**Sign up using Phone Number**

In today’s world, a user interacts with various apps/websites which require them to login or sign up via using their email address. There are a high chances of a person forgetting the username and the associated password to login or just completely giving up the sign up process as it asks a lot of information. Furthermore, from the app developer point of view, they will need to perform extra steps to verify the user’s email address.

This leads to another solution of using phone number as a way to login to the service. It is easy to remember the phone number and we can implement a mechanism to verify that the user is in possession of this number via text messages. In order to sign up with phone number the user Birth Date, Name, Gender ,Email this all need to be collected first.

* Birth Date: The user Date Of Birth must be collected that should be in dd/mm/yyyy format.
* Name: The complete name of the user is collected.
* Family Name: The complete Family details is necessary and that to be stored.
* Gender: The user gender to be collected as male, Female.
* Email: The user’s associated email address to be collected.

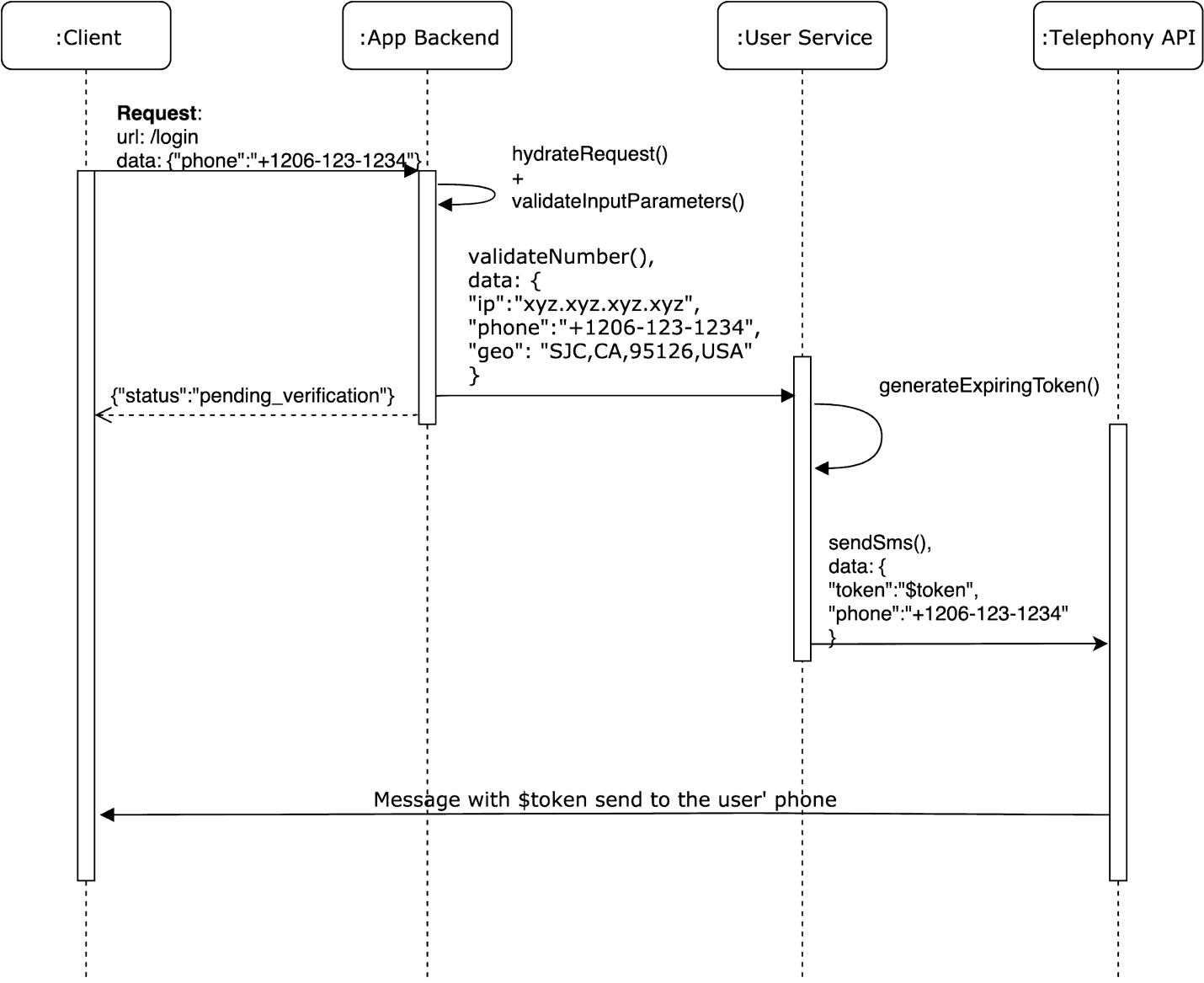
**sign up process is divided into two parts:**

