Lab 11 - How to secure your application (Client side)

In the previous lab we update the application **Mandatory application scope** to validate the user before we issue token. In this lab we are going to modify the client side app to send the credentials when the application start. We going to implement a challenge handler, The challenge handler will demonstrate a few additional features (APIs) such as the **preemptive login, logout** and obtainAccessToken.

> Note: For this lab there are snippets files included in the **/snippets** folder of your workspace which can be used to quickly copy/paste the large source code changes in the lab steps below.

Steps:

Update the wlClientInit

1. Open the **app.js** file and change the **wlClientInit()** method, remove the snippets file and copy the WLAuthorizationManager.obtainAccessToken() block

   Before

```
75  function wlCommonInit() {
76      console.log(">> wlCommonInit() ..." );
77      var serverUrl = WL.App.getServerUrl(function(success){
78          console.log(success);
79      }, function(fail){
80          console.log(fail);
81      });
82
83      //Calling to the MobileFirst Server
84      WLAuthorizationManager.obtainAccessToken().then(
85            function (accessToken) {
86              console.log(">> Success - Connected to MobileFirst Server");
87            },
88            function (error) {
89              console.log(">> Failed to connect to MobileFirst Server");
90            }
91      );
92      WL.Analytics.send();
93  };
```

After

```
function wlCommonInit() {
    console.log(">> wlCommonInit() ..." );
    var serverUrl = WL.App.getServerUrl(function(success){
        console.log(success);
    }, function(fail){
        console.log(fail);
    });
    WL.Analytics.send();
};
```

> Note: We going to obtain a access token when we inside the new implementation of the login method in the next steps.

Create new challenge handler

2. Open the **controllers.js** and change the look for the **ibmApp.controller('splashCtrl'...** controller and add the location service $location to the method signature

Before

```
51
52 ibmApp.controller('splashCtrl', ['$scope', '$stateParams', '$timeout', '$state', 'AuthenticateUserService',
   '$ionicPopup',  function ($scope, $stateParams, $timeout, $state, AuthenticateUserService, $ionicPopup) {
53      console.log(">> splashCtrl - ... ");
54
```

After

```
51
52 ibmApp.controller('splashCtrl', ['$scope', '$stateParams', '$timeout', '$state', 'AuthenticateUserService',
   '$ionicPopup', '$location', function ($scope, $stateParams, $timeout, $state, AuthenticateUserService,
   $ionicPopup, $location) {
53      console.log(">> splashCtrl - ... ");
54
```

3. Next search the **$scope.doLogin = function ()** delete entire function and replace it with the code below

```
/* using mfp challenge handler */
    $scope.doLogin = function () {
    console.log(">> loginCtrl - doLogin - $scope.user:" + $scope.user);

    if ($scope.isChallenged){
        console.log(">> loginCtrl - doLogin -  $scope.isChallenged == true");

        $scope.userLoginChallengeHandler.submitChallengeAnswer({
            'username': $scope.user.username,
            'password': $scope.user.password
        });
    } else {
        console.log(">> loginCtrl - doLogin -  $scope.isChallenged == false");

        WLAuthorizationManager.login("UserLogin",{
            'username':$scope.user.username,
            'password':$scope.user.password
        }).then( function () {
            console.log(">> WLAuthorizationManager.login - onSuccess");
            $state.transitionTo("main");
        },
        function (response) {
            console.log(">> WLAuthorizationManager.login - onFailure: " + JSON.st
ringify(response));
            $scope.showLoginError();
        });
    }
}
```

> Note: that in our scanrio we want to login a user **without any challenge being received.** showing a login screen as the first screen of the application, or showing a login screen after a logout, or a login failure. We call those scenarios **preemptive logins**.
>
> You cannot call the submitChallengeAnswer API if there is no challenge to answer. For those scenarios, the MobileFirst Platform Foundation SDK includes the login API: **WLAuthorizationManager.login()**, If the credentials are wrong, the security check will send back a challenge.
>
> It is the developer's responsibility to know when to use login vs submitChallengeAnswer based on the application's needs. One way to achieve this is to define a boolean flag, for example isChallenged, and set it to true when reaching handleChallenge or set it to false in any other cases (failure, success, initializing, etc). When the user clicks the Login button, you can dynamically choose which API to use:

