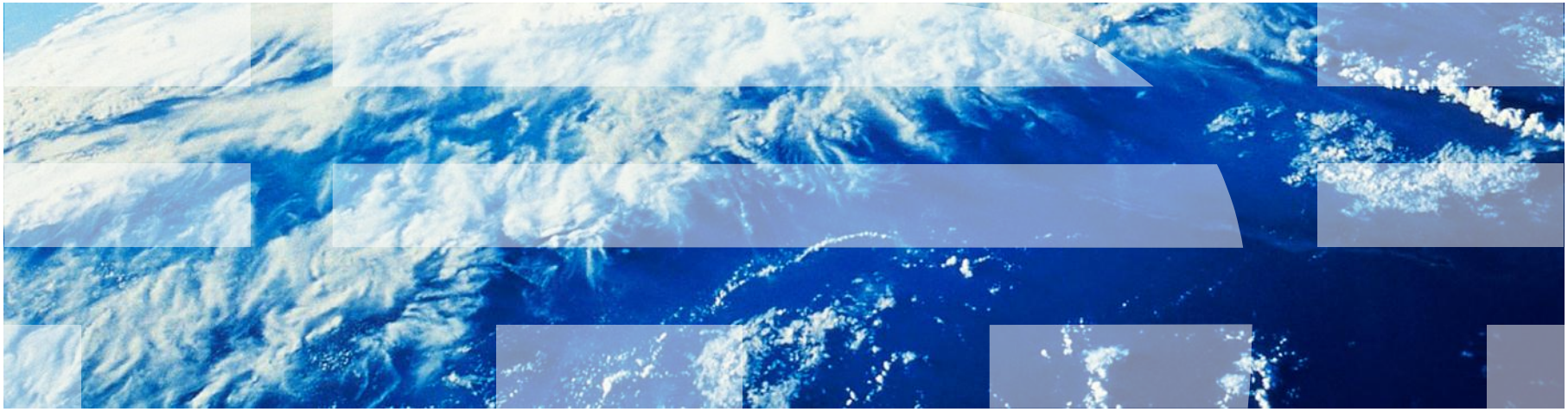
IBM Worklight Foundation V6.2.0 Getting Started

Adapter-based authentication in hybrid applications



20 June 2014

Trademarks

- IBM, the IBM logo, ibm.com, and Worklight are trademarks or registered trademarks of International Business Machines Corporation, registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at www.ibm.com/legal/copytrade.shtml.
- Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.
- Other company products or service names may be trademarks or service marks of others.
- This document may not be reproduced in whole or in part without the prior written permission of IBM.

About IBM®

- See http://www.ibm.com/ibm/us/en/

Agenda

- Introduction to adapter-based authentication
- Configuring the authenticationConfig.xml file
- Creating the server-side authentication components
- Creating the client-side authentication components
- Examining the result
- Exercise
- Quiz

Introduction to adapter-based authentication

- Adapter-based authentication is the most flexible type of authentication to implement and contains all the benefits of the Worklight® Server authentication framework.
- When you use adapter-based authentication, you can implement the entire authentication logic, including validation of the credentials, an adapter by using plain JavaScript™.
- Nevertheless, you can also use any login module as an extra authentication layer.
- In this module, you implement an adapter-based authentication mechanism that relies on a user name and a password.

Agenda

- Introduction to adapter-based authentication
- Configuring the authenticationConfig.xml file
- Creating the server-side authentication components
- Creating the client-side authentication components
- Examining the result
- Exercise
- Quiz

Configuring the authenticationConfig.xml file

- Add two authentication realms to the `<realms>` section of the **authenticationConfig.xml** file.

```
<realm loginModule="AuthLoginModule" name="SingleStepAuthRealm">
    <className>com.worklight.integration.auth.AdapterAuthenticator</className>
    <parameter name="login-function" value="SingleStepAuthAdapter.onAuthRequired"/>
    <parameter name="logout-function" value="SingleStepAuthAdapter.onLogout"/>
</realm>
<realm loginModule="AuthLoginModule" name="DoubleStepAuthRealm">
    <className>com.worklight.integration.auth.AdapterAuthenticator</className>
    <parameter name="login-function" value="DoubleStepAuthAdapter.onAuthRequired"/>
    <parameter name="logout-function" value="DoubleStepAuthAdapter.onLogout"/>
</realm>
```

- These realms use the `AuthLoginModule` login module, which is defined later.

