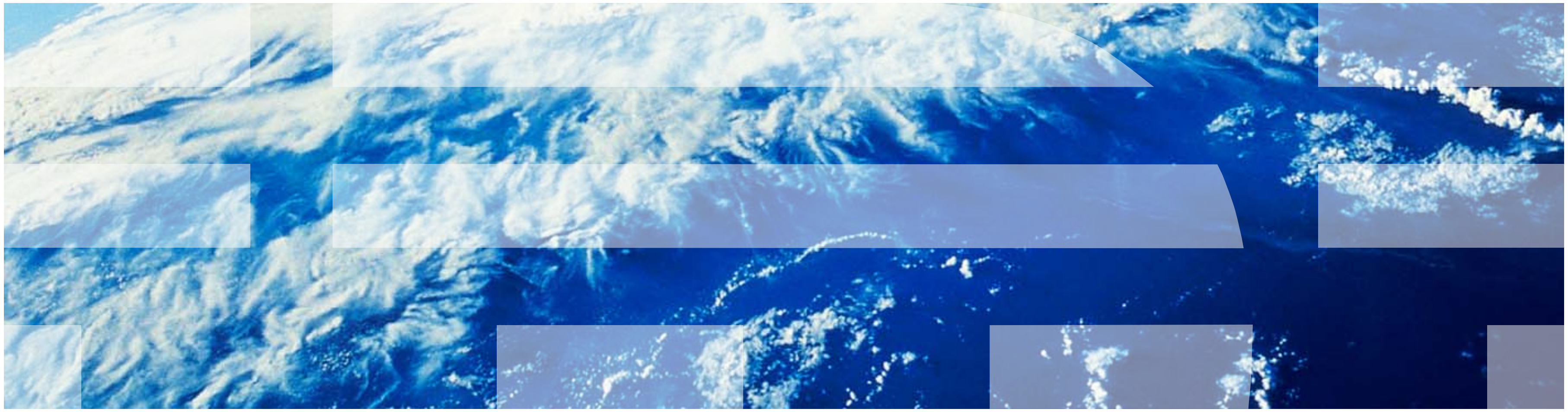
IBM Worklight V6.1.0
Getting Started

Adapter-based authentication



7 March 2014

Trademarks

About IBM®

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

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
- Check yourself questions

Adapter-based authentication introduction

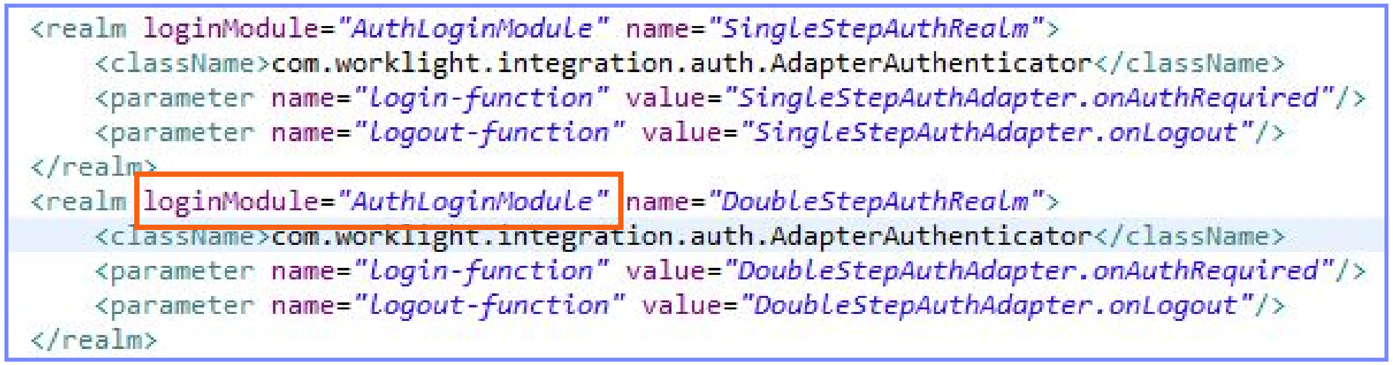
- Adapter-based authentication is the most flexible type of authentication to implement and contains all the benefits of the Worklight® Server authentication framework.
- When you use the adapter-based authentication, the entire authentication logic, including the credentials validation, can be implemented in an adapter by using plain JavaScript™.
- Nevertheless, any login module can be used in the adapter-based authentication 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

- 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
- Check yourself questions

Configuring the authenticationConfig.xml file

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

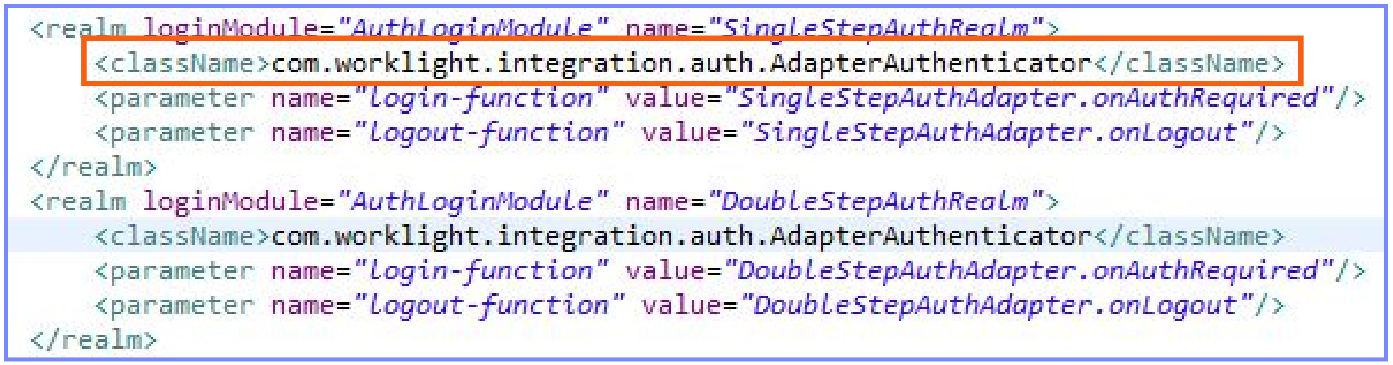


```
<realm loginModule="AuthLoginModule" name="SingleStepAuthRealm">
    <className>com.worklight.integration.auth.AdapterAuthenticator</className>
    <parameter name="login-function" value="SingleStepAuthAdapter.onAuthRequired"/>
    <parameter name="logout-function" value="SingleStepAuthAdapter.onLogout"/>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

Configuring the authenticationConfig.xml file

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

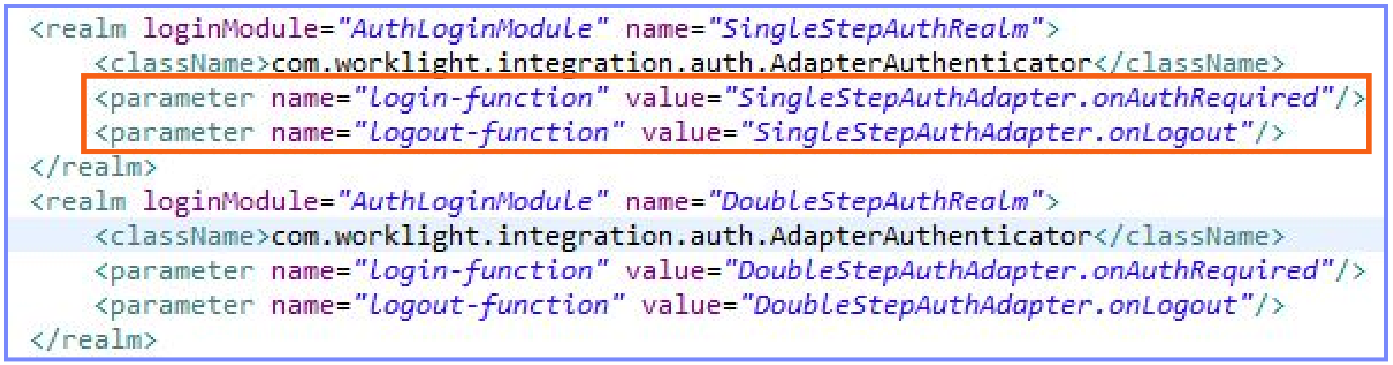


