Lab 10 - How to secure your application (Server side)

The MobileFirst Foundation authentication framework uses the **OAuth 2.0** protocol. The OAuth 2 protocol is based on the acquisition of an access token that encapsulates the granted permissions to the client.

In that context, the IBM MobileFirst Server serves as an authorization server and is able to generate access tokens. The client can then use these tokens to access resources on a resource server, which can be either the MobileFirst Server itself or an external server.

The resource server checks the validity of the token to make sure that the client can be granted access to the requested resource. The separation between resource server and authorization server allows to enforce security on resources that are running outside MobileFirst Server.

Security Check A security check is an entity that is responsible for obtaining and validating client credentials. Security checks are instantiated by Adapters.

The security check defines the process to be used to authenticate users. It is often associated with a SecurityCheckConfiguration that defines properties to be used by the security check. The same security check can also be used to protect several resources.

On the client-side, the application logic needs to implement a challenge handler to handle challenges sent by the security check.

In this lab we are going to use the **CredentialsValidationSecurityCheck** which fit the most common use-cases of simple user authentication. In addition to validating the credentials, it creates a user identity that will be accessible from various parts of the framework, allowing you to identify the current user. Optionally, UserAuthenticationSecurityCheck also provides Remember Me capabilities.

In this labe we going to use a security check asking for a username and password and uses the username to represent an authenticated user.

Steps:

Create new adapter

1. In the console nevigate to **AdapterServices** folder

2. Create a new adapter

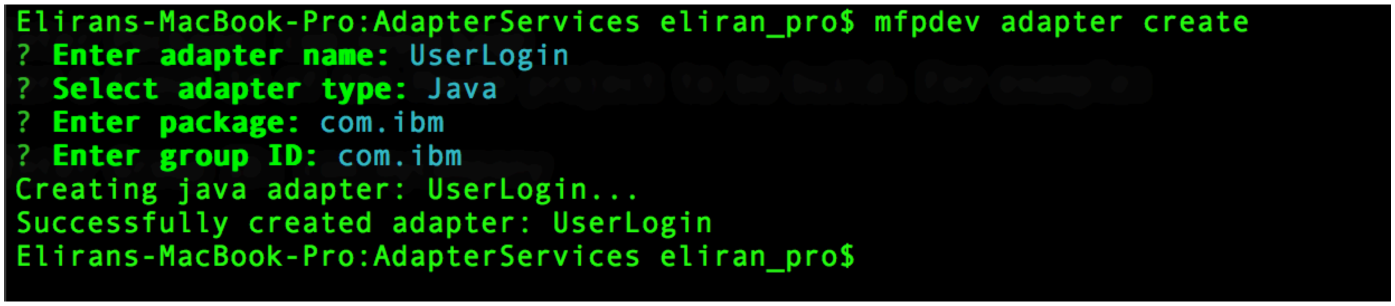
```
mfpdev adapter create
```

3. Enter the adapter name: **UserLogin**

4. Select an adapter type **(Java)** using the arrows and the enter keys

5. Enter an adapter package For example: **com.ibm**

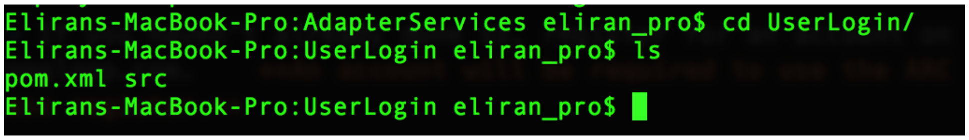
6. Enter a Group Id of the Maven project to be build: **com.ibm**



```
Elirans-MacBook-Pro:AdapterServices eliran_pro$ mfpdev adapter create
? Enter adapter name: UserLogin
? Select adapter type: Java
? Enter package: com.ibm
? Enter group ID: com.ibm
Creating java adapter: UserLogin...
Successfully created adapter: UserLogin
Elirans-MacBook-Pro:AdapterServices eliran_pro$
```

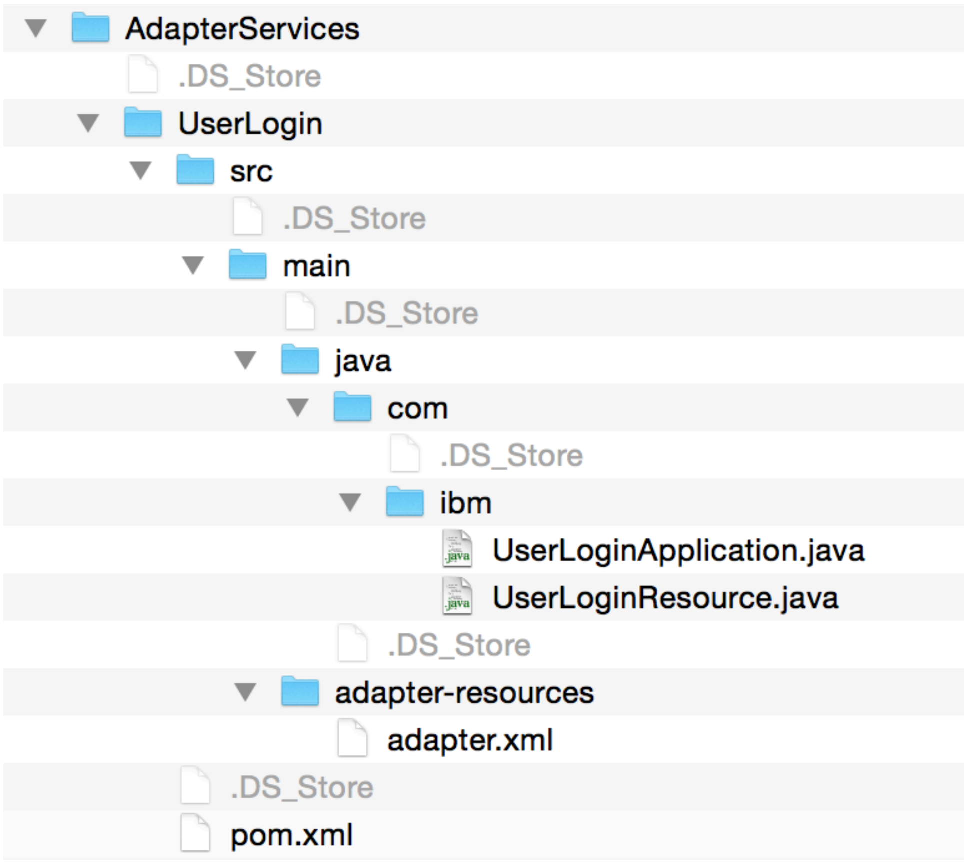
7. Change the direcotry to **UserLogin**

```
cd UserLogin
```



```
Elirans-MacBook-Pro:AdapterServices eliran_pro$ cd UserLogin/
Elirans-MacBook-Pro:UserLogin eliran_pro$ ls
pom.xml src
Elirans-MacBook-Pro:UserLogin eliran_pro$
```