1. Login Flow : In this phase, user submits the phone number to app backend and receives a token.
2. Verify Flow : In this phase, user submits the received token in the previous phase and sends to backend. On validation, user is logged in and a session cookie is issued.
3. **Login Flow:** In order to authenticate the user account, user submits the phone number to the App Backend . On submission, user is shown a screen where a token can be submitted. App backend validates the input parameter and enrichies/ hydrates the request with extra information like (IP address, geo information, device information etc). All this information can be used to generate a fingerprint of a user request which can be used for later security measures.This enriched and validated request is submitted to user service which generates a token and associate with these request. Token generation is an important process here.

**Token Generation**

The aim of the token is to make sure we can

* Expire the token so that same token cannot be used multiple times.
* Invalidate a token so that if a user enters the phone number multiple times then we can invalidate the previously generated token issue a new one.
* as the key and a randomly generated 6 digit token as the value which expires in 5 minutes.{“Key” : “+1206-123-1234”, “Value”:”123456”}



**Send SMS:** User service then calls a telephony API to send this token to the user’s phone number. There are various companies which provides this service (ex: Twilio, Plivo etc).

1. **Verify Phone Number Flow**

user enters the token on the form. This form consists of a hidden phone number field which is submitted to an endpoint (for ex: /verify\_phone) with the the token. Once again the app backend hydrates the request and validates the input parameters. The user service does a look up with the phone number as the key from the memcache data store. A session cookie is issued If the token exists and matches with the input token. This session cookie is sent as response header to the client which can persist it for later requests.

* **Sign up vs Login:** On successful token match, user service should do a look up in the user database to see if there is an already existing account with the same phone number, if that phone number is present then the user should be logged into that account. Another issue is to consider what happens if a user changes the phone number. This involves implementing a similar mechanism of verifying the new phone number and updating the user account if that is successful.
* **Advantage:** This definitely removes the mental load on the user to remember the email and passwords across various services. This is the best way.

**Signup using email**

To sign up in any site we get more options like we can login with mail id, Phone number, Google, Face book, Amazon. I worked on Email.

Email was actually invented before the publicly accessible internet as we know it. An email address is unique identifier for an email account. It is used to both send and receive email messages over the internet. Similar to physical mail, an email message requires an address for both the sender and recipient in order to be sent successfully. Every email address has two main parts: a username and domain name. The username comes first, followed by an at (@) symbol, followed by the domain name.

When a message is sent SMTP the sending mail server checks for another mail server on the internet that corresponds with the domain name of the recipient address. If someone sends a message to user, it will check with the mail server to see if the username is valid. If the user exists, the message will be delivered.

While a basic email address consists of only a username and domain name, most email clients and webmail system include with names with email address. An email address contains a name is formatted with the first name, followed by the email address. While creating an email, it consists of Name, Birth date, Family name, Gender, Phone number and User name. Whenever we are signing up with email we have to enter all the details.

Birth Date: The user date of birth must be collected that should be in dd/mm/yyyy format.

Family Name: The complete family details is necessary and that to be stored.

Name: The complete name of the user is collected.

Gender: The user gender to be collected as male, female.

Phone number: The user’s phone number to be collected.

Email is based on these tools:

Qmail (http://www.qmail.org/), a robust mail transfer agent written by Dan Bernstein. It was designed to be simple, modular, secure, and reliable. It is easy to understand, and easy to integrate with other programs.

Oracle RDBMS (http://www.oracle.com).

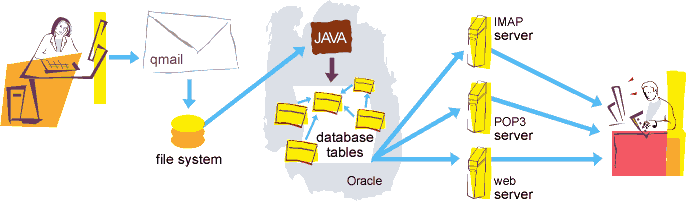
JavaSoft's JavaMail (http://java.sun.com/products/javamail/) and Oracle's SQLJ (http://www.oracle.com/java/index.html?jdbc\_sqlj.html) APIs, running within Oracle.