Configuring the authenticationConfig.xml file

- Add two authentication realms to the `<realms>` section of the **authenticationConfig.xml** file.

```
<realm loginModule="AuthLoginModule" name="SingleStepAuthRealm">
    <className>com.worklight.integration.auth.AdapterAuthenticator</className>
    <parameter name="login-function" value="SingleStepAuthAdapter.onAuthRequired"/>
    <parameter name="logout-function" value="SingleStepAuthAdapter.onLogout"/>
</realm>
<realm loginModule="AuthLoginModule" name="DoubleStepAuthRealm">
    <className>com.worklight.integration.auth.AdapterAuthenticator</className>
    <parameter name="login-function" value="DoubleStepAuthAdapter.onAuthRequired"/>
    <parameter name="logout-function" value="DoubleStepAuthAdapter.onLogout"/>
</realm>
```

- Using the `com.worklight.integration.auth.AdapterAuthenticator` class means that the server-side part of the authenticator is defined in the adapter.

Configuring the authenticationConfig.xml file

- Add two authentication realms to the `<realms>` section of the **authenticationConfig.xml** file.

```
<realm loginModule="AuthLoginModule" name="SingleStepAuthRealm">
    <className>com.worklight.integration.auth.AdapterAuthenticator</className>
    <parameter name="login-function" value="SingleStepAuthAdapter.onAuthRequired"/>
    <parameter name="logout-function" value="SingleStepAuthAdapter.onLogout"/>
</realm>
<realm loginModule="AuthLoginModule" name="DoubleStepAuthRealm">
    <className>com.worklight.integration.auth.AdapterAuthenticator</className>
    <parameter name="login-function" value="DoubleStepAuthAdapter.onAuthRequired"/>
    <parameter name="logout-function" value="DoubleStepAuthAdapter.onLogout"/>
</realm>
```

- Whenever the Worklight authentication framework detects an attempt to access a protected resource, an adapter function that is defined in a **login-function parameter** is called automatically.
- When logout is detected (explicit or session timeout), a **logout-function** is called automatically.
- In both cases, the parameter value syntax is `adapterName.functionName`.

Configuring the authenticationConfig.xml file

- Add a login module to the `<loginModules>` section of the **authenticationConfig.xml** file and call it `AuthLoginModule`.

```
<loginModule name="AuthLoginModule">
    <className>com.worklight.core.auth.ext.NonValidatingLoginModule</className>
</loginModule>
```

- Using a `NonValidatingLoginModule` class name means that no additional validation is performed by the Worklight platform, and the developer takes responsibility for the validation of credentials within the adapter.
- Because all authentication-related actions are done in the adapter code, using `NonValidatingLoginModule` is mandatory for adapter-based authentication.

Configuring the authenticationConfig.xml file

- Add security tests to the `<securityTests>` section of the **authenticationConfig.xml** file.
- You must use this security test to protect the adapter procedure, so use the `<customSecurityTest>` element.

```
<customSecurityTest name="SingleStepAuthAdapter-securityTest">
    <test isInternalUserID="true" realm="SingleStepAuthRealm"/>
</customSecurityTest>
<customSecurityTest name="DoubleStepAuthAdapter-securityTest">
    <test isInternalUserID="true" realm="DoubleStepAuthRealm"/>
</customSecurityTest>
```

