API Reference

The API reference documentation provides detailed information for each of the the classes and methods in the Firebase SDK. Choose your preferred platform from the list below.

plat_android

Android Full API Reference

plat_ios

iOS Full API Reference

plat_web

Web Full API Reference

plat_cpp

C++ Full API Reference

Other reference documentation

Find detailed information about app configuration settings, or Firebase related tools.

- CLI Reference
- Firebase Cloud Messaging
- HTTP Server Protocol
- XMPP Server Protocol

Last updated November 19, 2016.

Authenticate Using Facebook Login on Android

You can let your users authenticate with Firebase using their Facebook accounts by integrating Facebook Login into your app.

Before you begin

1. Add Firebase to your Android project.
2. If you haven't yet connected your app to your Firebase project, do so from the Firebase console.
3. Add the dependency for Firebase Authentication to your app-level `build.gradle` file:

```
compile 'com.google.firebase:firebase-auth:10.0.1'
```

4. On the Facebook for Developers site, get the **App ID** and an **App Secret** for your app.
5. Enable Facebook Login:

a. In the Firebase console, open the **Auth** section.

b. On the **Sign in method** tab, enable the **Facebook** sign-in method and specify the **App ID** and **App Secret** you got from Facebook.

c. Then, make sure your **OAuth redirect URI** (e.g. `my-app-12345.firebaseapp.com/__/auth/handler`) is listed as one of your **OAuth redirect URIs** in your Facebook app's settings page on the Facebook for Developers site in the **Product Settings > Facebook Login** config.

Authenticate with Firebase

1. Integrate Facebook Login into your app by following the developer's documentation. When you configure the`LoginButton` or `LoginManager` object, request the `public_profile` and `email` permissions. If you integrated Facebook Login using a `LoginButton`, your sign-in activity has code similar to the following:

```
// Initialize Facebook Login button
mCallbackManager = CallbackManager.Factory.create();
LoginButton loginButton = (LoginButton)
findViewById(R.id.button_facebook_login);
loginButton.setReadPermissions("email", "public_profile");
```

```
loginButton.registerCallback(mCallbackManager, new
FacebookCallback<LoginResult>() {
    @Override
    public void onSuccess(LoginResult loginResult) {
        Log.d(TAG, "facebook:onSuccess:" + loginResult);
        handleFacebookAccessToken(loginResult.getAccessToken());
    }

    @Override
    public void onCancel() {
        Log.d(TAG, "facebook:onCancel");
        // ...
    }

    @Override
    public void onError(FacebookException error) {
        Log.d(TAG, "facebook:onError", error);
        // ...
    }
});

// ...

@Override
protected void onActivityResult(int requestCode, int resultCode, Intent
data) {
    super.onActivityResult(requestCode, resultCode, data);

    // Pass the activity result back to the Facebook SDK
    mCallbackManager.onActivityResult(requestCode, resultCode, data);
}
```

FacebookLoginActivity.java

2. In your sign-in activity's `onCreate` method, get the shared instance of the `FirebaseAuth` object:

```
private FirebaseAuth mAuth;
// ...
// Initialize Firebase Auth
mAuth = FirebaseAuth.getInstance();
```

FacebookLoginActivity.java

3. Set up an `AuthStateListener` that responds to changes in the user's sign-in state:

```
private FirebaseAuth.AuthStateListener mAuthListener;

// ...

@Override
protected void onCreate(Bundle savedInstanceState) {
    // ...
    mAuthListener = new FirebaseAuth.AuthStateListener() {
        @Override
        public void onAuthStateChanged(@NonNull FirebaseAuth firebaseAuth)
{
            FirebaseUser user = firebaseAuth.getCurrentUser();
            if (user != null) {
                // User is signed in
                Log.d(TAG, "onAuthStateChanged:signed_in:" +
user.getUid());
            } else {
                // User is signed out
                Log.d(TAG, "onAuthStateChanged:signed_out");
            }
            // ...
        }
    };
    // ...
}

@Override
public void onStart() {
    super.onStart();
    mAuth.addAuthStateListener(mAuthListener);
}

@Override
public void onStop() {
    super.onStop();
    if (mAuthListener != null) {
        mAuth.removeAuthStateListener(mAuthListener);
    }
}
```

FacebookLoginActivity.java

4. After a user successfully signs in, in the `LoginButton`'s `onSuccess` callback method, get an access token for the signed-in user, exchange it for a Firebase credential, and authenticate with Firebase using the Firebase credential:

```
private void handleFacebookAccessToken(AccessToken token) {
    Log.d(TAG, "handleFacebookAccessToken:" + token);

    AuthCredential credential =
FacebookAuthProvider.getCredential(token.getToken());
    mAuth.signInWithCredential(credential)
            .addOnCompleteListener(this, new
OnCompleteListener<AuthResult>() {
                @Override
                public void onComplete(@NonNull Task<AuthResult> task) {
                    Log.d(TAG, "signInWithCredential:onComplete:" +
task.isSuccessful());

                    // If sign in fails, display a message to the user. If
sign in succeeds
                    // the auth state listener will be notified and logic
to handle the
                    // signed in user can be handled in the listener.
                    if (!task.isSuccessful()) {
                        Log.w(TAG, "signInWithCredential",
task.getException());
                        Toast.makeText(FacebookLoginActivity.this,
"Authentication failed.",
                                Toast.LENGTH_SHORT).show();
                    }

                    // ...
                }
            });
}
```

FacebookLoginActivity.java

If the call to `signInWithCredential` succeeds, the `AuthStateListener` runs the `onAuthStateChanged`callback. In the callback, you can use the `getCurrentUser` method to get the user's account data.

Next steps

After a user signs in for the first time, a new user account is created and linked to the credentials—that is, the user name and password, or auth provider information—the user signed in with. This new account is stored as part of your Firebase project, and can be used to identify a user across every app in your project, regardless of how the user signs in.