A presentation layer implemented in Tcl using AOLserver.

Qmail handles the traditional duties of a mail transfer agent: receipt of incoming email via SMTP, bouncing of incorrectly addressed or formatted messages, notification of errors during reception, and reliable delivery of the email into the file system. Aside from being configured in a somewhat idiosyncratic fashion, qmail is used without modification to do what it was designed to do. Non-interference with qmail means we can trust it to handle its MTA duties (to the extent that it is reliable).

After delivery to the file system, the message must be moved into the database as rows in tables. This job falls to Java code running in the database, relying on the JavaMail and SQLJ APIs to parse and insert the message. The headers are parsed and stored separately to facilitate searching and filtering, and any MIME attachments are decoded and stored separately. Once this processing is complete, the message is removed from the file system. This polling process is triggered once a minute by the DBMS\_JOB PL/SQL package.

Once the message is in the database, it is a simple matter to serve it up again through an IMAP server, or present it through Web pages produced by Tcl scripts running within AOL server. In the latter case, the flexibility of SQL make it easy to implement user interfaces that allow readers to view and manipulate their email in many powerful ways.



**Design Goals**

In writing email, I had the following goals:

Avoid writing code unnecessarily: Mail transfer agents are responsible for the reliable sending, receipt, and delivery of email in the face of network outages, full disks, and hardware failures. This is a solved problem; other people have solved it and solved it well, with MTAs that go to heroic lengths to preserve email. Similarly, parsing RFC822 and MIME messages correctly is something other people have written libraries for.

Make it easy for a human to handle large amounts of email:I receive more mail than most people. Much of it is from automated email alerts and monitors. I don't want to stop receiving them, but it is rare that I actually care to read them. Most web-based email interfaces seem to be designed with the notion that you actually care to read the email you receive. Webmail was designed to facilitate dealing with email in bulk.

New Signup using Facebook

To signup in any site we get more options like we can login with mail id, phone number, Google, face book, Amazon and so on. In that I worked on Face book Login.

The Facebook SDK for Android enables people to sign into our app with Face book Login. When people log into our app with Face book they can grant permissions to our app so we can retrieve information or perform actions on Face book on their behalf.

To do it we need to know about some important access controls provided as below.

* Go to Face book developer Portal

We need to go to the Face book developer site i.e. <https://developers.facebook.com/>. Here we get the information for the steps that we need to follow to create SDK.

To create any SDK we need access the tokens from the site those are as follows:

An access token is an opaque string that identifies a user, app, or Page and can be used by the app to make graph API calls. When someone connects with an app using Face book Login and approves the request for permissions, the app obtains an access token that provides temporary, secure access to Face book APIs. Access tokens are obtained via a number of methods.

The token includes information about when the token will expire and which app generated the token. Because of privacy checks, the majority of API calls on Face book need to include an access token. There are different types of access tokens to support different use cases:

* [User Access Token](https://developers.facebook.com/docs/facebook-login/access-tokens#usertokens): This kind of access token is needed any time the app calls an API to read, modify or write a specific person's Facebook data on their behalf. User access tokens are generally obtained via a login dialog and require a person to permit your app to obtain one.
* [App Access Token](https://developers.facebook.com/docs/facebook-login/access-tokens#apptokens): This kind of access token is needed to modify and read app settings. It can also be used to publish Open Graph actions. It is generated using a pre-agreed secret between the app and Face book and is then used during calls that change app-wide settings. You obtain an app access token via a server-to-server call.
* [Page Access Token](https://developers.facebook.com/docs/facebook-login/access-tokens#pagetokens): This kind of access token is similar to user access tokens, except that they provide permission to APIs that read, write or modify the data belonging to a Facebook Page. To obtain a page access token you need to start by obtaining a user access token and asking for the Page permission or permissions you need. Once you have the user access token you then get the page access token via the Graph API.
* [Client Token](https://developers.facebook.com/docs/facebook-login/access-tokens#clienttokens): The client token is an identifier that you can embed into native mobile binaries or desktop apps to identify your app. The client token isn't meant to be a secret identifier because it's embedded in apps. The client token is used to access app-level APIs, but only a very limited subset. The client token is found in your app's dashboard. Since the client token is used rarely, we won't talk about it in this document. Instead it's covered in any API documentation that uses the client token.