8. Let's look at the UserLogin folder



```
▼ AdapterServices
      .DS_Store
   ▼ UserLogin
      ▼ src
            .DS_Store
         ▼ main
               .DS_Store
            ▼ java
               ▼ com
                     .DS_Store
                  ▼ ibm
                        UserLoginApplication.java
                        UserLoginResource.java
                  .DS_Store
            ▼ adapter-resources
                  adapter.xml
         .DS_Store
         pom.xml
```

9. Open the UserLogin folder with your favorite IDE

10. Delete both of the files :

    - UserLoginApplication.java
    - UserLoginResource.java

11. Create a new java class name and call it **UserLoginSecurityCheck.java**

12. **Open** the UserLoginSecurityCheck.java under **UserLogin/src/main/java/com/ibm/UserLoginSecurityCheck.java**

13. **Copy** the following code

```
package com.ibm;
```

import com.ibm.mfp.server.registration.external.model.AuthenticatedUser; import com.ibm.mfp.security.checks.base.UserAuthenticationSecurityCheck;

import java.util.HashMap; import java.util.Map;

public class UserLogin extends UserAuthenticationSecurityCheck {

```
@Override
protected AuthenticatedUser createUser() {
    return null;
}

@Override
protected boolean validateCredentials(Map<String, Object> credentials) {
    return false;
}

@Override
protected Map<String, Object> createChallenge() {
    return null;
}
```

}```

Create the challenge

1. Copy the following code and replace the **createChallenge()** method

```
@Override
protected Map<String, Object> createChallenge() {
Map challenge = new HashMap();
challenge.put("errorMsg",errorMsg);
challenge.put("remainingAttempts",getRemainingAttempts());
return challenge;
}
```

Validating the user credentials

When the client sends the challenge's answer, the answer is passed to **validateCredentials** as a Map. This method should implement your logic and return true if the credentials are valid.

In this example, credentials are considered "valid" when username and password are the same:

~~~
```java
~~~

@Override protected boolean validateCredentials(Map credentials) { if(credentials!=null && credentials.containsKey("username") && credentials.containsKey("password")){ String username = credentials.get("username").toString(); String password = credentials.get("password").toString(); if(!username.isEmpty() && !password.isEmpty() && username.equals(password)) { return true; } else {

errorMsg = "Wrong Credentials"; } } else{ errorMsg = "Credentials not set properly"; } return false; }
```

Creating the AuthenticatedUser object

The **UserAuthenticationSecurityCheck** stores a representation of the current client (user, device, application) in persistent data, allowing you to retrieve the current user in various parts of your code, such as the challenge handlers or the adapters. Users are represented by an instance of the class AuthenticatedUser. Its constructor receives a id, displayName and securityCheckName.

In this example, we are using the username for both the id and displayName.

1. First, modify the validateCredentials method to save the username:

```
private String userId, displayName;
@Override
protected boolean validateCredentials(Map<String, Object> credentials) {
if(credentials!=null && credentials.containsKey("username") && credentials.contai
nsKey("password")){
    String username = credentials.get("username").toString();
    String password = credentials.get("password").toString();
    if(!username.isEmpty() && !password.isEmpty() && username.equals(password)) {
        userId = username;
        displayName = username;
        return true;
    } else {
        errorMsg = "Wrong Credentials";
    }
} else{
    errorMsg = "Credentials not set properly";
}
return false;
}
```

2. Then, override the createUser method to return a new instance of AuthenticatedUser:

```
@Override
protected AuthenticatedUser createUser() {
return new AuthenticatedUser(userId, displayName, this.getName());
}
```