```
<realm loginModule="AuthLoginModule" name="SingleStepAuthRealm">
    <className>com.worklight.integration.auth.AdapterAuthenticator</className>
    <parameter name="login-function" value="SingleStepAuthAdapter.onAuthRequired"/>
    <parameter name="logout-function" value="SingleStepAuthAdapter.onLogout"/>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

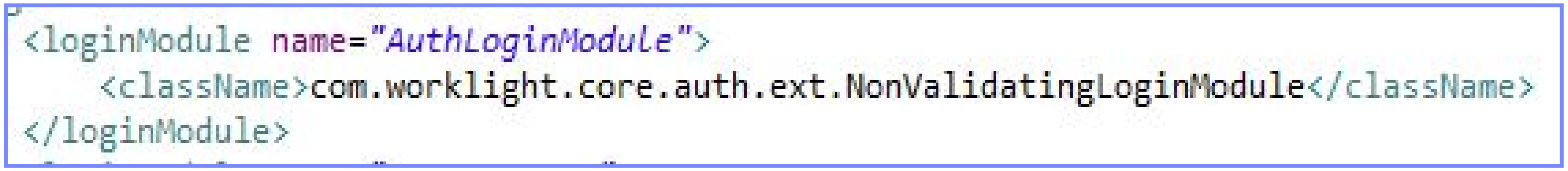


```
<realm loginModule="AuthLoginModule" name="SingleStepAuthRealm">
    <className>com.worklight.integration.auth.AdapterAuthenticator</className>
    <parameter name="login-function" value="SingleStepAuthAdapter.onAuthRequired"/>
    <parameter name="logout-function" value="SingleStepAuthAdapter.onLogout"/>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

- When the Worklight authentication framework detects an attempt to access a protected resource, an adapter function that is defined in a **login-function parameter** is invoked automatically.
- When logout is detected (explicit or session timeout), a **logout-function** is invoked 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`.

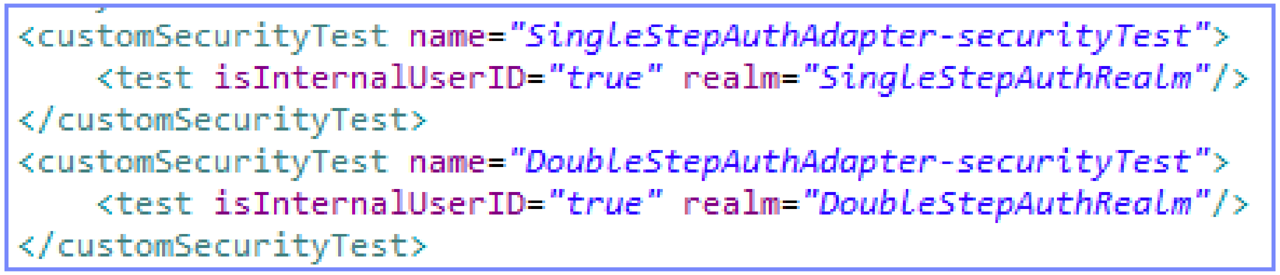