4. Add the code below right after the "$soope.doLogin" code block ends.

> Note: the code below handle the challenge that return by the server when the application tries to acces protected resource or when it tries to get access token.

```
    $scope.isChallenged = false;
    $scope.securityCheckName = 'UserLogin';
    $scope.userLoginChallengeHandler = null;

    $scope.registerChallengeHandler = function(){
    console.log(">> in $scope.registerChllangeHandler ... ");
    $scope.userLoginChallengeHandler = WL.Client.createWLChallengeHandler($scope.
securityCheckName);
    $scope.userLoginChallengeHandler.securityCheckName = $scope.securityCheckName
;

    $scope.userLoginChallengeHandler.handleChallenge = function(challenge) {
        console.log(">> in UserLoginChallengeHandler - userLoginChallengeHandler.
handleChallenge ...");
        //show the login ...
        $scope.user = { username: "", password: ""};
        $scope.currentPath = $location.path();
        console.log(">> $location.path(): " + $location.path());
        $state.transitionTo("splash");
        $scope.isChallenged = true;
        var statusMsg = "Remaining Attempts: " + challenge.remainingAttempts;
        if (challenge.errorMsg !== null){
            statusMsg = statusMsg + "<br/>" + challenge.errorMsg;
            $timeout(function(){
                //want to show only when submit user/pass not when token expired
...
                if($scope.currentPath == "/"){
                    $scope.showLoginError(statusMsg);
                }
             }, 300);
        }
        console.log(">>> statusMsg : " + statusMsg);
    };

    $scope.userLoginChallengeHandler.processSuccess = function(data) {

        console.log(">> in UserLoginChallengeHandler - userLoginChallengeHandler.
processSuccess ...");
        $scope.isChallenged = false;
        $timeout(function(){
```

```
                $scope.user = { username: "", password: ""};
        }, 200);
        $state.transitionTo("main");
    };

    $scope.userLoginChallengeHandler.handleFailure = function(error) {
        console.log(">> in UserLoginChallengeHandler - userLoginChallengeHandler.
handleFailure ...");
        console.log(">> handleFailure: " + error.failure);
        $scope.isChallenged = false;
        if (error.failure !== null){
            alert(error.failure);
        } else {
            alert("Failed to login.");
        }
    };
}
```

> Note: A **challengeHandler** is responsible for handling challenges sent by the MobileFirst server, such as displaying a login screen, collecting credentials and submitting them back to the security check. we use the **WL.Client.createWLChallengeHandler()** API method to create and register a challenge Handler, In order for the challenge handler to listen for the right challenges, you must tell the framework to associate the challenge handler with a specific security check name. This is done by creating the challenge handler with the security check like this: **someChallengeHandler = WL.Client.createWLChallengeHandler("the-securityCheck-name");**
>
> Note: The minimum requirement from the WLChallengeHandler protocol is to implement the handleChallenge() method, that is responsible for asking the user to provide the credentials. The handleChallenge method receives the challenge as a JSON Object.
>
> Note: Once the credentials have been collected from the UI, use WLChallengeHandler's **submitChallengeAnswer()** to send an answer back to the security check.
>
> Note: **Handling failures** Some scenarios may trigger a failure (such as maximum attempts reached). To handle these, implement WLChallengeHandler's **handleFailure()**
>
> Note: **Handling successes** In general successes are automatically processed by the framework to allow the rest of the application to continue, Optionally you can also choose to do something before the framework closes the challenge handler flow, by implementing WLChallengeHandler's **processSuccess()**. Here again, the content and structure of the success JSON object depends on what the security check sends.

5. Let's also modify the **showLoginError** method so we can display the error messages that are been

returned from the server, replace the **showLoginError** with the code below

```
//show alert login error ...
$scope.showLoginError = function(msg) {
    if(msg == null || msg == undefined) msg = 'Please check your username and pas
sword and try again';
    var alertPopup = $ionicPopup.alert({
        title: 'Login Error!',
        template: msg
    });
    alertPopup.then(function(res) {
        console.log('>> Thank you for trying ...');
    });
};
```

Before

```
167       //show alert login error ...
168       $scope.showLoginError = function() {
169           var alertPopup = $ionicPopup.alert({
170               title: 'Login Error!',
171               template: 'Please check your username and password and try again'
172           });
173           alertPopup.then(function(res) {
174               console.log('>> Thank you for trying ...');
175           });
176       };
```

