” tenemos toda la lógica para iniciar sesión tanto con email/password y con Google.

Primero declaramos todas las variables que vamos a utilizar.

Luego declaramos las funcionas de validación.

```
Column(
    modifier = Modifier.fillMaxSize(), verticalArrangement = Arrangement.Center
) { this: ColumnScope
    Image(
        painter = painterResource(id = R.drawable.logo_title_black),
        contentDescription = "Nurse control logo",
        contentScale = ContentScale.FillWidth,
        modifier = Modifier.padding(horizontal = 10.dp)
    )
    Text(
        text = "Login",
        textAlign = TextAlign.Center,
        fontSize = 32.sp,
        fontWeight = FontWeight.Bold,
        modifier = Modifier
            .fillMaxWidth()
            .padding(top = 10.dp)
    )
```

```
OutlinedTextField(
    value = email,
    onValueChange = { it: String
        email = it
        validarEmail()
    },
    label = { Text( text: "Correo") },
    isError = emailError,
    supportingText = {
        if (emailError) {
            Text( text: "Escriba un correo válido")
        }
    },
    modifier = Modifier
        .fillMaxWidth()
        .padding(horizontal = 20.dp)
        .padding(top = 40.dp),
    keyboardOptions = KeyboardOptions(keyboardType = KeyboardType.Email)
)
```

```
OutlinedTextField(
    value = password,
    onValueChange = { it: String
        password = it
        validarPass()
    },
    label = { Text( text: "Contraseña") },
    isError = passwordError,
    supportingText = {
        if (passwordError) {
            Text( text: "Contraseña es requerida")
        }
    },
    visualTransformation = if (passwordVisible) VisualTransformation.None else PasswordVisualTransformation(),
    keyboardOptions = KeyboardOptions(keyboardType = KeyboardType.Password),
    trailingIcon = {
        val image = if (passwordVisible) R.drawable.visibility
        else R.drawable.visibility_off

        // Localized description for accessibility services
        val description = if (passwordVisible) "Hide password" else "Show password"

        // Toggle button to hide or display password
        IconButton(onClick = { passwordVisible = !passwordVisible }) {
            Image(
                painter = painterResource(id = image),
                contentDescription = description,
            )
        }
    },
    modifier = Modifier
        .fillMaxWidth()
        .padding(horizontal = 20.dp)
        .padding(top = 10.dp)
)
```

```
Button(
    onClick = {
        validarEmail()
        validarPass()

        if (!emailError && !passwordError) {
            FirebaseAuth.getInstance().signInWithEmailAndPassword(email, password)
                .addOnCompleteListener { task ->
                    if (task.isSuccessful) {
                        onLoginSuccess()
                    } else {
                        errorMessage =
                            when (val exception = task.exception) {
                                is FirebaseAuthInvalidCredentialsException -> "Credenciales incorrectar o vencidas."
                                else -> exception?.message
                                    ?: "Hubo un error la iniciar sesión."
                            }
                        openDialog.value = true
                    }
                }
        }
    },
    modifier = Modifier
        .fillMaxWidth()
        .padding(horizontal = 20.dp)
        .padding(top = 20.dp),
) { this: RowScope
    Text( text: "Iniciar Sesión", modifier = Modifier.padding(vertical = 5.dp))
}
```

```
OutlinedButton(
    onClick = onGoogleSignInClick,
    modifier = Modifier
        .fillMaxWidth()
        .padding(horizontal = 20.dp)
        .padding(top = 20.dp),
) { this: RowScope
    Image(
        painter = painterResource(id = R.drawable.google_g_logo),
        contentDescription = "Google logo",
        modifier = Modifier.padding(horizontal = 10.dp)
    )
    Text( text: "Iniciar con Google", modifier = Modifier.padding(vertical = 5.dp))
}

if (openDialog.value) {
    AlertDialog(
        onDismissRequest = {
            // Dismiss the dialog when the user clicks outside the dialog or on the back
            // button. If you want to disable that functionality, simply use an empty
            // onDismissRequest.
            openDialog.value = false
        },
        title = { Text(text = "Error") },
        text = { Text(text = errorMessage) },
        confirmButton = {
            TextButton(onClick = { openDialog.value = false }) { Text( text: "Ok") }
        },
        dismissButton = {
            TextButton(onClick = { openDialog.value = false }) { Text( text: "Cerrar") }
        }
    )
}
```

Luego renderizamos todos los componentes necesarios para el login (textos, imágenes, botones, inputs, etc) usando jetpack compose

Para los inputs usamos el componente OutlinedTextField de jetpack compose, cambiamos el tipo de teclado correspondiente y además para el campo password añadimos la funcionalidad de mostrar y ocultar contraseña.

Conclusiones

La autenticación de usuarios es un componente esencial en el desarrollo de aplicaciones móviles modernas, ya que permite proteger datos sensibles y garantizar una experiencia segura para los usuarios. En este trabajo, se ha explorado y documentado la integración de métodos de autenticación utilizando Firebase en aplicaciones desarrolladas con Kotlin, destacando las alternativas de autenticación mediante correo electrónico y contraseña, así como la autenticación con Google.

Firebase, como plataforma integral, ofrece soluciones robustas, seguras y fáciles de implementar, lo que lo convierte en una herramienta ideal para desarrolladores que buscan simplificar el proceso de autenticación sin comprometer la seguridad. Además, el uso de Kotlin potencia esta integración, gracias a su diseño moderno, conciso y altamente compatible con Android.

El desarrollo de una pantalla de inicio de sesión con múltiples métodos de autenticación demuestra la flexibilidad y efectividad de combinar diversas herramientas para mejorar tanto la seguridad como la experiencia del usuario. Las imágenes, explicaciones paso a paso y análisis del código fuente proporcionan una guía práctica para desarrolladores que deseen implementar funcionalidades similares en sus proyectos.

En conclusión, este trabajo enfatiza la importancia de adoptar tecnologías modernas como Firebase y Kotlin para optimizar el desarrollo de aplicaciones móviles. Al implementar sistemas de autenticación seguros y eficientes, no solo se mejora la confianza de los usuarios, sino que también se sientan las bases para el desarrollo de aplicaciones más robustas, escalables y centradas en las necesidades de las personas.