You can use **this.getName()** to get the current security check name.
```

> **Note: UserAuthenticationSecurityCheck** will call your **createUser()** implementation after a successful validateCredentials.

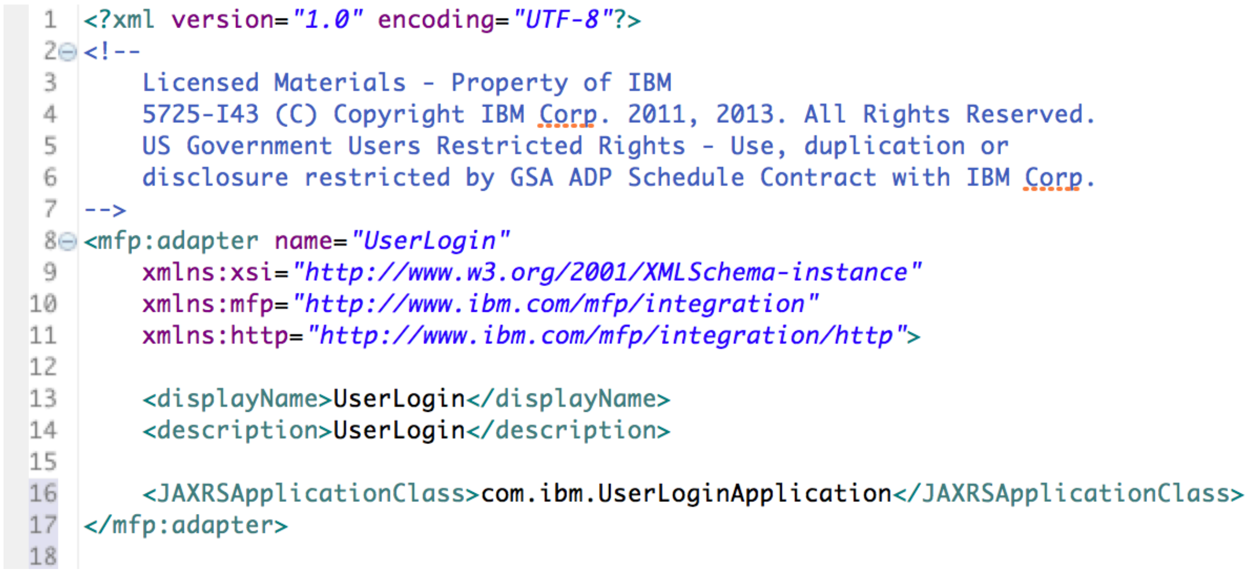
## Configuring the SecurityCheck

1. **Open** the **adapter.xml** under **UserLogin/src/main/adapter-resources/adapter.xml**
2. n the adapter.xml file, find the **com.ibm.UserLoginApplication** and delete it from the file
3. In the adapter.xml file, add a element:

```
<securityCheckDefinition name="UserLogin" class="com.ibm.UserLogin">
<property name="maxAttempts" defaultValue="3" description="How many attempts are allowed"/>
<property name="blockedStateExpirationSec" defaultValue="10" description="How long before the client can try again (seconds)"/>
<property name="successStateExpirationSec" defaultValue="60" description="How long is a successful state valid for (seconds)"/>
<property name="rememberMeDurationSec" defaultValue="120" description="How long is the user remembered when using RememberMe (seconds)"/>
</securityCheckDefinition>
```

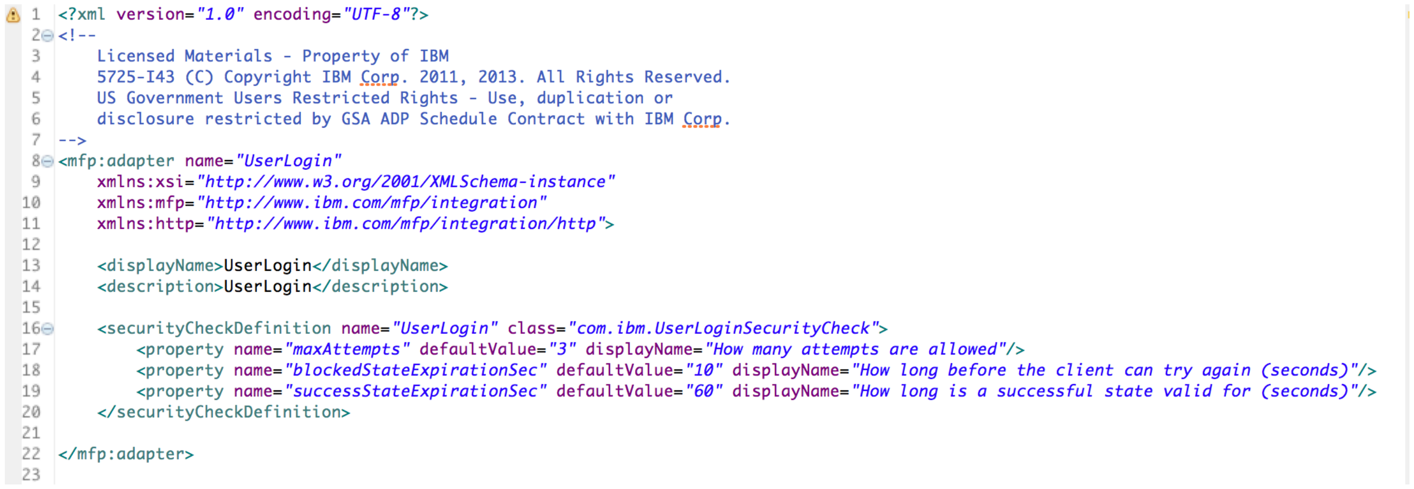
Your **adapter.xml** should look like this:

Before



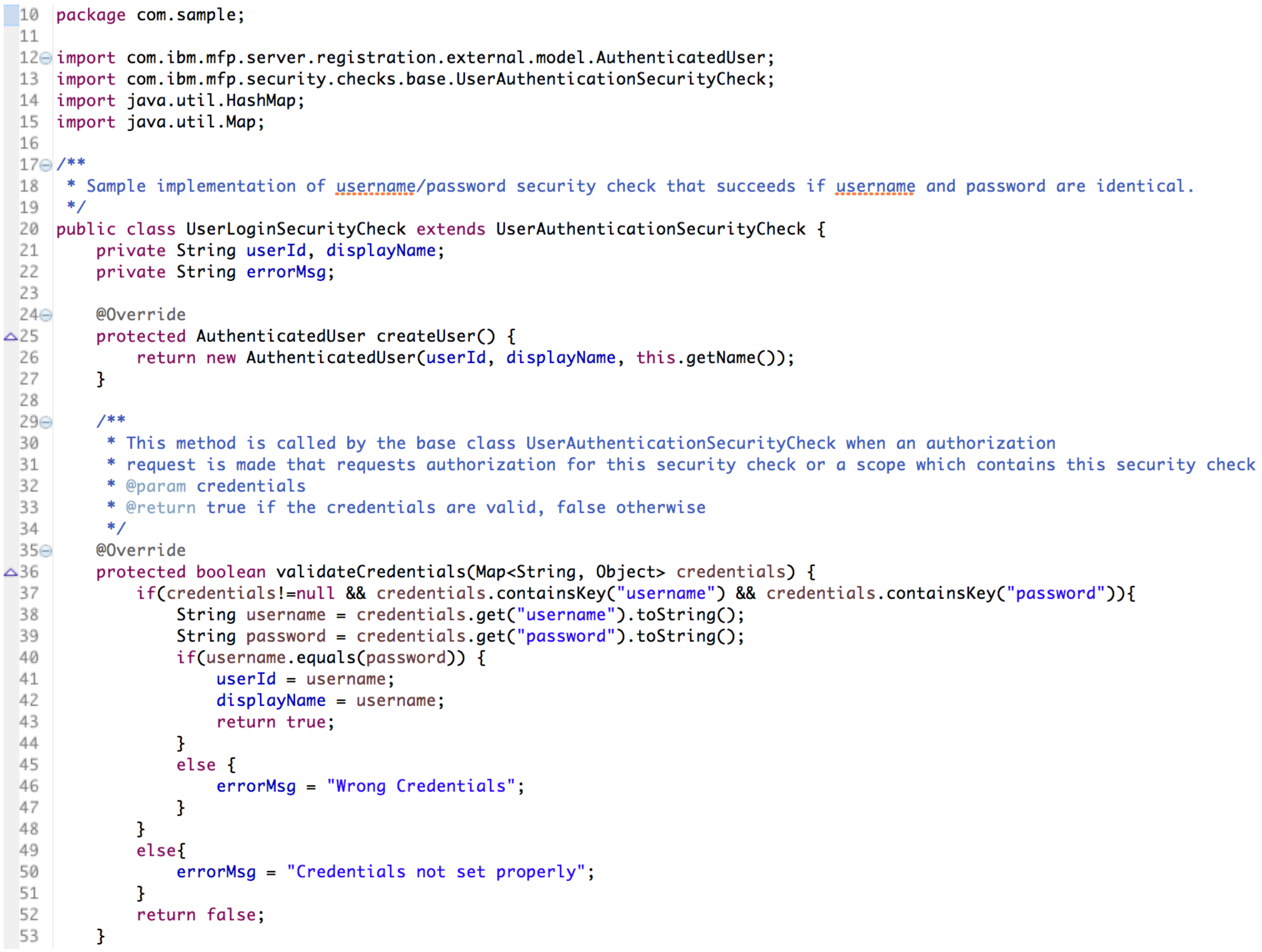
```
<?xml version="1.0" encoding="UTF-8"?>
<!--
    Licensed Materials - Property of IBM
    5725-I43 (C) Copyright IBM Corp. 2011, 2013. All Rights Reserved.
    US Government Users Restricted Rights - Use, duplication or
    disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
-->
<mfp:adapter name="UserLogin"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:mfp="http://www.ibm.com/mfp/integration"
    xmlns:http="http://www.ibm.com/mfp/integration/http">