In these we are discussing about user access token and App access token in detail.

* User Access Token

Although each platform generates access tokens through different APIs, all platforms follow the basic strategy to get a user token:

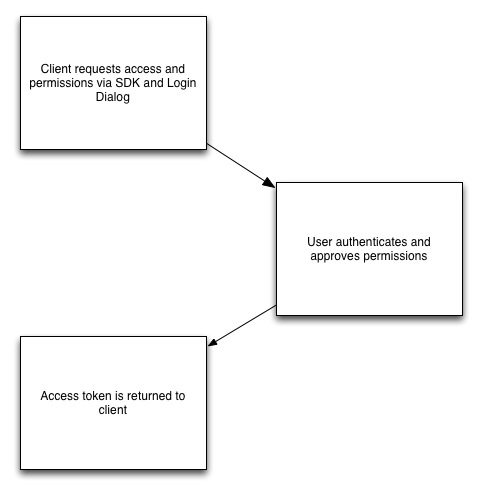


Fig 1: User Access Token

User access tokens come in two forms: short-lived tokens and long-lived tokens. Short-lived tokens usually have a lifetime of about an hour or two, while long-lived tokens usually have a lifetime of about 60 days.

One important aspect to understand about access tokens is that they are portable. Once we have an access token we can use it to make calls from a mobile client, a web browser, or from your server to Face Book's servers. If a token is obtained on a client, we can ship that token down to our server and use it in server-to-server calls. If a token is obtained via a server call, we can also ship that token up to a client and then make the calls from the client.

Different platforms have different methods to kick off this process.

* App access Tokens

App access tokens are used to make requests to Face book APIs on behalf of an app rather than a user. This can be used to modify the parameters of our app, create and manage test users, or read our app's insights.

Some user data that would normally be visible to an app making a request with a user access token isn't always visible with an app access token. If we're reading user data and using it in our app, we should use a user access token instead of an app access token.

Generating an App access token – To generate an app access token we need:

1. Our App ID

Before you can implement any of our products and SDKs or access any of our APIs, you must first register as a Face book developer and use our App Dashboard to provide information about your app. These documents explain how to register as a developer, how to use the App Dashboard to configure your app's settings, and how to build, test, and release your app. To do it we need to follow steps as below:

1. Register: Register as a Face book developer to gain access to Face book app development tools.
2. Create an App: Use the App Dashboard to create an app and access app and account settings.
3. Build and Test: Tools and information to help us with the app development process.
4. Release: How to make our app available to Face book users.
5. Maintaining Data Access: How to avoid losing access to Face book products, APIs, and SDKs.
6. Terms and Policies: Terms and Policies that we must agree to.
7. Support: Face book developer support resources and how to access them.
8. Our App Secret

The App Secret is used in some of the Login flows to generate access tokens and the Secret itself is intended to secure usage of our App to only those that are trusted. The secret can be used to easily create an App Access Token which can make API requests on behalf of any user of the app, which makes it extremely important that an App Secret is not compromised.

Therefore the App Secret or an App Access token should never be included in any code that could be accessed by anyone other than a developer of the app. This applies to all methods of code that are not secured like client-side code (such as HTML or JavaScript) or native apps (such as iOS, Android or Windows desktop apps) that could be decompiled.

We recommend that App Access Tokens should only be used directly from our app's servers in order to provide the best security. For native apps, Face Book suggest that the app communicates with our own server and the server then makes the API requests to Face book using the App Access Token. For this reason, if our 'App Type' under [Advanced Settings in the App Dashboard](https://developers.facebook.com/apps/) is set to Native/Desktop Face book assume that our native app contains the App Secret or an App Access Token in the binary, and Face book do not allow calls signed with an App Access Token to proceed. The API will behave as though no access token was provided.

If our App Secret is compromised, we should reset it immediately in the [Basic Settings of your App Dashboard](https://developers.facebook.com/apps/). When we start the reset process, we can specify a number of hours that the compromised secret will continue to work for when making requests, however anything sent from Face book (such as signed requests) will use the new secret straight away, so we must adjust our code to expect it as soon as possible.

* Permissions:

When a person logs into your app via Face book Login you can access a subset of that person's data stored on Face book. Permissions are how you ask someone if you can access that data. A person's privacy settings combined with what you ask for will determine what you can access.

