Concept clear of firebase

What is Firebase?

- A product of Google
- Helps developers to build apps faster and securely.
- No programming is required on the firebase side which makes it easy to use its features more efficiently.
- Provides services to android, ios, web, and unity. It provides cloud storage.
- Uses NoSQL for the database for the storage of data.

Firebase SDK

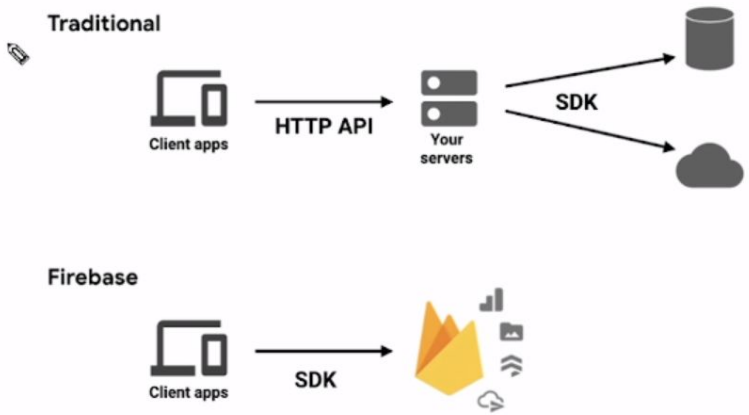
Software Development Kit একটা package -> একসাথে কতগুলো software development tools এর collection -> যা নির্দিষ্ট application তৈরীর কাজকে সহজ করে দেয়। Firebase SDK- এক বা একের অধিক sign-in methods এর সমন্বয় ঘটিয়ে Authentication System কে handle করার কাজ সহজ করে দেয়।

Firebase SDK

- The SDKs provided by Firebase, directly interact with backend services.
- There is no need to establish any connection between the app and the service.
- If you're using one of the Firebase database options, you typically write code to query the database in your client app.

Firebase SDK

- The traditional app development process requires writing both frontend and backend software. The frontend code just implements the API endpoints exposed by the backend, and the backend code actually does the work.
- With Firebase products, the traditional backend is bypassed, putting the work into the client.
- **Serverless**



Firebase Authentication

- Firebase Authentication provides backend services, easy-to-use SDKs, and ready-made UI libraries to authenticate users to your app.
- It supports authentication using passwords, phone numbers, popular federated identity providers like Google, Facebook and Twitter, and more.
- Firebase Authentication integrates tightly with other Firebase services, and it **leverages industry standards like OAuth 2.0**

Firebase Authentication

- OAuth 2.0, which stands for "Open Authorization", is a standard designed to allow a website or application to access resources hosted by other web apps on behalf of a user.
- OAuth 2.0 provides consented access and restricts actions of what the client app can perform on resources on behalf of the user, without ever sharing the user's credentials.

Since Firebase Authentication follows industry standards like **OAuth 2.0,** let's discuss more about OAuth2.0 to know how firebase authentication works behind the scene...

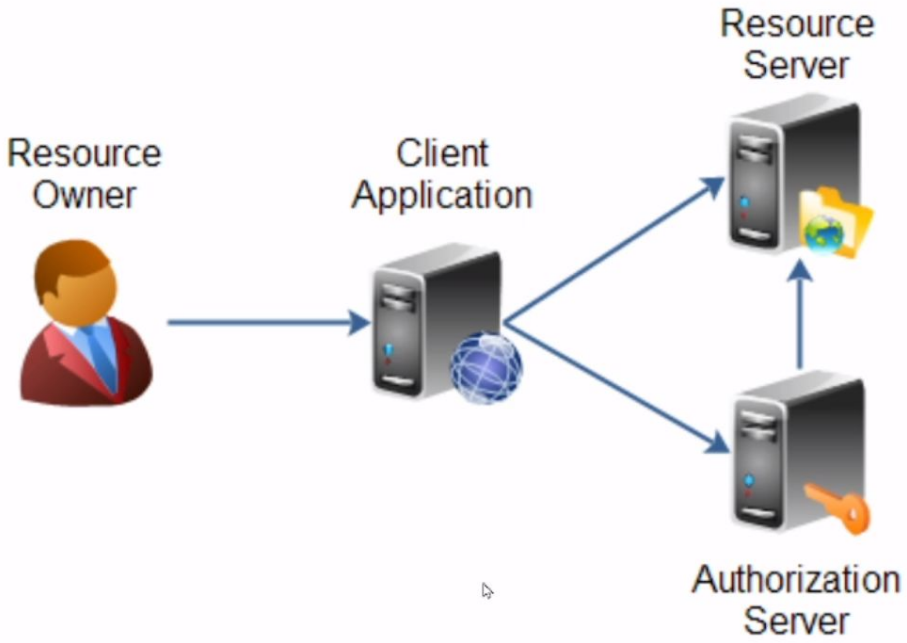
Principles of OAuth 2.0

- OAuth 2.0 is an authorization protocol and NOT an authentication protocol.
- **Authentication** verifies the identity of a user or service, and **authorization** determines their access rights.
- OAuth 2.0 is designed primarily as a means of granting access to a set of resources, for example, remote APIs or user's data.
- OAuth 2.0 uses Access Tokens. An **Access Token** is a piece of data that represents the authorization to access resources on behalf of the end-user.

