Tutorial/Lab

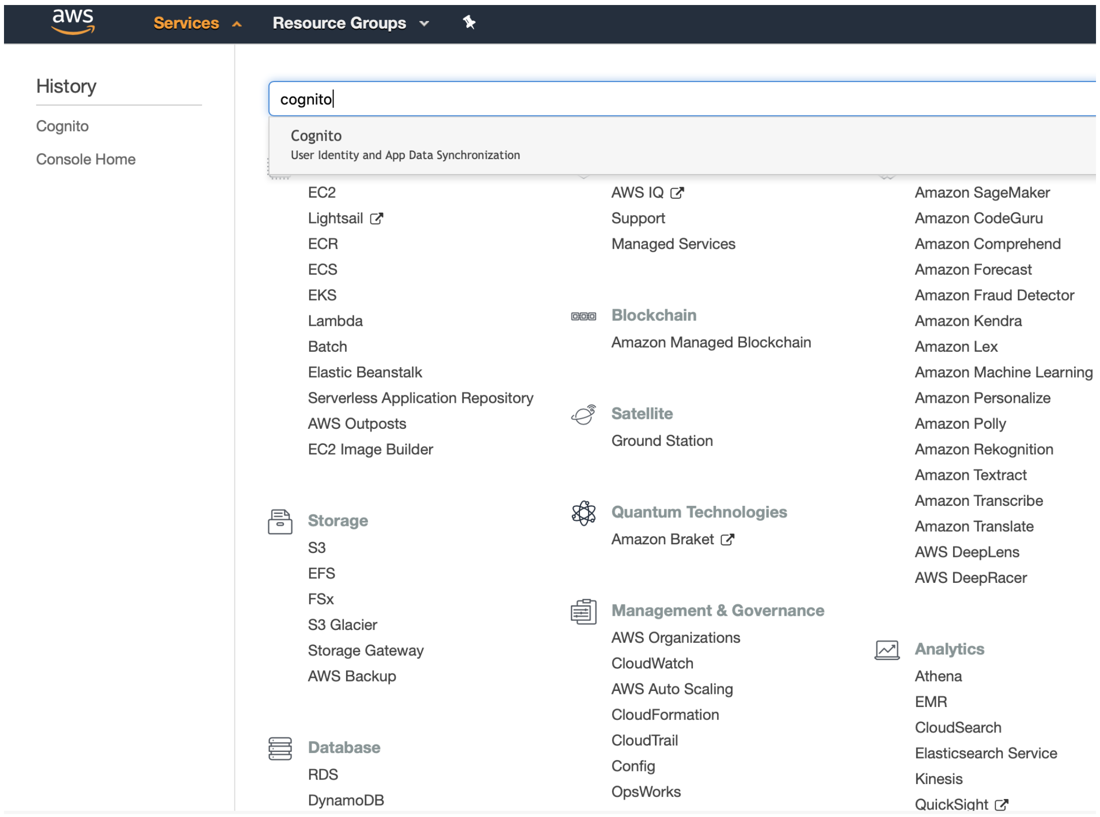
More AWS: Cognito

For this tutorial/lab you will first walk through the steps to create an AWS Cognito User Pool. After the tutorial, you will create a simple Android application to use the user pool you have created.

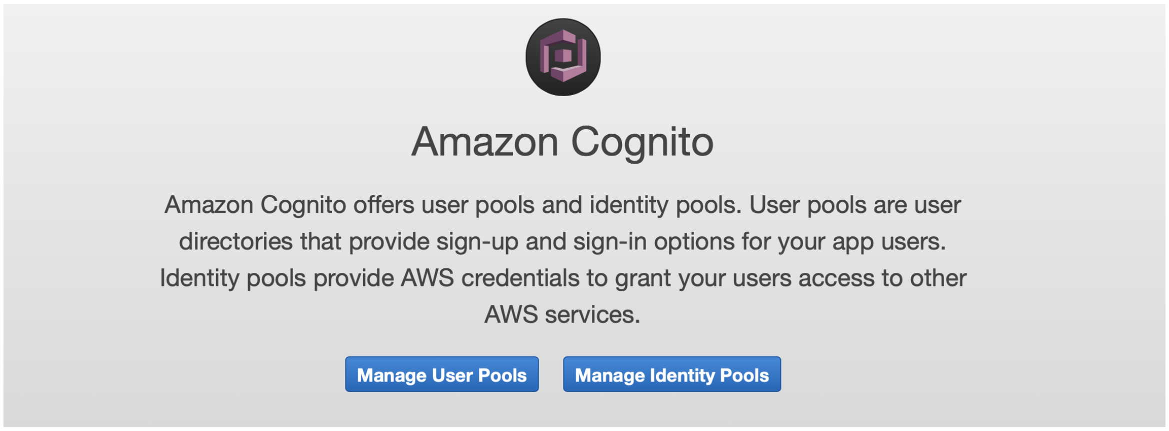
Create an AWS Cognito User Pool

Note: When creating your user pool, use the default settings for each page except where a step tells you to change a setting.

1. Search for Cognito in the AWS Console

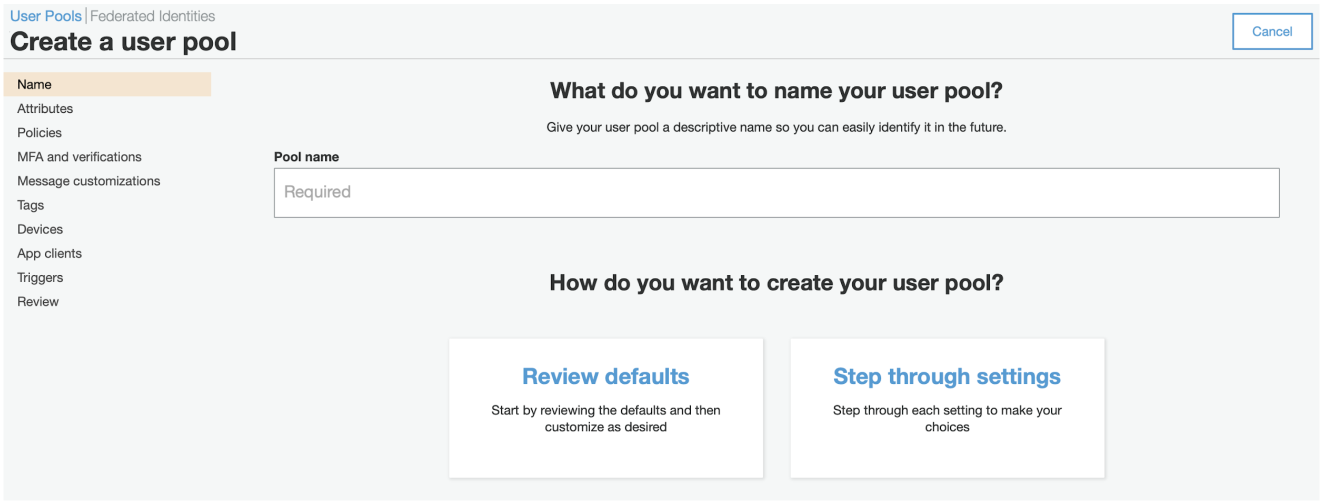


2. Click on Cognito and you should get to a screen that looks like this.


Amazon Cognito

Amazon Cognito offers user pools and identity pools. User pools are user directories that provide sign-up and sign-in options for your app users. Identity pools provide AWS credentials to grant your users access to other AWS services.