- In your apps, you can get the user's basic profile information from the `FirebaseUser` object. See Manage Users.
- In your Firebase Realtime Database and Firebase Storage Security Rules, you can get the signed-in user's unique user ID from the `auth` variable, and use it to control what data a user can access.

You can allow users to sign in to your app using multiple authentication providers by linking auth provider credentials to an existing user account.

To sign out a user, call `signOut`:

```
FirebaseAuth.getInstance().signOut();
```

Authenticate Using Google Sign-In on Android

You can let your users authenticate with Firebase using their Google Accounts by integrating Google Sign-In into your app.

Before you begin

1. Add Firebase to your Android project.
2. Add the dependencies for Firebase Authentication and Google Sign-In to your app-level `build.gradle` file:

```
3. compile 'com.google.firebase:firebase-auth:10.0.1'

4. compile 'com.google.android.gms:play-services-auth:10.0.1'
```

5. If you haven't yet connected your app to your Firebase project, do so from the Firebase console.
6. If you haven't yet specified your app's SHA-1 fingerprint, do so from the Settings page of the Firebase console. See Authenticating Your Client for details on how to get your app's SHA-1 fingerprint.
7. Enable Google Sign-In in the Firebase console:

a. In the Firebase console, open the **Auth** section.

b. On the **Sign in method** tab, enable the **Google** sign-in method and click **Save**.

Authenticate with Firebase

1. Integrate Google Sign-In into your app by following the steps on the Integrating Google Sign-In into Your Android App page. When you configure the `GoogleSignInOptions` object, call `requestIdToken`:

```
        // Configure Google Sign In
        GoogleSignInOptions gso = new
GoogleSignInOptions.Builder(GoogleSignInOptions.DEFAULT_SIGN_IN)
                .requestIdToken(getString(R.string.default_web_client_id))
                .requestEmail()
                .build();
```

GoogleSignInActivity.java

You must pass your server's client ID to the `requestIdToken` method. To find the OAuth 2.0 client ID:

a. Open the Credentials page in the API Console.

b. The **Web application** type client ID is your backend server's OAuth 2.0 client ID.

After you integrate Google Sign-In, your sign-in activity has code similar to the following:

```
    private void signIn() {
        Intent signInIntent =
Auth.GoogleSignInApi.getSignInIntent(mGoogleApiClient);
        startActivityForResult(signInIntent, RC_SIGN_IN);
    }

    @Override
    public void onActivityResult(int requestCode, int resultCode, Intent
data) {
        super.onActivityResult(requestCode, resultCode, data);

        // Result returned from launching the Intent from
GoogleSignInApi.getSignInIntent(...);
        if (requestCode == RC_SIGN_IN) {
            GoogleSignInResult result =
Auth.GoogleSignInApi.getSignInResultFromIntent(data);
            if (result.isSuccess()) {
                // Google Sign In was successful, authenticate with
Firebase
                GoogleSignInAccount account = result.getSignInAccount();
                firebaseAuthWithGoogle(account);
            } else {
                // Google Sign In failed, update UI appropriately
                // ...
            }
        }
    }
```

GoogleSignInActivity.java

2. In your sign-in activity's `onCreate` method, get the shared instance of the `FirebaseAuth` object:

```
    private FirebaseAuth mAuth;
// ...
        mAuth = FirebaseAuth.getInstance();
```

GoogleSignInActivity.java

3. Set up an `AuthStateListener` that responds to changes in the user's sign-in state:

```
    private FirebaseAuth.AuthStateListener mAuthListener;

// ...

@Override
protected void onCreate(Bundle savedInstanceState) {
    // ...
    mAuthListener = new FirebaseAuth.AuthStateListener() {
        @Override
        public void onAuthStateChanged(@NonNull FirebaseAuth firebaseAuth)
{
            FirebaseUser user = firebaseAuth.getCurrentUser();
            if (user != null) {
                // User is signed in
                Log.d(TAG, "onAuthStateChanged:signed_in:" +
user.getUid());
            } else {
                // User is signed out
                Log.d(TAG, "onAuthStateChanged:signed_out");
            }
            // ...
        }
    };
    // ...
}

    @Override
    public void onStart() {
        super.onStart();
        mAuth.addAuthStateListener(mAuthListener);
    }

    @Override
    public void onStop() {
        super.onStop();
```

```
        if (mAuthListener != null) {
            mAuth.removeAuthStateListener(mAuthListener);
        }
    }
```

GoogleSignInActivity.java

4. After a user successfully signs in, get an ID token from the `GoogleSignInAccount` object, exchange it for a Firebase credential, and authenticate with Firebase using the Firebase credential:

```
    private void firebaseAuthWithGoogle(GoogleSignInAccount acct) {
        Log.d(TAG, "firebaseAuthWithGoogle:" + acct.getId());

        AuthCredential credential =
GoogleAuthProvider.getCredential(acct.getIdToken(), null);
        mAuth.signInWithCredential(credential)
                .addOnCompleteListener(this, new
OnCompleteListener<AuthResult>() {
                    @Override
                    public void onComplete(@NonNull Task<AuthResult> task)
{
                        Log.d(TAG, "signInWithCredential:onComplete:" +
task.isSuccessful());

                        // If sign in fails, display a message to the
user. If sign in succeeds
                        // the auth state listener will be notified and
logic to handle the
                        // signed in user can be handled in the listener.
                        if (!task.isSuccessful()) {
                            Log.w(TAG, "signInWithCredential",
task.getException());
                            Toast.makeText(GoogleSignInActivity.this,
"Authentication failed.",
                                    Toast.LENGTH_SHORT).show();
                        }
                        // ...
                    }
                });
    }
```

GoogleSignInActivity.java

If the call to `signInWithCredential` succeeds, the `AuthStateListener` runs the `onAuthStateChanged`callback. In the callback, you can use the `getCurrentUser` method to get the user's account data.