    <displayName>UserLogin</displayName>
    <description>UserLogin</description>

    <JAXRSApplicationClass>com.ibm.UserLoginApplication</JAXRSApplicationClass>
</mfp:adapter>

```

After

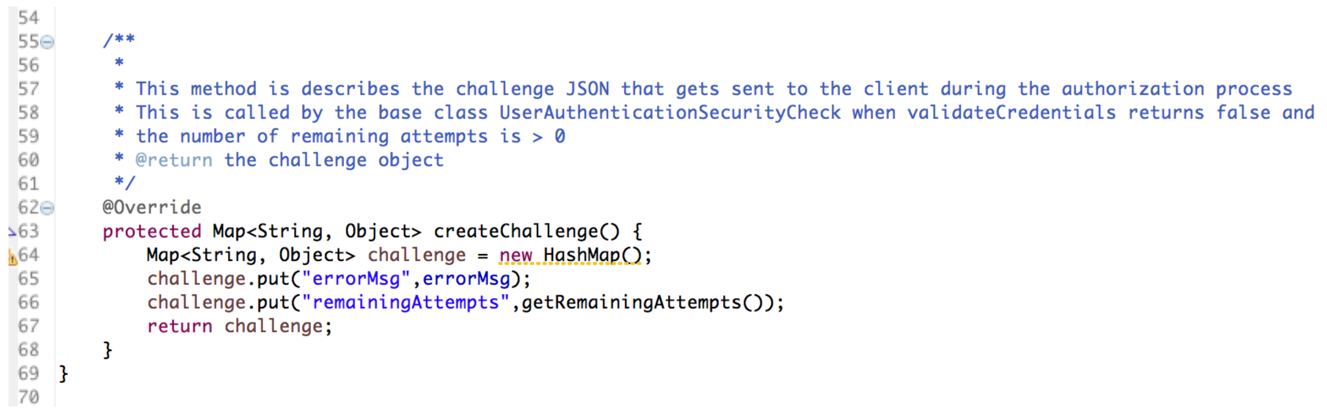


```xml
<?xml version="1.0" encoding="UTF-8"?>
<!--
    Licensed Materials - Property of IBM
    5725-I43 (C) Copyright IBM Corp. 2011, 2013. All Rights Reserved.
    US Government Users Restricted Rights - Use, duplication or
    disclosure restricted by GSA ADP Schedule Contract with IBM Corp.
-->
<mfp:adapter name="UserLogin"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:mfp="http://www.ibm.com/mfp/integration"
    xmlns:http="http://www.ibm.com/mfp/integration/http">