3. Click “Manage User Pools”
4. “Create a user pool” in the top right corner.
5. The setup screen should look like this. Give your user pool a descriptive name such as Your_App_Name_Users.



Create a user pool

What do you want to name your user pool?

Give your user pool a descriptive name so you can easily identify it in the future.

Pool name

Required

How do you want to create your user pool?

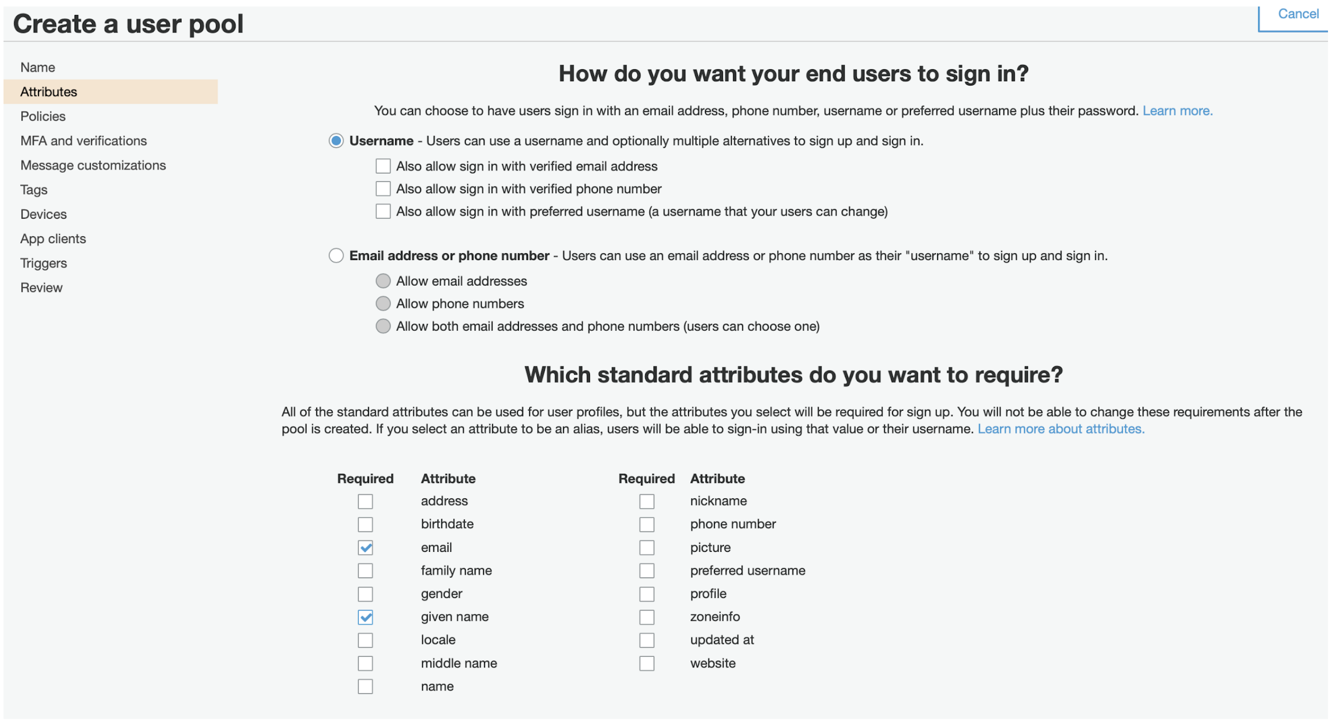
Review defaults

Start by reviewing the defaults and then customize as desired

Step through settings

Step through each setting to make your choices

6. At this point there are two options: “Review defaults” or “Step through settings”.
7. Choose “Step through settings” to help you better understand what some of the different parts mean.
8. The next step is “Attributes”. For the first part, keep the username selected. This means that users will have to sign up and sign in with a username instead of email or phone number. For the required standard attributes we want to select email and given name. This means we are requiring them to enter both of these when they sign up. Password is a mandatory field which is why you don’t see it listed. We won’t add any custom attributes for this user pool.


Create a user pool

How do you want your end users to sign in?

You can choose to have users sign in with an email address, phone number, username or preferred username plus their password. Learn more.

[✓ **Username** - Users can use a username and optionally multiple alternatives to sign up and sign in.]
- [ ] Also allow sign in with verified email address
- [ ] Also allow sign in with verified phone number
- [ ] Also allow sign in with preferred username (a username that your users can change)

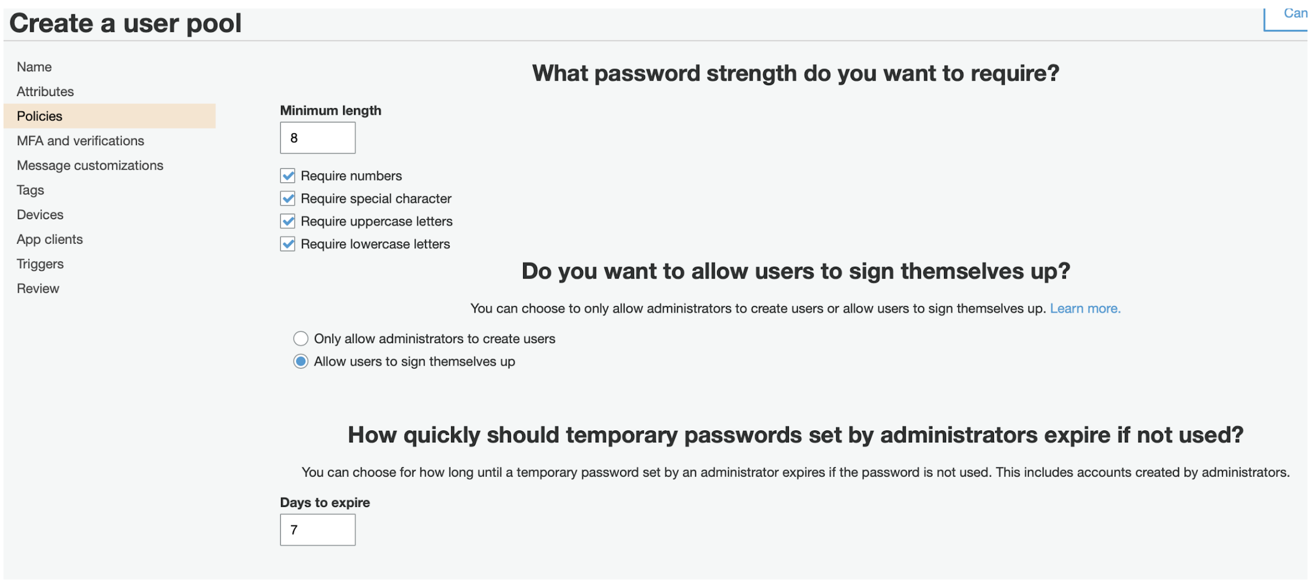
Email address or phone number - Users can use an email address or phone number as their "username" to sign up and sign in.
- Allow email addresses
- Allow phone numbers
- Allow both email addresses and phone numbers (users can choose one)

Which standard attributes do you want to require?