Next steps

After a user signs in for the first time, a new user account is created and linked to the credentials—that is, the user name and password, or auth provider information—the user signed in with. This new account is stored as part of your Firebase project, and can be used to identify a user across every app in your project, regardless of how the user signs in.

- In your apps, you can get the user's basic profile information from the `FirebaseUser` object. See Manage Users.
- In your Firebase Realtime Database and Firebase Storage Security Rules, you can get the signed-in user's unique user ID from the `auth` variable, and use it to control what data a user can access.

You can allow users to sign in to your app using multiple authentication providers by linking auth provider credentials to an existing user account.

To sign out a user, call `signOut`:

```
FirebaseAuth.getInstance().signOut();
```

Authenticate with Firebase Anonymously on Android

You can use Firebase Authentication to create and use temporary anonymous accounts to authenticate with Firebase. These temporary anonymous accounts can be used to allow users who haven't yet signed up to your app to to work with data protected by security rules. If an anonymous user decides to sign up to your app, you canlink their sign-in credentials to the anonymous account so that they can continue to work with their protected data in future sessions.

Before you begin

1. Add Firebase to your Android project.
2. Add the dependency for Firebase Authentication to your app-level `build.gradle` file:

```
compile 'com.google.firebase:firebase-auth:10.0.1'
```

3. If you haven't yet connected your app to your Firebase project, do so from the Firebase console.
4. Enable anonymous auth:

a. In the Firebase console, open the **Auth** section.

b. On the **Sign-in Methods** page, enable the **Anonymous** sign-in method.

Authenticate with Firebase anonymously

When a signed-out user uses an app feature that requires authentication with Firebase, sign in the user anonymously by completing the following steps:

1. In your activity's `onCreate` method, get the shared instance of the `FirebaseAuth` object:

```
private FirebaseAuth mAuth;
// ...
mAuth = FirebaseAuth.getInstance();
```

AnonymousAuthActivity.java

2. Set up an `AuthStateListener` that responds to changes in the user's sign-in state:

```
private FirebaseAuth.AuthStateListener mAuthListener;

// ...

@Override
protected void onCreate(Bundle savedInstanceState) {
    // ...
    mAuthListener = new FirebaseAuth.AuthStateListener() {
        @Override
        public void onAuthStateChanged(@NonNull FirebaseAuth firebaseAuth)
{
            FirebaseUser user = firebaseAuth.getCurrentUser();
            if (user != null) {
                // User is signed in
                Log.d(TAG, "onAuthStateChanged:signed_in:" +
user.getUid());
            } else {
                // User is signed out
                Log.d(TAG, "onAuthStateChanged:signed_out");
            }
            // ...
        }
    };
    // ...
}

@Override
public void onStart() {
    super.onStart();
    mAuth.addAuthStateListener(mAuthListener);
}

@Override
public void onStop() {
    super.onStop();
    if (mAuthListener != null) {
        mAuth.removeAuthStateListener(mAuthListener);
    }
}
```

AnonymousAuthActivity.java

3. Finally, call `signInAnonymously` to sign in as an anonymous user:

```
mAuth.signInAnonymously()
        .addOnCompleteListener(this, new OnCompleteListener<AuthResult>()
{
            @Override
            public void onComplete(@NonNull Task<AuthResult> task) {
                Log.d(TAG, "signInAnonymously:onComplete:" +
task.isSuccessful());

                // If sign in fails, display a message to the user. If
sign in succeeds
                // the auth state listener will be notified and logic to
handle the
                // signed in user can be handled in the listener.
                if (!task.isSuccessful()) {
                    Log.w(TAG, "signInAnonymously", task.getException());
                    Toast.makeText(AnonymousAuthActivity.this,
"Authentication failed.",
                            Toast.LENGTH_SHORT).show();
                }

                // ...
            }
        });
```

AnonymousAuthActivity.java

If sign-in succeeds, the `AuthStateListener` runs the `onAuthStateChanged` callback, in which you can use the `getCurrentUser` method to get the user's account data.

Convert an anonymous account to a permanent account

When an anonymous user signs up to your app, you might want to allow them to continue their work with their new account—for example, you might want to make the items the user added to their shopping cart before they signed up available in their new account's shopping cart. To do so, complete the following steps:

1. When the user signs up, complete the sign-in flow for the user's authentication provider up to, but not including, calling one of the `FirebaseAuth.signInWith` methods. For example, get the user's Google ID token, Facebook access token, or email address and password.
2. Get an `AuthCredential` for the new authentication provider:

Google Sign-In

```
AuthCredential credential =
GoogleAuthProvider.getCredential(googleIdToken, null);
```

Facebook Login

```
AuthCredential credential =
FacebookAuthProvider.getCredential(token.getToken());
```

Email-password sign-in

```
AuthCredential credential =
EmailAuthProvider.getEmailAuthCredential(email, password);
```

3. Pass the `AuthCredential` object to the sign-in user's `linkWithCredential` method:

```
mAuth.getCurrentUser().linkWithCredential(credential)
        .addOnCompleteListener(this, new OnCompleteListener<AuthResult>()
{
            @Override
            public void onComplete(@NonNull Task<AuthResult> task) {
                Log.d(TAG, "linkWithCredential:onComplete:" +
task.isSuccessful());

                // If sign in fails, display a message to the user. If
sign in succeeds
                // the auth state listener will be notified and logic to
handle the
                // signed in user can be handled in the listener.
                if (!task.isSuccessful()) {
                    Toast.makeText(AnonymousAuthActivity.this,
"Authentication failed.",
                            Toast.LENGTH_SHORT).show();
                }

                // ...
            }
        });
```

AnonymousAuthActivity.java

If the call to `linkWithCredential` succeeds, the user's new account can access the anonymous account's Firebase data.

This technique can also be used to link any two accounts.