Principles of OAuth 2.0

- The OAuth 2.0 authorization framework contains **authorization server**
- **Authorization Server** -receives requests from the Client for Access Tokens and issues them upon successful authentication and consent by the **Resource Owner(the user or system that owns the protected resources and can grant access to them.)**. The authorization server exposes two endpoints: the Authorization endpoint, which handles the interactive authentication and consent of the user, and the Token endpoint, which is involved in a machine to machine interaction.

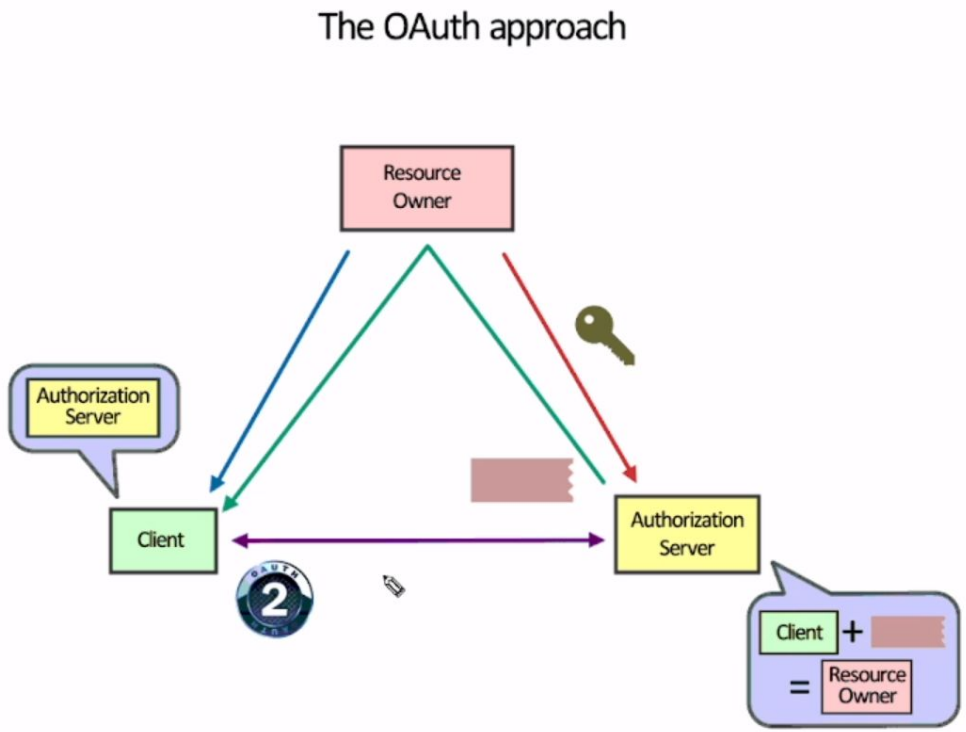
OAuth roles at a glance



How OAuth 2.0 works?

1. The Client requests authorization (*authorization request*) from the Authorization server, supplying the **client id** and **secret** to as identification; it also provides the scopes and an endpoint URI (***redirect URI***) to send the Access Token or the Authorization Code to.
2. The Authorization server authenticates the Client and verifies that the requested scopes are permitted.
3. The Resource owner interacts with the Authorization server to grant access.
4. The Authorization server redirects back to the Client with either an Authorization Code or Access Token, depending on the grant type.
5. With the Access Token, the Client requests access to the resource from the Resour