After

```
140       //show alert login error ...
141       $scope.showLoginError = function(msg) {
142           if(msg == null || msg == undefined) msg = 'Please check your username and password and try again';
143           var alertPopup = $ionicPopup.alert({
144               title: 'Login Error!',
145               template: msg
146           });
147           alertPopup.then(function(res) {
148               console.log('>> Thank you for trying ...');
149           });
150       };
```

6. Next add a call to register our challengeHandler navigate to the bottom of the page and add the following code below just before the **},3000);**

```
$scope.registerChallengeHandler();
```

Before

```
183      $timeout(function(){
184          $scope.moveSplashBox();
185          var event = {viewLoad: 'Splash View'};
186          if(WL!=null && WL!=undefined) WL.Analytics.log(event, 'Splash View - loaded');
187          $scope.registerChallengeHandler();
188      }, 3000);
189
```

After

```
183      $timeout(function(){
184          $scope.moveSplashBox();
185          var event = {viewLoad: 'Splash View'};
186          if(WL!=null && WL!=undefined) WL.Analytics.log(event, 'Splash View - loaded');
187      }, 3000);
188
```

You new code should like this below

```
51
52  ibmApp.controller('splashCtrl', ['$scope', '$stateParams', '$timeout', '$state', 'AuthenticateUserService',
    '$ionicPopup', '$location', function ($scope, $stateParams, $timeout, $state, AuthenticateUserService,
    $ionicPopup, $location) {
53      console.log(">> splashCtrl - ... ");
54      $scope.user = { username: "", password: "" }
55      var code = document.getElementsByClassName('code-wrapper');
56      for (var i = 0; i < code.length; i++) {
57          code[i].addEventListener('click', function() {
58              this.classList.toggle('active');
59          });
60      }
61
62      /* using mfp challenge handler */
63      $scope.doLogin = function () {
64              console.log(">> loginCtrl - doLogin - $scope.user:" + $scope.user);
65              if ($scope.isChallenged){
66                  console.log(">> loginCtrl - doLogin -  $scope.isChallenged == true");
67                  $scope.userLoginChallengeHandler.submitChallengeAnswer({
68                      'username': $scope.user.username,
69                      'password': $scope.user.password
70                  });
71              } else {
72                  console.log(">> loginCtrl - doLogin -  $scope.isChallenged == false");
73                  WLAuthorizationManager.login("UserLogin",{
74                      'username':$scope.user.username,
75                      'password':$scope.user.password
76                  }).then( function () {
77                      console.log(">> WLAuthorizationManager.login - onSuccess");
78                      $state.transitionTo("main");
79                  },
80                  function (response) {
81                      console.log(">> WLAuthorizationManager.login - onFailure: " + JSON.stringify(response));
82                      $scope.showLoginError();
83                  });
84              }
85      }
86
```

```
87
88      $scope.isChallenged = false;
89      $scope.securityCheckName = 'UserLogin';
90      $scope.userLoginChallengeHandler = null;
91
92      $scope.registerChallengeHandler = function(){
93          console.log(">> in $scope.registerChllangeHandler ... ");
94          $scope.userLoginChallengeHandler = WL.Client.createWLChallengeHandler($scope.securityCheckName);
95          $scope.userLoginChallengeHandler.securityCheckName = $scope.securityCheckName;
96
97          $scope.userLoginChallengeHandler.handleChallenge = function(challenge) {
98              console.log(">> in UserLoginChallengeHandler - userLoginChallengeHandler.handleChallenge ...");
99              //show the login ...
100             $scope.user = { username: "", password: ""};
101             $scope.currentPath = $location.path();
102             console.log(">> $location.path(): " + $location.path());
103             $state.transitionTo("splash");
104             $scope.isChallenged = true;
105             var statusMsg = "Remaining Attempts: " + challenge.remainingAttempts;
106             if (challenge.errorMsg !== null){
107                 statusMsg = statusMsg + "<br/>" + challenge.errorMsg;
108                 $timeout(function(){
109                     //want to show only when submit user/pass not when token expired ...
110                     if($scope.currentPath == "/"){
111                         $scope.showLoginError(statusMsg);
112                     }
113                  }, 300);
114             }
115             console.log(">>> statusMsg : " + statusMsg);
116         };
117
118         $scope.userLoginChallengeHandler.processSuccess = function(data) {
119             console.log(">> in UserLoginChallengeHandler - userLoginChallengeHandler.processSuccess ...");
120             $scope.isChallenged = false;
121             $timeout(function(){
122                     $scope.user = { username: "", password: ""};
123             }, 200);
124             $state.transitionTo("main");
125         };
126
127         $scope.userLoginChallengeHandler.handleFailure = function(error) {
128             console.log(">> in UserLoginChallengeHandler - userLoginChallengeHandler.handleFailure ...");
129             console.log(">> handleFailure: " + error.failure);
130             $scope.isChallenged = false;
131             if (error.failure !== null){
132                 alert(error.failure);
133             } else {
134                 alert("Failed to login.");
135             }
136         };
137     }
138
```

```
138
139
140         //show alert login error ...
141         $scope.showLoginError = function(msg) {
142             if(msg == null || msg == undefined) msg = 'Please check your username and password and try again';
143             var alertPopup = $ionicPopup.alert({
144                 title: 'Login Error!',
145                 template: msg
146             });
147             alertPopup.then(function(res) {
148                 console.log('>> Thank you for trying ...');
149             });
150         };
151
152         $scope.doShowLogin = function(){
153             console.log(">> SplashCtrl - doShowLogin() ... ");
154             $scope.hideSplashBox();
155         }
156
157         $scope.moveSplashBox = function() {
158             var splashNextBox = document.getElementById('splash-next-box');
159             move(splashNextBox).ease('in-out').y(-415).duration('0.5s').end();
160             //move('.signInMsg').rotate(360).end();
161         };
162
163         $scope.hideSplashBox = function() {
164             var splashNextBox = document.getElementById('splash-next-box');
165             move(splashNextBox).ease('in-out').y(415).duration('0.5s').end(
166                 function(){
167                     console.log(">>> showLogin ... ");
168                     var loginBox = document.getElementById('login-box');
169                     move(loginBox).ease('in-out').y(-415).duration('0.5s').end();
170                 }
171             );
172             //move(loginBox).ease('in-out').y(-385).duration('0.5s').end
173         };
174
175         $timeout(function(){
176             //fix android bug where render splash screen incorrect.
177             var splashNextBox = document.getElementById('splash-next-box');
178             var loginBox = document.getElementById('login-box');
179             splashNextBox.style.display = 'block';
180             loginBox.style.display = 'block'
181         }, 415);
182
183         $timeout(function(){
184             $scope.moveSplashBox();
185             var event = {viewLoad: 'Splash View'};
186             if(WL!=null && WL!=undefined) WL.Analytics.log(event, 'Splash View - loaded');
187             $scope.registerChallengeHandler();
188         }, 3000);
189
190     }])
```

