

Custom Authenticator and Login Module in native Android applications

This is a continuation of Custom Authenticator and Login Module (../).

Creating the client-side authentication components

Create a native Android application and add the IBM MobileFirst Platform Foundation native APIs following the documentation.

Add an Activity, *LoginCustomLoginModule*, that will handle and present the login form.

Remember to add this Activity to the *AndroidManifest.xml* file as well.

Create a *MyChallengeHandler* class as a subclass of *ChallengeHandler*.

MyChallengeHandler should implement 2 main methods:

- *isCustomResponse*
- *HandleChallenge*

In our sample we add another method to present and handle the received data from our form (*submitLogin*).

isCustomResponse

This method checks every custom response received from the MobileFirst Server to see if that's the challenge we are expecting.

```

1  public boolean isCustomResponse(WLResponse response) {
2      if (response == null || response.getResponseJSON() == null) {
3          return false;
4      }
5      if(response.toString().indexOf("authStatus") > -1){
6          return true;
7      }
8      else{
9          return false;
10     }
11 }

```

handleChallenge

This method is called after the *isCustomResponse* method returned *true*.

Here we use this method to present our login form.

```

1  public void handleChallenge(WLResponse response){
2      try {
3          if(response.getResponseJSON().getString("authStatus") == "complete"){
4              submitSuccess(response);
5          }
6          else {
7              cachedResponse = response;
8              Intent login = new Intent(parentActivity, LoginCustomLoginModule.class);
9              parentActivity.startActivityForResult(login, 1);
10         }
11     } catch (JSONException e) {
12         e.printStackTrace();
13     }
14 }

```

submitLogin

If the user asked to abort this action we use *submitFailure()* method, otherwise we send the information we collected from our login form to our custom authenticator using *submitLoginForm()* method.

```
1 public void submitLogin(int resultCode, String userName, String password, boolean back){
2     if (resultCode != Activity.RESULT_OK || back) {
3         submitFailure(cachedResponse);
4     } else {
5         HashMap<String, String> params = new HashMap<String, String>();
6         params.put("username", userName);
7         params.put("password", password);
8         submitLoginForm("/my_custom_auth_request_url", params, null, 0, "post");
9     }
10 }
```

Main Activity

In the Main Activity class connect to the MobileFirst server, register your *challengeHandler* and invoke the protected adapter procedure.

The procedure invocation will trigger the MobileFirst server to send a challenge that will trigger our *challengeHandler*.

```
1 final WLCClient client = WLCClient.createInstance(this);
2 client.connect(new MyConnectionListener());
3 challengeHandler = new AndroidChallengeHandler(this, realm);
4 client.registerChallengeHandler(challengeHandler);
5 invokeBtn = (Button) findViewById(R.id.invoke);
6 invokeBtn.setOnClickListener(new View.OnClickListener() {
7     @Override
8     public void onClick(View v) {
9         WLProcedureInvocationData invocationData = new WLProcedureInvocationData("DummyAdapter", "getSecretData");
10        WLRequestOptions options = new WLRequestOptions();
11        options.setTimeout(30000);
12        client.invokeProcedure(invocationData, new MyResponseListener(), options);
13    }
14 });
```

Sample application

Click to download

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

Click to download

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