Next steps

Now that users can authenticate with Firebase, you can control their access to data in your Firebase database using Firebase rules.

Authenticate with Firebase on Android Using a Custom Authentication System

You can integrate Firebase Authentication with a custom authentication system by modifying your authentication server to produce custom signed tokens when a user successfully signs in. Your app receives this token and uses it to authenticate with Firebase.

Before you begin

1. Add Firebase to your Android project.
2. Add the dependency for Firebase Authentication to your app-level `build.gradle` file:

```
compile 'com.google.firebase:firebase-auth:10.0.1'
```

3. Get your project's server keys:

a. Go to the Service Accounts page in your project's settings.

b. Click *Generate New Private Key* at the bottom of the *Firebase Admin SDK* section of the *Service Accounts* page.

c. The new service account's public/private key pair is automatically saved on your computer. Copy this file to your authentication server.

Authenticate with Firebase

1. In your sign-in activity's `onCreate` method, get the shared instance of the `FirebaseAuth` object:

```
private FirebaseAuth mAuth;
// ...
mAuth = FirebaseAuth.getInstance();
```

CustomAuthActivity.java

2. Set up an `AuthStateListener` that responds to changes in the user's sign-in state:

```
private FirebaseAuth.AuthStateListener mAuthListener;

// ...
```

```java
@Override
protected void onCreate(Bundle savedInstanceState) {
    // ...
    mAuthListener = new FirebaseAuth.AuthStateListener() {
        @Override
        public void onAuthStateChanged(@NonNull FirebaseAuth firebaseAuth)
{
            FirebaseUser user = firebaseAuth.getCurrentUser();
            if (user != null) {
                // User is signed in
                Log.d(TAG, "onAuthStateChanged:signed_in:" +
user.getUid());
            } else {
                // User is signed out
                Log.d(TAG, "onAuthStateChanged:signed_out");
            }
            // ...
        }
    };
    // ...
}

@Override
public void onStart() {
    super.onStart();
    mAuth.addAuthStateListener(mAuthListener);
    // ...
}

@Override
public void onStop() {
    super.onStop();
    if (mAuthListener != null) {
        mAuth.removeAuthStateListener(mAuthListener);
    }
    // ...
}
```

CustomAuthActivity.java

3. When users sign in to your app, send their sign-in credentials (for example, their username and password) to your authentication server. Your server checks the credentials and returns a custom token if they are valid.
4. After you receive the custom token from your authentication server, pass it to `signInWithCustomToken` to sign in the user:

```
mAuth.signInWithCustomToken(mCustomToken)
        .addOnCompleteListener(this, new OnCompleteListener<AuthResult>()
{
            @Override
            public void onComplete(@NonNull Task<AuthResult> task) {
                Log.d(TAG, "signInWithCustomToken:onComplete:" +
task.isSuccessful());

                // If sign in fails, display a message to the user. If
sign in succeeds
                // the auth state listener will be notified and logic to
handle the
                // signed in user can be handled in the listener.
                if (!task.isSuccessful()) {
                    Log.w(TAG, "signInWithCustomToken",
task.getException());
                    Toast.makeText(CustomAuthActivity.this,
"Authentication failed.",
                            Toast.LENGTH_SHORT).show();
                }
            }
        });
```

CustomAuthActivity.java

If sign-in succeeds, the `AuthStateListener` runs the `onAuthStateChanged` callback, in which you can use the `getCurrentUser` method to get the user's account data.

Next steps

After a user signs in for the first time, a new user account is created and linked to the credentials—that is, the user name and password, or auth provider information—the user signed in with. This new account is stored as part of your Firebase project, and can be used to identify a user across every app in your project, regardless of how the user signs in.

- In your apps, you can get the user's basic profile information from the `FirebaseUser` object. See Manage Users.
- In your Firebase Realtime Database and Firebase Storage Security Rules, you can get the signed-in user's unique user ID from the `auth` variable, and use it to control what data a user can access.

  You can allow users to sign in to your app using multiple authentication providers by linking auth provider credentials to an existing user account.

  To sign out a user, call `signOut`:

```
FirebaseAuth.getInstance().signOut();
```

Authenticate with Firebase using Password-Based Accounts on Android

You can use Firebase Authentication to let your users authenticate with Firebase using their email addresses and passwords, and to manage your app's password-based accounts.

Before you begin

1. Add Firebase to your Android project.
2. Add the dependency for Firebase Authentication to your app-level `build.gradle` file:

```
compile 'com.google.firebase:firebase-auth:10.0.1'
```

3. If you haven't yet connected your app to your Firebase project, do so from the Firebase console.
4. Enable Email/Password sign-in:

a. In the Firebase console, open the **Auth** section.

b. On the **Sign in method** tab, enable the **Email/password** sign-in method and click **Save**.

Create a password-based account

To create a new user account with a password, complete the following steps in your app's sign-in activity:

1. In your sign-up activity's `onCreate` method, get the shared instance of the `FirebaseAuth` object:

```
private FirebaseAuth mAuth;
// ...
mAuth = FirebaseAuth.getInstance();
```

EmailPasswordActivity.java

2. Set up an `AuthStateListener` that responds to changes in the user's sign-in state:

```
private FirebaseAuth.AuthStateListener mAuthListener;

// ...

@Override
protected void onCreate(Bundle savedInstanceState) {
    // ...
    mAuthListener = new FirebaseAuth.AuthStateListener() {
        @Override
        public void onAuthStateChanged(@NonNull FirebaseAuth firebaseAuth)
{
            FirebaseUser user = firebaseAuth.getCurrentUser();
            if (user != null) {
                // User is signed in
                Log.d(TAG, "onAuthStateChanged:signed_in:" +
user.getUid());
            } else {
                // User is signed out
                Log.d(TAG, "onAuthStateChanged:signed_out");
            }
            // ...
        }
    };
    // ...
}

@Override
public void onStart() {
    super.onStart();
    mAuth.addAuthStateListener(mAuthListener);
}

@Override
public void onStop() {
    super.onStop();
    if (mAuthListener != null) {
        mAuth.removeAuthStateListener(mAuthListener);
    }
}
```