OAuth 2.0 approach visualization

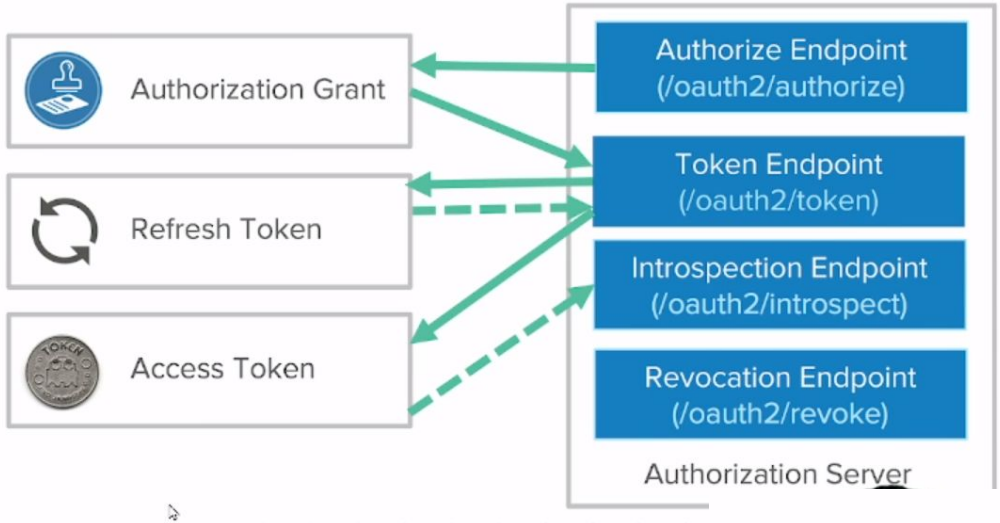


OAuth access tokens and authorization code

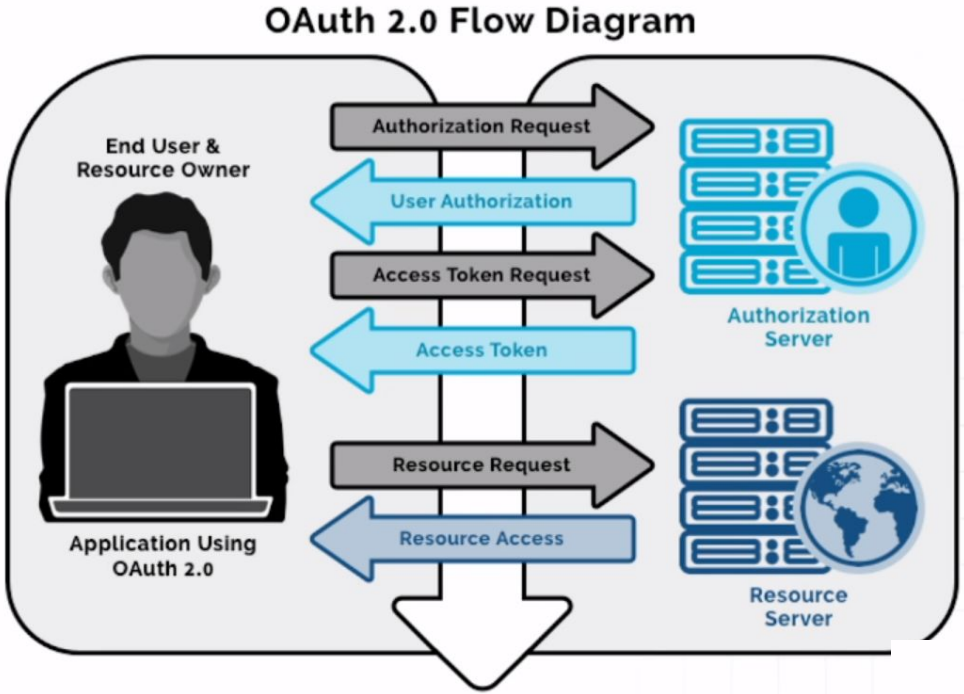
Authorization Code may be returned, which is then exchanged for an Access Token.

The Authorization server may also issue a Refresh Token with the **Access Token**.

