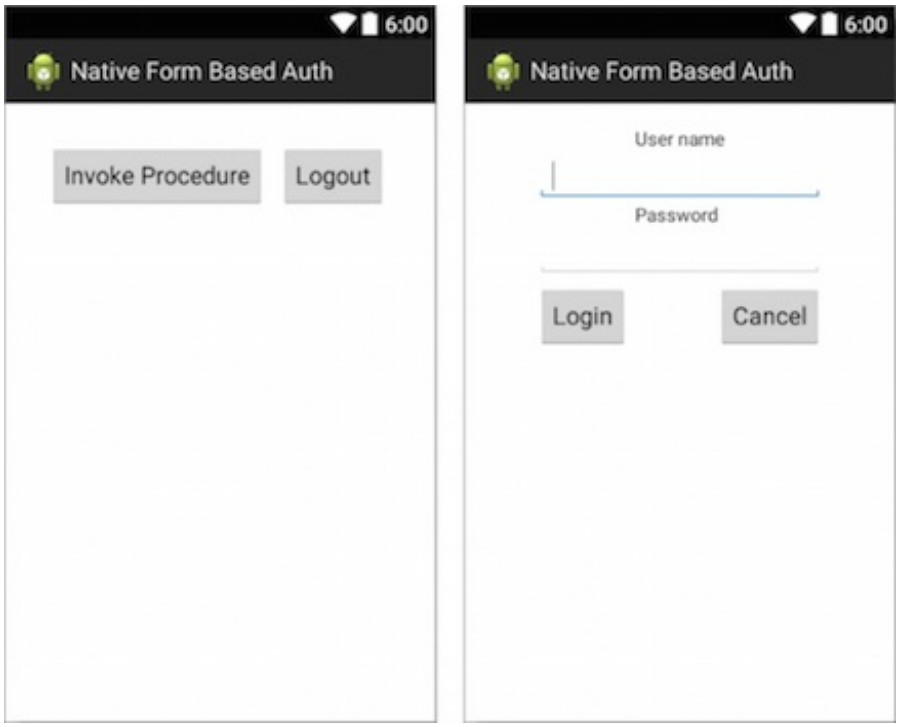
Form-based authentication in native Android applications

Overview

This tutorial explains how to implement the client-side of form-based authentication in native Android.
Prerequisite: Make sure that you read the Form-based authentication (../) tutorial first.

Implementing the client-side authentication

- Create a native Android application and add the MobileFirst native APIs as explained in the Configuring a native Android application with the MobileFirst Platform SDK (../../../hello-world/configuring-a-native-android-application-with-the-mfp-sdk/) tutorial.
- Add an activity which handles and presents a login form.



Challenge Handler

- Create a `MyChallengeHandler` class as a subclass of `ChallengeHandler`.

```
public class AndroidChallengeHandler extends ChallengeHandler
```

- Call the `super` method:

Copy

```
public AndroidChallengeHandler(String realm) {
 super(realm);
}
```

- Add an implementation of the following `ChallengeHandler` methods to handle the form-based challenge:
  1. **`isCustomResponse` method:**
     The `isCustomResponse` method is invoked each time a response is received from the MobileFirst Server. It is used to detect whether the response contains data that is related to this challenge handler. It must return either `true` or `false`.
     The default login form that returns from the MobileFirst Server contains the `j_security_check` string. If the response contains the string, the challenge handler returns `true`.

     Copy

     ```
     public boolean isCustomResponse(WLResponse response) {
        if (response == null || response.getResponseText() == null ||
              response.getResponseText().indexOf("j_security_check") == -1)
     {
           return false;
        }
        return true;
     }
     ```

  2. **`handleChallenge` method:**
     If `isCustomResponse` returns `true`, the framework calls the `handleChallenge` method. This function is used to perform required actions, such as hiding the application screen and showing the login screen.

     Copy

     ```
     public void handleChallenge(WLResponse response){
        if (!isCustomResponse(response)) {
           submitSuccess(response);
        } else {
           cachedResponse = response;
           Intent login = new Intent(parentActivity, LoginFormBasedAuth.class)
     ;
           parentActivity.startActivityForResult(login, 1);
        }
     }
     ```

  3. **`onSuccess` and `onFailure` methods:**
     At the end of the authentication flow, `onSuccess` or `onFailure` will be triggered
     Call the `submitSuccess` method in order to inform the framework that the authentication

process completed successfully and for the `onSuccess` handler of the invocation to be called. Call the `submitFailure` method in order to inform the framework that the authentication process failed and for the `onFailure` handler of the invocation to be called.

Copy

```
public void onFailure(WLFailResponse response) {
 submitFailure(response);
}
public void onSuccess(WLResponse response) {
 submitSuccess(response);
}
```

submitLoginForm

When the user taps to submit the credentials, you need to call the `submitLoginForm` method in order to send the `j_security_check` string and the credentials to the MobileFirst Server.
For example, in here we implemented a `submitLogin method` that called by the MainActivity after the login process is completed.

Copy

```
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("j_username", userName);
      params.put("j_password", password);
      submitLoginForm("/j_security_check", params, null, 0, "post");
   }
}
```

The Main Activity

In the sample project, in order to trigger the challenge handler we use the `WLClient invokeProcedure` method.
The protected procedure invocation triggers MobileFirst Server to send the challenge.

- Create a `WLClient` instance and use the `connect` method to connect to the MobileFirst Server:

  Copy

  ```
  final WLClient client = WLClient.createInstance(this);
  client.connect(new MyConnectionListener());
  ```

- In order to listen to incoming challenges, make sure to register the challenge handler by using the

`registerChallengeHandler` method:

```
challengeHandler = new AndroidChallengeHandler(this, realm);
client.registerChallengeHandler(challengeHandler);
```

- Invoke the protected adapter procedure:

```
URI adapterPath = new URI("/adapters/AuthAdapter/getSecretData");
WLResourceRequest request = new WLResourceRequest(adapterPath,WLResourceRequest.GET);
request.send(new MyResponseListener());
```

Sample application

Click to download (https://github.com/MobileFirst-Platform-Developer-Center/FormBasedAuth) the MobileFirst project.
Click to download (https://github.com/MobileFirst-Platform-Developer-Center/FormBasedAuthAndroid) the Native project.

- The `FormBasedAuth` project contains a MobileFirst native API that you can deploy to your MobileFirst server.
- The `FormBasedAuthAndroid` project contains a native Android application that uses a MobileFirst native API library.
- Make sure to update the `wlclient.properties` file in the native project with the relevant server settings.