EmailPasswordActivity.java

3. When a new user signs up using your app's sign-up form, complete any new account validation steps that your app requires, such as verifying that the new account's password was correctly typed and meets your complexity requirements.
4. Create a new account by passing the new user's email address and password to `createUserWithEmailAndPassword`:

```
mAuth.createUserWithEmailAndPassword(email, password)
        .addOnCompleteListener(this, new OnCompleteListener<AuthResult>()
{
            @Override
            public void onComplete(@NonNull Task<AuthResult> task) {
                Log.d(TAG, "createUserWithEmail:onComplete:" +
task.isSuccessful());

                // If sign in fails, display a message to the user. If
sign in succeeds
                // the auth state listener will be notified and logic to
handle the
                // signed in user can be handled in the listener.
                if (!task.isSuccessful()) {
                    Toast.makeText(EmailPasswordActivity.this,
R.string.auth_failed,
                            Toast.LENGTH_SHORT).show();
                }

                // ...
            }
        });
```

EmailPasswordActivity.java

If the new account was created, the user is also signed in, and the `AuthStateListener` runs the `onAuthStateChanged` callback. In the callback, you can use the `getCurrentUser` method to get the user's account data.

Sign in a user with an email address and password

The steps for signing in a user with a password are similar to the steps for creating a new account. In your app's sign-in activity, do the following:

1. In your sign-in activity's `onCreate` method, get the shared instance of the `FirebaseAuth` object:

```
private FirebaseAuth mAuth;
// ...
mAuth = FirebaseAuth.getInstance();
```

EmailPasswordActivity.java

2. Set up an `AuthStateListener` that responds to changes in the user's sign-in state:

```
private FirebaseAuth.AuthStateListener mAuthListener;

// ...

@Override
protected void onCreate(Bundle savedInstanceState) {
    // ...
    mAuthListener = new FirebaseAuth.AuthStateListener() {
        @Override
        public void onAuthStateChanged(@NonNull FirebaseAuth firebaseAuth)
{
            FirebaseUser user = firebaseAuth.getCurrentUser();
            if (user != null) {
                // User is signed in
                Log.d(TAG, "onAuthStateChanged:signed_in:" +
user.getUid());
            } else {
                // User is signed out
                Log.d(TAG, "onAuthStateChanged:signed_out");
            }
            // ...
        }
    };
    // ...
}

@Override
public void onStart() {
    super.onStart();
    mAuth.addAuthStateListener(mAuthListener);
}

@Override
public void onStop() {
    super.onStop();
```

```
    if (mAuthListener != null) {
        mAuth.removeAuthStateListener(mAuthListener);
    }
}
```

EmailPasswordActivity.java

3. When a user signs in to your app, pass the user's email address and password to `signInWithEmailAndPassword`:

```
mAuth.signInWithEmailAndPassword(email, password)
        .addOnCompleteListener(this, new OnCompleteListener<AuthResult>()
{
            @Override
            public void onComplete(@NonNull Task<AuthResult> task) {
                Log.d(TAG, "signInWithEmail:onComplete:" +
task.isSuccessful());

                // If sign in fails, display a message to the user. If
sign in succeeds
                // the auth state listener will be notified and logic to
handle the
                // signed in user can be handled in the listener.
                if (!task.isSuccessful()) {
                    Log.w(TAG, "signInWithEmail:failed",
task.getException());
                    Toast.makeText(EmailPasswordActivity.this,
R.string.auth_failed,
                            Toast.LENGTH_SHORT).show();
                }

                // ...
            }
        });
```

EmailPasswordActivity.java

If sign-in succeeded, the `AuthStateListener` runs the `onAuthStateChanged` callback. In the callback, you can use the `getCurrentUser` method to get the user's account data.

Next steps

After a user signs in for the first time, a new user account is created and linked to the credentials—that is, the user name and password, or auth provider information—the user signed in with. This new account is stored as part of your Firebase project, and can be used to identify a user across every app in your project, regardless of how the user signs in.

- In your apps, you can get the user's basic profile information from the `FirebaseUser` object. See Manage Users.
- In your Firebase Realtime Database and Firebase Storage Security Rules, you can get the signed-in user's unique user ID from the `auth` variable, and use it to control what data a user can access.

  You can allow users to sign in to your app using multiple authentication providers by linking auth provider credentials to an existing user account.

  To sign out a user, call `signOut`:

```
FirebaseAuth.getInstance().signOut();
```

et up Firebase Authentication for Android

Connect your app to Firebase

1. Install the Firebase SDK.
2. In the Firebase console, add your app to your Firebase project.

Add Firebase Authentication to your app

Add the dependency for Authentication to your app-level `build.gradle` file:

```
compile 'com.google.firebase:firebase-auth:10.0.1'
```

To use an authentication provider, you need to enable it in the Firebase console. Go to the Sign-in Method page in the Firebase Authentication section to enable Email/Password sign-in and any other identity providers you want for your app.