All of the standard attributes can be used for user profiles, but the attributes you select will be required for sign up. You will not be able to change these requirements after the pool is created. If you select an attribute to be an alias, users will be able to sign-in using that value or their username. Learn more about attributes.

Required	Attribute	Required	Attribute
☐	address	☐	nickname
☐	birthdate	☐	phone number
✓	email	☐	picture
☐	family name	☐	preferred username
☐	gender	☐	profile
✓	given name	☐	zoneinfo
☐	locale	☐	updated at
☐	middle name	☐	website
☐	name		

9. The next step is “Policies”. We won’t change anything on this page, but if you wanted to require your users to have a more or less secure password, you would do that here.


Create a user pool

What password strength do you want to require?

Minimum length: 8

- [x] Require numbers
- [x] Require special character
- [x] Require uppercase letters
- [x] Require lowercase letters

Do you want to allow users to sign themselves up?

You can choose to only allow administrators to create users or allow users to sign themselves up. Learn more.

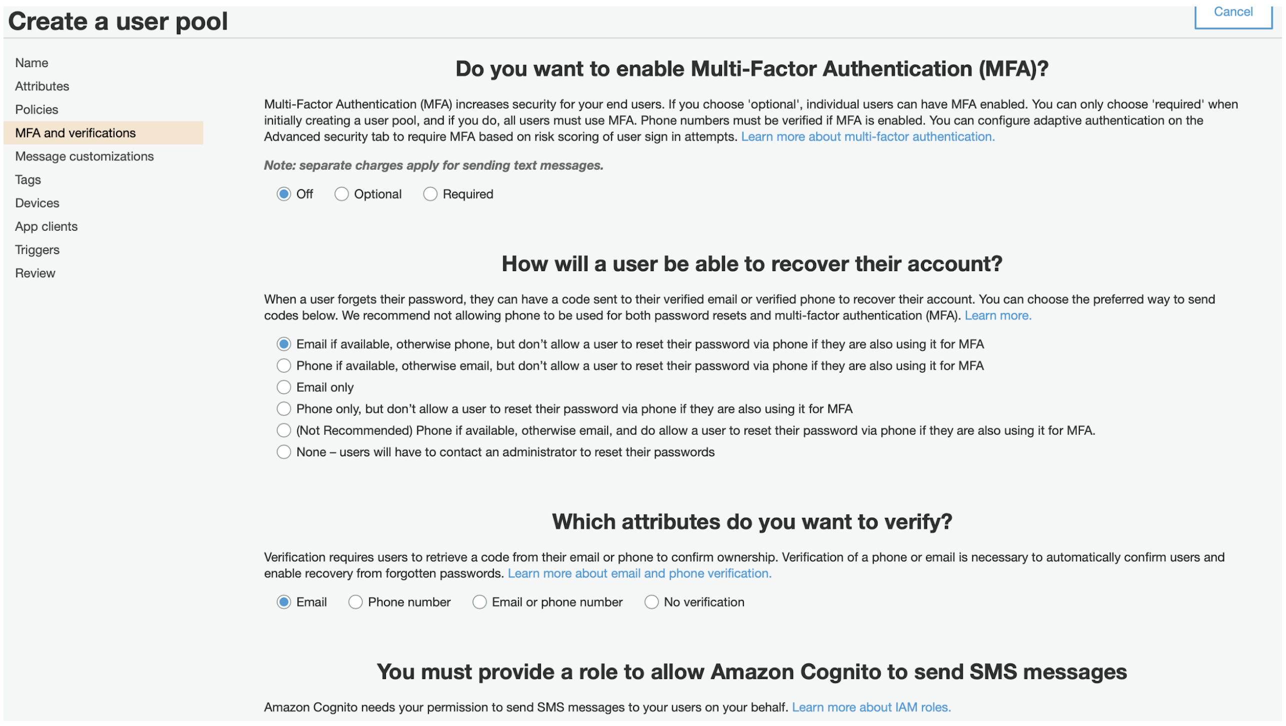
Only allow administrators to create users [✓ Allow users to sign themselves up]

How quickly should temporary passwords set by administrators expire if not used?

You can choose for how long until a temporary password set by an administrator expires if the password is not used. This includes accounts created by administrators.

Days to expire: 7

10. The next step is “MFA and verifications”. Here we will make sure that MFA is off. Under “which attribute do you want to verify?” make sure that **email** is selected. We won’t need to change anything else.



Create a user pool

Cancel

Do you want to enable Multi-Factor Authentication (MFA)?

Multi-Factor Authentication (MFA) increases security for your end users. If you choose 'optional', individual users can have MFA enabled. You can only choose 'required' when initially creating a user pool, and if you do, all users must use MFA. Phone numbers must be verified if MFA is enabled. You can configure adaptive authentication on the Advanced security tab to require MFA based on risk scoring of user sign in attempts. Learn more about multi-factor authentication.

Note: separate charges apply for sending text messages.

[✓ Off] Optional Required

How will a user be able to recover their account?

When a user forgets their password, they can have a code sent to their verified email or verified phone to recover their account. You can choose the preferred way to send codes below. We recommend not allowing phone to be used for both password resets and multi-factor authentication (MFA). Learn more.

- [✓ Email if available, otherwise phone, but don't allow a user to reset their password via phone if they are also using it for MFA]
- Phone if available, otherwise email, but don't allow a user to reset their password via phone if they are also using it for MFA
- Email only
- Phone only, but don't allow a user to reset their password via phone if they are also using it for MFA
- (Not Recommended) Phone if available, otherwise email, and do allow a user to reset their password via phone if they are also using it for MFA.
- None – users will have to contact an administrator to reset their passwords

Which attributes do you want to verify?