Your app has requested a person's email address and the things they like but that request also automatically asks for access to a person's public profile. Your app can ask for additional permissions at any time, even after a person logs in for the first time. The additional permissions like ads\_management, ads\_read, business\_namangement, groups\_access\_member, Instagram\_basics, pages\_events, user\_birthday, user\_age\_range, user\_friends, user\_gender, user\_hometown, user\_likes, user\_link, user\_location, user\_photos, user\_paste, user\_vidios etc.

Face book Login allows **a person to grant only a subset of permissions** that we ask for to our app, except for public profile, which is always required. This is available as a separate screen in the login dialog when we ask for permissions. Our app should handle the case where someone had declined to grant our app one of the permissions we requested.

Once the development is over our app undergoes App Review. App Review is part of [app development](https://developers.facebook.com/docs/apps) that enables us to verify that our app uses Face Book Products and APIs in an approved manner. If our app will be used by anyone without a [Role on the app](https://developers.facebook.com/docs/apps#roles) or a [role in a Business](https://www.facebook.com/business/help/442345745885606?id=180505742745347) that has claimed the app, it must first undergo App Review. So that by doing this Face Book can manage the access of user details in the Face Book profile and restrict the unnecessary access, give permissions to only that app needs to function. It conforms that the app uses the data in intended ways and safeguards user privacy.

**Google Login Configuration**

The hassle of managing different accounts for different sites specifically, when we have several passwords for different services and a website asks us to create yet another account on their site.

To handle with this, we could offer a single sign-on feature to allow visitors to use their existing credentials to open an account on our site. In this users can login by using their existing Google account. This is a more convenient way for new users to register with a third-party site instead of signing up for a new account with yet another username and password.

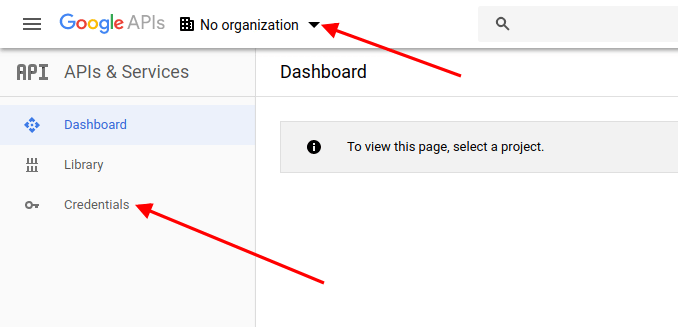
This document describes how to complete a basic Google Login configuration.

1. Create authorization credentials
2. Load the Google platform library
3. Specify our app’s client ID
4. Add a Google Login button
5. Get profile information

**We will go through the basic setup which is required to configure Google Login with our web application.**

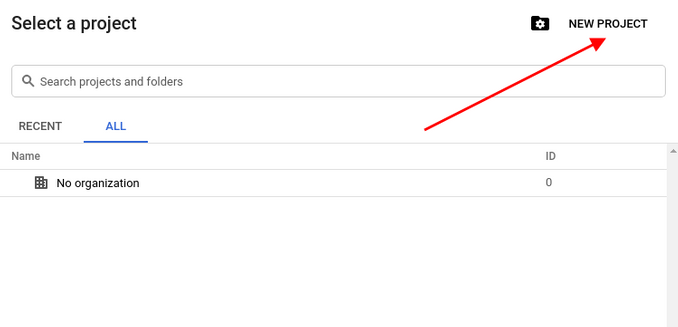
Firstly, we need to create an application with Google which will allow us to register our site with Google. It allows us to set up basic information about our website and a couple of technical details as well.

Once we were logged in with Google, open the **Google Developers Console** (<https://console.cloud.google.com>). That should open up the **Google Dashboard page**.

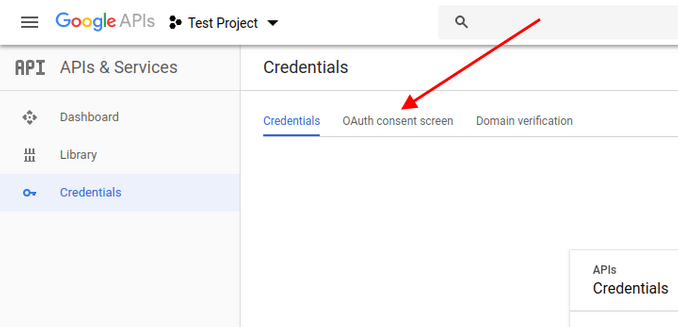


In that page from the top-left menu, click on the **Select a project link**. That should open up a popup.