    <displayName>UserLogin</displayName>
    <description>UserLogin</description>

    <securityCheckDefinition name="UserLogin" class="com.ibm.UserLoginSecurityCheck">
        <property name="maxAttempts" defaultValue="3" displayName="How many attempts are allowed"/>
        <property name="blockedStateExpirationSec" defaultValue="10" displayName="How long before the client can try again (seconds)"/>
        <property name="successStateExpirationSec" defaultValue="60" displayName="How long is a successful state valid for (seconds)"/>
    </securityCheckDefinition>

</mfp:adapter>

```

4. Your new class should look like this:



```java
package com.sample;

import com.ibm.mfp.server.registration.external.model.AuthenticatedUser;
import com.ibm.mfp.security.checks.base.UserAuthenticationSecurityCheck;
import java.util.HashMap;
import java.util.Map;

/**
 * Sample implementation of username/password security check that succeeds if username and password are identical.
 */
public class UserLoginSecurityCheck extends UserAuthenticationSecurityCheck {
    private String userId, displayName;
    private String errorMsg;

    @Override
    protected AuthenticatedUser createUser() {
        return new AuthenticatedUser(userId, displayName, this.getName());
    }

    /**
     * This method is called by the base class UserAuthenticationSecurityCheck when an authorization
     * request is made that requests authorization for this security check or a scope which contains this security check
     * @param credentials
     * @return true if the credentials are valid, false otherwise
     */
    @Override
    protected boolean validateCredentials(Map<String, Object> credentials) {
        if(credentials!=null && credentials.containsKey("username") && credentials.containsKey("password")){
            String username = credentials.get("username").toString();
            String password = credentials.get("password").toString();
            if(username.equals(password)) {
                userId = username;
                displayName = username;
                return true;
            }
            else {
                errorMsg = "Wrong Credentials";
            }
        }
        else{
            errorMsg = "Credentials not set properly";
        }
        return false;
    }
```

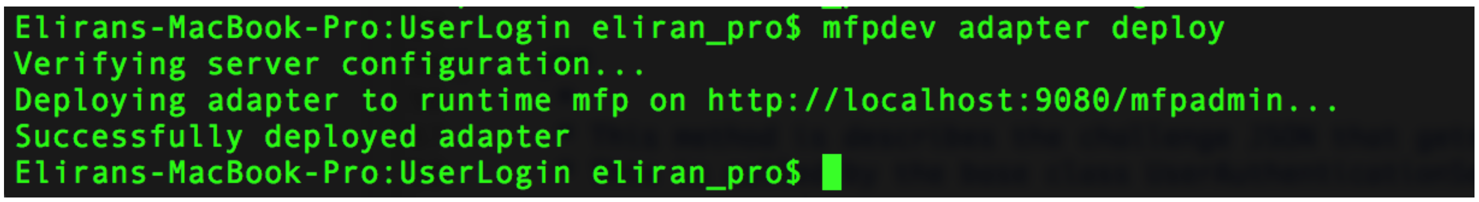


```java

    /**
     *
     * This method is describes the challenge JSON that gets sent to the client during the authorization process
     * This is called by the base class UserAuthenticationSecurityCheck when validateCredentials returns false and
     * the number of remaining attempts is > 0
     * @return the challenge object
     */
    @Override
    protected Map<String, Object> createChallenge() {
        Map<String, Object> challenge = new HashMap();
        challenge.put("errorMsg",errorMsg);
        challenge.put("remainingAttempts",getRemainingAttempts());
        return challenge;
    }
}

```

5. **Save** your changes

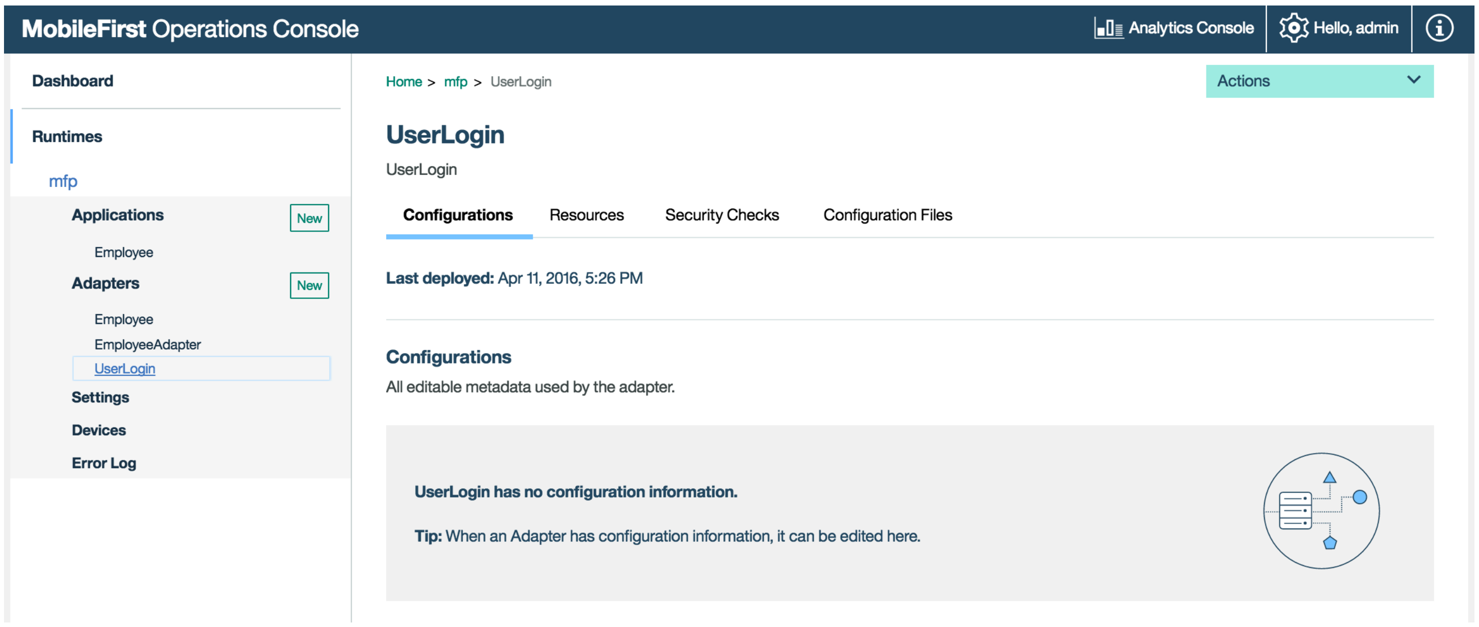
6. **Deploy** the new adapter to the console.