7. Next let's change the logout method, locate the **ibmApp.controller('appCtrl'** and replace the **$scope.logout** function with the code below :

```
$scope.logout = function() {
    console.log(">> in appCtrl - logout");
    $timeout(function(){
                $scope.user = { username: "", password: ""};
    }, 200);
    WLAuthorizationManager.logout("UserLogin").then(
        function () {
            console.log(">> logout onSuccess");
            $state.transitionTo("splash");
        },
        function (response) {
            console.log(">> logout onFailure: " + JSON.stringify(response));
            $state.transitionTo("splash");
        });
}
```

> Note : The MobileFirst Platform Foundation SDK provides a logout API to logout from a specific security check.

Before

```
ibmApp.controller('appCtrl', function ($scope, $state) {
    $scope.logout = function () {
        console.log(">> in appCtrl >> logout ... ");
        $scope.user = {
            username: "",
            password: ""
        };
    }
})
```

After

```
ibmApp.controller('appCtrl', function ($scope, $state) {
    $scope.logout = function() {
        console.log(">> in appCtrl - logout");
        $timeout(function(){
                    $scope.user = { username: "", password: ""};
        }, 200);
        WLAuthorizationManager.logout("UserLogin").then(
            function () {
                console.log(">> logout onSuccess");
                $state.transitionTo("splash");
            },
            function (response) {
                console.log(">> logout onFailure: " + JSON.stringify(response));
                $state.transitionTo("splash");
            });
        }
})
```

