Adapter-based authentication in hybrid applications

fork and edit tutorial (https://github.ibm.com/MFPSamples/DevCenter/tree/master/tutorials/en/foundation/7.0/authentication-security/adapter-based-authentication/adapter-based-authentication-hybrid-applications.html) | report issue (https://github.ibm.com/MFPSamples/DevCenter/issues/new)

This tutorial illustrates the client-side authentication components for adapter-based authentication in hybrid applications.
Prerequisite: Make sure that you read Adapter-based authentication (../) first.

Creating the client-side authentication components

The application consists of two main `div` tags:

- The `AppDiv` element is used to display the application content.
- The `AuthDiv` element is used for authentication forms.

When authentication is required, the application hides the `AppDiv` element and shows the `AuthDiv` element.
When authentication is complete, it does the opposite.

AppDiv

```
<div id="AppDiv">
    <input type="button" id="getSecretDataButton" value="Get secret data" onclick="getSecretData()" />
    <input type="button" class="appButton" value="Logout" onclick="WL.Client.logout('SingleStepAuthRealm', {onSucce
ss:WL.Client.reloadApp})" />
    <div id="ResponseDiv"></div>
</div>
```

The buttons are used to call the `getSecretData` procedure and to log out.
The `ResponseDiv` element is used to display the `getSecretData` response.

AuthDiv

```
 <br />
<div id="AuthDiv" style="display:none">
  <p id="AuthInfo"></p>
    <input type="text" placeholder="Enter username" id="AuthUsername"/>
    <input type="password" placeholder="Enter password" id="AuthPassword"/>
    <input type="button" class="formButton" value="Submit" id="AuthSubmitButton" /><input type="button" class="for
mButton" value="Cancel" id="AuthCancelButton" />
</div>
```

- The `AuthInfo` element displays error messages.
- The `AuthUsername` and `AuthPassword` elements are used to enter credentials.
- The `AuthSubmitButton` and `AuthCancelButton` submits or cancels the authentication request.
- The `AuthDiv` element is styled as `display:none` because it must not be displayed before authentication is requested by the server.

Challenge handler

Use the `WL.Client.createChallengeHandler` method to create a challenge handler object.
Supply a realm name as a parameter.

```
var singleStepAuthRealmChallengeHandler = WL.Client.createChallengeHandler("SingleStepAuthRealm");
```

The `isCustomResponse` function of the challenge handler is called each time a response is received from the server. That

function is used to detect whether the response contains data that is related to this challenge handler. The function returns `true` or `false`.

```
singleStepAuthRealmChallengeHandler.isCustomResponse = function(response) {
    if (!response || !response.responseJSON || response.responseText === null) {
        return false;
    }
    if (typeof(response.responseJSON.authRequired) !== 'undefined'){
        return true;
    } else {
        return false;
    }
};
```

If the `isCustomResponse` function returns `true`, the framework calls the `handleChallenge` function. This function is used to perform required actions, such as hide the application screen or show the login screen.

```
singleStepAuthRealmChallengeHandler.handleChallenge = function(response){
    var authRequired = response.responseJSON.authRequired;
    if (authRequired == true){
        $("#AppDiv").hide();
        $("#AuthDiv").show();
        $("#AuthPassword").empty();
        $("#AuthInfo").empty();
        if (response.responseJSON.errorMessage)
            $("#AuthInfo").html(response.responseJSON.errorMessage);
    } else if (authRequired == false){
        $("#AppDiv").show();
        $("#AuthDiv").hide();
        singleStepAuthRealmChallengeHandler.submitSuccess();
    }
};
```

- If `authRequired` is `true`, the function shows the login screen, cleans up the password field, and shows an error message (if applicable).
- If `authRequired` is `false`, the function shows `AppDiv`, hides `AuthDiv`, and notifies the framework that authentication completed successfully.

In addition to the methods that the developer must implement, the challenge handler contains mandatory challenge handler functions that the developer must use:

- The `submitAdapterAuthentication` function sends collected credentials to a specific adapter procedure. It has the same signature as the `WL.Client.invokeProcedure` API.
- The `submitSuccess` function notifies the framework that the authentication process completed successfully. The framework then automatically issues the original request that triggered authentication.
- The `submitFailure` function notifies the framework that the authentication process completed with failure. The framework then disposes of the original request that triggered authentication.

Important: You **must** attach each of these functions to its object. For example: `myChallengeHandler.submitSuccess()`

Clicking the submit button triggers the function that collects the user name and the password from the HTML input fields and submits them to the adapter. Note the use of the `submitAdapterAuthentication` method.

```
$("#AuthSubmitButton").bind('click', function () {
    var username = $("#AuthUsername").val();
    var password = $("#AuthPassword").val();
    var invocationData = {
        adapter : "SingleStepAuthAdapter",
        procedure : "submitAuthentication",
        parameters : [ username, password ]
    };
    singleStepAuthRealmChallengeHandler.submitAdapterAuthentication(invocationData, {})
;
});
```

Sample application

Click to download (http://public.dhe.ibm.com/software/products/en/MobileFirstPlatform/docs/v700/AdapterBasedAuthenticationHybridProject.zip) the Studio project.