```
mfpdev adapter build
mfpdev adapter deploy
```



```
Elirans-MacBook-Pro:UserLogin eliran_pro$ mfpdev adapter deploy
Verifying server configuration...
Deploying adapter to runtime mfp on http://localhost:9080/mfpadmin...
Successfully deployed adapter
Elirans-MacBook-Pro:UserLogin eliran_pro$
```

1. If we look at the MFP console we can see that the adapter **UserLogin** was successfully deployed.



UserLogin

UserLogin

Last deployed: Apr 11, 2016, 5:26 PM

Configurations

All editable metadata used by the adapter.

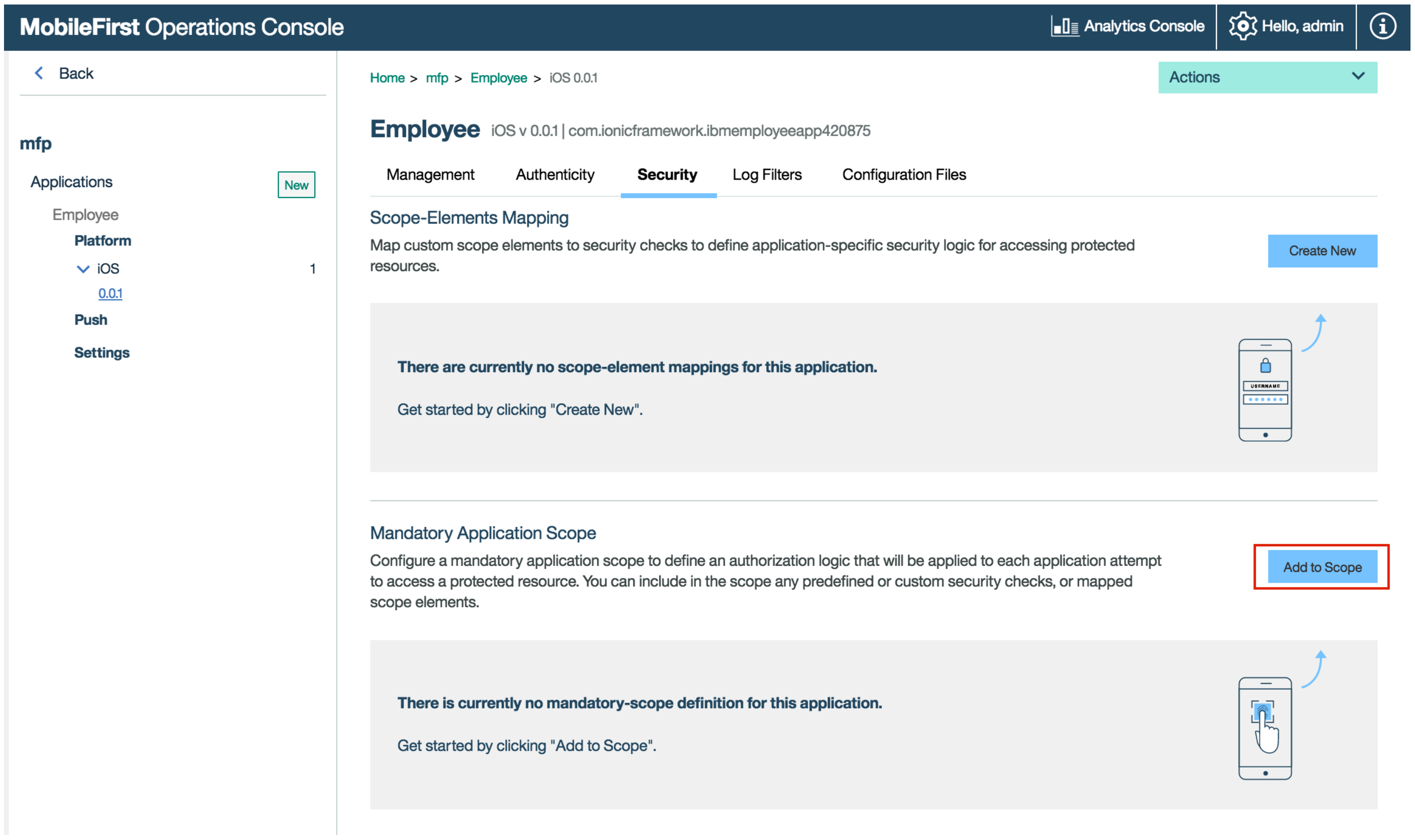
UserLogin has no configuration information.

Tip: When an Adapter has configuration information, it can be edited here.

Mandatory application scope

At the application level, you can define a scope that will apply to all the resources used by this application.

1. In the MobileFirst Operations Console, select **Employee** app then select the → **Security** tab. Under Mandatory Application Scope click on Add to Scope.



Scope-Elements Mapping

Map custom scope elements to security checks to define application-specific security logic for accessing protected resources.

There are currently no scope-element mappings for this application.

Get started by clicking "Create New".

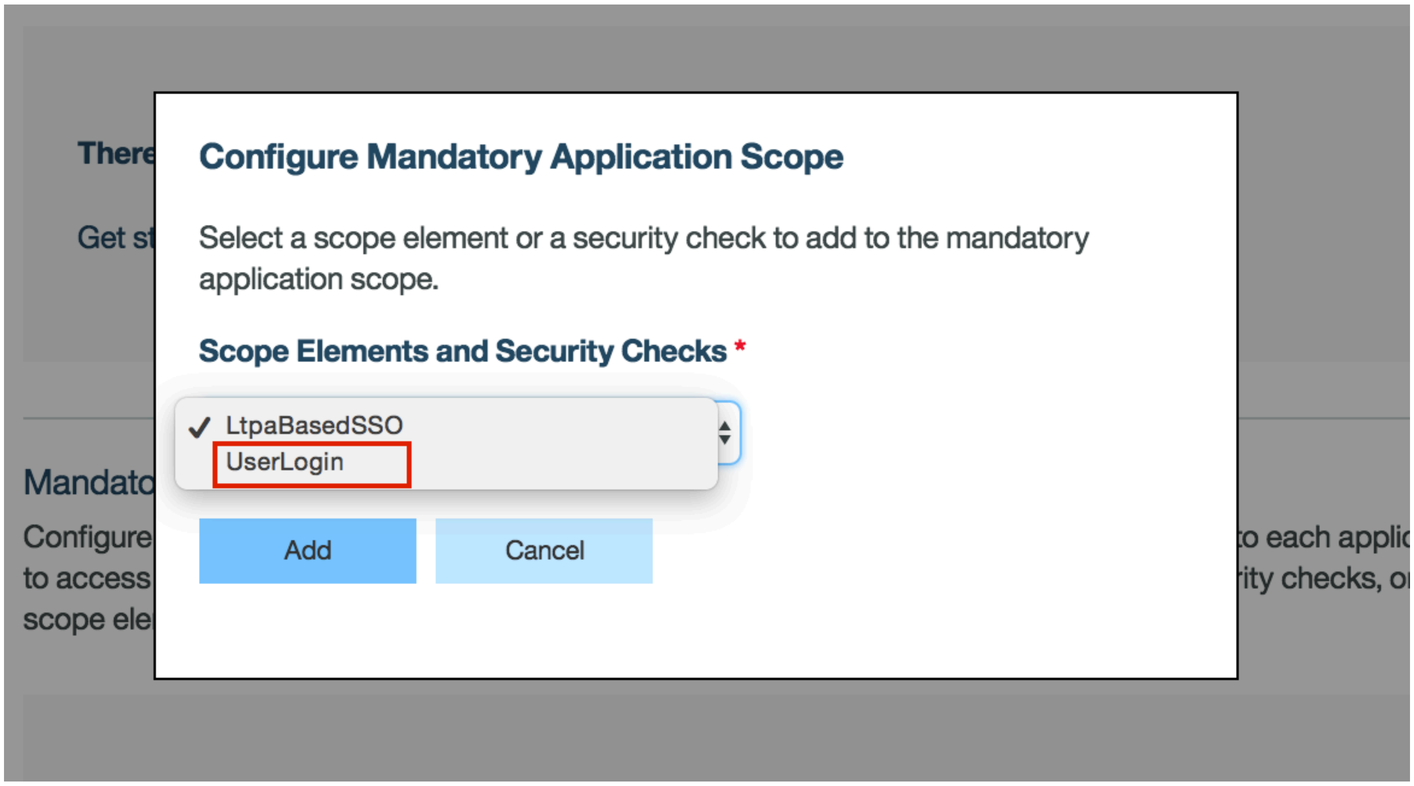
Mandatory Application Scope