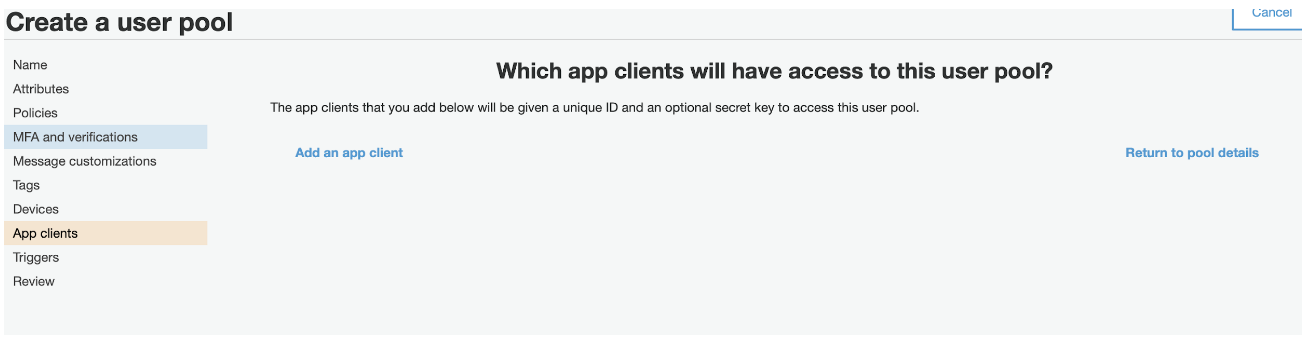
Verification requires users to retrieve a code from their email or phone to confirm ownership. Verification of a phone or email is necessary to automatically confirm users and enable recovery from forgotten passwords. Learn more about email and phone verification.

[✓ Email] Phone number Email or phone number No verification

You must provide a role to allow Amazon Cognito to send SMS messages

Amazon Cognito needs your permission to send SMS messages to your users on your behalf. Learn more about IAM roles.

11. We are now going to skip down to "App clients" (do this by clicking on "App clients" in the left sidebar menu) and click on "Add an app client". Feel free to read the sections we passed over to see what they do.



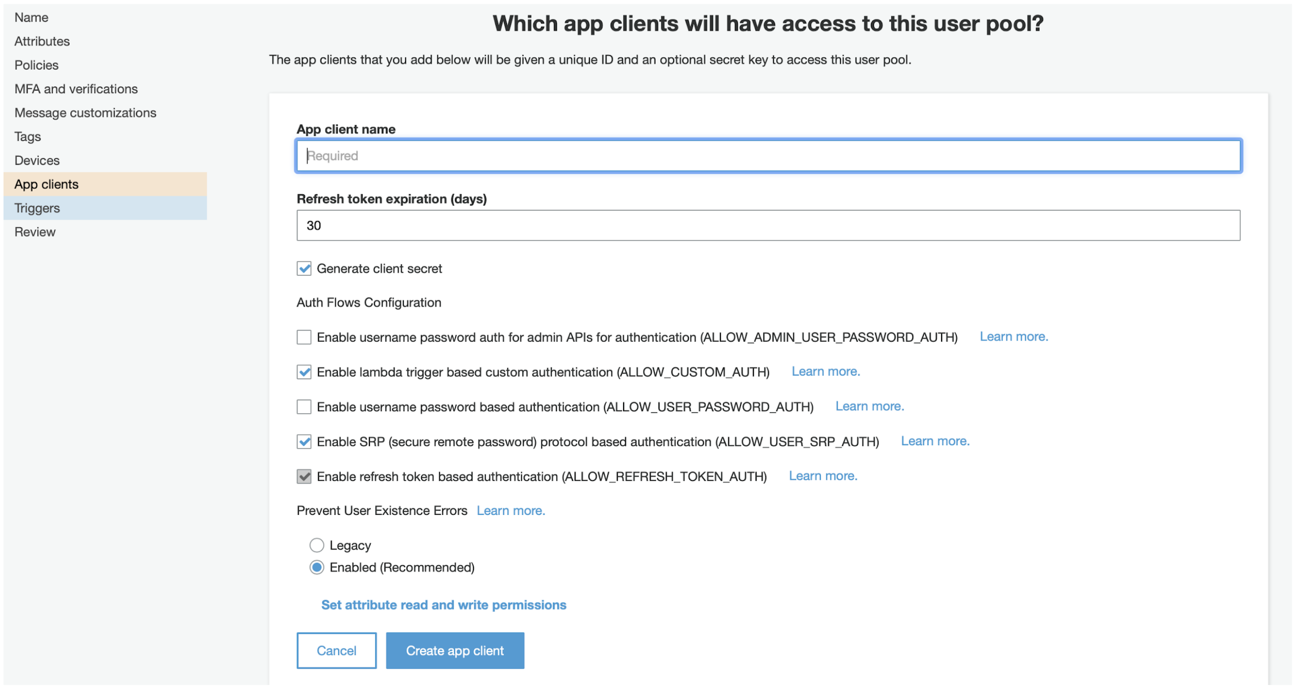
Create a user pool

Cancel

Which app clients will have access to this user pool?

The app clients that you add below will be given a unique ID and an optional secret key to access this user pool.

12. Your screen should look like this. Now enter an app client name (this will be an Android app that you create shortly).
13. Make sure "Generate client secret" is selected and hit "Create app client".



Which app clients will have access to this user pool?

The app clients that you add below will be given a unique ID and an optional secret key to access this user pool.

App client name: Required

Refresh token expiration (days): 30

[✓] Generate client secret

Auth Flows Configuration

[ ] Enable username password auth for admin APIs for authentication (ALLOW_ADMIN_USER_PASSWORD_AUTH) Learn more.

[✓] Enable lambda trigger based custom authentication (ALLOW_CUSTOM_AUTH) Learn more.

[ ] Enable username password based authentication (ALLOW_USER_PASSWORD_AUTH) Learn more.