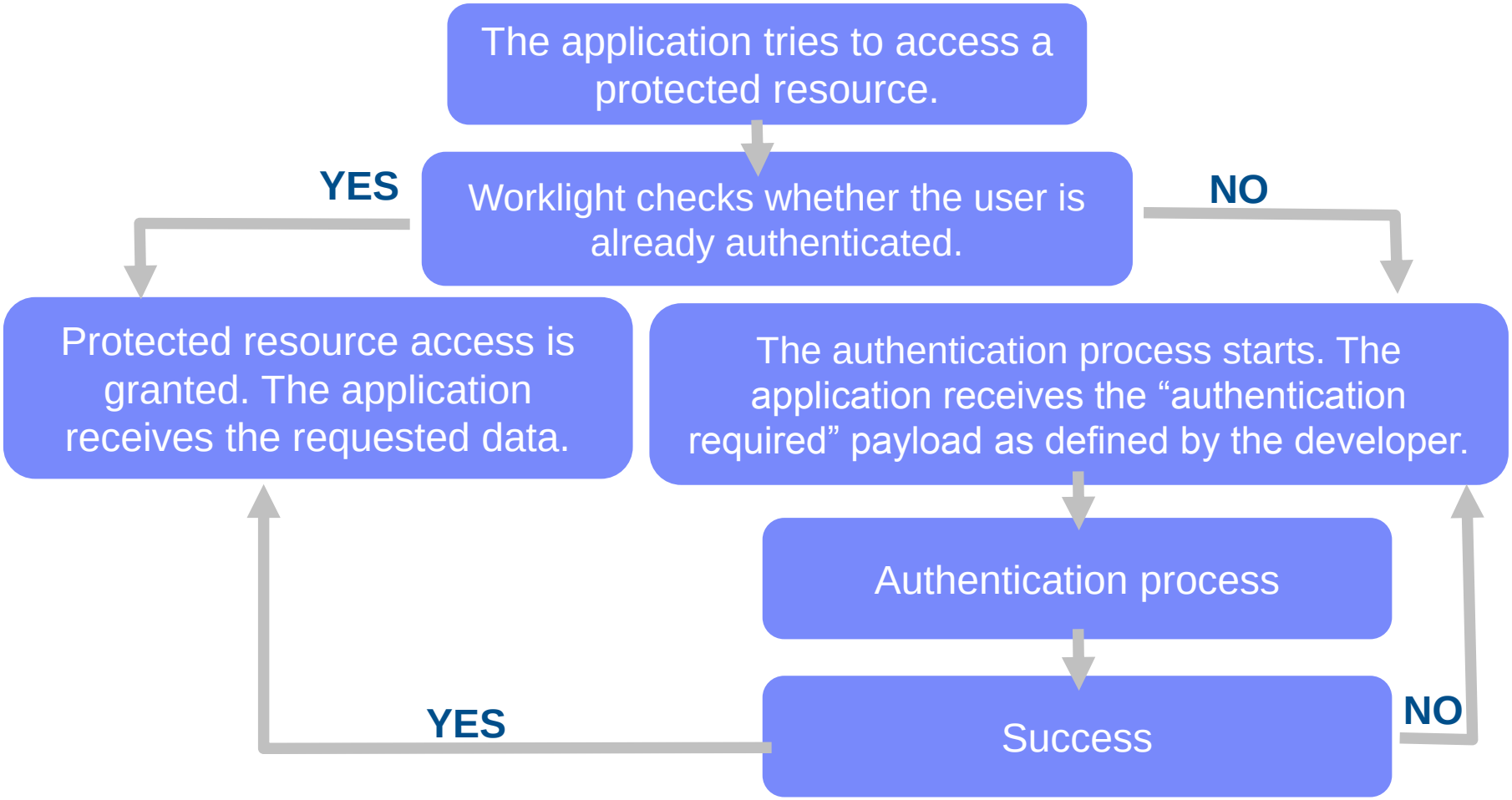
- Remember the security test names. You must use them in subsequent slides.

Agenda

- Introduction to adapter-based authentication
- Configuring the authenticationConfig.xml file
- Creating the server-side authentication components
- Creating the client-side authentication components
- Examining the result
- Exercise
- Quiz

Creating the server-side authentication components

- The following diagram illustrates the adapter-based authentication process:



Creating the server-side authentication components

- The sample that is provided with this training module uses two applications and two adapters. The next slides focus on the SingleStepAuth application and adapter. The DoubleStepAuth application and adapter are just an extension of the same technique.
- Create an adapter that takes care of the authentication process. Name it **SingleStepAuthAdapter.**
- **SingleStepAuthAdapter** includes the following two procedures:

```
<procedure name="submitAuthentication"/>

<procedure name="getSecretData" securityTest="AdapterSecurityTest"/>
```

- The `submitAuthentication` procedure takes care of the authentication process and authentication is not required to call it.
- The second procedure, however, is available to authenticated users only.

Creating the server-side authentication components

- The following diagram shows the flow to implement:

Creating the server-side authentication components

- Whenever the Worklight framework detects an unauthenticated attempt to access a protected resource, the `onAuthRequired` function is called, as defined in the **authenticationConfig.xml** file).

```
function onAuthRequired(headers, errorMessage){
    errorMessage = errorMessage ? errorMessage : null;
    return {
        authRequired: true,
        errorMessage: errorMessage
    };
}
```

This object is a **custom** challenge object that is sent to the application.