```
<loginModule name="AuthLoginModule">
    <className>com.worklight.core.auth.ext.NonValidatingLoginModule</className>
</loginModule>
```

- Using a `NonValidatingLoginModule` class name means that no additional validation is performed by the Worklight platform, and the developer takes responsibility for the credential validation 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 make it a `<customSecurityTest>.`



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

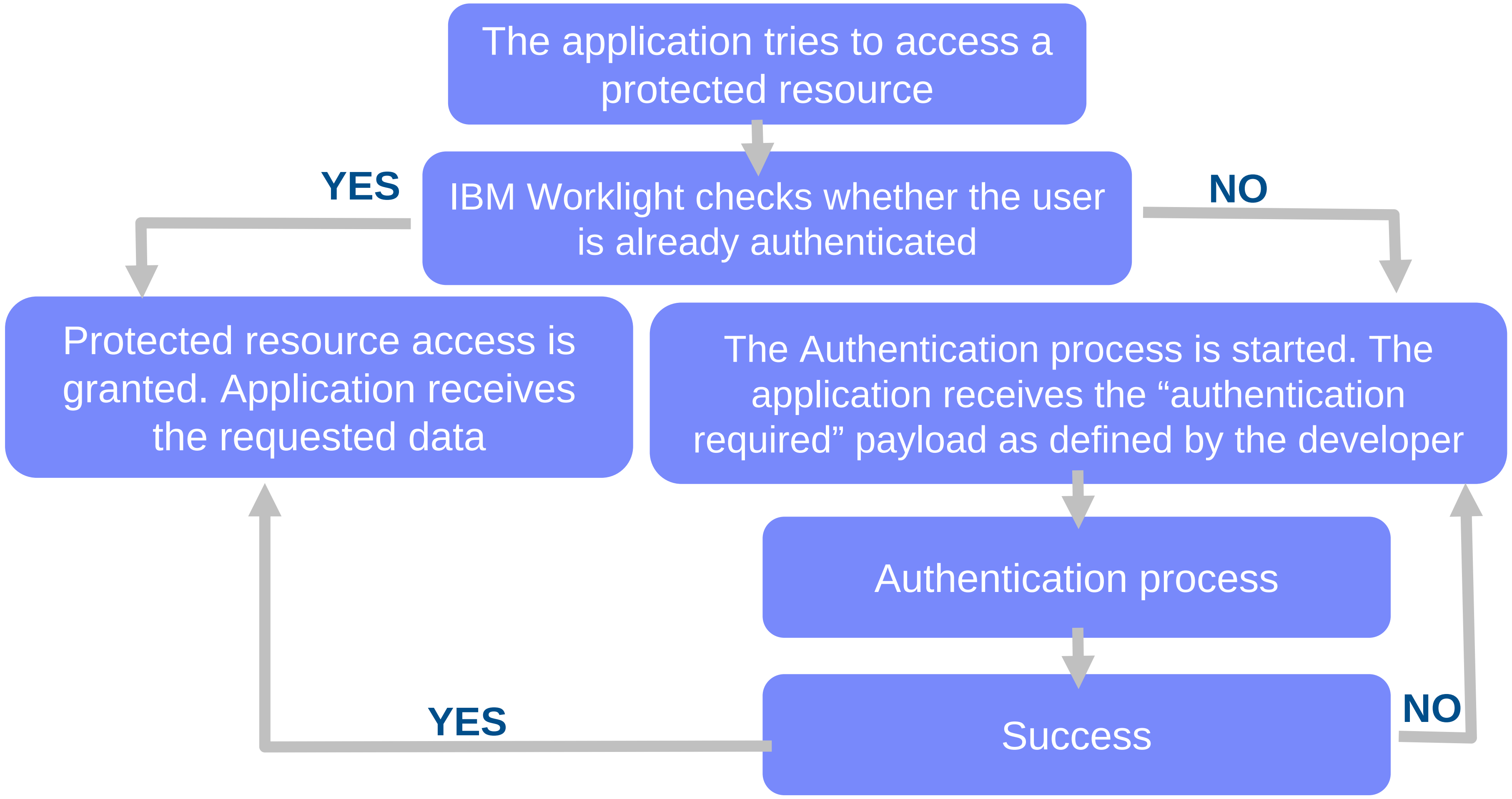
- Remember the security test names. You must use them in subsequent slides.

Agenda

- 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
- Check yourself questions

Creating the server-side authentication components

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


The application tries to access a protected resource
IBM Worklight checks whether the user is already authenticated
YES
NO
Protected resource access is granted. Application receives the requested data
The Authentication process is started. The application receives the "authentication required" payload as defined by the developer
Authentication process
Success
YES
NO

Creating the server-side authentication components

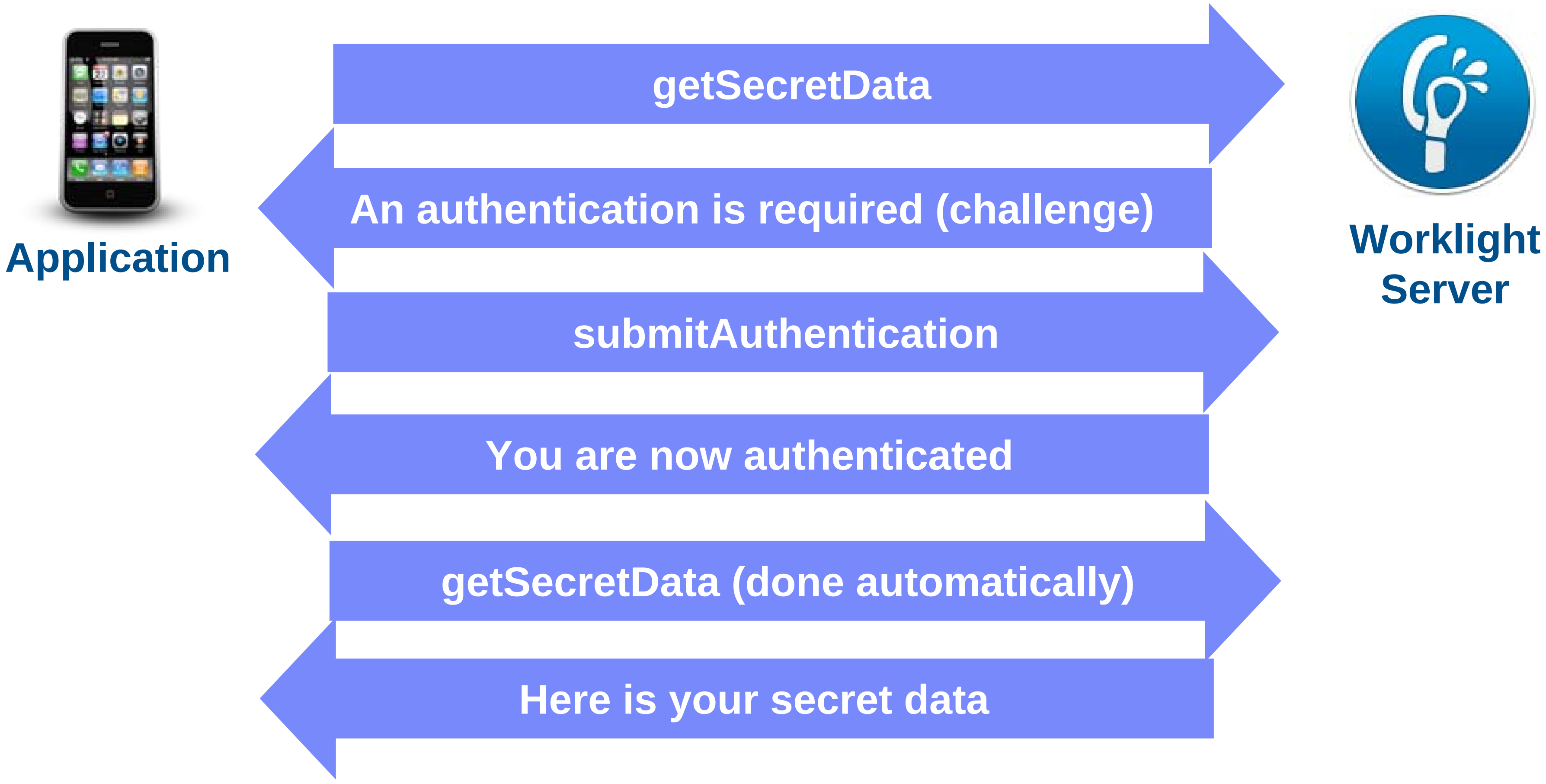
- In the sample provided with this training module we use two applications and two adapters. In the next slides we will focus on the SingleStepAuth application and adapter. The DoubleStepAuth application and adapter is just an extension of the same technique.
- Create an adapter that takes care of the authentication process. Name it **SingleStepAuthAdapter.**
- **SingleStepAuthAdapter** has 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 invoke it.
- The second procedure, however, is available to authenticated users only.

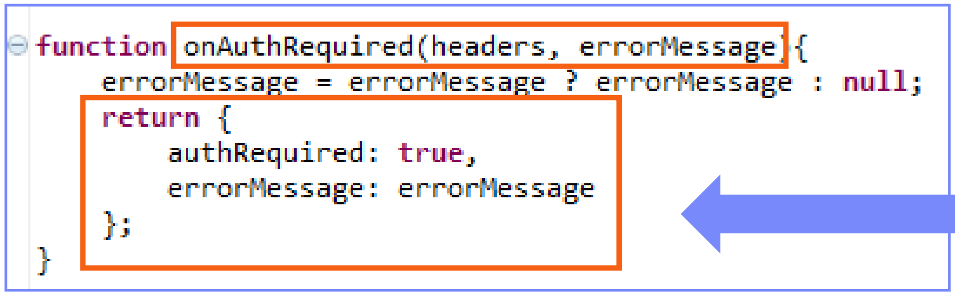
Creating the server-side authentication components

- The following diagram shows the flow to implement:


Application
getSecretData
An authentication is required (challenge)
submitAuthentication
You are now authenticated
getSecretData (done automatically)
Here is your secret data
Worklight Server

Creating the server-side authentication components

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



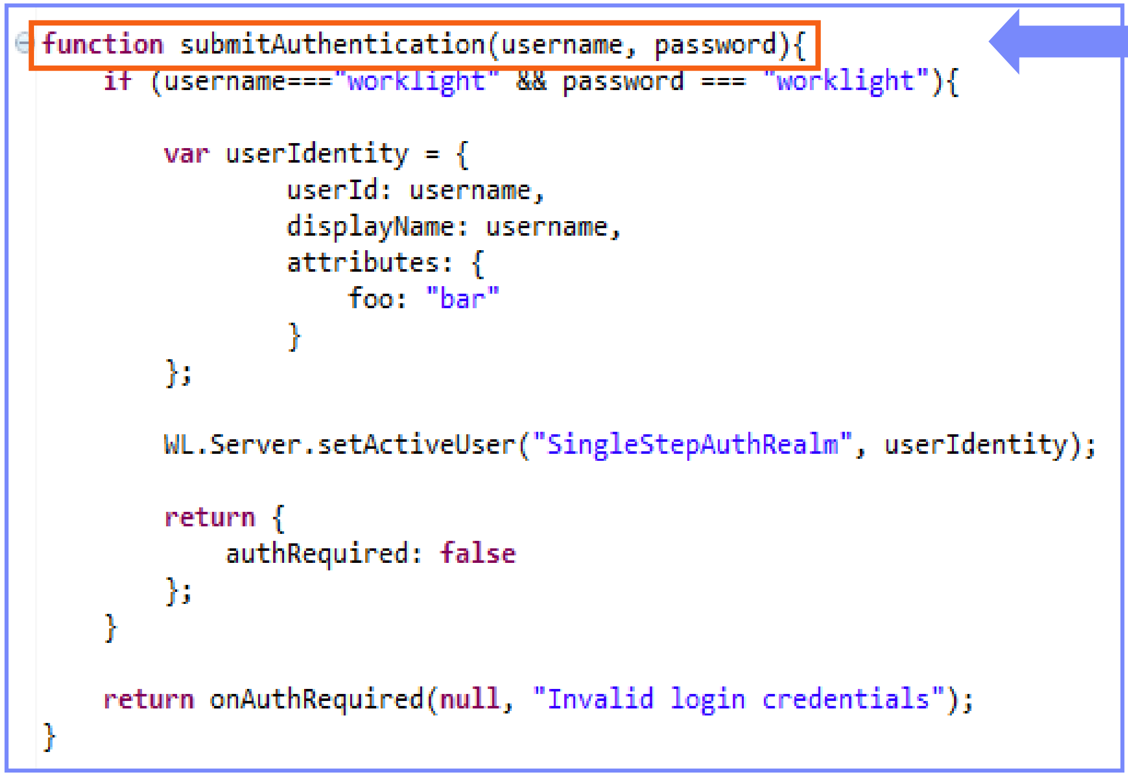
```
function onAuthRequired(headers, errorMessage){
    errorMessage = errorMessage ? errorMessage : null;
    return {
        authRequired: true,
        errorMessage: errorMessage
    };
}
```

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 invoked by a client application to validate user name and password.

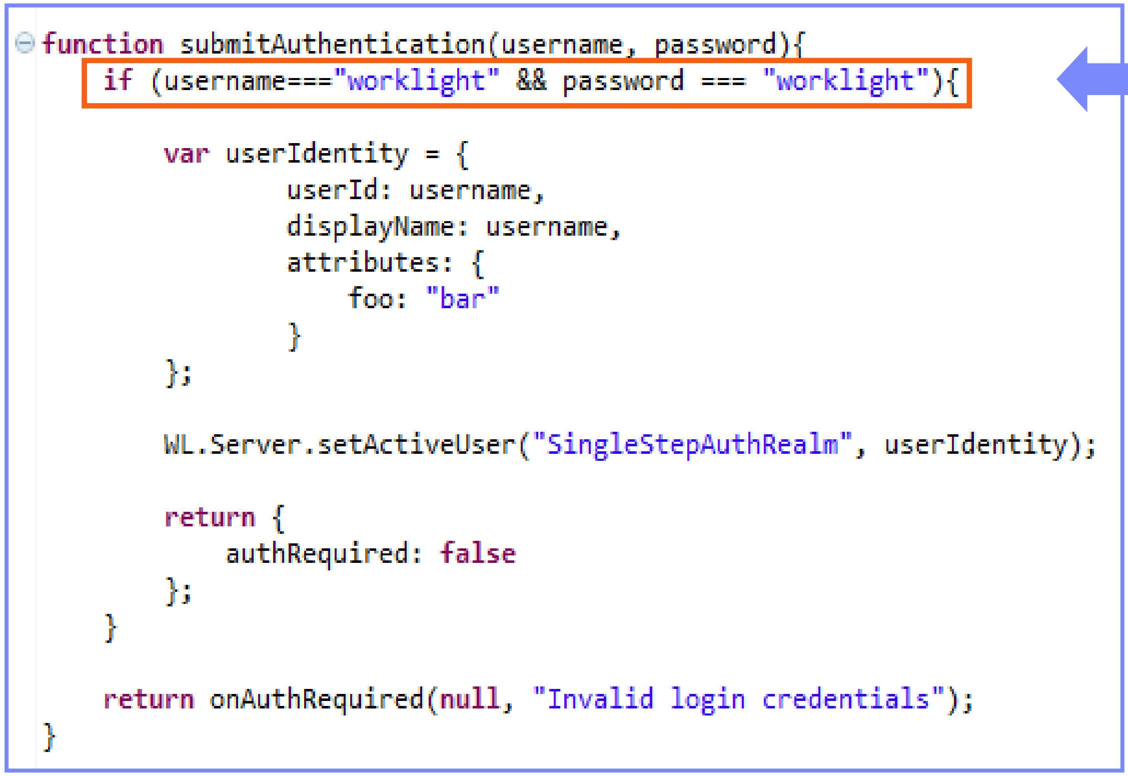