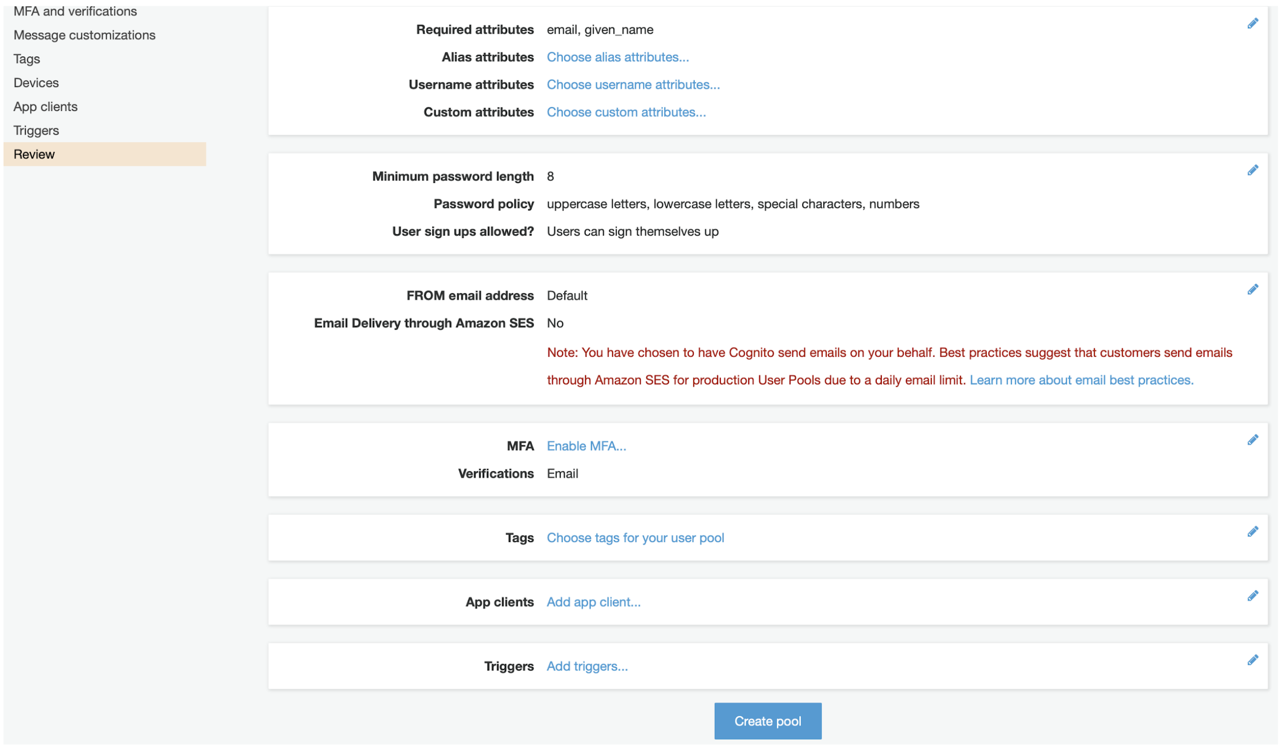
[✓] Enable SRP (secure remote password) protocol based authentication (ALLOW_USER_SRP_AUTH) Learn more.

[✓] Enable refresh token based authentication (ALLOW_REFRESH_TOKEN_AUTH) Learn more.

Prevent User Existence Errors Learn more.

Legacy [✓ Enabled (Recommended)]

14. After you have created an app client, go to the “Review” step.
15. Scroll to the bottom of the page and click “create pool”.



Required attributes	email, given_name
Alias attributes	Choose alias attributes...
Username attributes	Choose username attributes...
Custom attributes	Choose custom attributes...
Minimum password length	8
Password policy	uppercase letters, lowercase letters, special characters, numbers
User sign ups allowed?	Users can sign themselves up
FROM email address	Default
Email Delivery through Amazon SES	No
	Note: You have chosen to have Cognito send emails on your behalf. Best practices suggest that customers send emails through Amazon SES for production User Pools due to a daily email limit. Learn more about email best practices.
MFA	Enable MFA...
Verifications	Email
Tags	Choose tags for your user pool
App clients	Add app client...
Triggers	Add triggers...

Now your user pool is configured to use in an Android app!

Use AWS Cognito to Authenticate Users in an Android app

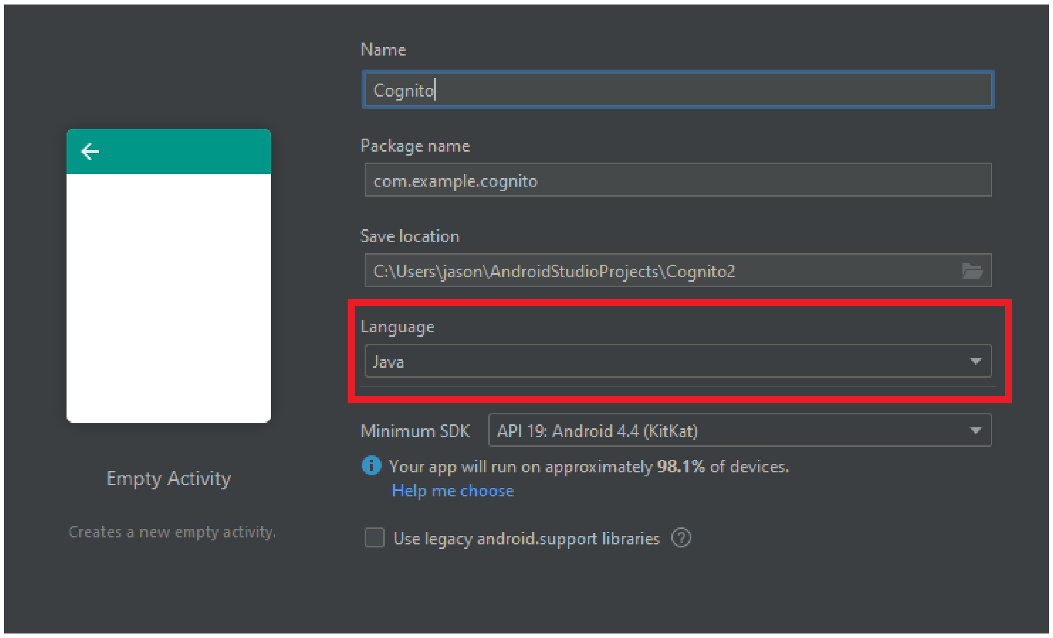
Now you are going to create a simple Android application where you will integrate AWS Cognito to authenticate users.

What your Android app will be able to do

1. Take in information (username,password, name, email).
2. Create an AWS cognito user with the Signup button.
3. Confirm an AWS cognito user with the confirm button.
4. Sign in an existing confirmed AWS cognito user using the sign-in button.

