Adapter-based authentication in native Android applications

fork and edit tutorial (https://github.ibm.com/MFPSamples/DevCenter/tree/master/tutorials/en/foundation/7.0/authentication-security/adapter-based-authentication-native-android-applications.html) | report issue (https://github.ibm.com/MFPSamples/DevCenter/issues/new) This tutorial illustrates the native Android client-side authentication components for adapter-based authentication. **Prerequisite:** Make sure that you read Adapter-based authentication (../) first.

Creating the client-side authentication components

- 1. Create a native Android application and add the MobileFirst native APIs as explained in the documentation.
- 2. Add an activity, LoginAdapterBasedAuth, which will handle and present the login form.
- 3. Remember to add this activity to the AndroidManifest.xml file, too.
- 4. Create a MyChallengeHandler class as a subclass of ChallengeHandler.

The <u>isCustomResponse</u> method checks every custom response received from MobileFirst Server to verify whether it is the expected challenge. In the sample adapter code, a <u>authRequired</u> variable is sent for this purpose.

```
public boolean isCustomResponse(WLResponse response) {
   try {
      if(response!= null &&
        response.getResponseJSON()!=null &&
        response.getResponseJSON().isNull("authRequired") != true &&
        response.getResponseJSON().getBoolean("authRequired") == true)
   {
      return true;
   }
   } catch (JSONException e) {
      e.printStackTrace();
   }
   return false;
}
```

The handleChallenge method is called after the isCustomResponse method returns true.

5. Use this method to present the login form.

```
public void handleChallenge(WLResponse response){
   cachedResponse = response;
   Intent login = new Intent(parentActivity, LoginAdapterBasedAuth.class)
;
   parentActivity.startActivityForResult(login, 1);
}
```

6. In the submitLogin method, if the user asked to abort this action, use the submitFailure() method, otherwise invoke the adapter authentication procedure by using the submitAdapterAuthentication() method.

```
public void submitLogin(int resultCode, String userName, String password, boolean back) {
   if (resultCode != Activity.RESULT_OK || back) {
        submitFailure(cachedResponse);
   } else {
        Object[] parameters = new Object[]{userName, password};
        WLProcedureInvocationData invocationData = new WLProcedureInvocationData("NativeAdapterBasedAdapter", "submitAuthentication");
        invocationData.setParameters(parameters);
        WLRequestOptions options = new WLRequestOptions();
        options.setTimeout(30000);
        submitAdapterAuthentication(invocationData, options);
   }
}
```

7. In the main activity class, connect to MobileFirst Server, register your challengeHandler method, and invoke the protected adapter procedure.

The procedure invocation triggers MobileFirst Server to send a challenge that will trigger the challengeHandler.

```
final WLClient client = WLClient.createInstance(this);
client.connect(new MyConnectionListener());
challengeHandler = new AndroidChallengeHandler(this, realm);
client.registerChallengeHandler(challengeHandler);
invokeBtn = (Button) findViewByld(R.id.invoke);
invokeBtn.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        WLProcedureInvocationData invocationData = new WLProcedureInvocationData("DummyAdapter", "getS ecretData");
        WLRequestOptions options = new WLRequestOptions();
        options.setTimeout(30000);
        client.invokeProcedure(invocationData, new MyResponseListener(), options);
    }
});
```

Sample application

Click to download

(http://public.dhe.ibm.com/software/products/en/MobileFirstPlatform/docs/v700/NativeAdapterBasedAuthProject.zip) the Studio project.

Click to download

(http://public.dhe.ibm.com/software/products/en/MobileFirstPlatform/docs/v700/AndroidNativeAdapterBasedAuthProject.zip) the Native project.