```
function submitAuthentication(username, password){
    if (username==="worklight" && password === "worklight"){

        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 invoked by a client application to validate user name and password.

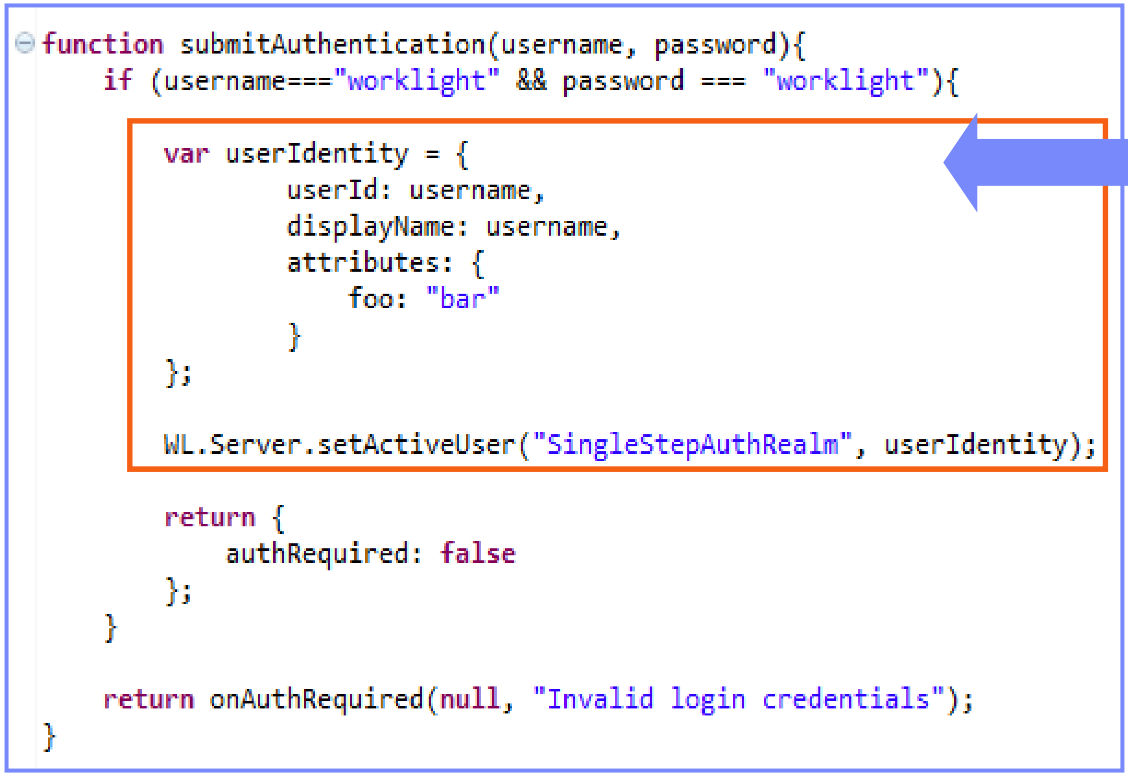


```
function submitAuthentication(username, password){
    if (username==="worklight" && password === "worklight"){

        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 can be performed, for example by using SQL or WebServices.

Creating the server-side authentication components

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

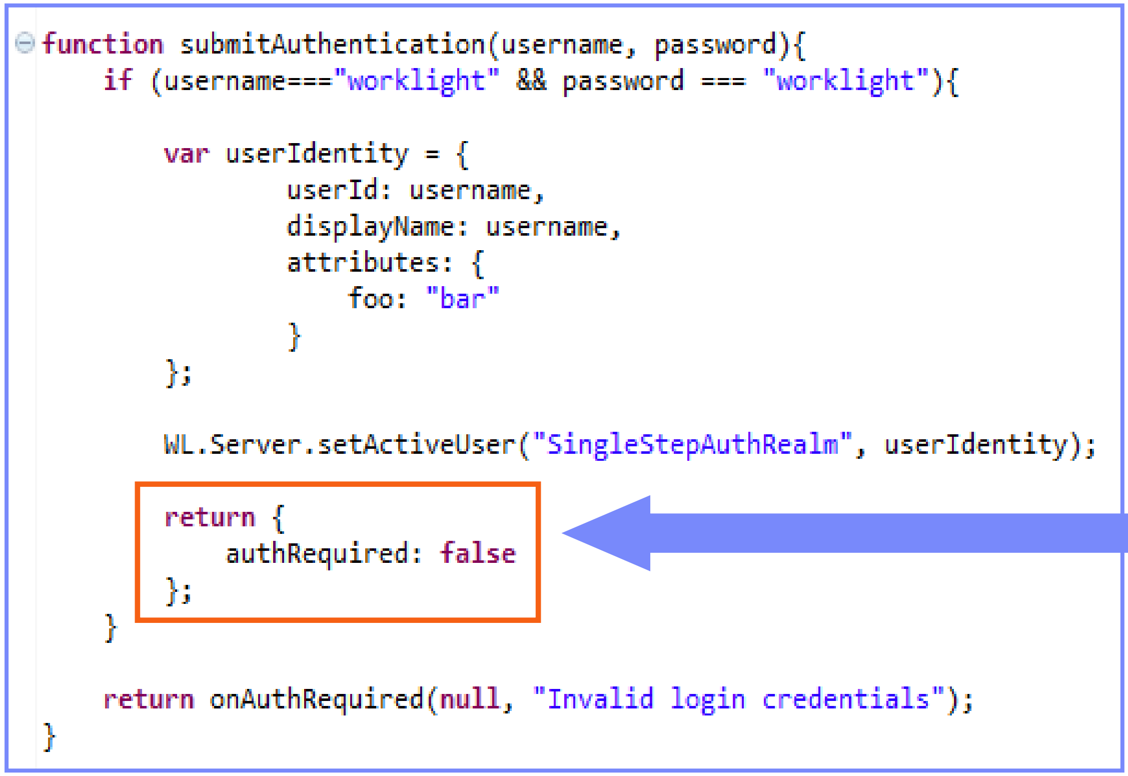