Configure a mandatory application scope to define an authorization logic that will be applied to each application attempt to access a protected resource. You can include in the scope any predefined or custom security checks, or mapped scope elements.

There is currently no mandatory-scope definition for this application.

Get started by clicking "Add to Scope".

2. In the drop down list selete the **UserLogin** scope



Configure Mandatory Application Scope

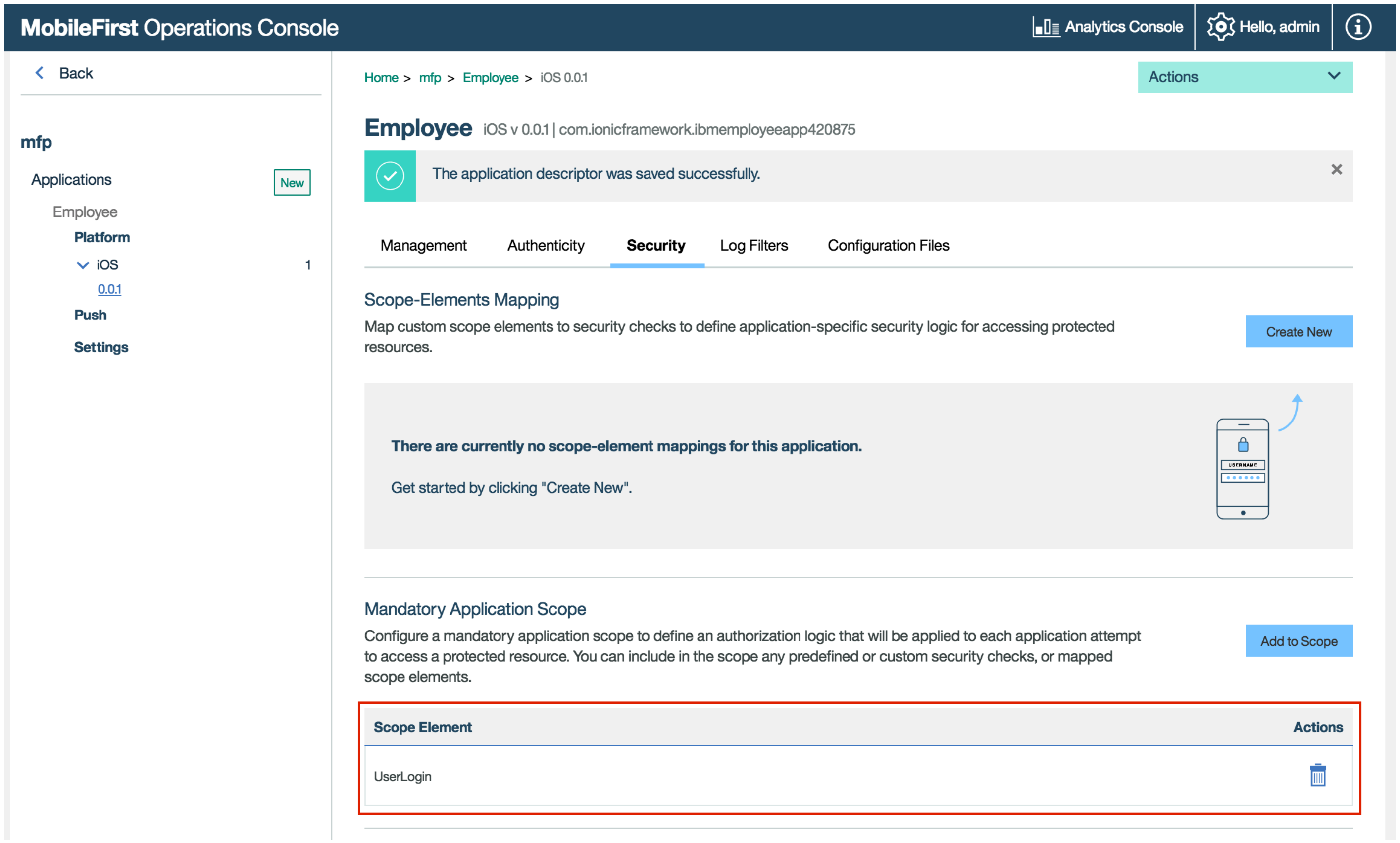
Select a scope element or a security check to add to the mandatory application scope.

Scope Elements and Security Checks *

✓ LtpaBasedSSO
UserLogin

Add Cancel

3. **Press** the "Add" button



MobileFirst Operations Console

Employee iOS v 0.0.1 | com.ionicframework.ibmemployeeapp420875

The application descriptor was saved successfully.

Scope-Elements Mapping

Map custom scope elements to security checks to define application-specific security logic for accessing protected resources.

There are currently no scope-element mappings for this application.

Get started by clicking "Create New".

Mandatory Application Scope

Configure a mandatory application scope to define an authorization logic that will be applied to each application attempt to access a protected resource. You can include in the scope any predefined or custom security checks, or mapped scope elements.

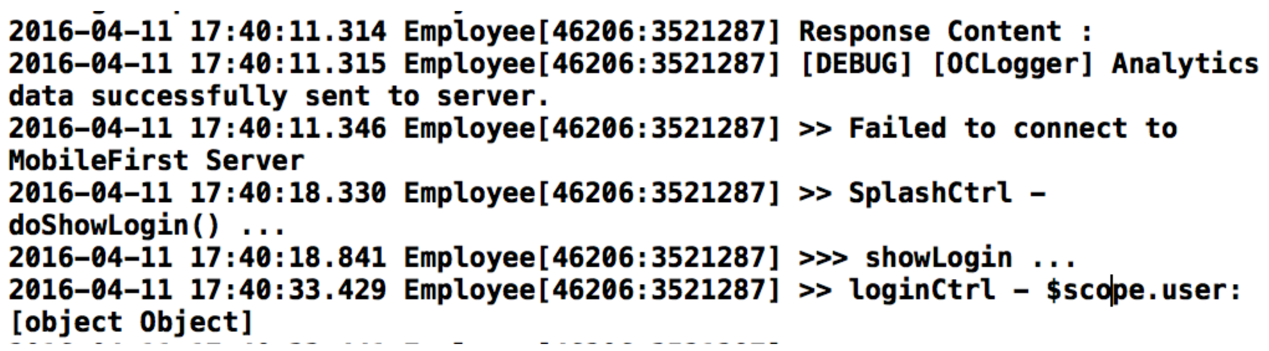
Scope Element	Actions
