Implementing the challenge handler in JavaScript (Cordova, Web) applications

Overview

Prerequisite: Make sure to read the **CredentialsValidationSecurityCheck**'s challenge handler implementation (../../credentials-validation/javascript) 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, which tells the security check to remember this user for a longer period. In the sample application, this is collected by a Boolean value from a checkbox in the login form.

```
userLoginChallengeHandler.submitChallengeAnswer({'username':username, 'password':password, rememberMe: 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 Foundation SDK includes the `login` API:

```
WLAuthorizationManager.login(securityCheckName,{'username':username, 'password':password, rememberMe: 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 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:

```
if (isChallenged){
    userLoginChallengeHandler.submitChallengeAnswer({'username':username, 'password':password, rememberMe: rememberMeState});
} else {
    WLAuthorizationManager.login(securityCheckName,{'username':username, 'password':password, rememberMe: rememberMeState}).the
n(
//...
    );
}
```

> **Note:** The `WLAuthorizationManager` `login()` API has its own `onSuccess` and `onFailure` methods, the `processSuccess` 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.obtainAccessToken(userLoginChallengeHandler.securityCheckName).then(
    function (accessToken) {
        WL.Logger.debug("obtainAccessToken onSuccess");
        showProtectedDiv();
    },
    function (response) {
        WL.Logger.debug("obtainAccessToken onFailure: " + JSON.stringify(response));
        showLoginDiv();
});
```

> **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 receives `data` 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:

```
userLoginChallengeHandler.handleSuccess = function(data) {
    WL.Logger.debug("handleSuccess");
    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 Foundation SDK also provides a `logout` API to log out 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

Two samples are associated with this tutorial:

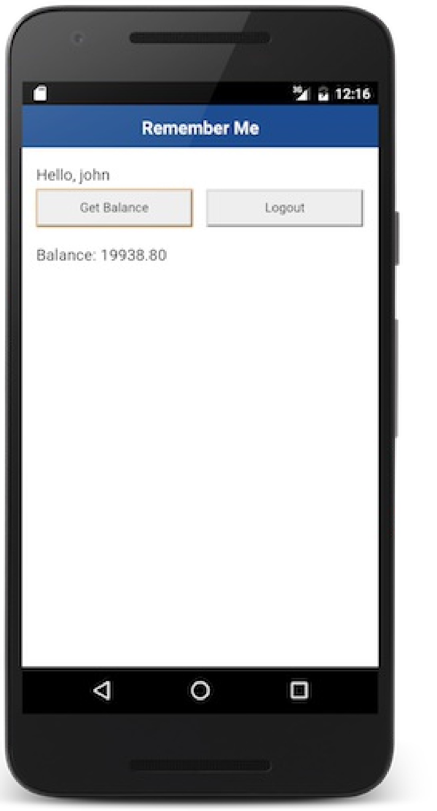
- **PreemptiveLogin**: An application that always starts with a login screen, using the preemptive `login` API.
- **RememberMe**: An application with a *Remember Me* checkbox. The user can bypass the login screen the next time the application is opened.

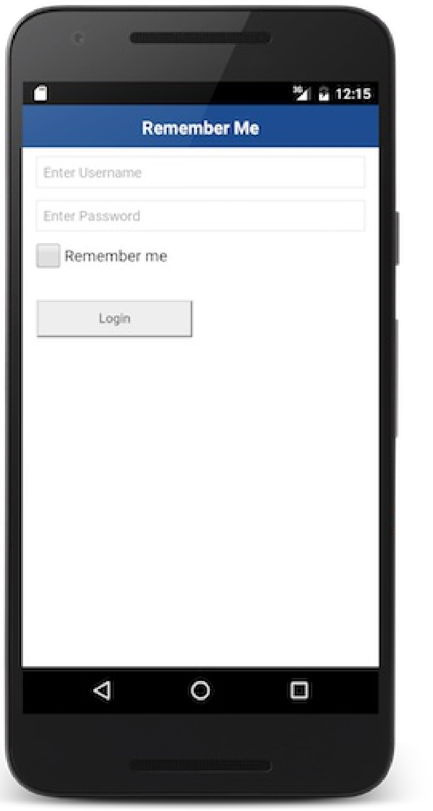
Both samples use the same `UserLogin` security check from the **SecurityCheckAdapters** adapter Maven project.

- 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 RememberMe Cordova project.
- Click to download (https://github.com/MobileFirst-Platform-Developer-Center/PreemptiveLoginCordova/tree/release80) the PreemptiveLogin Cordova project.
- Click to download (https://github.com/MobileFirst-Platform-Developer-Center/RememberMeWeb/tree/release80) the RememberMe Web project.
- Click to download (https://github.com/MobileFirst-Platform-Developer-Center/PreemptiveLoginWeb/tree/release80) the PreemptiveLogin Web 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

IBM
Legal notices
(file:////home/travis/build/MFPSamples/DevCenter/_site/legal-notices/)
Privacy
(http://www.ibm.com/privacy/us/en/)
Terms of use
(file:////home/travis/build/MFPSamples/DevCenter/_site/terms-of-use/)
Third party notice
(file:////home/travis/build/MFPSamples/DevCenter/_site/third-party-notice/)

Social
Facebook
(https://www.facebook.com/ibmmobilefirstplatform/)
Twitter
(https://twitter.com/ibmmobiledev)
YouTube
(https://www.youtube.com/channel/UC7itA4arKsnci2Qusu97Q)
GitHub
(https://github.com/MobileFirst-Platform-Developer-Center)

Site
RSS feed
(file:////home/travis/build/MFPSamples/DevCe
Open issue
(https://github.com/MobileFirst-Platform-Developer-Center/DevCenter/issues/new)
Contribute
(https://github.com/MobileFirst-Platform-Developer-Center/DevCenter/blob/master/contributing.m
Report abuse
(https://www.ibm.com/developerworks/comm