Implementing the challenge handler in Android applications

Overview

Prerequisite: Make sure to read the **CredentialsValidationSecurityCheck** challenge handler implementation (../../credentials-validation/android) tutorial.

The challenge handler tutorial demonstrates a few additional features (APIs) such as preemptive `login`, `logout`, and `obtainAccessToken`.

Login

In this example, `UserLogin` expects *key:value*s called `username` and `password`. Optionally, it also accepts a Boolean `rememberMe` key, which tells the security check to remember this user for a longer period. In the sample application, this is collected using a Boolean value from a checkbox in the login form.

The `credentials` argument is a `JSONObject` containing `username`, `password`, and `rememberMe`:

```
submitChallengeAnswer(credentials);
```

You might also want to log in a user without any challenge being received. For example, you can show a login screen as the first screen of the application, or show a login screen after a logout, or a login failure. Those scenarios are called **preemptive logins**.

You cannot call the `submitChallengeAnswer` API if there is no challenge to answer. For those scenarios, the MobileFirst Foundation SDK includes the `login` API:

```
WLAuthorizationManager.getInstance().login(securityCheckName, credentials, new WLLoginResponseListener() {
    @Override
    public void onSuccess() {
        Log.d(securityCheckName, "Login Preemptive Success");

    }

    @Override
    public void onFailure(WLFailResponse wlFailResponse) {
        Log.d(securityCheckName, "Login Preemptive Failure");
    }
});
```

If the credentials are wrong, the security check sends back a **challenge**.

It is the developer's responsibility to know when to use `login`, as opposed to `submitChallengeAnswer`, based on the application's needs. One way to achieve this is to define a Boolean flag, for example `isChallenged`, and set it to `true` when `handleChallenge` is reached, or set it to `false` in any other cases (failure, success, initialization, etc).

When the user clicks the **Login** button, you can dynamically choose which API to use:

```
public void login(JSONObject credentials){
    if(isChallenged){
        submitChallengeAnswer(credentials);
    }
    else{
        WLAuthorizationManager.getInstance().login(securityCheckName, credentials, new WLLoginResponseListener() {
//...
        });
    }
}
```

> **Note:** The `WLAuthorizationManager` `login()` API has its own `onSuccess` and `onFailure` methods, the `handleSuccess` or `handleFailure` methods of the relevant challenge handler are **also** called.

Obtaining an access token

Because this security check supports the **RememberMe** functionality (as the `rememberMe` Boolean key), it would be useful to check whether the client is currently logged in when the application starts.

The MobileFirst Foundation SDK provides the `obtainAccessToken` API to ask the server for a valid token:

```
WLAuthorizationManager.getInstance().obtainAccessToken(scope, new WLAccessTokenListener() {
    @Override
    public void onSuccess(AccessToken accessToken) {
        Log.d(securityCheckName, "auto login success");
    }

    @Override
    public void onFailure(WLFailResponse wlFailResponse) {
        Log.d(securityCheckName, "auto login failure");
    }
});
```

> **Note:** The `WLAuthorizationManager` `obtainAccessToken()` API has its own `onSuccess` and `onFailure` methods, the `handleSuccess` or `handleFailure` methods of the relevant challenge handler are **also** called.

If the client is already logged-in or is in the *remembered* state, the API triggers a success. If the client is not logged in, the security check sends back a challenge.

The `obtainAccessToken` API takes in a **scope**. The scope can be the name of your **security check**.

> Learn more about **scopes** in the Authorization concepts (../../) tutorial

Retrieving the authenticated user

The challenge handler `handleSuccess` method takes a `JSONObject identity` as a parameter. If the security check sets an `AuthenticatedUser`, this object contains the user's properties. You can use `handleSuccess` to save the current user:

```
@Override
public void handleSuccess(JSONObject identity) {
    super.handleSuccess(identity);
    isChallenged = false;
    try {
        //Save the current user
        SharedPreferences preferences = context.getSharedPreferences(Constants.PREFERENCES_FILE, Context.MODE_PRIVATE);
        SharedPreferences.Editor editor = preferences.edit();
        editor.putString(Constants.PREFERENCES_KEY_USER, identity.getJSONObject("user").toString());
        editor.commit();
    } catch (JSONException e) {
        e.printStackTrace();
    }
}
```