Listen for auth state

Declare the `FirebaseAuth` and `AuthStateListener` objects.

```
private FirebaseAuth mAuth;

                                                    EmailPasswordActivity.java

private FirebaseAuth.AuthStateListener mAuthListener;

                                                    AnonymousAuthActivity.java
```

In the `onCreate()` method, initialize the `FirebaseAuth` instance and the `AuthStateListener` method so you can track whenever the user signs in or out.

```
mAuth = FirebaseAuth.getInstance();

                                                    EmailPasswordActivity.java

mAuthListener = new FirebaseAuth.AuthStateListener() {
    @Override
    public void onAuthStateChanged(@NonNull FirebaseAuth firebaseAuth) {
```

```
        FirebaseUser user = firebaseAuth.getCurrentUser();
        if (user != null) {
            // User is signed in
            Log.d(TAG, "onAuthStateChanged:signed_in:" + user.getUid());
        } else {
            // User is signed out
            Log.d(TAG, "onAuthStateChanged:signed_out");
        }
        // ...
    }
};
```

AnonymousAuthActivity.java

Attach the listener to your `FirebaseAuth` instance in the `onStart()` method and remove it on `onStop()`.

```
@Override
public void onStart() {
    super.onStart();
    mAuth.addAuthStateListener(mAuthListener);
}
```

AnonymousAuthActivity.java

```
@Override
public void onStop() {
    super.onStop();
    if (mAuthListener != null) {
        mAuth.removeAuthStateListener(mAuthListener);
    }
}
```

AnonymousAuthActivity.java

Sign up new users

Create a new `createAccount` method which takes in an email address and password, validates them and then creates a new user with the `createUserWithEmailAndPassword` method.

```
mAuth.createUserWithEmailAndPassword(email, password)
        .addOnCompleteListener(this, new OnCompleteListener<AuthResult>()
```

```
{
            @Override
            public void onComplete(@NonNull Task<AuthResult> task) {
                Log.d(TAG, "createUserWithEmail:onComplete:" +
task.isSuccessful());

                // If sign in fails, display a message to the user. If
sign in succeeds
                // the auth state listener will be notified and logic to
handle the
                // signed in user can be handled in the listener.
                if (!task.isSuccessful()) {
                    Toast.makeText(EmailPasswordActivity.this,
R.string.auth_failed,
                            Toast.LENGTH_SHORT).show();
                }

                // ...
            }
        });
```

EmailPasswordActivity.java

Add a form to register new users with their email and password and call this new method when it is submitted. You can see an example in our quickstart sample.

Sign in existing users

Create a new `signIn` method which takes in an email address and password, validates them, and then signs a user in with the `signInWithEmailAndPassword` method.

```
mAuth.signInWithEmailAndPassword(email, password)
        .addOnCompleteListener(this, new OnCompleteListener<AuthResult>()
{
            @Override
            public void onComplete(@NonNull Task<AuthResult> task) {
                Log.d(TAG, "signInWithEmail:onComplete:" +
task.isSuccessful());

                // If sign in fails, display a message to the user. If
sign in succeeds
                // the auth state listener will be notified and logic to
```

```
handle the
                // signed in user can be handled in the listener.
                if (!task.isSuccessful()) {
                    Log.w(TAG, "signInWithEmail:failed",
task.getException());
                    Toast.makeText(EmailPasswordActivity.this,
R.string.auth_failed,
                                   Toast.LENGTH_SHORT).show();
                }

                // ...
            }
        });
```

EmailPasswordActivity.java

Add a form to sign in users with their email and password and call this new method when it is submitted. You can see an example in our quickstart sample.

Access user information

If a user has signed in successfully you can get their account data at any point with the `getCurrentUser`method.

```
FirebaseUser user = FirebaseAuth.getInstance().getCurrentUser();
if (user != null) {
    // Name, email address, and profile photo Url
    String name = user.getDisplayName();
    String email = user.getEmail();
    Uri photoUrl = user.getPhotoUrl();

    // The user's ID, unique to the Firebase project. Do NOT use this
value to
    // authenticate with your backend server, if you have one. Use
    // FirebaseUser.getToken() instead.
    String uid = user.getUid();
}
```

Optional: Configure ProGuard

When using Firebase Authentication in your app along with ProGuard add the following flags to your `proguard-rules.pro` file to ensure that your app works correctly:

```
-keepattributes Signature
-keepattributes *Annotation*
```

Next Steps

Explore the guides on adding other identity and authentication services:

- Google Sign-in
- Facebook Login
- Anonymous Authentication

Link Multiple Auth Providers to an Account on Android

You can allow users to sign in to your app using multiple authentication providers by linking auth provider credentials to an existing user account. Users are identifiable by the same Firebase user ID regardless of the authentication provider they used to sign in. For example, a user who signed in with a password can link a Google account and sign in with either method in the future. Or, an anonymous user can link a Facebook account and then, later, sign in with Facebook to continue using your app.

Before you begin

Add support for two or more authentication providers (possibly including anonymous authentication) to your app.

Link auth provider credentials to a user account

To link auth provider credentials to an existing user account:

1. Sign in the user using any authentication provider or method.
2. Complete the sign-in flow for the new authentication provider up to, but not including, calling one of the `FirebaseAuth.signInWith` methods. For example, get the user's Google ID token, Facebook access token, or email and password.
3. Get a `AuthCredential` for the new authentication provider:

Google Sign-In

```
AuthCredential credential =
GoogleAuthProvider.getCredential(googleIdToken, null);
```

Facebook Login

```
AuthCredential credential =
FacebookAuthProvider.getCredential(token.getToken());
```

Email-password sign-in

```
AuthCredential credential =
EmailAuthProvider.getEmailAuthCredential(email, password);
```

4. Pass the `AuthCredential` object to the signed-in user's `linkWithCredential` method:

```
mAuth.getCurrentUser().linkWithCredential(credential)
        .addOnCompleteListener(this, new OnCompleteListener<AuthResult>()
{
            @Override
            public void onComplete(@NonNull Task<AuthResult> task) {
                Log.d(TAG, "linkWithCredential:onComplete:" +
task.isSuccessful());

                // If sign in fails, display a message to the user. If
sign in succeeds
                // the auth state listener will be notified and logic to
handle the
                // signed in user can be handled in the listener.
                if (!task.isSuccessful()) {
                    Toast.makeText(AnonymousAuthActivity.this,
"Authentication failed.",
                            Toast.LENGTH_SHORT).show();
                }

                // ...
            }
        });
```

AnonymousAuthActivity.java

The call to `linkWithCredential` will fail if the credentials are already linked to another user account. In this situation, you must handle merging the accounts and associated data as appropriate for your app:

```
FirebaseUser prevUser = currentUser;
currentUser = auth.signInWithCredential(credential).await().getUser();
// Merge prevUser and currentUser accounts and data
// ...
```

If the call to `linkWithCredential` succeeds, the user can now sign in using any linked authentication provider and access the same Firebase data.