```
function submitAuthentication(username, password){
    if (username==="worklight" && password === "worklight"){

        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, WL.Server.setActiveUser API is called to create an authenticated session for the SingleStepAuthRealm with a user data stored in a userIdentity object. Note that you can add your own custom properties to the user identity attributes.

Creating the server-side authentication components

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

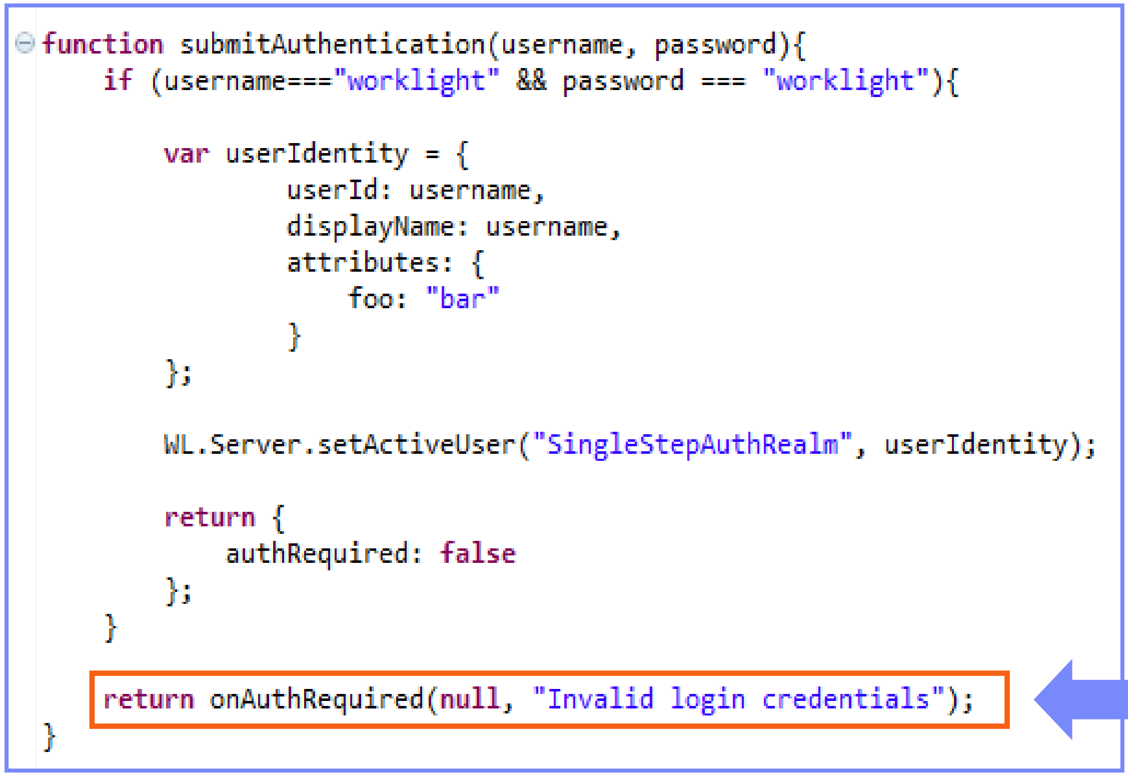


```
function submitAuthentication(username, password){
    if (username==="worklight" && password === "worklight"){

        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 invoked by a client application to validate user name and password.

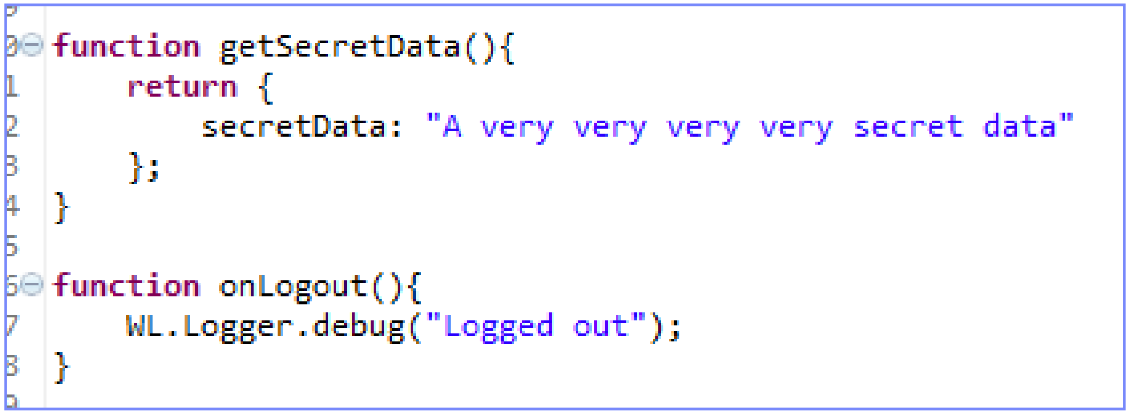


```
function submitAuthentication(username, password){
    if (username==="worklight" && password === "worklight"){

        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 corresponding 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 invoked 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

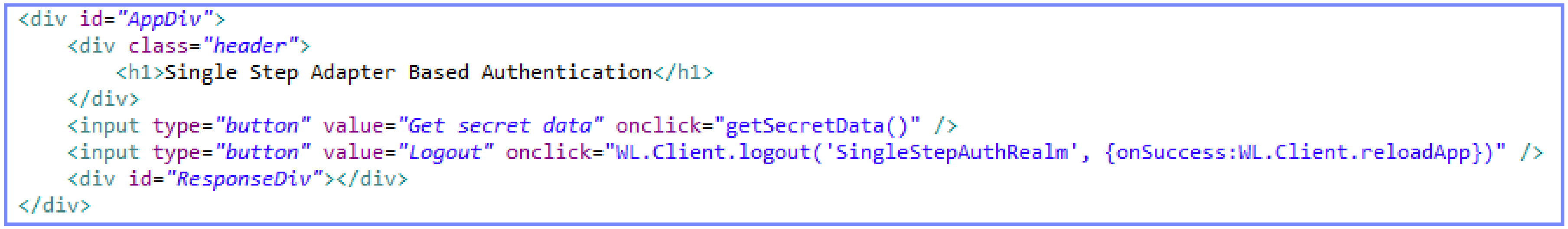
- Adapter-based authentication introduction
- Configuring the authenticationConfig.xml file
- Creating the server-side authentication components
- Creating the client-side authentication components
- Examining the result
- Exercise
- Check yourself questions