Here, `identity` has a key called `user` which itself contains a `JSONObject` representing the `AuthenticatedUser`:

```
{
  "user": {
    "id": "john",
    "displayName": "john",
    "authenticatedAt": 1455803338008,
    "authenticatedBy": "UserLogin"
  }
}
```

Logout

The MobileFirst Foundation SDK also provides a `logout` API to log out from a specific security check:

```
WLAuthorizationManager.getInstance().logout(securityCheckName, new WLLogoutResponseListener() {
    @Override
    public void onSuccess() {
        Log.d(securityCheckName, "Logout Success");
    }

    @Override
    public void onFailure(WLFailResponse wlFailResponse) {
        Log.d(securityCheckName, "Logout Failure");
    }
});
```

Sample applications

Two samples are associated with this tutorial:

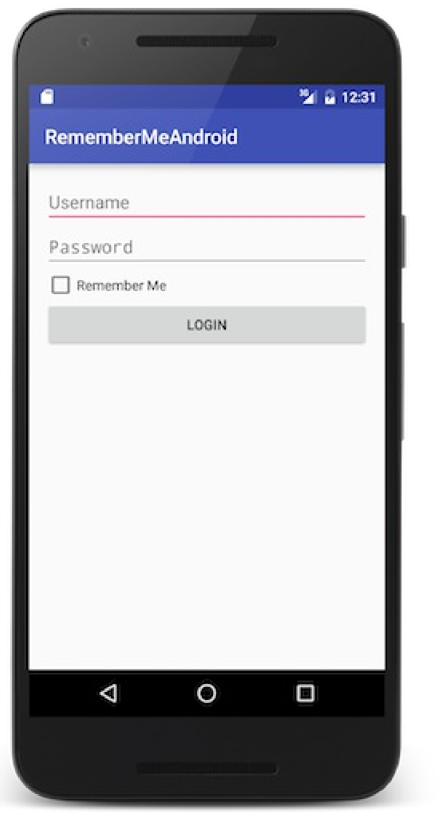
- **PreemptiveLoginAndroid**: An application that always starts with a login screen, using the preemptive `login` API.
- **RememberMeAndroid**: An application with a *Remember Me* checkbox. The user can bypass the login screen the next time the application is opened.

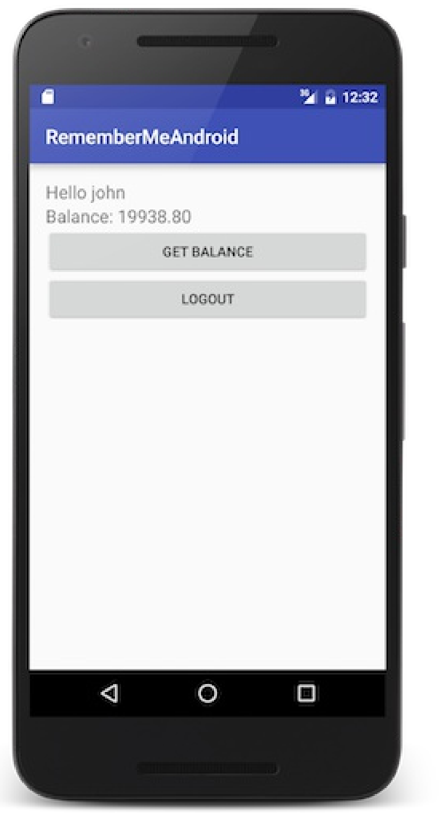
Both samples use the same `UserLogin` security check from the **SecurityCheckAdapters** adapter Maven project.

Click to download (https://github.com/MobileFirst-Platform-Developer-Center/SecurityCheckAdapters/tree/release80) the SecurityCheckAdapters Maven project.
Click to download (https://github.com/MobileFirst-Platform-Developer-Center/RememberMeAndroid/tree/release80) the Remember Me project.
Click to download (https://github.com/MobileFirst-Platform-Developer-Center/PreemptiveLoginAndroid/tree/release80) the Preemptive Login project.

Sample usage

Follow the sample's README.md file for instructions.
The username/password for the app must match, i.e. "john"/"john".





Last modified on

Third party notice (file:////home/travis/build/MFPSamples/DevCenter_site/third-party-notice/)

(https://github.com/MobileFirst-Platform-Developer-Center)

Platform-Developer-Center/DevCenter/blob/master/contributing.m

Report abuse (https://www.ibm.com/developerworks/commu