UserLogin	🗑

4. Change directory to the root of your IBMEmployeeApp directory

5. **Run** the following command to start the application

   ```
   cordova prepare
   cordova emulate
   ```

6. You have noticed that nothing happend after you press he login button, if you look at the debugger area within xcode you can see the following message: **Failed to connect to MobileFirst Server **



```
2016-04-11 17:40:11.314 Employee[46206:3521287] Response Content :
2016-04-11 17:40:11.315 Employee[46206:3521287] [DEBUG] [OCLogger] Analytics
data successfully sent to server.
2016-04-11 17:40:11.346 Employee[46206:3521287] >> Failed to connect to
MobileFirst Server
2016-04-11 17:40:18.330 Employee[46206:3521287] >> SplashCtrl -
doShowLogin() ...
2016-04-11 17:40:18.841 Employee[46206:3521287] >>> showLogin ...
2016-04-11 17:40:33.429 Employee[46206:3521287] >> loginCtrl - $scope.user:
[object Object]
```

Summary

Now that you set **Mandatory application scope** vs **Default_scope** the application cannot access the MobileFirst server unless the server validate the user first and issue a token, in the next lab we are going to modify the client side log to in-order to validate the user and gain access to the server and the adapter.

In case you got lost on the way

You can easily get to this stage by running the following command :

```
git checkout -f step-10
```