- This function receives the response headers and an optional `errorMessage` parameter. The object that is returned by this function is sent to the client application.
- Note the `authRequired: true` property. You use this property in a challenge handler to detect that the server is requesting authentication.

Creating the server-side authentication components

- The `submitAuthentication` function is called by a client application to validate the user name and password.

```
function submitAuthentication(username, password){
    if (username==="worklight" && password === "worklight"){

        var userIdentity = {
                userId: username,
                displayName: username,
                attributes: {
                    foo: "bar"
                }
        };

        WL.Server.setActiveUser("SingleStepAuthRealm", userIdentity);

        return {
            authRequired: false
        };
    }

    return onAuthRequired(null, "Invalid login credentials");
}
```

The user name and password are received from the application as parameters.

Creating the server-side authentication components

- The `submitAuthentication` function is called by a client application to validate the user name and password.

```
function submitAuthentication(username, password){
    if (username==="worklight" && password === "worklight"){

        var userIdentity = {
                userId: username,
                displayName: username,
                attributes: {
                    foo: "bar"
                }
        };

        WL.Server.setActiveUser("SingleStepAuthRealm", userIdentity);

        return {
            authRequired: false
        };
    }

    return onAuthRequired(null, "Invalid login credentials");
}
```

In this sample, the credentials are validated against some hardcoded values, but any other validation mode is valid, for example by using SQL or web services.

Creating the server-side authentication components

- The `submitAuthentication` function is called by a client application to validate the user name and password.

```
function submitAuthentication(username, password){
    if (username==="worklight" && password === "worklight"){

        var userIdentity = {
                userId: username,
                displayName: username,
                attributes: {
                    foo: "bar"
                }
        };

        WL.Server.setActiveUser("SingleStepAuthRealm", userIdentity);

        return {
            authRequired: false
        };
    }

    return onAuthRequired(null, "Invalid login credentials");
}
```

If the validation passed successfully, the WL.Server.setActiveUser method is called to create an authenticated session for the SingleStepAuthRealm, with user data stored in a userIdentity object. You can add your own custom properties to the user identity attributes.

Creating the server-side authentication components

- The `submitAuthentication` function is called by a client application to validate the user name and password.

```
function submitAuthentication(username, password){
    if (username==="worklight" && password === "worklight"){

        var userIdentity = {
                userId: username,
                displayName: username,
                attributes: {
                    foo: "bar"
                }
        };

        WL.Server.setActiveUser("SingleStepAuthRealm", userIdentity);

        return {
            authRequired: false
        };
    }

    return onAuthRequired(null, "Invalid login credentials");
}
```

An object is sent to the application, stating that the authentication screen is no longer required.

Creating the server-side authentication components

- The `submitAuthentication` function is called by a client application to validate the user name and password.

```
function submitAuthentication(username, password){
    if (username==="worklight" && password === "worklight"){

        var userIdentity = {
                userId: username,
                displayName: username,
                attributes: {
                    foo: "bar"
                }
        };

        WL.Server.setActiveUser("SingleStepAuthRealm", userIdentity);

        return {
            authRequired: false
        };
    }

    return onAuthRequired(null, "Invalid login credentials");
}
```

If the credentials validation fails, an object that is built by the onAuthRequired function is returned to the application with a suitable error message.

Creating the server-side authentication components

- For training purposes, the `getSecretData` function returns a hardcoded value. Keep in mind that `getSecretData` is protected by a security test, as defined in the adapter XML.
- The `onLogout` function is defined in the **authenticationConfig.xml** file to be called automatically on logout, for example to perform a cleanup.

```
function getSecretData(){
    return {
        secretData: "A very very very very secret data"
    };
}

function onLogout(){
    WL.Logger.debug("Logged out");
}
```

Agenda

- Introduction to adapter-based authentication
- Configuring the authenticationConfig.xml file
- Creating the server-side authentication components
- Creating the client-side authentication components
- Examining the result
- Exercise
- Quiz

Creating the client-side authentication components

- Create a Worklight application.
- The application consists of two main `<div>` elements:
  - The **`<div id="AppDiv">`** element is used to display the application content.
  - The **`<div id="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.