Unlink an auth provider from a user account

You can unlink an auth provider from an account, so that the user can no longer sign in with that provider.

To unlink an auth provider from a user account, pass the provider ID to the `unlink` method. You can get the provider IDs of the auth providers linked to a user by calling `getProviderData`.

```
FirebaseAuth.getInstance().getCurrentUser().unlink(providerId)
        .addOnCompleteListener(this, new OnCompleteListener<AuthResult>()
{
            @Override
            public void onComplete(@NonNull Task<AuthResult> task) {
                if (!task.isSuccessful()) {
                    // Auth provider unlinked from account
                }
            }
        });
```

Manage Users in Firebase

Create a user

You create a new user in your Firebase project by calling the `createUserWithEmailAndPassword` method or by signing in a user for the first time using a federated identity provider, such as Google Sign-In or Facebook Login.

You can also create new password-authenticated users from the Authentication section of the Firebase console, on the Users page.

Get the currently signed-in user

The recommended way to get the current user is by setting a listener on the Auth object:

```
mAuthListener = new FirebaseAuth.AuthStateListener() {
    @Override
    public void onAuthStateChanged(@NonNull FirebaseAuth firebaseAuth) {
        FirebaseUser user = firebaseAuth.getCurrentUser();
        if (user != null) {
            // User is signed in
            Log.d(TAG, "onAuthStateChanged:signed_in:" + user.getUid());
        } else {
            // User is signed out
            Log.d(TAG, "onAuthStateChanged:signed_out");
        }
        // ...
    }
};
```

By using a listener, you ensure that the Auth object isn't in an intermediate state—such as initialization—when you get the current user.

You can also get the currently signed-in user by calling `getCurrentUser`. If a user isn't signed in, `getCurrentUser` returns null:

```
FirebaseUser user = FirebaseAuth.getInstance().getCurrentUser();
if (user != null) {
    // User is signed in
```

```
} else {
    // No user is signed in
}
```

Note: `getCurrentUser` might also return null because the auth object has not finished initializing. If you use a listener to keep track of the user's sign-in status, you don't need to handle this case.

Get a user's profile

To get a user's profile information, use the accessor methods of an instance of `FirebaseUser`. For example:

```
FirebaseUser user = FirebaseAuth.getInstance().getCurrentUser();
if (user != null) {
    // Name, email address, and profile photo Url
    String name = user.getDisplayName();
    String email = user.getEmail();
    Uri photoUrl = user.getPhotoUrl();

    // The user's ID, unique to the Firebase project. Do NOT use this
value to
    // authenticate with your backend server, if you have one. Use
    // FirebaseUser.getToken() instead.
    String uid = user.getUid();
}
```

Get a user's provider-specific profile information

To get the profile information retrieved from the sign-in providers linked to a user, use the `getProviderData`method. For example:

```
FirebaseUser user = FirebaseAuth.getInstance().getCurrentUser();
if (user != null) {
    for (UserInfo profile : user.getProviderData()) {
        // Id of the provider (ex: google.com)
        String providerId = profile.getProviderId();

        // UID specific to the provider
        String uid = profile.getUid();
```

```
        // Name, email address, and profile photo Url
        String name = profile.getDisplayName();
        String email = profile.getEmail();
        Uri photoUrl = profile.getPhotoUrl();
    };
}
```

Update a user's profile

You can update a user's basic profile information—the user's display name and profile photo URL—with the `updateProfile` method. For example:

```
FirebaseUser user = FirebaseAuth.getInstance().getCurrentUser();

UserProfileChangeRequest profileUpdates = new
UserProfileChangeRequest.Builder()
        .setDisplayName("Jane Q. User")
        .setPhotoUri(Uri.parse("https://example.com/jane-q-
user/profile.jpg"))
        .build();

user.updateProfile(profileUpdates)
        .addOnCompleteListener(new OnCompleteListener<Void>() {
            @Override
            public void onComplete(@NonNull Task<Void> task) {
                if (task.isSuccessful()) {
                    Log.d(TAG, "User profile updated.");
                }
            }
        });
```

Set a user's email address

You can set a user's email address with the `updateEmail` method. For example:

```
FirebaseUser user = FirebaseAuth.getInstance().getCurrentUser();

user.updateEmail("user@example.com")
        .addOnCompleteListener(new OnCompleteListener<Void>() {
```

```
            @Override
            public void onComplete(@NonNull Task<Void> task) {
                if (task.isSuccessful()) {
                    Log.d(TAG, "User email address updated.");
                }
            }
        });
```

Important: To set a user's email address, the user must have signed in recently. See Re-authenticate a user.

Send a user a verification email

You can send an address verification email to a user with the `sendEmailVerification` method. For example:

```
FirebaseUser user = FirebaseAuth.getInstance().getCurrentUser();

user.sendEmailVerification()
        .addOnCompleteListener(new OnCompleteListener<Void>() {
            @Override
            public void onComplete(@NonNull Task<Void> task) {
                if (task.isSuccessful()) {
                    Log.d(TAG, "Email sent.");
                }
            }
        });
```

You can customize the email template that is used in Authentication section of the Firebase console, on the Email Templates page. See Email Templates in Firebase Help Center.