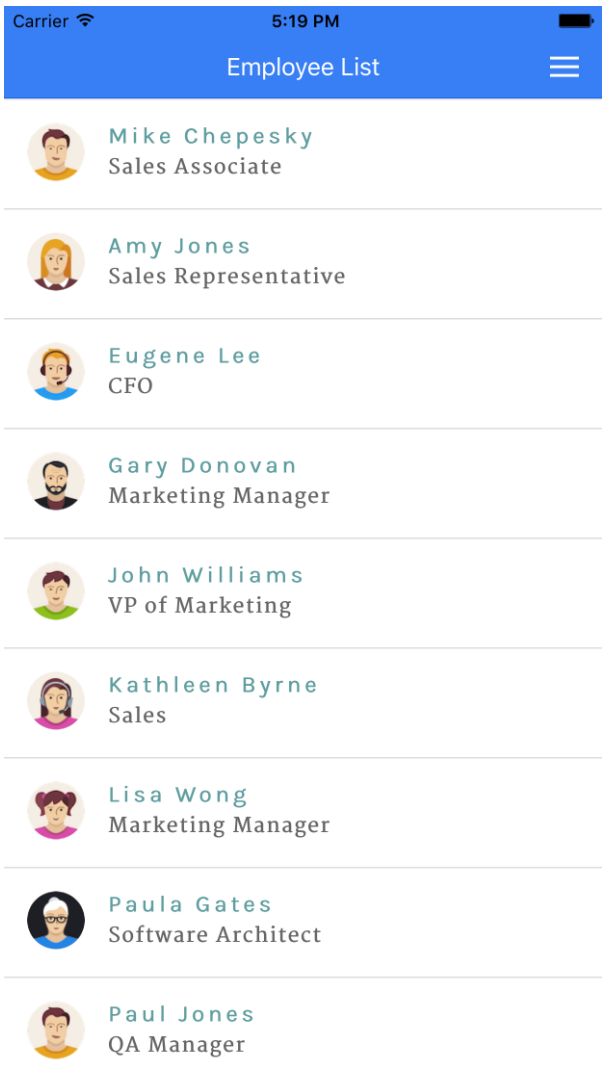
8. **Save** your changes.