Creating the client-side authentication components

- Start by creating an `AppDiv` element.
- It has a basic structure and functions:

```
<div id="AppDiv">
    <div class="header">
        <h1>Single Step Adapter Based Authentication</h1>
    </div>
    <input type="button" value="Get secret data" onclick="getSecretData()" />
    <input type="button" value="Logout" onclick="WL.Client.logout('SingleStepAuthRealm', {onSuccess:WL.Client.reloadApp})" />
    <div id="ResponseDiv"></div>
</div>
```

- The buttons are used to call the `getSecretData` procedure and to log out.
- The `<div id="ResponseDiv">` element is used to display the `getSecretData` response.

Creating the client-side authentication components

- The `AuthDiv` element contains the following subelements:

```
<div id="AuthDiv" style="display:none">
    <div class="header">
        <h1>Single Step Adapter Based Authentication</h1>
    </div>
    <p id="AuthInfo"></p>
    <input type="text" placeholder="Enter username" id="AuthUsername"/><br />
    <input type="password" placeholder="Enter password" id="AuthPassword"/><br />
    <input type="button" value="Submit" id="AuthSubmitButton" />
    <input type="button" value="Cancel" id="AuthCancelButton" />
</div>
```

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

Creating the client-side authentication components

- Finally, create a challenge handler.
- Use the following API to create this handler and implement its functionality.

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

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

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

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.

Creating the client-side authentication components

- Finally, create a challenge handler.
- Use the following API to create this handler and implement its functionality.

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

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

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`.

Creating the client-side authentication components

- Finally, create a challenge handler.
- Use the following API to create this handler and implement its functionality.

```
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 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.

Creating the client-side authentication components

- Finally, create a challenge handler.
- Use the following API to create this handler and implement its functionality.

```
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`, it shows the login screen, cleans up the password field, and shows an error message (if applicable).

Creating the client-side authentication components

- Finally, create a challenge handler.
- Use the following API to create this handler and implement its functionality.

```
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.error

    } else if (authRequired == false){
        $("#AppDiv").show();
        $("#AuthDiv").hide();
        singleStepAuthRealmChallengeHandler.submitSuccess();
    }
};
```

If `authRequired` is `false`, it shows AppDiv, hides AuthDiv, and notifies the Worklight framework that authentication completed successfully.

Creating the client-side authentication components

- In addition to the methods that the developer must implement, the challenge handler contains functionalities that the developer might want to use:
  - The `submitAdapterAuthentication` function is used to send collected credentials to a specific adapter procedure. It has the same signature as the `WL.Client.invokeProcedure` API.
  - The `submitSuccess` function notifies the Worklight framework that the authentication process completed successfully. The Worklight framework then automatically issues the original request that triggered authentication.
  - The `submitFailure` function notifies the Worklight framework that the authentication process completed with failure. The Worklight framework then disposes of the original request that triggered authentication.

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

Creating the client-side authentication components

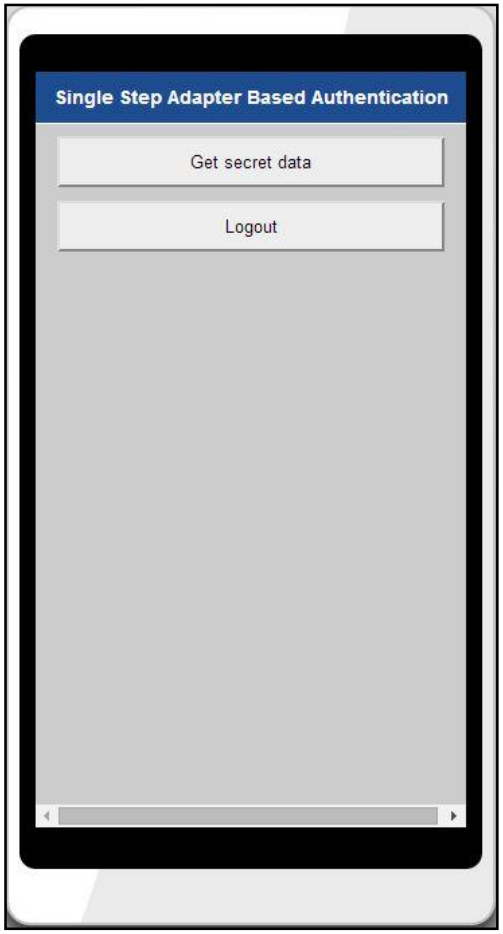
- Clicking the submit button triggers the function that collects the user name and the password from the HTML input fields and submits them to the adapter.
- Note that the challenge handler uses the `submitAdapterAuthentication` method.

