Form-based authentication in hybrid applications

Overview

This tutorial illustrates the JavaScript client-side authentication components for form-based authentication.
Prerequisite: Make sure that you read Form-based authentication (../) first.

This tutorial covers the following topics:

- Creating the client-side authentication components
- Sample application

Creating the client-side authentication components

The application consists of two main `div` elements:

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

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

AppDiv

```
<div id="AppDiv">
  <input type="button" id="getSecretDataButton" value="Call protected adapter procedure" onclick="getS
ecretData()" />
  <input type="button" class="appButton" value="Logout" onclick="WL.Client.logout('SampleAppRealm', {
onSuccess:  WL.Client.reloadApp});" />
  <div id="ResponseDiv"></div>
</div>
```

The buttons are used to call the `getSecretData` procedure and to log out.

AuthDiv

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

The `AuthDiv` element is styled as `display:none` because it must not be displayed before authentication is requested by the server.

Challenge handler

1. Use the `WL.Client.createChallengeHandler()` method to create a challenge handler object. The realm name is a mandatory parameter.

   ```
   var sampleAppRealmChallengeHandler = WL.Client.createChallengeHandler("SampleAppRealm"
   );
   ```

   The `isCustomResponse` function of the challenge handler is invoked each time that a response is received from the 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`.

   ```
   sampleAppRealmChallengeHandler.isCustomResponse = function(response) {
      return false;
   };
   ```

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

   ```
   sampleAppRealmChallengeHandler.handleChallenge = function(response) {
   <p>};
   ```

2. Create a challenge handler to define a customized authentication flow. In your challenge handler, do not add code that modifies the user interface when this modification is not related to the authentication flow.
   In addition to the methods that the developer must implement, the challenge handler contains functionality that the developer might want to use:

   - `submitLoginForm()` sends the collected credentials to a specific URL. The developer can also specify request parameters, headers, and callback.
   - `submitSuccess()` notifies the MobileFirst framework that the authentication successfully finished. The framework then automatically issues the original request that triggered the authentication.
   - `submitFailure()` notifies the MobileFirst framework that the authentication process completed with failure. The framework then disposes of the original request that triggered the authentication.

   Important: Each function must be attached to its object. For example:
   `SampleAppRealmChallengeHandler.submitSuccess()`

The default login form that is returned from the MobileFirst server contains the `j_security_check` string. If the challenge handler detects it in the response, it returns `true`.

```
sampleAppRealmChallengeHandler.isCustomResponse = function(response) {
    if (!response || response.responseText === null) {
        return false;
    }
    var indicatorIdx = response.responseText.search('j_security_check');
    if (indicatorIdx >= 0){
        return true;
    }
    return false;
};
```

After the client application detects that the server sent a login form, which means that the server is requesting authentication, the application hides the `AppDiv` element, shows the `AuthDiv` element, and cleans up the `AuthPassword`.

```
sampleAppRealmChallengeHandler.handleChallenge = function(response) {
    $('#AppDiv').hide();
    $('#AuthDiv').show();
    $('#AuthPassword').val('');
};
```

Clicking the `login` button triggers a function that collects the user name and password from the HTML input fields and submits them to the server.
It is possible to set request headers here, and specify callback.
The form-based authenticator uses a hardcoded `j_security_check` URL component. You cannot have more than one instance of it.

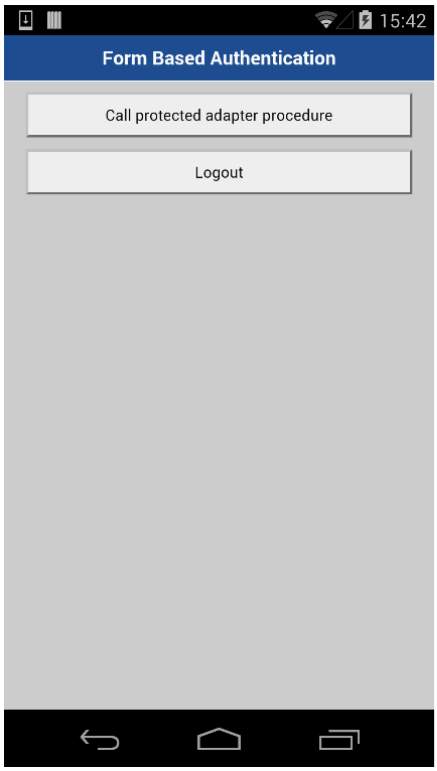
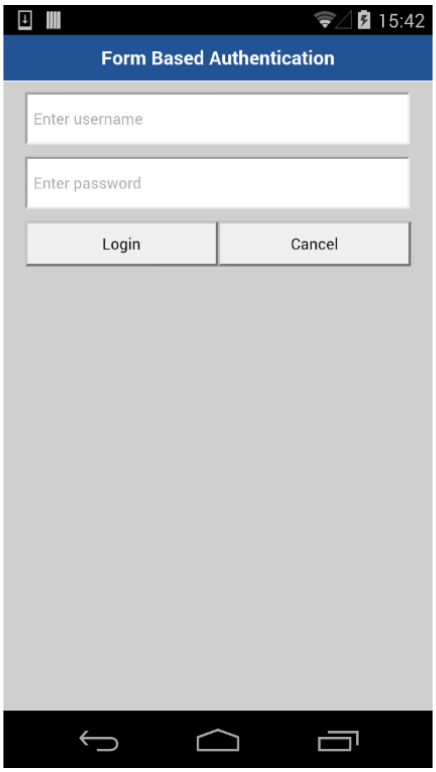
```
$('#AuthSubmitButton').bind('click', function () {
    var reqURL = '/j_security_check';
    var options = {};
    options.parameters = {
        j_username : $('#AuthUsername').val(),
        j_password : $('#AuthPassword').val()
    };
    options.headers = {};
    sampleAppRealmChallengeHandler.submitLoginForm(reqURL, options, sampleAppRealmChallengeH
andler.submitLoginFormCallback);
});
```

Clicking the cancel button hides the `authDiv` element, shows the `appDiv` element, and notifies the framework that authentication failed.

```
$('#AuthCancelButton').bind('click', function () {
    sampleAppRealmChallengeHandler.submitFailure()
;
    $('#AppDiv').show();
    $('#AuthDiv').hide();
});
```

The callback function checks the response for the containing server challenge again. If a challenge is found, the `handleChallenge()` function is called again.
No challenge present in the server response means that the authentication process successfully completed. In this case, `AppDiv` is shown, `AuthDiv` is hidden, and the framework is notified about the authentication success.

```
sampleAppRealmChallengeHandler.submitLoginFormCallback = function(response) {
    var isLoginFormResponse = sampleAppRealmChallengeHandler.isCustomResponse(response)
;
    if (isLoginFormResponse){
        sampleAppRealmChallengeHandler.handleChallenge(response);
    } else {
        $('#AppDiv').show();
        $('#AuthDiv').hide();
        sampleAppRealmChallengeHandler.submitSuccess();
    }
};
```

Sample application

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