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 form purposes.
- When the authentication is required, the application hides the AppDiv element and shows the AuthDiv element. When the 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:

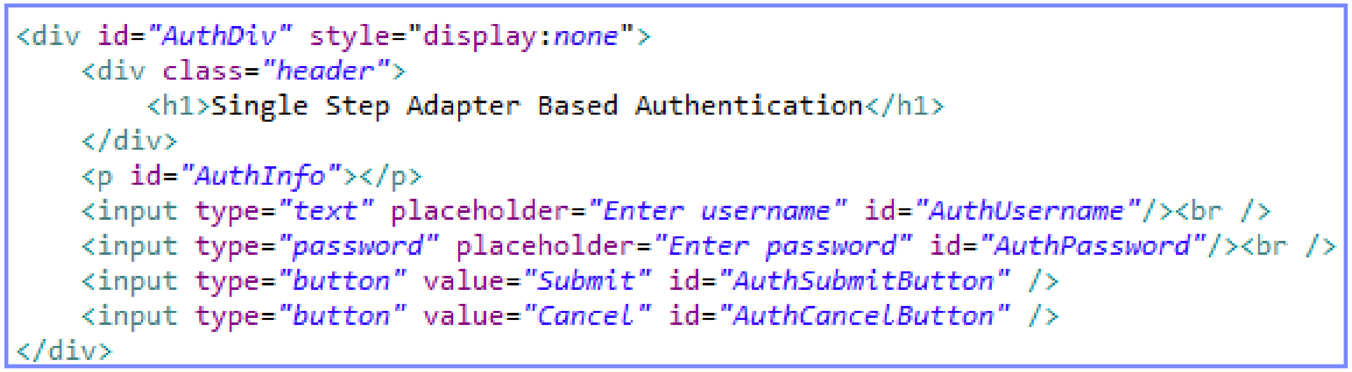


```
<div id="AppDiv">
    <div class="header">
        <h1>Single Step Adapter Based Authentication</h1>
    </div>
    <input type="button" value="Get secret data" onclick="getSecretData()" />
    <input type="button" value="Logout" onclick="WL.Client.logout('SingleStepAuthRealm', {onSuccess:WL.Client.reloadApp})"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

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

Creating the client-side authentication components

- The `AuthDiv` element contains the following sub-elements:



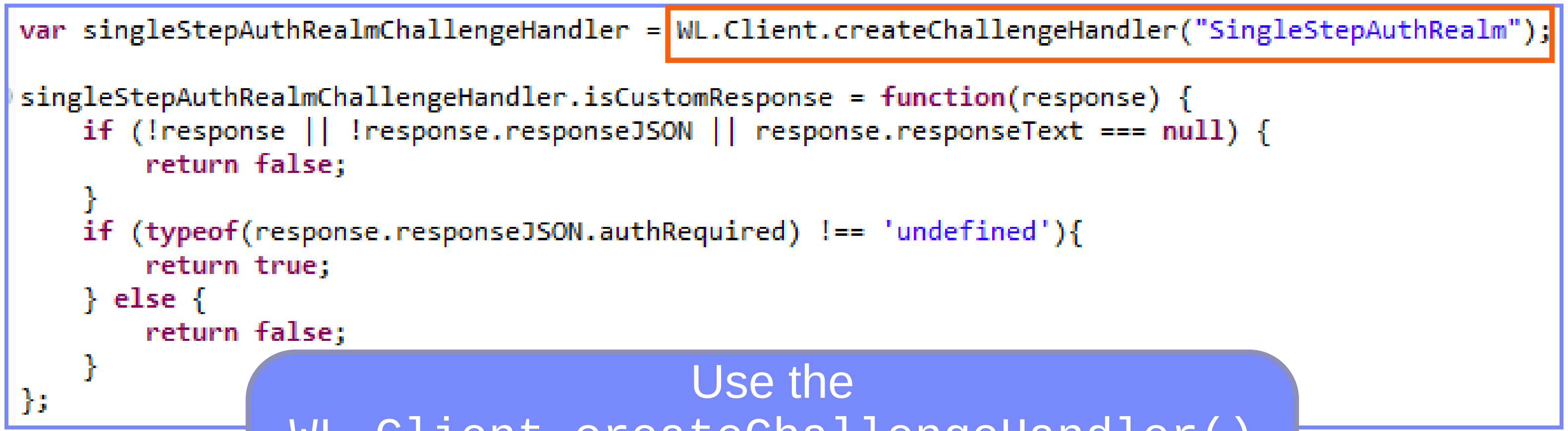
```
<div id="AuthDiv" style="display:none">
    <div class="header">
        <h1>Single Step Adapter Based Authentication</h1>
    </div>
    <p id="AuthInfo"></p>
    <input type="text" placeholder="Enter username" id="AuthUsername"/><br />
    <input type="password" placeholder="Enter password" id="AuthPassword"/>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

- The AuthDiv element is styled as `display:none` because it must not be displayed before the 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.



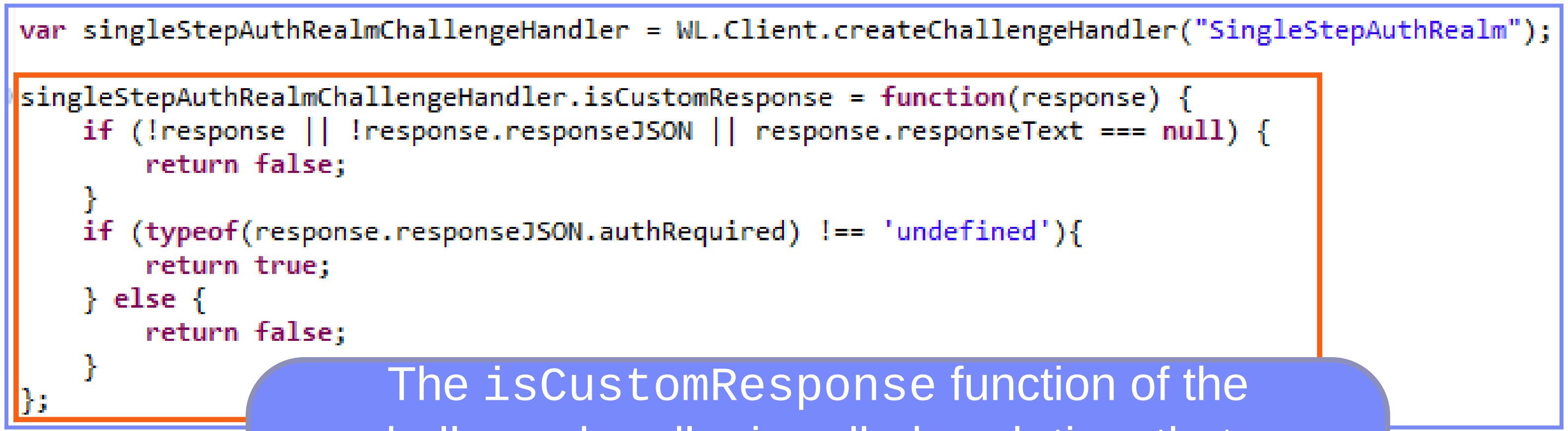
```
var singleStepAuthRealmChallengeHandler = WL.Client.createChallengeHandler("SingleStepAuthRealm");

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()` API method to create a challenge handler object. A realm name must be supplied 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.

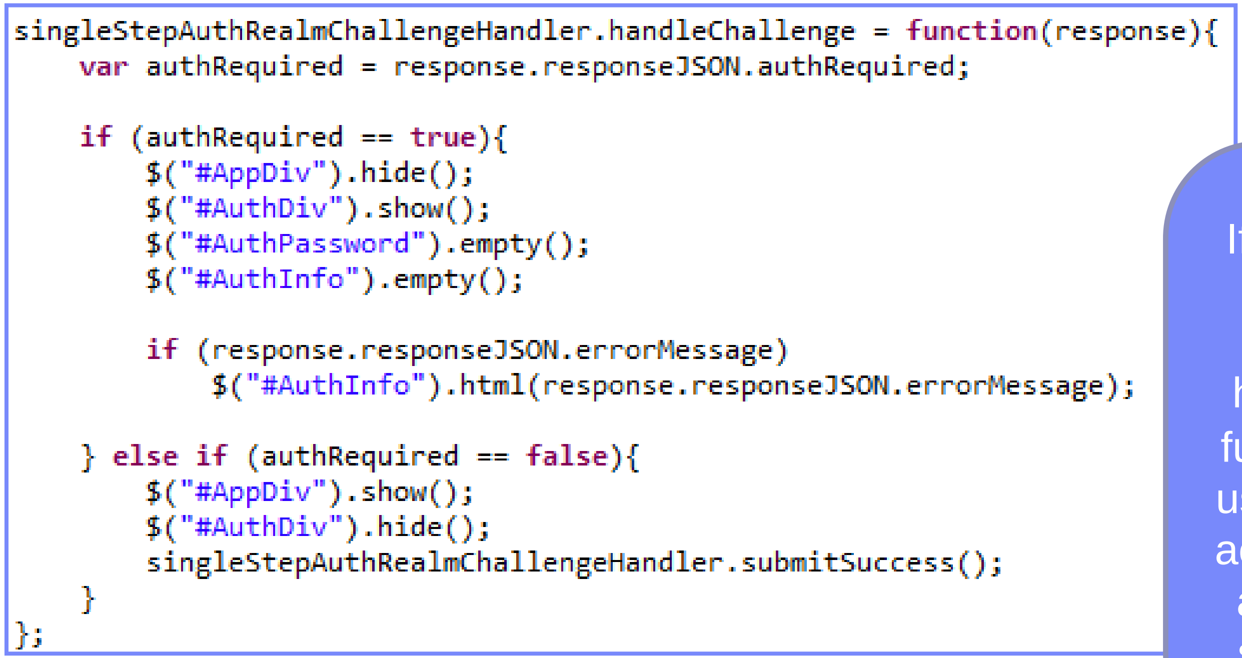