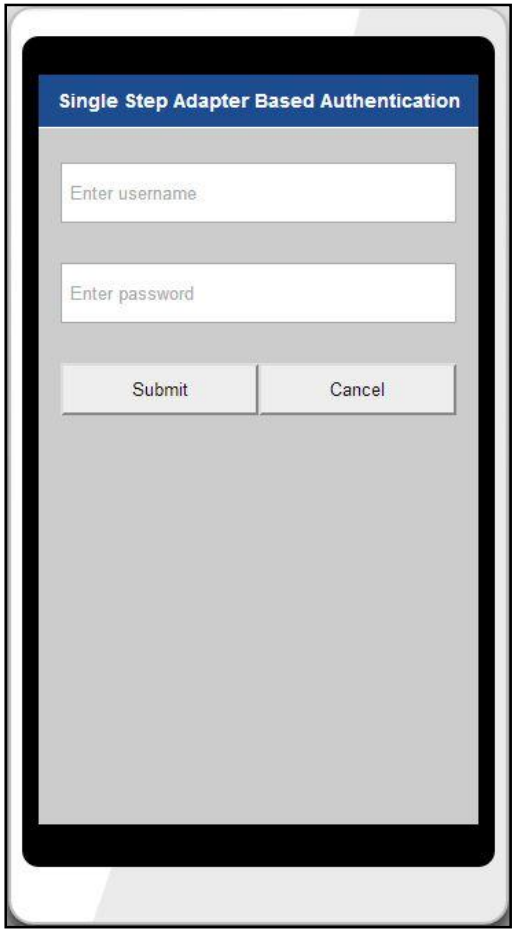
```
$("#AuthSubmitButton").bind('click', function () {
    var username = $("#AuthUsername").val();
    var password = $("#AuthPassword").val();

    var invocationData = {
        adapter : "SingleStepAuthAdapter",
        procedure : "submitAuthentication",
        parameters : [ username, password ]
    };

    singleStepAuthRealmChallengeHandler.submitAdapterAuthentication(invocationData, {});
});
```

Agenda

- Introduction to adapter-based authentication
- Configuring the authenticationConfig.xml file
- Creating the server-side authentication components
- Creating the client-side authentication components
- Examining the result
- Exercise
- Quiz

Examining the result





Agenda

- Introduction to adapter-based authentication
- Configuring the authenticationConfig.xml file
- Creating the server-side authentication components
- Creating the client-side authentication components
- Examining the result
- Exercise
- Quiz

Exercise

- Implement the adapter authentication as described in this training module.
- You can find the sample for this training module in the **Getting Started** page of the IBM® Worklight ® Foundation sdocumentation website at http://www.ibm.com/mobile-docs.

Agenda

- Adapter-based authentication introduction
- Configuring the authenticationConfig.xml file
- Creating the server-side authentication components
- Creating the client-side authentication components
- Examining the result
- Exercise
- Quiz

Quiz
Test your knowledge – Answers on the next slide

- When you define a realm that uses an adapter-based authentication in the authenticationConfig.xml, which two parameters are mandatory?
  - The login-function, the logout-function.
  - The adapter-name, the realm-name.
  - The adapter-name, the login-function.
  - The login-function, the login-module.
- How can a developer specify which adapter procedures are protected by an authentication realm?
  - When the authentication realm is specified in the adapter XML file, all the adapter procedures are protected by it.
  - The developer does not have to specify it. Authentication credentials are added on the client side when you use WL.Client.invokeProcedure for the procedure to work.
  - By adding a securityTest property to the procedure definition in the adapter XML.
  - You cannot protect the adapter procedures by an authentication realm. The protection is for applications only.
- What client-side mechanism is used to detect that the server requires authentication for the client request?
  - The challengeHandler.isAuthenticationRequired
  - The challengeHandler.isUserAuthenticated
  - The challengeHandler.analyzeServerResponse
  - The challengeHandler.isCustomResponse