Next we have to click on the **New Project** link and it will ask us to enter the **Project Name** and other details.

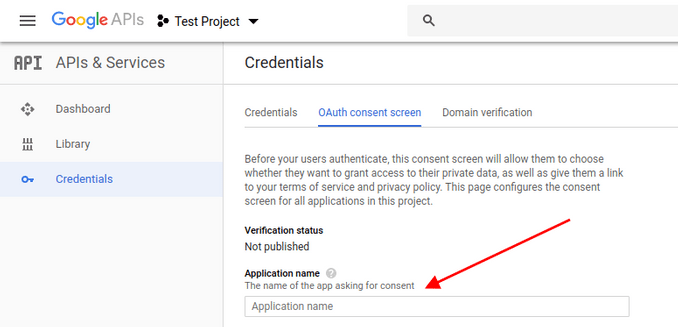


Click on the **Create** button to save our new project. We will be redirected to the Dashboard page. Click on the **Credentials** from the left sidebar, and go to the **OAuth consent screen** tab.

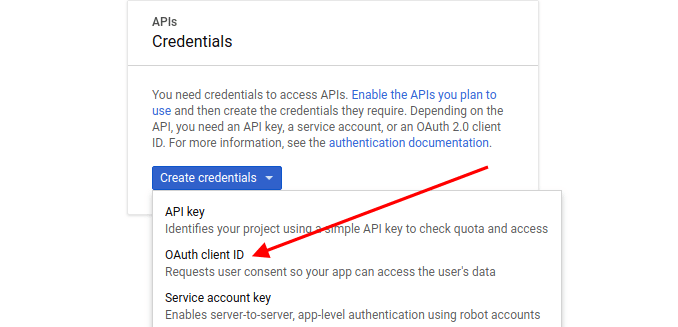


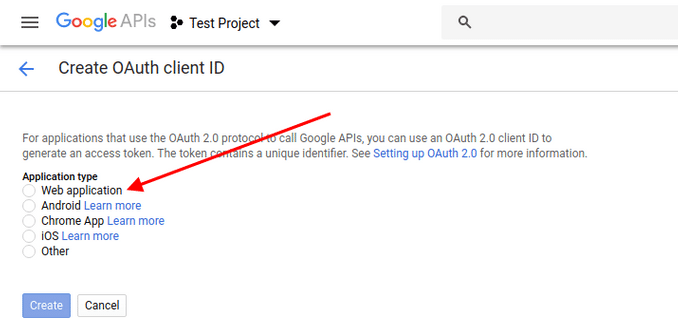
On this tab we need to enter the details about our application, like the application name and few other details. Fill in the necessary details and save them.

Next, click on Credentials box under the Credentials tab.



Click **Client credentials>OAuth client ID** to create a new set of credentials for our application. That should present us with a screen that asks us to choose an appropriate application type. It is because a client ID is used to identify a single app to Google’s OAuth servers. If our app runs on multiple platforms, each will need its own client ID. See **Settings up OAuth 2.0** for more information.



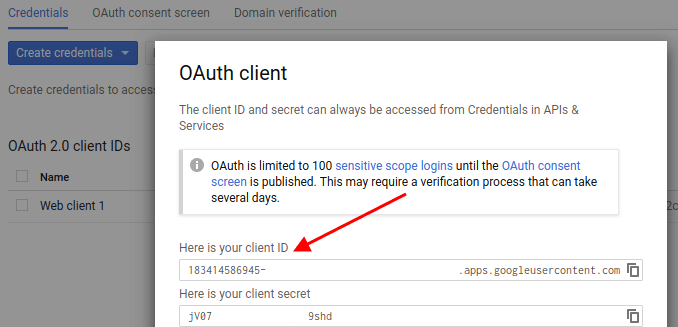


Choose an application type.

Next click on the **Create** button. It will be asked us to provide a few more details about our application.

Enter all the details and save it. After that we need to set the **Redirect URI** as per our application settings. It is the URL where the user will be redirected after login.

At this point, we have created the **Google OAuth2 client application**, and now we should be able to use this application to integrate Google login.



Lastly we have to note down or copy the **Client ID** and **Client Secret** values that will be required during the application configuration on our end.