Test the client side application

1. Run the application by running

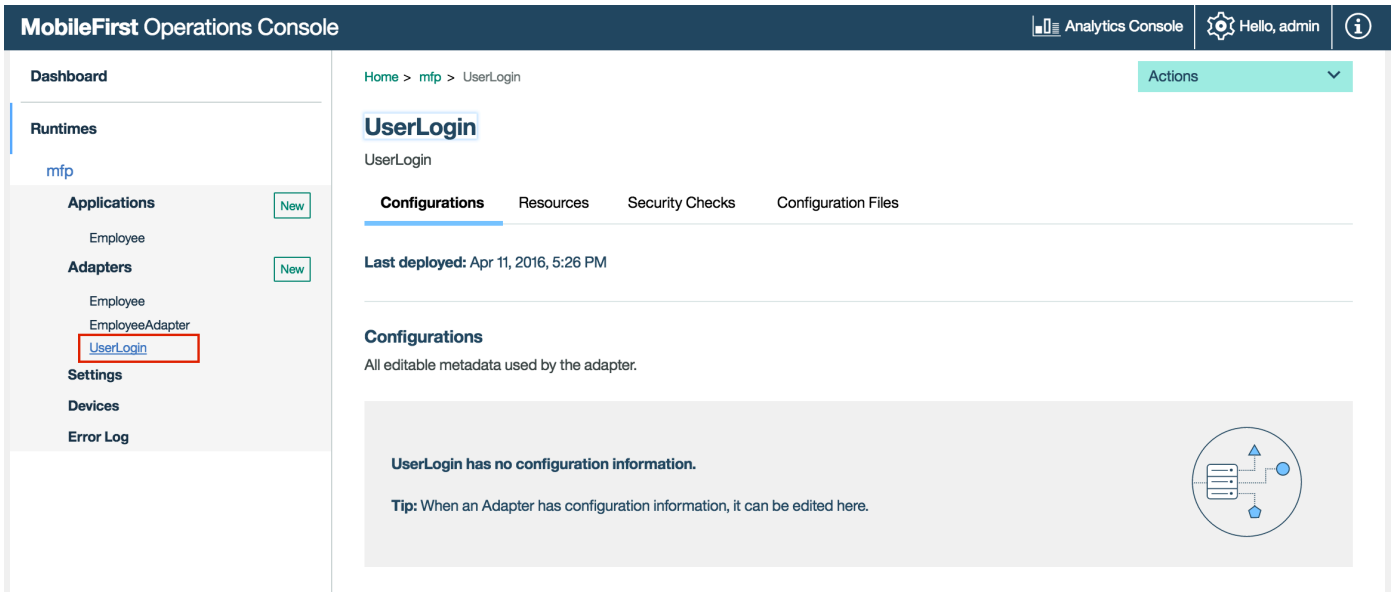
```
cordova prepare
cordova emulate
```

2. Use any combination of matching username and password to login for example demo/demo, you will see the employee list below.

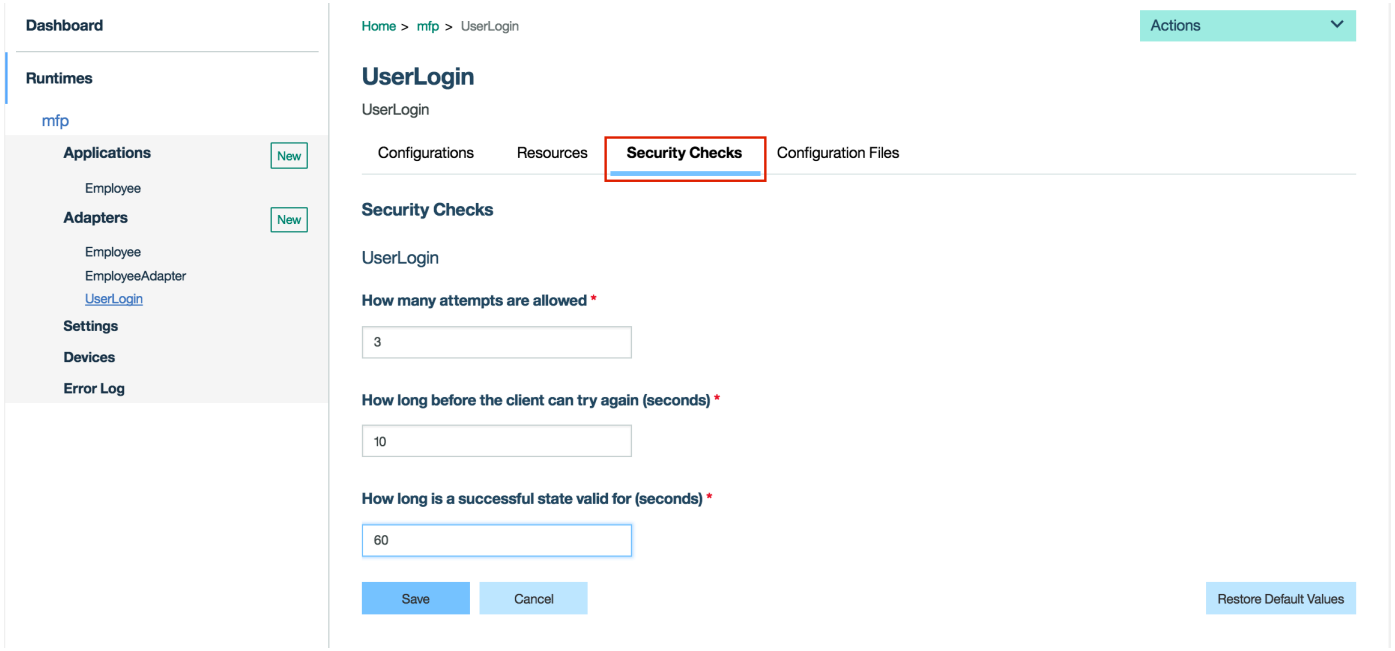


3. By default the application timeout is 60 seconds, lets open the web console and change the "How long the successful sate valid for (seconds)" to 10 seconds.

