Implementing the challenge handler in Cordova applications

Overview

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

The challenge handler will demonstrate a few additional features (APIs) such as the 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 that will tell 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.

userLoginChallengeHandler.submitChallengeAnswer({'username':username, 'password':password, reme mberMe: rememberMeState});

You may also want to login a user without any challenge being received. For example, showing a login screen as the first screen of the application, or showing a login screen after a logout, or a login failure. We call those scenarios **preemptive logins**.

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

```
WLAuthorizationManager.login(securityCheckName,{'username':username, 'password':password, remem
berMe: rememberMeState}).then(
    function () {
        WL.Logger.debug("login onSuccess");
    },
    function (response) {
        WL.Logger.debug("login onFailure: " + JSON.stringify(response));
    });
}
```

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

It is the developer's responsibility to know when to use <code>login</code> vs <code>submitChallengeAnswer</code> based on the application's needs. One way to achieve this is to define a boolean flag, for example <code>isChallenged</code>, and set it to <code>true</code> when reaching <code>handleChallenge</code> or set it to <code>false</code> in any other cases (failure, success, initializing, etc).

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

```
if (isChallenged){
    userLoginChallengeHandler.submitChallengeAnswer({'username':username, 'password':password, rem
    emberMe: rememberMeState});
} else {
    WLAuthorizationManager.login(securityCheckName,{'username':username, 'password':password, reme
    mberMe: rememberMeState}).then(
    //...
    );
}
```

Note: WLAuthorizationManager's login() API has its own onSuccess and onFailure methods, the relevant challenge handler's processSuccess or handleFailure will **also** be called.

Obtaining an access token

Since this security check supports *remember me* functionality, it would be useful to check if the client is currently logged in, during the application startup.

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

```
WLAuthorizationManager.obtainAccessToken(userLoginChallengeHandler.securityCheckName).then(
function (accessToken) {
    WL.Logger.debug("obtainAccessToken onSuccess");
    showProtectedDiv();
},
function (response) {
    WL.Logger.debug("obtainAccessToken onFailure: " + JSON.stringify(response));
    showLoginDiv();
});
```

Note: WLAuthorizationManager's obtainAccessToken() API has its own onSuccess and onFailure methods, the relevant challenge handler's processSuccess or handleFailure will **also** be called.

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

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

Learn more about **scope** in the Authorization concepts (../../authorization-concepts) tutorial

Retrieving the authenticated user

The challenge handler's processSuccess method receives a data as a parameter. If the security check sets an AuthenticatedUser, this object will contain the user's properties. You can use processSuccess to save the current user:

```
userLoginChallengeHandler.processSuccess = function(data) {
   WL.Logger.debug("processSuccess");
   isChallenged = false;
   document.getElementById ("rememberMe").checked = false;
   document.getElementById('username').value = "";
   document.getElementById('password').value = "";
   document.getElementById("helloUser").innerHTML = "Hello, " + data.user.displayName;
   showProtectedDiv();
}
```

Here, data 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 Platform Foundation SDK also provides a logout API to logout from a specific security check:

```
WLAuthorizationManager.logout(securityCheckName).then(
   function () {
      WL.Logger.debug("logout onSuccess");
      location.reload();
   },
   function (response) {
      WL.Logger.debug("logout onFailure: " + JSON.stringify(response));
   });
```

Sample applications

There are two samples associated with this tutorial:

- **PreemptiveLoginCordova**: An application that always starts with a login screen, using the preemptive login API.
- RememberMeCordova: 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/RememberMeCordova/tree/release80) the Remember Me project.

Click to download (https://github.com/MobileFirst-Platform-Developer-

Center/PreemptiveLoginCordova/tree/release80) the Preemptive Login project.

Sample usage

- Use either Maven or MobileFirst Developer CLI to build and deploy the available **ResourceAdapter** and **UserLogin** adapters (../../adapters/creating-adapters/).
- From a **Command-line** window, navigate to the project's root folder and run the command: mfpdev app register.
- Map the accessRestricted scope to the UserLogin security check:
 - In the MobileFirst Operations Console, under Applications → [your-application] → Security
 → Map scope elements to security checks, add a scope mapping from accessRestricted to UserLogin.
 - Alternatively, from the **Command-line**, navigate to the project's root folder and run the command: mfpdev app push.

Learn more about the mfpdev app push/push commands in the Using MobileFirst Developer CLI to manage MobilefFirst artifacts (../../using-the-mfpf-sdk/using-mobilefirst-developer-cli-to-manage-mobilefirst-artifacts).