Set a user's password

You can set a user's password with the `updatePassword` method. For example:

```
FirebaseUser user = FirebaseAuth.getInstance().getCurrentUser();
String newPassword = "SOME-SECURE-PASSWORD";
```

```
user.updatePassword(newPassword)
        .addOnCompleteListener(new OnCompleteListener<Void>() {
            @Override
            public void onComplete(@NonNull Task<Void> task) {
                if (task.isSuccessful()) {
                    Log.d(TAG, "User password updated.");
                }
            }
        });
```

Important: To set a user's password, the user must have signed in recently. See Re-authenticate a user.

Send a password reset email

You can send a password reset email to a user with the `sendPasswordResetEmail` method. For example:

```
FirebaseAuth auth = FirebaseAuth.getInstance();
String emailAddress = "user@example.com";

auth.sendPasswordResetEmail(emailAddress)
        .addOnCompleteListener(new OnCompleteListener<Void>() {
            @Override
            public void onComplete(@NonNull Task<Void> task) {
                if (task.isSuccessful()) {
                    Log.d(TAG, "Email sent.");
                }
            }
        });
```

You can customize the email template that is used in Authentication section of the Firebase console, on the Email Templates page. See Email Templates in Firebase Help Center.

You can also send password rest emails from the Firebase console.

Delete a user

You can delete a user account with the `delete` method. For example:

```
FirebaseUser user = FirebaseAuth.getInstance().getCurrentUser();

user.delete()
        .addOnCompleteListener(new OnCompleteListener<Void>() {
            @Override
            public void onComplete(@NonNull Task<Void> task) {
                if (task.isSuccessful()) {
                    Log.d(TAG, "User account deleted.");
                }
            }
        });
```

Important: To delete a user, the user must have signed in recently. See Re-authenticate a user.

You can also delete users from the Authentication section of the Firebase console, on the Users page.

Re-authenticate a user

Some security-sensitive actions—such as deleting an account, setting a primary email address, and changing a password—require that the user has recently signed in. If you perform one of these actions, and the user signed in too long ago, the action fails and throws `FirebaseAuthRecentLoginRequiredException`. When this happens, re-authenticate the user by getting new sign-in credentials from the user and passing the credentials to `reauthenticate`. For example:

```
FirebaseUser user = FirebaseAuth.getInstance().getCurrentUser();

// Get auth credentials from the user for re-authentication. The example
below shows
// email and password credentials but there are multiple possible
providers,
// such as GoogleAuthProvider or FacebookAuthProvider.
AuthCredential credential = EmailAuthProvider
        .getCredential("user@example.com", "password1234");

// Prompt the user to re-provide their sign-in credentials
user.reauthenticate(credential)
        .addOnCompleteListener(new OnCompleteListener<Void>() {
            @Override
            public void onComplete(@NonNull Task<Void> task) {
```

```
                Log.d(TAG, "User re-authenticated.");
            }
        });
```

Import user accounts

You can import user accounts from a file into your Firebase project by using the Firebase CLI's `auth:import` command. For example:

```
firebase auth:import users.json --hash-algo=scrypt --rounds=8 --mem-
cost=14
```

SamplesSamples

Title	Description	Links
Analytics	Quickstart for Analytics	plat_android plat_ios plat_cppvideogame_asset
Realtime Database	Quickstart for Realtime Database	plat_android plat_ios plat_web videogame_a
Authentication	Quickstart for Authentication	plat_android plat_ios plat_webplat_cpp video
Cloud Messaging	Quickstart for Cloud Messaging	plat_android plat_ios plat_cppvideogame_asset
Remote Config	Quickstart for Remote Config	plat_android plat_ios plat_cppvideogame_asset

Title	Description	Links
Storage	Quickstart for Storage	plat_android plat_ios plat_web
Crash Reporting	Quickstart for Crash Reporting	plat_android plat_ios
App Indexing	Quickstart for App Indexing	plat_android
Dynamic Links	Quickstart for Dynamic Links	plat_android plat_ios
Invites	Quickstart for Invites	plat_android plat_ios plat_cppvideogame_asset
AdMob	Quickstart for AdMob	plat_android plat_ios plat_cpp

Multi-Feature Samples

Title	Description	Links
FriendlyPix Analytics, Realtime Database,Crash Reporting, Authentication, Storage	App	plat_android plat_ios
FirePad Realtime Database, Authentication, Hosting	App	plat_web
FireChat Realtime Database, Authentication, Hosting	App	plat_web

Title	Description	Links

Title	Description	Links
Analytics	Quickstart for Analytics	plat_android plat_ios plat_cppvideogame_asset
Realtime Database	Quickstart for Realtime Database	plat_android plat_ios plat_web videogame_a
Authentication	Quickstart for Authentication	plat_android plat_ios plat_webplat_cpp video
Cloud Messaging	Quickstart for Cloud Messaging	plat_android plat_ios plat_cppvideogame_asset
Remote Config	Quickstart for Remote Config	plat_android plat_ios plat_cppvideogame_asset
Storage	Quickstart for Storage	plat_android plat_ios plat_web

Title	Description	Links
Crash Reporting	Quickstart for Crash Reporting	plat_android plat_ios
App Indexing	Quickstart for App Indexing	plat_android
Dynamic Links	Quickstart for Dynamic Links	plat_android plat_ios
Invites	Quickstart for Invites	plat_android plat_ios plat_cppvideogame_asset
AdMob	Quickstart for AdMob	plat_android plat_ios plat_cpp

Multi-Feature Samples

Title	Description	Links
FriendlyPix Analytics, Realtime Database,Crash Reporting, Authentication, Storage	App	plat_android plat_ios
FirePad Realtime Database, Authentication, Hosting	App	plat_web
FireChat Realtime Database, Authentication, Hosting	App	plat_web

Codelabs

Title	Description	Links
FriendlyChat Analytics, Realtime Database, Authentication, Hosting, Storage,	Learn how to use Firebase through building a chat app. Browse the source on GitHub.	plat_ios plat_android

Title	Description	Links
AdMob, Crash Reporting, Cloud Messaging		