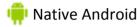
## MobileFirst Platform {dev}

# **Custom Authentication in native Android** applications

Relevant to:



This is a continuation of Custom Authentication.

# 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.

```
public boolean isCustomResponse(WLResponse response) {
    if (response == null | response.getResponseJSON() == null) {
        return false:
    if(response.toString().indexOf("authStatus") > -1){
        return true:
    else{
        return false;
}
```

#### handleChallenge

This method is called after the *isCustomResponse* method returned *true*. Here we use this method to present our login form.

```
public void handleChallenge(WLResponse response){
       if(response.getResponseJSON().getString("authStatus") ==
```

```
"complete"){
            submitSuccess(response);
        else {
            cachedResponse = response;
            Intent login = new Intent(parentActivity,
LoginCustomLoginModule.class);
            parentActivity.startActivityForResult(login, 1);
        }
    } catch (JSONException e) {
        e.printStackTrace();
}
```

#### 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.

```
public void submitLogin(int resultCode, String userName, String
password, boolean back){
    if (resultCode != Activity.RESULT OK || back) {
        submitFailure(cachedResponse);
    } else {
        HashMap<String, String> params = new HashMap<String, String>();
        params.put("username", userName);
        params.put("password", password);
        submitLoginForm("/my_custom auth request url", params, null, 0,
"post");
```

#### **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.

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

## **Worklight Protocol**

If your custom authenticator uses WorklightProtocolAuthenticator, some simplifications can be made:

- Subclass your challenge handler using WLChallengeHandler instead of ChallengeHandler. Note the WL.
- You no longer need to implement isCustomResponse as the challenge handler will automatically check that the realm name matches.
- handleChallenge will receive the challenge as a parameter, not the entire response object.
- Instead of submitLoginForm, use submitChallengeAnswer to send your challenge response as a JSON.
- There is no need to call submitSuccess or submitFailure as the framework will do it for you.

For an example that uses WorklightProtocolAuthenticator, see the Remember Me tutorial or this video blog post.

# Sample application

Click to download the MobileFirst project.

Click to download the Native project.

- The CustomAuth project contains a MobileFirst native API that you can deploy to your MobileFirst server.
- The CustomAuthAndroid project contains a native Android application that uses a MobileFirst native API library.
- Make sure to update the worklight.plist file in the native project with the relevant server settings.