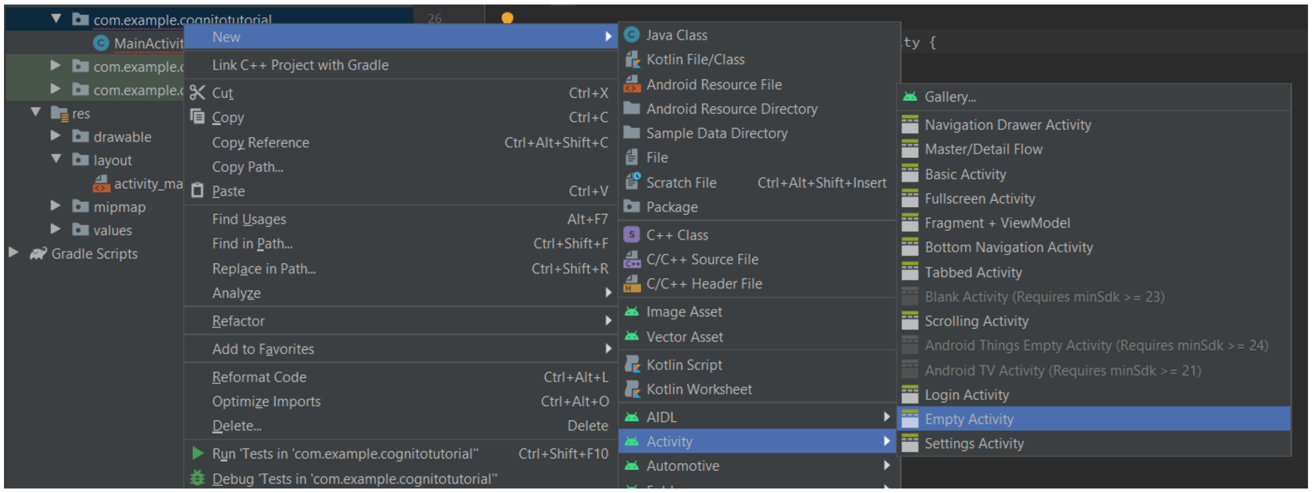
Setting up the Lab

1. Download the Cognito User Files:
2. Create a new Android studio project with an **empty activity.**
    a. Any name is ok
    b. Make sure the language is Java
        i. Look at the screenshot below to confirm your settings

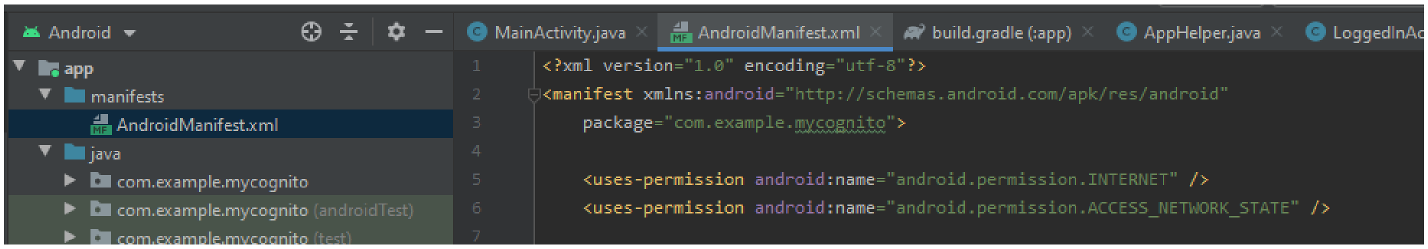


Name: Cognito
Package name: com.example.cognito
Save location: C:\Users\jason\AndroidStudioProjects\Cognito2
Language: Java
Minimum SDK: API 19: Android 4.4 (KitKat)
Your app will run on approximately **98.1%** of devices.
Help me choose
☐ Use legacy android.support libraries

3. Copy the **MainActivity.java** and **activity_main.xml** *(Located in the res/layout folder)* over the ones generated for you and update the package statement in your MainActivity.java if necessary.
4. Create a **new empty Activity** with the name **LoggedInActivity** and **replace its .java and .xml files** with the provided code as well.

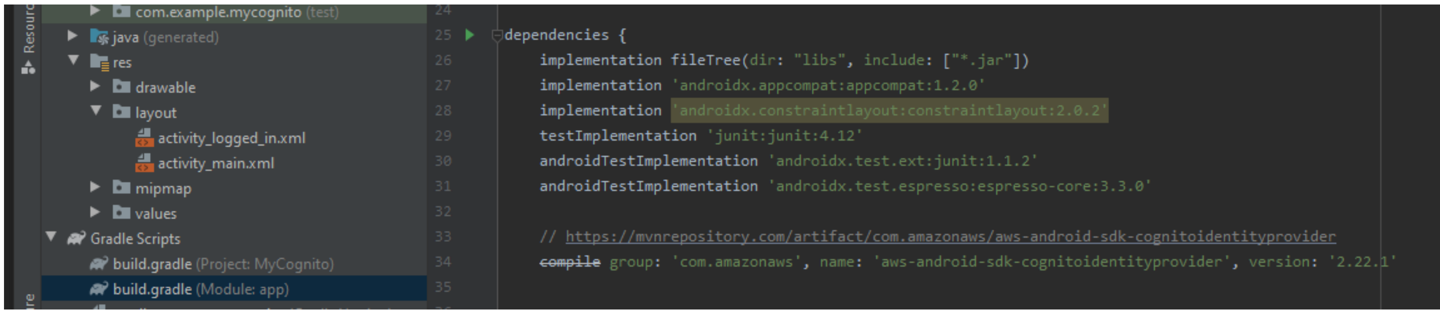


5. **Copy in the provided AppHelper.java** file into the same package as your
6. MainActivity.java
7. Add internet permissions in your Android Manifest file
    a. `<uses-permission android:name="android.permission.INTERNET" />`
    b. `<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />`



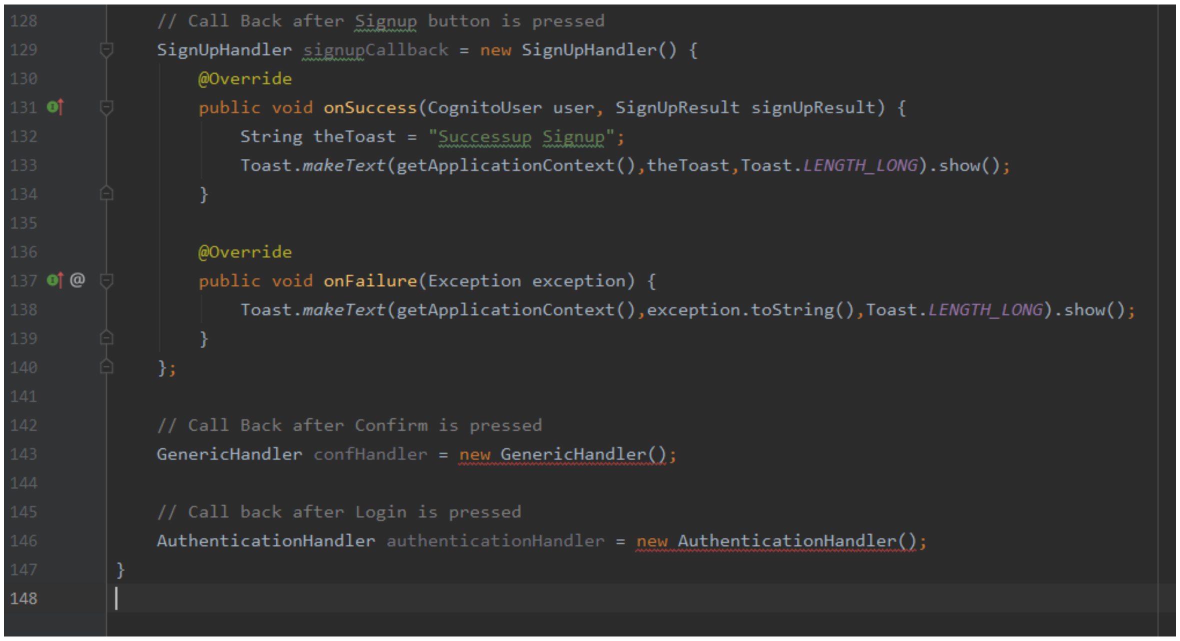
```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.mycognito">

    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
```