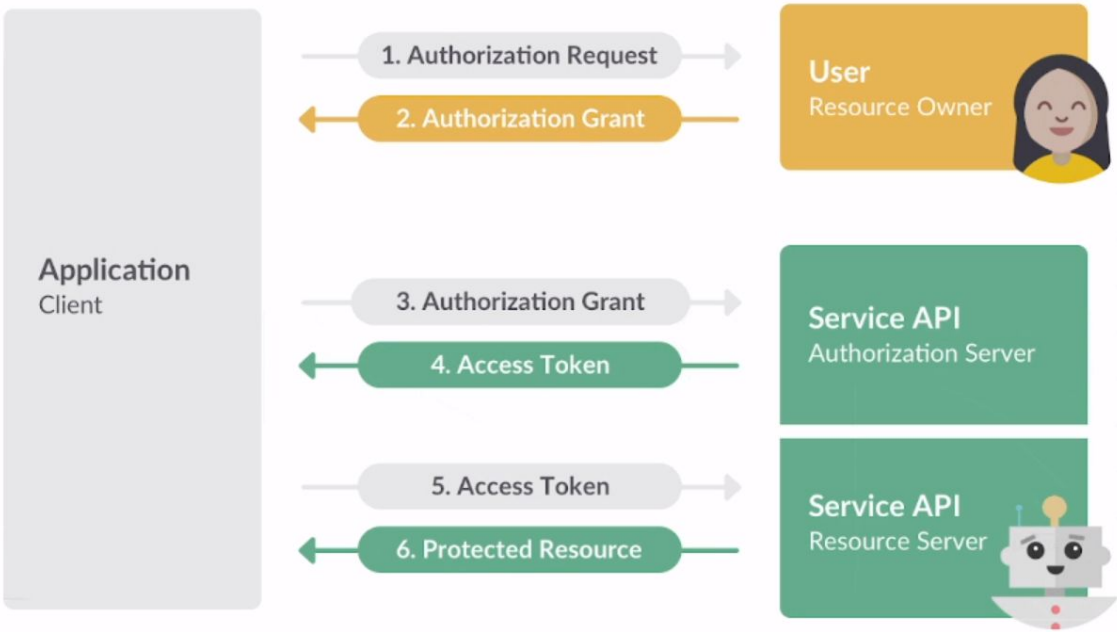
Refresh Tokens normally have long expiry times and may be exchanged for new Access Tokens when the latter expires. Because Refresh Tokens have these properties, they have to be stored securely by clients.



OAuth 2.0 working Flow diagram



OAuth 2.0 working for slack



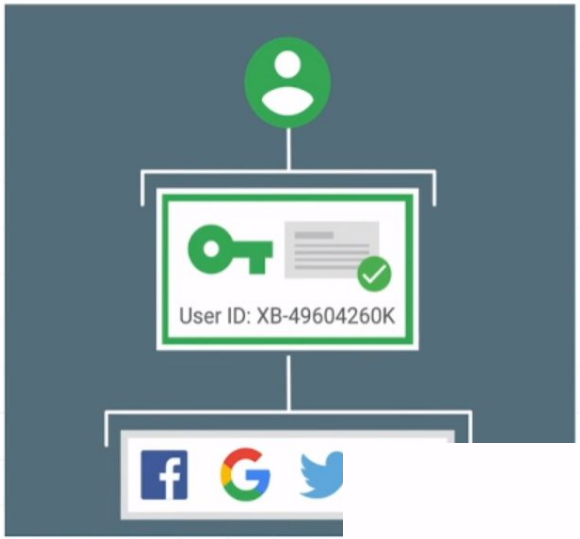
Firebase Authentication

- Once a user authenticates, 3 things happen:

  1) Information about the user is returned to the application via callbacks to allow us to personalize our app's user experience for the specific user

  2) The user information contains a unique ID which is guaranteed to be unique distinct across all providers

  3) This unique ID is used to identify the user and what parts of the backend system they are authorized to access.



Some Firebase Reference

- **initializeApp()**- Creates and initializes a FirebaseApp instance.
- **FirebaseApp** - A FirebaseApp holds the initialization information for a collection of services.
- **getAuth(app)**- Returns the Auth instance associated with the provided FirebaseApp. If no instance exists, initializes an Auth instance with platform-specific default dependencies.
- **OAuthProvider** - Provider for generating generic OAuthCredential.
- **OAuthCredential** - specify the details about each auth provider's credential requirements.

Why use an OAuth Provider?

As you make an app that accesses a solid web based back-end it's important to consider the following aspects of web security:

- Requiring strong passwords.
- The use of strong encryption.
- Ensuring secure communication (between client and server).
- Securing password storage within an encrypted database.
- Implementing password recovery.(Which also has to be secure).
- Adding 2 factor authentication. (Highly recommended extra layer of security)
- Including protection against man in the middle attacks.

Why use an OAuth Provider?

Having said all this even the simplest of applications may become fairly difficult to make when we factor in security.

Thankfully we won't have to worry about this if we are going to use an OAuth provider which will handle this for us. OAuth providers allow your users to login into your application using credentials from a trusted website such as Facebook, Github, Google+or even Twitter.

Send a div as children inside another component

```
<div className="countries-container">
                {
                    countries.map(country => <Country
                        key={country.capital}
                        country={country}
                    >
                        <div>
                            <h2>Sent as children inside prop</h2>
                        </div>
                    </Country>)
                }
</div>
```

```
const Country = (props) => {
    const { name, flags, region, population } = props.country;
    // console.log(props?.country?.capital[0]);
    console.log(props);
    return (
        <div className="country">
            <h4 >This is: {name.common}</h4>
            <img src={flags.png} alt="" />
            <p><small>Region: {region}</small></p>
            <p>Capital is: {props.country?.capital} </p>
            <p>population: {population}</p>
            <small>{props.children}</small>
        </div>
    );
};
```

Get div as children inside child component