Quiz - Answers

- When you define a realm that uses an adapter-based authentication in the authenticationConfig.xml, which two parameters are mandatory?
  - The login-function, the logout-function.
  - The adapter-name, the realm-name.
  - The adapter-name, the login-function.
  - The login-function, the login-module.
- How can a developer specify which adapter procedures are protected by an authentication realm?
  - When the authentication realm is specified in the adapter XML file, all the adapter procedures are protected by it.
  - The developer does not have to specify it. Authentication credentials are added on the client side when you use WL.Client.invokeProcedure for the procedure to work.
  - By adding a securityTest property to the procedure definition in the adapter XML.
  - You cannot protect the adapter procedures by an authentication realm. The protection is for applications only.
- What client side mechanism is used to detect that the server requires authentication for the client request?
  - The challengeHandler.isAuthenticationRequired
  - The challengeHandler.isUserAuthenticated
  - The challengeHandler.analyzeServerResponse
  - The challengeHandler.isCustomResponse

Notices

- Permission for the use of these publications is granted subject to these terms and conditions.
- This information was developed for products and services offered in the U.S.A.
- IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.
- IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:
  - IBM Director of Licensing
    IBM Corporation
    North Castle Drive
    Armonk, NY 10504-1785
    U.S.A.
- For license inquiries regarding double-byte character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:
  - Intellectual Property Licensing
    Legal and Intellectual Property Law
    IBM Japan Ltd.
    1623-14, Shimotsuruma, Yamato-shi
    Kanagawa 242-8502 Japan
- **The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:** INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.
- This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.
- Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.
- IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.
- Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:
  - IBM Corporation
    Dept F6, Bldg 1
    294 Route 100
    Somers NY 10589-3216
    USA
- Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.
- The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.
- Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

COPYRIGHT LICENSE:

- This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.
- Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:
  - © (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. _enter the year or years_. All rights reserved.

Privacy Policy Considerations

- IBM Software products, including software as a service solutions, (“Software Offerings”) may use cookies or other technologies to collect product usage information, to help improve the end user experience, to tailor interactions with the end user or for other purposes. In many cases no personally identifiable information is collected by the Software Offerings. Some of our Software Offerings can help enable you to collect personally identifiable information. If this Software Offering uses cookies to collect personally identifiable information, specific information about this offering’s use of cookies is set forth below.
- Depending upon the configurations deployed, this Software Offering may use session cookies that collect session information (generated by the application server). These cookies contain no personally identifiable information and are required for session management. Additionally, persistent cookies may be randomly generated to recognize and manage anonymous users. These cookies also contain no personally identifiable information and are required.
- If the configurations deployed for this Software Offering provide you as customer the ability to collect personally identifiable information from end users via cookies and other technologies, you should seek your own legal advice about any laws applicable to such data collection, including any requirements for notice and consent. For more information about the use of various technologies, including cookies, for these purposes, see IBM’s Privacy Policy at http://www.ibm.com/privacy and IBM’s Online Privacy Statement at http://www.ibm.com/privacy/details the sections entitled “Cookies, Web Beacons and Other Technologies” and the “IBM Software Products and Software-as-a-Service Privacy Statement” at http://www.ibm.com/software/info/product-privacy.

Support and comments

- For the entire IBM Worklight documentation set, training material and online forums where you can post questions, see the IBM website at:
  - http://www.ibm.com/mobile-docs
- **Support**
  - Software Subscription and Support (also referred to as Software Maintenance) is included with licenses purchased through Passport Advantage and Passport Advantage Express. For additional information about the International Passport Advantage Agreement and the IBM International Passport Advantage Express Agreement, visit the Passport Advantage website at:
    - http://www.ibm.com/software/passportadvantage
  - If you have a Software Subscription and Support in effect, IBM provides you assistance for your routine, short duration installation and usage (how-to) questions, and code-related questions. For additional details, consult your IBM Software Support Handbook at:
    - http://www.ibm.com/support/handbook
- **Comments**
  - We appreciate your comments about this publication. Please comment on specific errors or omissions, accuracy, organization, subject matter, or completeness of this document. The comments you send should pertain to only the information in this manual or product and the way in which the information is presented.
  - For technical questions and information about products and prices, please contact your IBM branch office, your IBM business partner, or your authorized remarketer.
  - When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you. IBM or any other organizations will only use the personal information that you supply to contact you about the issues that you state.
  - Thank you for your support.
  - Submit your comments in the IBM Worklight Developer Edition support community at:
    - https://www.ibm.com/developerworks/mobile/worklight/connect.html
  - If you would like a response from IBM, please provide the following information:
    - Name
    - Address
    - Company or Organization
    - Phone No.
    - Email address

Thank You