Add the following dependency to your **Module** build.gradle file: [MVN Repository: AWS SDK For Android Amazon Cognito Identity Provider]()



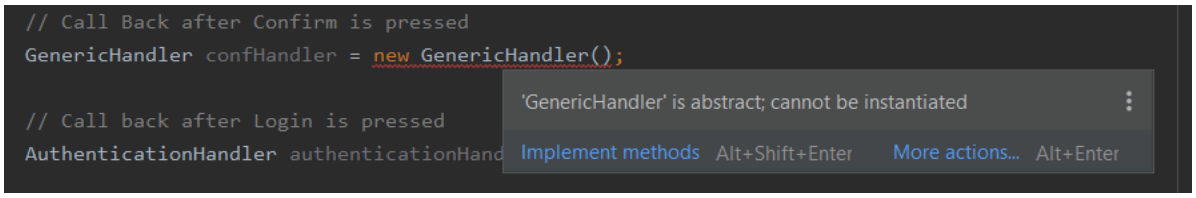
```
dependencies {
    implementation fileTree(dir: "libs", include: ["*.jar"])
    implementation 'androidx.appcompat:appcompat:1.2.0'
    implementation 'androidx.constraintlayout:constraintlayout:2.0.2'
    testImplementation 'junit:junit:4.12'
    androidTestImplementation 'androidx.test.ext:junit:1.1.2'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.3.0'

    // https://mvnrepository.com/artifact/com.amazonaws/aws-android-sdk-cognitoidentityprovider
    ~~compile~~ group: 'com.amazonaws', name: 'aws-android-sdk-cognitoidentityprovider', version: '2.22.1'
```

8. Sync your gradle build so all required AWS dependencies are downloaded.
9. Implement AWS `confHandler` and `authenticationHandler`:
    a. Open your MainActivity.java
    b. Scroll down to the bottom
    c. You will see 3 Callback Handlers, one of which is already implemented.



```java
    // Call Back after Signup button is pressed
    SignUpHandler signupCallback = new SignUpHandler() {
        @Override
        public void onSuccess(CognitoUser user, SignUpResult signUpResult) {
            String theToast = "Successup Signup";
            Toast.makeText(getApplicationContext(),theToast,Toast.LENGTH_LONG).show();
        }

        @Override
        public void onFailure(Exception exception) {
            Toast.makeText(getApplicationContext(),exception.toString(),Toast.LENGTH_LONG).show();
        }
    };

    // Call Back after Confirm is pressed
    GenericHandler confHandler = new GenericHandler();

    // Call back after Login is pressed
    AuthenticationHandler authenticationHandler = new AuthenticationHandler();
}

```

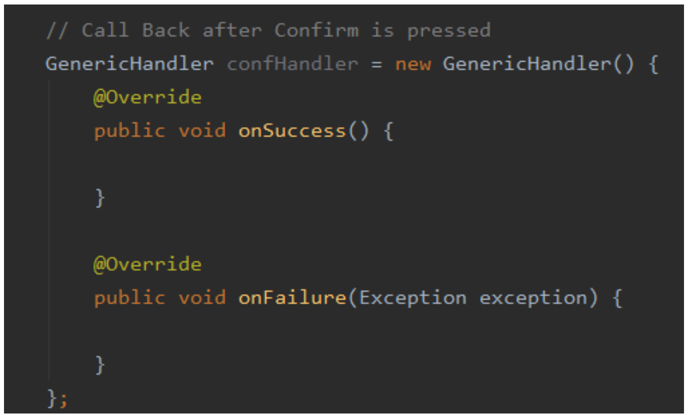
d. Hover over the method to get the “Implement methods” option.



```java
// Call Back after Confirm is pressed
GenericHandler confHandler = new GenericHandler();

// Call back after Login is pressed
AuthenticationHandler authenticationHand
```

'GenericHandler' is abstract; cannot be instantiated

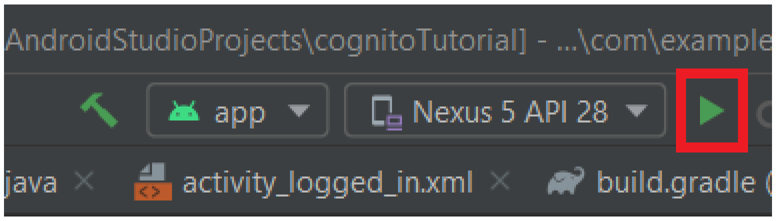
e. Make sure all methods are selected and click Ok. You will see that the confHandler methods are now ready to be implemented.



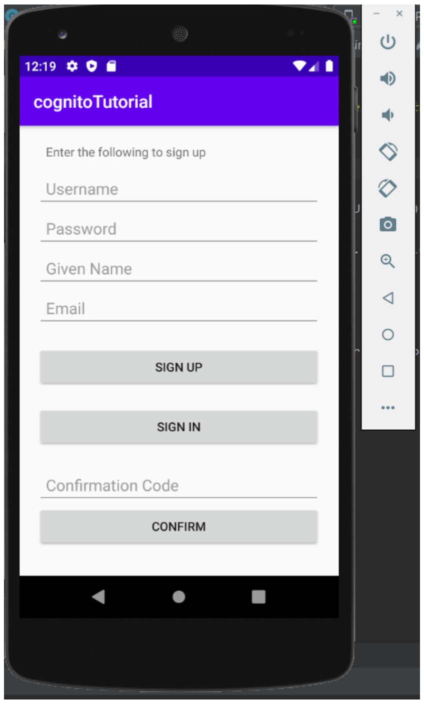
```java
// Call Back after Confirm is pressed
GenericHandler confHandler = new GenericHandler() {
    @Override
    public void onSuccess() {

    }

    @Override
    public void onFailure(Exception exception) {

    }
};
```

f. Do steps d and e for the authenticationHandler as well. Note that the authenticationHandler will have 5 methods instead of just two.