**Sign Up Configuration Using Amazon**

**To use Amazon for social login we must:**

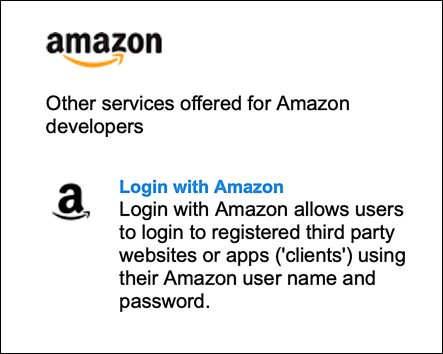
1. Obtain an Amazon developer account from [https://developer.amazon.com](https://developer.amazon.com/).
2. Do one of the following:
   * Open a previously-created Amazon application and copy the client ID and client secret.
   * Create a new application and copy the client ID and client secret.
3. Use the Social Login Engage Dashboard and the Amazon client ID and client secret to configure Amazon as a social login provider.

This section walks you through the process of creating an Amazon application for social login. If you already have an application, see the [Using an Existing Amazon Application for Social Login](https://janrain-education-center.knowledgeowl.com/home/amazon-social-login-configuration-guide" \l "using" \o ")section of this documentation.

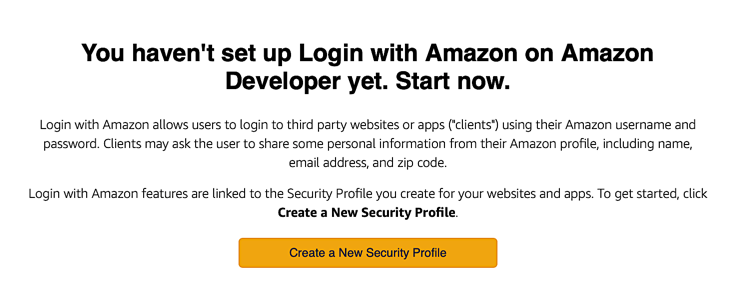
To create an Amazon application, complete the following procedure:

1. Log on to the Amazon developers center ([https://developer.amazon.com](https://developer.amazon.com/)).
2. On the developer home page, in the upper right corner, click **Developer Console:**



3.On the **Developer Console** home page, click **Login with Amazon:**

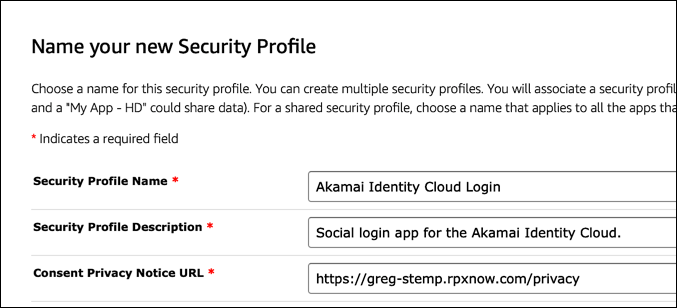
4. On the **Login with Amazon** home page, click **Create a New Security Profile**



5.On the Name your new Security Profilepage:

* Enter the name of your app in the **Security Profile Name** field.
* Enter a brief description of your app in the **Security Profile Description**field.
* Enter the URL for your privacy policy in the **Consent Privacy Notice URL** field.

For example:



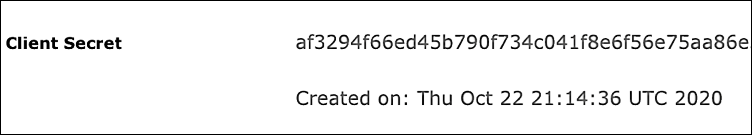
1. Click Save.
2. On the home page for your new app. click the configuration icon in the Manage column and then click Web Settings:



In the **Web Settings** section, copy the **Client ID:**

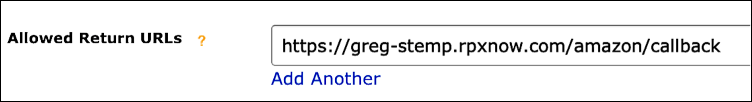


Click **Show Secret** and then copy the client secret



8.Both the client ID and the client secret are required to configure Amazon as a social login provider.

9. Click the Edit button, enter your Amazon redirect URI in the AllowedReturns URLs field, and then click Save. If you are using a standard Identity Cloud domain your redirect URL will look similar to this: *https://greg-stemp.rpxnow.com/amazon/callback*. For example:



1. Log off from the Amazon developers center.

**Password:**  We can set a custom password policy on our AWS account to specify complexity requirements and mandatory rotation periods for your IAM users' passwords. If we don't set a custom password policy, IAM user passwords must meet the default AWS password policy.