```
var singleStepAuthRealmChallengeHandler = WL.Client.createChallengeHandler("SingleStepAuthRealm");

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 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 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.

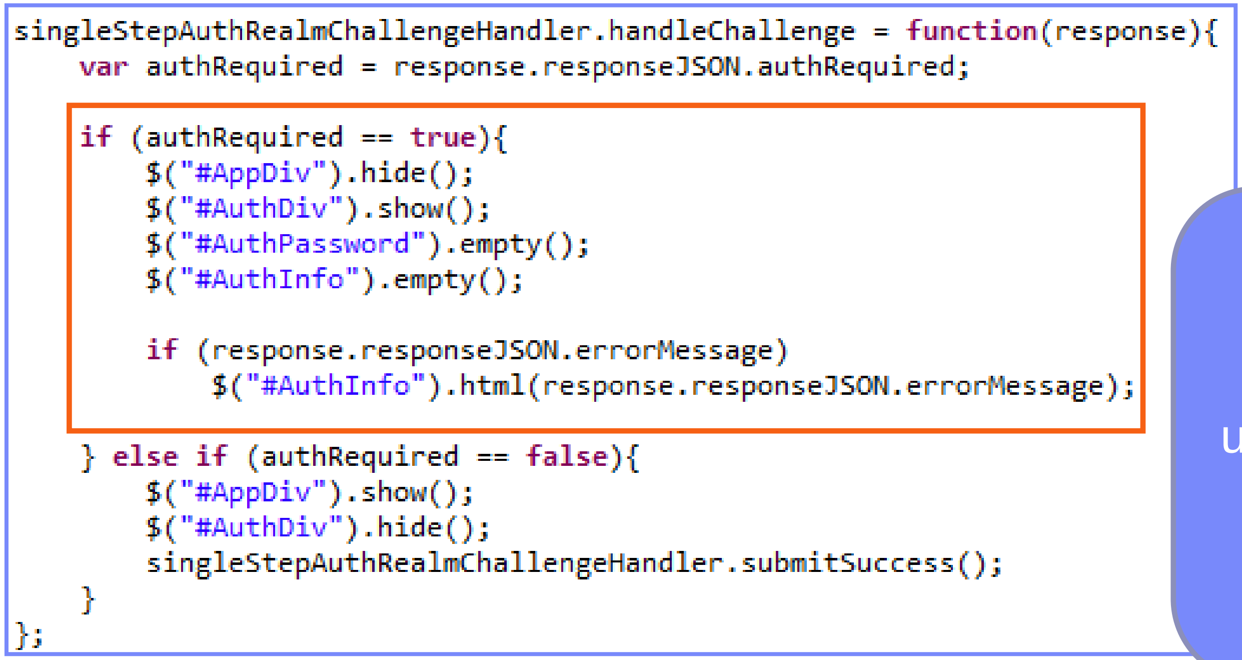


```
singleStepAuthRealmChallengeHandler.handleChallenge = function(response){
    var authRequired = response.responseJSON.authRequired;

    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 `isCustomResponse` returns `true`, the framework calls the `handleChallenge()` function. This function is used to perform required actions, such as hide the application screen and 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.

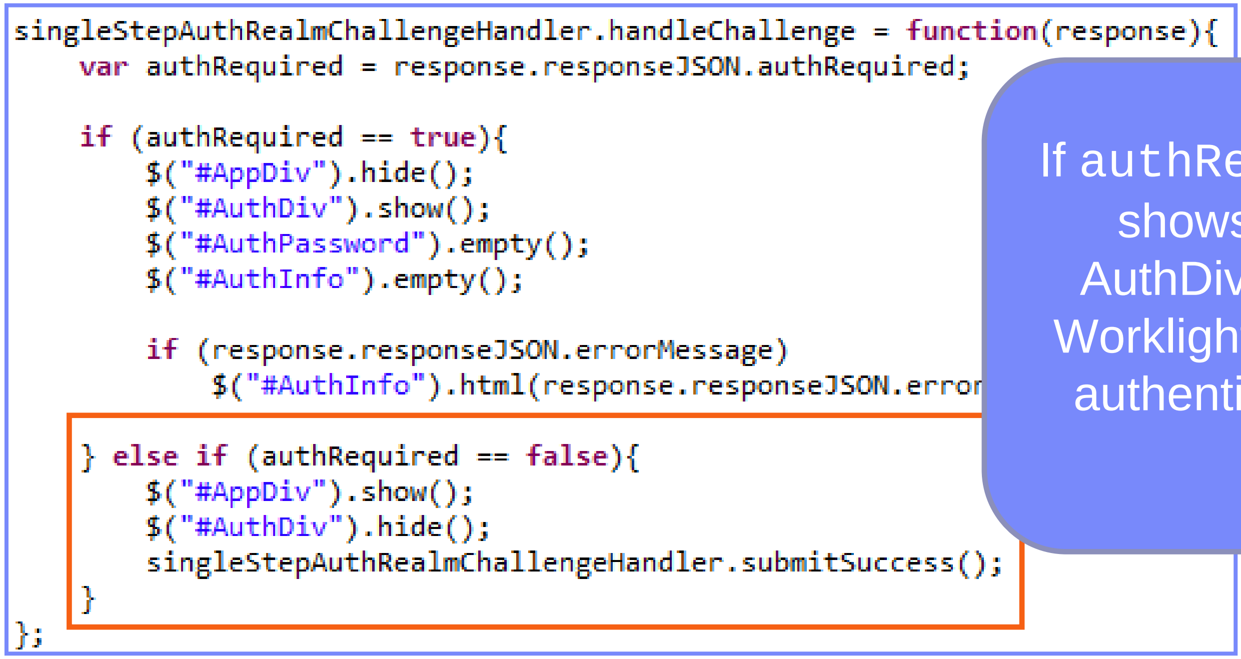


```
singleStepAuthRealmChallengeHandler.handleChallenge = function(response){
    var authRequired = response.responseJSON.authRequired;

    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 `authRequires` is `true`, it shows the login screen, cleans up the password field, and shows an `errorMessage` (if present).