10. Verify setup by running your app



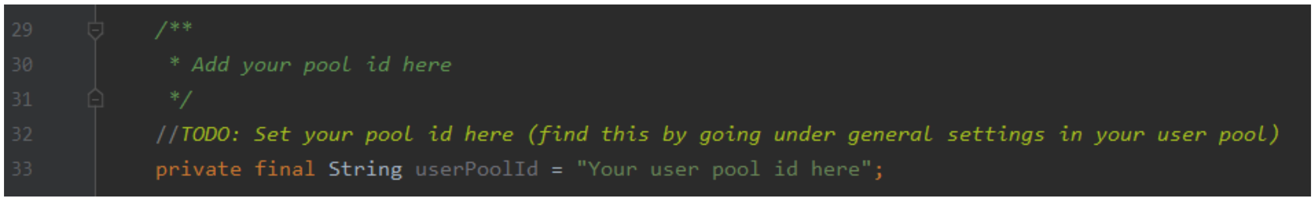
a. Your emulator should look similar to the screenshot below



Getting your Cognito Userpool information into your App

The `Apphelper.java` will be in charge of storing and retrieving the information required to access your specific Cognito user pool.

1. Open to your `AppHelper.java`.
2. On a separate window navigate to your AWS Cognito UserPool Dashboard.
3. From the dashboard you will need to get the following information:
   Note: Most of the information can be found under “App Clients” in the dashboard, but userPoolId is found under “General settings”.
   a. userPoolId
   b. clientId
   c. clientSecret (click “Show details” to see the secret key)
   d. cognitoRegion (this is located on the top right of your AWS console)
4. Input this information into the AppHelper.java at the specified fields. An example of the userpoolId is shown below:



```java
    /**
     * Add your pool id here
     */
    //TODO: Set your pool id here (find this by going under general settings in your user pool)
    private final String userPoolId = "Your user pool id here";
```

5. Implement the AppHelper init method by creating a new cognitoUserPool object and setting it as the value of the already declared userPool variable as shown below:
   a. `userPool = new CognitoUserPool(context, userPoolId, clientId, clientSecret, clientConfiguration, cognitoRegion);`
      i. The CognitoUserPool object represents the Cognito Userpool you created at the beginning of this tutorial. If any of the id or authentication information is incorrect, AWs will not authorize your object to access your userpool.

Wiring your App with AWS methods to Sign-up, Confirm, and Sign-in a user

Signup button

1. Open your MainActivity.java.
2. Find your signUpButton onclickListener.
   a. When the signup button is clicked, your app needs to:
      i. Create a CognitoUserAttributes object:
         1. `CognitoUserAttributes userAttributes = new CognitoUserAttributes();`
      ii. Set the attributes of the CognitoUserAttributes to have the email and name the user enters in the app:
         1. `userAttributes.addAttribute("given_name", givenName);`
         2. `userAttributes.addAttribute("email", email);`
      iii. Call the AWS Signup Method with your signupHandler as the callback function:
         1. `appHelper.getUserPool().signUpInBackground(username, password, userAttributes, null, signupCallback);`

            Note: We are getting the CognitoUserPool object that is stored in the AppHelper object. We then call the AWS method signUpInBackground() and pass it the information we retrieved from the application.
      iv. Display a toast indicating whether the signup was successful.
         1. This is already handled in the signupCallback handler that we provided.

3. Verify that your Signup button works.
    a. Run the application
    b. Fill out the information fields on the app.
        i. **Note:** make sure the password is at least 8 characters long and has an uppercase letter, a lowercase letter, a number, and a special character (like '!').
        ii. Make sure the email is a real email that you have access to.
    c. Click Signup.
    d. If everything worked correctly you will receive a notification on your app saying the signup was successful. You will also receive an email containing a confirmation code.

Confirm button

4. Find your confirmButton onClickListener in the MainActivity.java.
    a. When the confirm button is clicked, the app needs to:
        i. Get the user you want to confirm from the AWS Userpool
            1. `CognitoUser myUser = appHelper.getUserPool().getUser(username);`
               Note: this gets the userpool stored in the Apphelper and calls the `getUser()` method using the username string as an argument.
        ii. Pass the Confirmation code the user enters in the app (which will be the one received in the email from AWS) to the `confirmSignupInBackground` method to confirm your user.
            1. *`myUser.confirmSignUpInBackground(confirmationCode, false, confHandler);`*
        iii. Display a toast indicating whether the confirmation was successful. ***(Unlike the signup callback, which was written for you, you will need to write this code for confirmation as explained below):***
            1. This is handled in the confHandler that you started to implement.
            2. Display a toast indicating a successful confirmation if the confirmation was successful (if it reached the `onSuccess()` method).
            3. Display a toast indicating an unsuccessful confirmation if it reached the `onFailure()` method.
            4. Look at the SignupHandler for help.
5. Verify your confirmation Button
    a. Run your application.
    b. Enter the username of the user you registered.
    c. Enter the confirmation code with the code emailed to you by AWS.
    d. Press Confirm.
    e. If it was successfully confirmed, you will receive a notification that the confirmation was successful.

Sign In button

6. Find your signInButton onClickListener
    a. When the "Sign In" button is clicked, the app needs to:
        i. Get the user you want to confirm from the AWS Userpool.
            1. `CognitoUser myUser = appHelper.getUserPool().getUser(username);`

            Note: this gets the userpool stored in the Apphelper and calls the getUser() method using the username string as an argument.
        ii. Get a login session from AWS for the specified user
            1. `myUser.getSessionInBackground(authenticationHandler);`
7. Find your Authentication handler.
   Note: The Authentication handler is more complex than the signup and confirmation handlers so it has more methods. It first verifies that the user you want to login with exists. Then if the user exists, it asks your app for the login credentials.
    a. After the "Sign in" button is clicked and the **getAuthenticationDetail** method is invoked by aws we need to:
        i. Create an authenticationDetails object (in the getAuthenticationDetails method)
            1. `AuthenticationDetails authenticationDetails = new AuthenticationDetails(userId, password, null);`

               Note: userId is the username String. Paassword is the password as a string.
        ii. Set the authenticationContinuation object to have the authenticationDetails:
            1. `authenticationContinuation.setAuthenticationDetails(authenticationDetails);`
        iii. Continue the task so AWS can verify that the password for the user is correct
            1. `authenticationContinuation.continueTask();`
    b. When a user sign in attempt is successful your handler needs to start the loggedInActivity:
        i. In the `onSuccess()` method of the authentication handler, run the method to start the LoggedInActivity:
            1. `startLoggedInActivity();`
    c. When a user sign in attempt is unsuccessful your handler needs to output a notification saying it failed:
        i. Make a toast saying it failed. Look at the signup handler for an example.

You may have noticed that there are two methods we did not implement: `getMFACode(...)` and `authenticationChallenge(...).` These are methods we would implement if we wanted to have additional security for our application. However for this tutorial we disabled Multi-Factor Authentication and will not be adding additional security.

8. Verify that the sign-in works:
    a. Run the application.
    b. Enter the username and password for your **confirmed** user.
    c. Click Sign-in.
    d. Your app should be redirected to a simple page saying that you signed in.