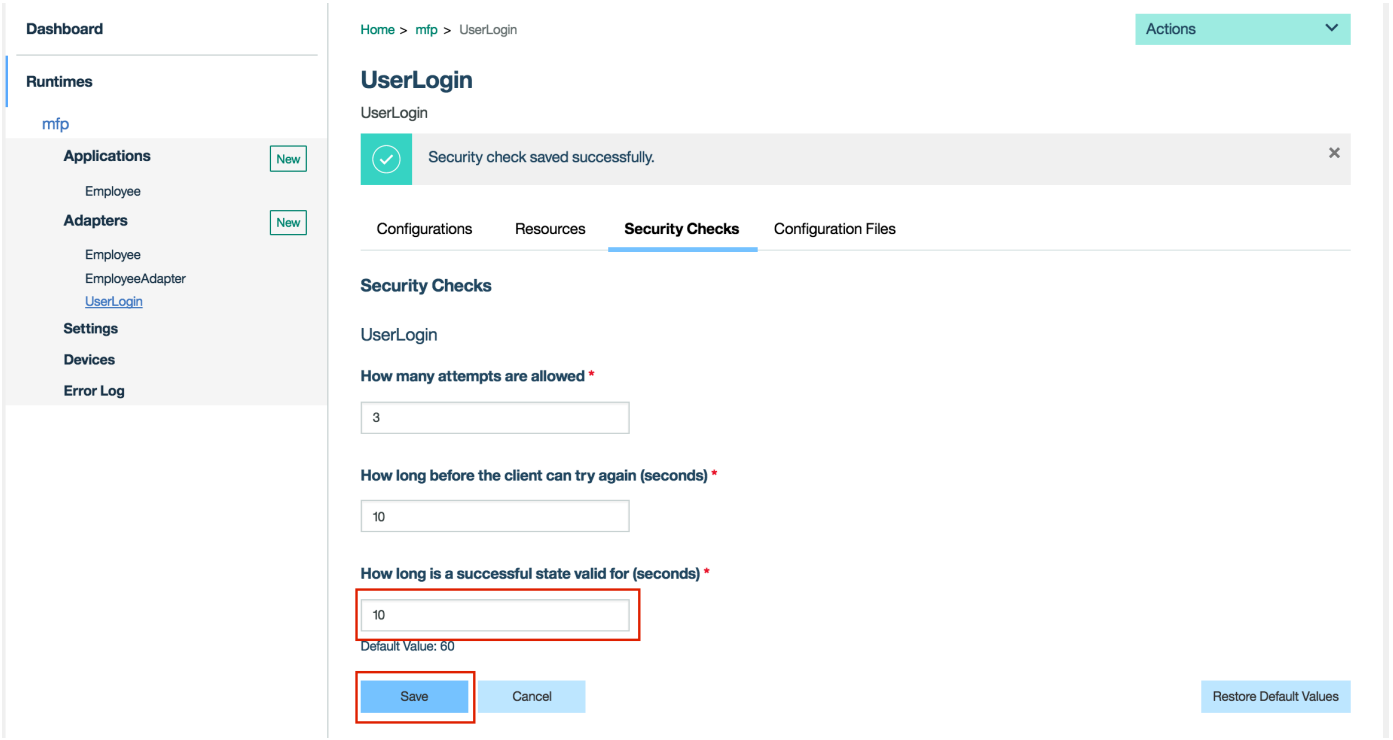
4. Select the **UserLogin** adapter under the adapter list on the left side menu.