**Rules for setting a password policy:** The IAM password policy does not apply to the AWS account root user password or IAM user access keys. If a password expires, the IAM user can't sign in to the AWS Management Console but can continue to use their access keys.

When we create or change a password policy, most of the password policy settings are enforced the next time our users change their passwords. However, some of the settings are enforced immediately. For example:

When the minimum length and character type requirements change, these settings are enforced the next time that our users change their passwords. Users are not forced to change their existing passwords, even if the existing passwords do not adhere to the updated password policy.

When we set a password expiration period, the expiration period is enforced immediately. For example, assume that you set a password expiration period of 90 days. In that case, the password expires for all IAM users whose existing password is older than 90 days. Those users are required to change their password the next time that they sign in.

We can't create a "lockout policy" to lock a user out of the account after a specified number of failed sign-in attempts. For enhanced security, we recommend that you combine a strong password policy with multi-factor authentication (MFA).

## Permissions required to set a password policy

We must configure permissions to allow an IAM entity (user or role) to view or edit their account password policy. You can include the following password policy actions in an IAM policy:

iam:GetAccountPasswordPolicy – Allows the entity to view the password policy for their account

iam:DeleteAccountPasswordPolicy – Allows the entity to delete the custom password policy for their account and revert to the default password policy

iam:UpdateAccountPasswordPolicy – Allows the entity to create or change the custom password policy for their account.

## Default password policy

If an administrator does not set a custom password policy, IAM user passwords must meet the default AWS password policy. The default password policy enforces the following conditions:

Minimum password length of 8 characters and a maximum length of 128 characters

Minimum of three of the following mix of character types: uppercase, lowercase, numbers, and ! @ # $ % ^ & \* ( ) \_ + - = [ ] { } | ' symbols

Not be identical to your AWS account name or email address

## Custom password policy options

When we configure a custom password policy for your account, you can specify the following conditions:

Password minimum length – We can specify a minimum of 6 characters and a maximum of 128 characters.

Password strength – We can select any of the following check boxes to define the strength of your IAM user passwords:

Require at least one uppercase letter from Latin alphabet (A–Z)

Require at least one lowercase letter from Latin alphabet (a–z)

Require at least one number

Require at least one nonalphanumeric character ! @ # $ % ^ & \* ( ) \_ + - = [ ] { } | '

Enable password expiration – We can select and specify a minimum of 1 and a maximum of 1,095 days that IAM user passwords are valid after they are set. For example, after 90 days a user's password expires and they must set a new password before accessing the AWS Management Console. The AWS Management Console warns IAM users when they are within 15 days of password expiration. IAM users can change their password at any time if they have permission. When they set a new password, the expiration period for that password starts over. An IAM user can have only one valid password at a time.

Password expiration requires administrator reset – Select this option to prevent IAM users from updating their own passwords after the password expires. Before you select this option, confirm that your AWS account has more than one user with administrative permissions to reset IAM user passwords. Also consider providing access keys to allow administrators to reset IAM user passwords programmatically. If you clear this check box, IAM users with expired passwords must still set a new password before they can access the AWS Management Console.

Allow users to change their own password – We can permit all IAM users in your account to use the IAM console to change their own passwords, as described in Permitting IAM users to change their own passwords. Alternatively, you can allow only some users to manage passwords, either for themselves or for others. To do so, you clear this check box. For more information about using policies to limit who can manage passwords, see Permitting IAM users to change their own passwords.

Prevent password reuse – We can prevent IAM users from reusing a specified number of previous passwords. We can specify a minimum number of 1 and a maximum number of 24 previous passwords that can't be repeated.

## Setting a password policy (console)

We can use the AWS Management Console to create, change, or delete a custom password policy.

**To create a custom password policy (console)**

Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.

In the navigation pane, choose Account settings.

In the Password policy section, choose Change password policy.

Select the options that you want to apply to your password policy and choose Save changes.

**To change a custom password policy (console)**

Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.

In the navigation pane, choose Account settings.

In the Password policy section, choose Change.

Select the options that you want to apply to your password policy and choose Save changes.

**To delete a password policy (console)**

Sign in to the AWS Management Console and open the IAM console at https://console.aws.amazon.com/iam/.

In the navigation pane, choose Account settings.

In the Password policy section, choose Delete.

Confirm that you want to delete the custom password policy by choosing Delete custom.