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, it hides AuthDiv, and it notifies the Worklight framework that the authentication successfully completed.

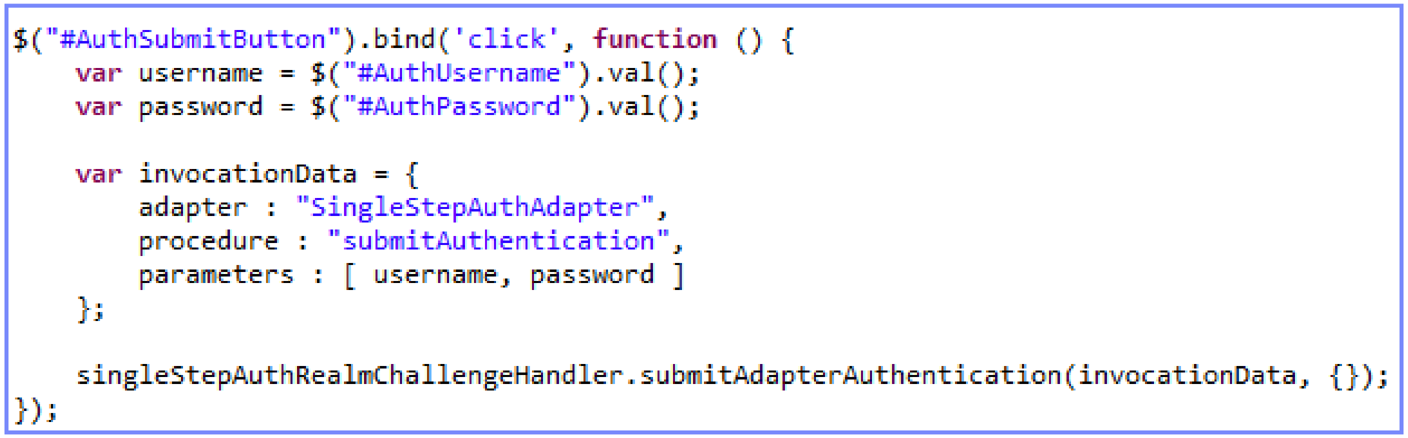
Creating the client-side authentication components

- In addition to the methods that the developer must implement, the challenge handler contains functionalities that the developer may 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 successfully finished. The Worklight framework then automatically issue the original request that triggered the authentication.
  - The `submitFailure()` function notifies the Worklight framework that the authentication completed with failure. The Worklight framework then disposes the original request that triggered the authentication.

**** Note that each one of these functions should be attached 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 in the challenge handler, the `submitAdapterAuthentication` method is used.

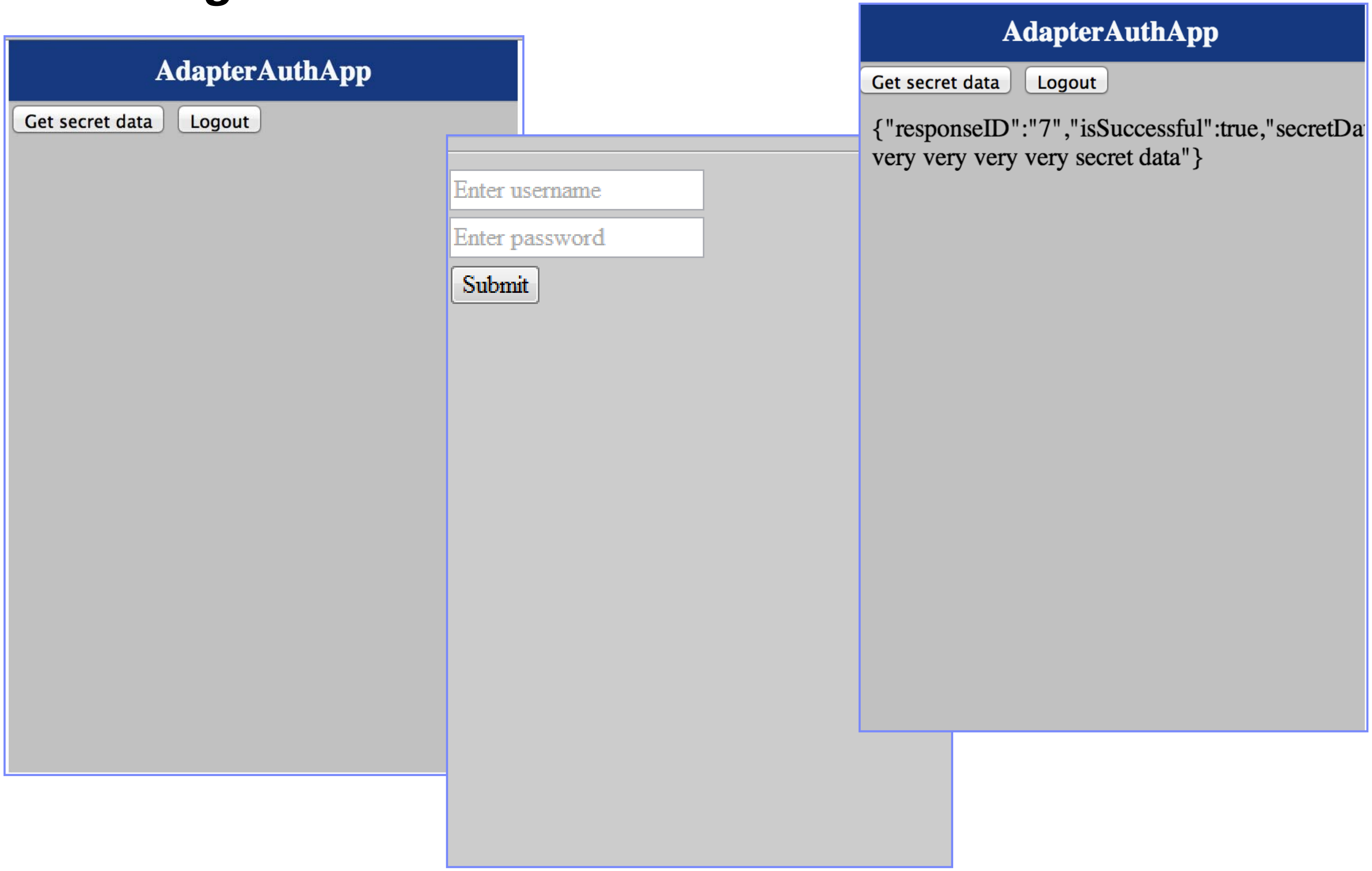


```
$("#AuthSubmitButton").bind('click', function () {
    var username = $("#AuthUsername").val();
    var password = $("#AuthPassword").val();

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

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

Agenda

- 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
- Check yourself questions

Examining the result



```
{"responseID":"7","isSuccessful":true,"secretDa
very very very very secret data"}
```

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
- Check yourself questions

Exercise

- Implement the adapter authentication as described in this training module.
- The sample for this training module can be found in the **Getting Started** page of the IBM® Worklight documentation 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
- Check yourself questions

Check yourself questions (1 of 2)

- When you define a realm that is using 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 an authentication for the client request?
  - The challengeHandler.isAuthenticationRequired
  - The challengeHandler.isUserAuthenticated
  - The challengeHandler.analyzeServerResponse
  - The challengeHandler.isCustomResponse

Check yourself questions (2 of 2)

- When you define a realm that is using an adapter-based authentication in the authenticationConfig.xml, which two parameters are mandatory?
  - $\boxed{\text{The login-function, the logout-function.}}$
  - 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.
  - $\boxed{\text{By adding a securityTest property to the procedure definition in the adapter XML.}}$
  - 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 an authentication for the client request?
  - $\boxed{\text{The challengeHandler.isAuthenticationRequired}}$
  - The challengeHandler.isUserAuthenticated
  - The challengeHandler.analyzeServerResponse
  - The challengeHandler.isCustomResponse

Notices

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