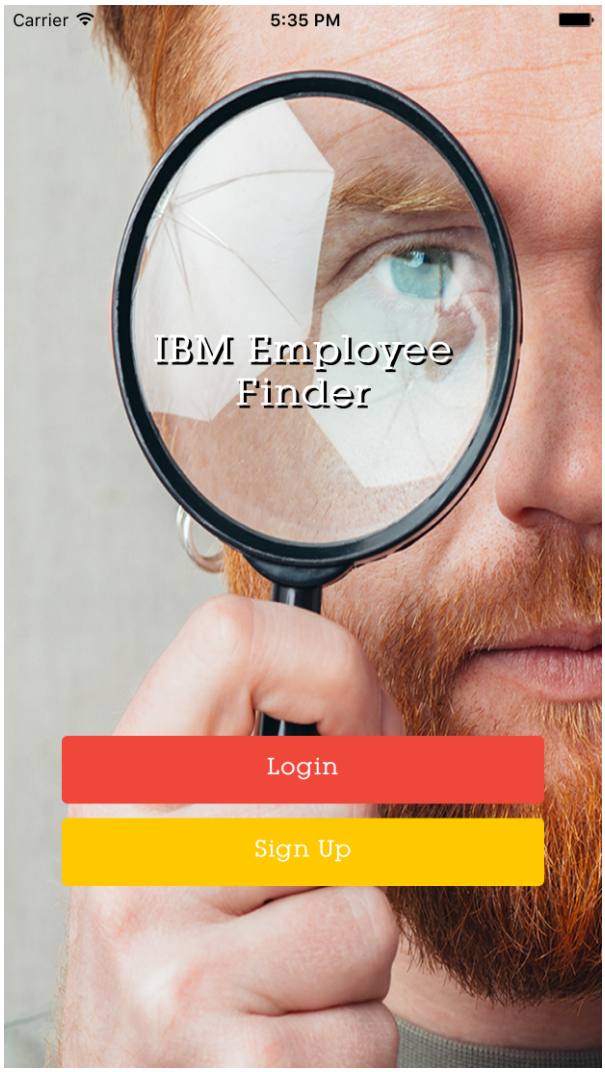
5. Select the *SecurityChecks* tabs.



6. Change the value to 10 seconds and press the "Save" button.

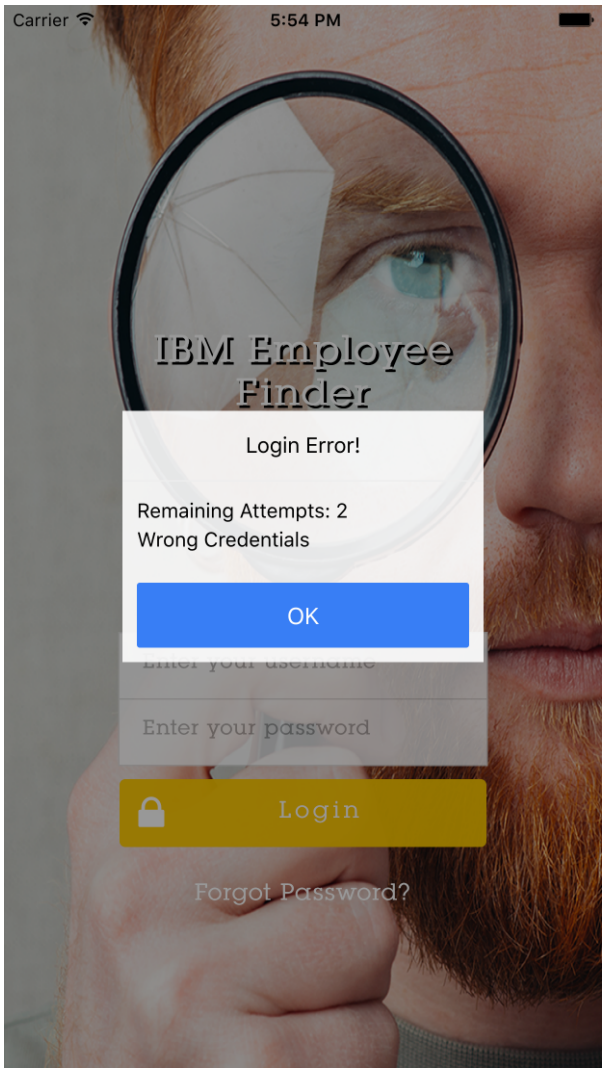


7. Go back the the device emulator and press on one of the employees.



> Note: The application log you out automatically since the token is not valid any more.

8. **Login** again but this time use un-matching values for the username and password and let's take a look at the error message that we got.



> Note: This is the error message we that was returned by the server, you can change the number of attempts and the error message by updating the UserLogin security check, or change the the attempts using the MFP web console.

Summary

You In this lab, you used the used the preemptive logins concept which is classic solution for scenarios where showing a login screen as the first screen of the application, or showing a login screen after a logout, or a login failure. you used the **WLAuthorizationManager.login()** API method to login the user without any challenge being received, you also use the Use the **WL.Client.createWLChallengeHandler()** API method to create and register a challenge Handler.

In case you got lost on the way

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

```
git checkout -f step-11
```