Lab 3: Mobile Foundation V8 - Securing Backend Calls with User Authentication

In this lab we will demonstrate how you can leverage MobileFirst Platform Foundation to secure access to enterprise backend resources. You will add security checks that will get called when accessing protected backend resources via MobileFirst Platform Foundation adapters. This application is a hybrid mobile application that leverages the Ionic2/Angular2 framework. Experience with Javascript, Ionic2 and Angular2 is recommended but not necessary to complete this lab.

Prerequisites

Both Lab 1 (Adding the MFP SDK) and Lab 2 (Using Java and Javascript adapters to make Backend Calls) should be completed prior to working on this lab or you can pull the required code from GitHub as per step 5 below. We also will be using the following required tools (these are already included and installed if you are using the supplied VMWare image):

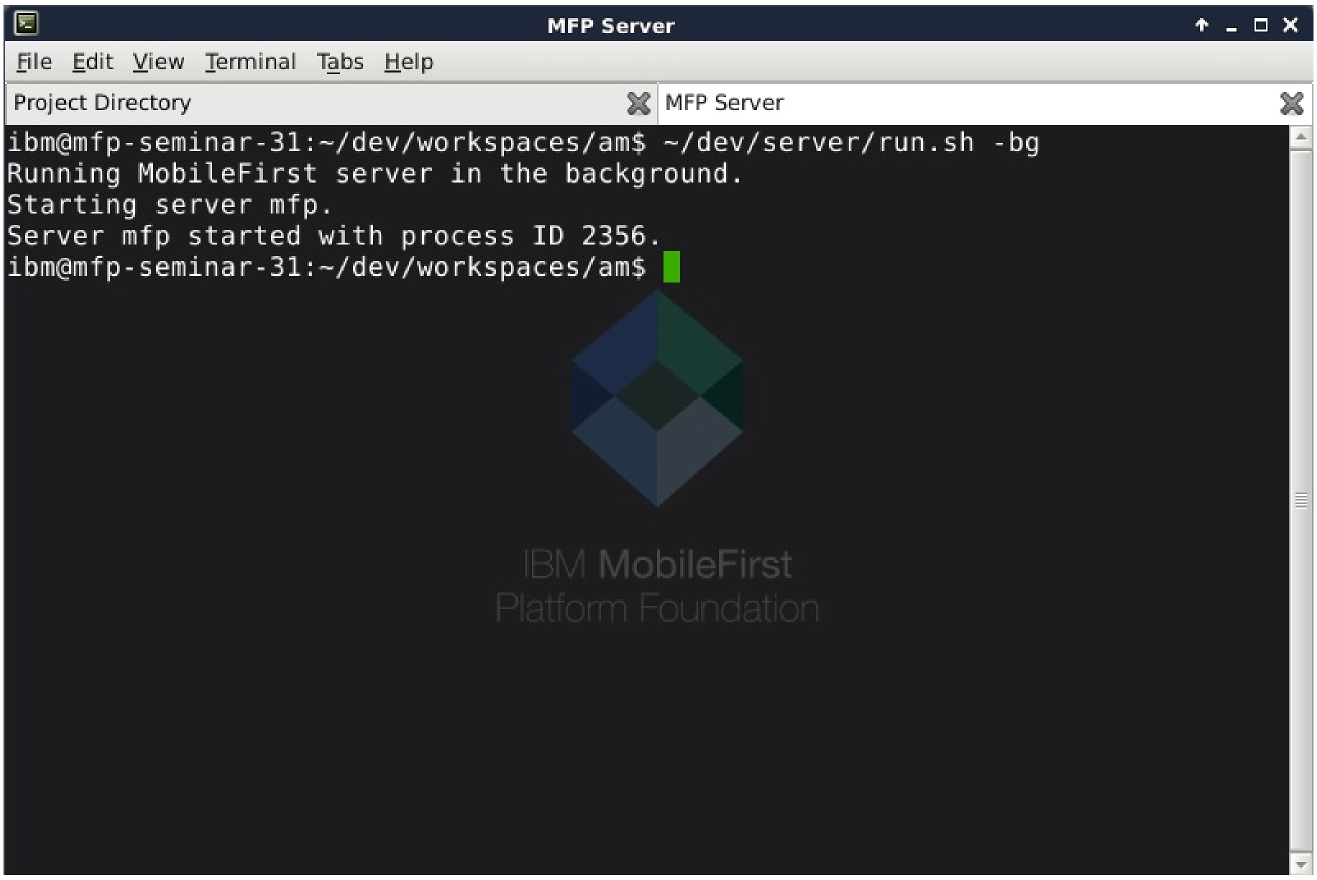
- An Integrated Development Environment (IDE) such as Visual Studio Code, Brackets or Atom to modify code as part of this lab.
- Apache Maven https://maven.apache.org/ needs to be installed for MobileFirst builds.
- Android development tools should be installed (SDK, AVD).

Check out the MobileFirst Foundation 8.0 Developer Labs Setup page at https://mobilefirstplatform.ibmcloud.com/labs/developers/8.0/setup/ for more details regarding lab details. Now let's get started!

Setup

If you completed all the previous labs to this point, then you can safely skip this section. If however you are starting with this lab, you need to complete these setup steps so that you can work through this lab. Let's proceed.

1. Start the lab virtual machine and log in with username **ibm** and password **QQqq1234**.
2. Open a terminal window by clicking the icon on bottom left. Set the terminal title by clicking *Terminal-Set Title* and naming it **Project Directory**.
3. Navigate to this directory by typing in the command `cd ~/dev/workspaces`
4. Create a working directory for this lab by typing `mkdir am` and then change into this directory by typing `cd am`.
5. We will pull in the code needed for the lab now. Type the following commands into the terminal.
   ```
   git init
   git remote add origin https://GitHub.com/andriivasylchenko/advancedMessenger
   git pull origin lab2.63
   ```
6. Open a second terminal tab where we will start our MFP Dev Server. Set the terminal title by clicking *Terminal-Set Title* and naming it **MFP Server**. Type the following commands.
   ```
   ~/dev/server/run.sh -bg
   ```



```
ibm@mfp-seminar-31:~/dev/workspaces/am$ ~/dev/server/run.sh -bg
Running MobileFirst server in the background.
Starting server mfp.
Server mfp started with process ID 2356.
ibm@mfp-seminar-31:~/dev/workspaces/am$
```

You will see the server start in about 15-20 seconds. *Note: If you are not using the supplied vm image, your server will likely be in a different location.*

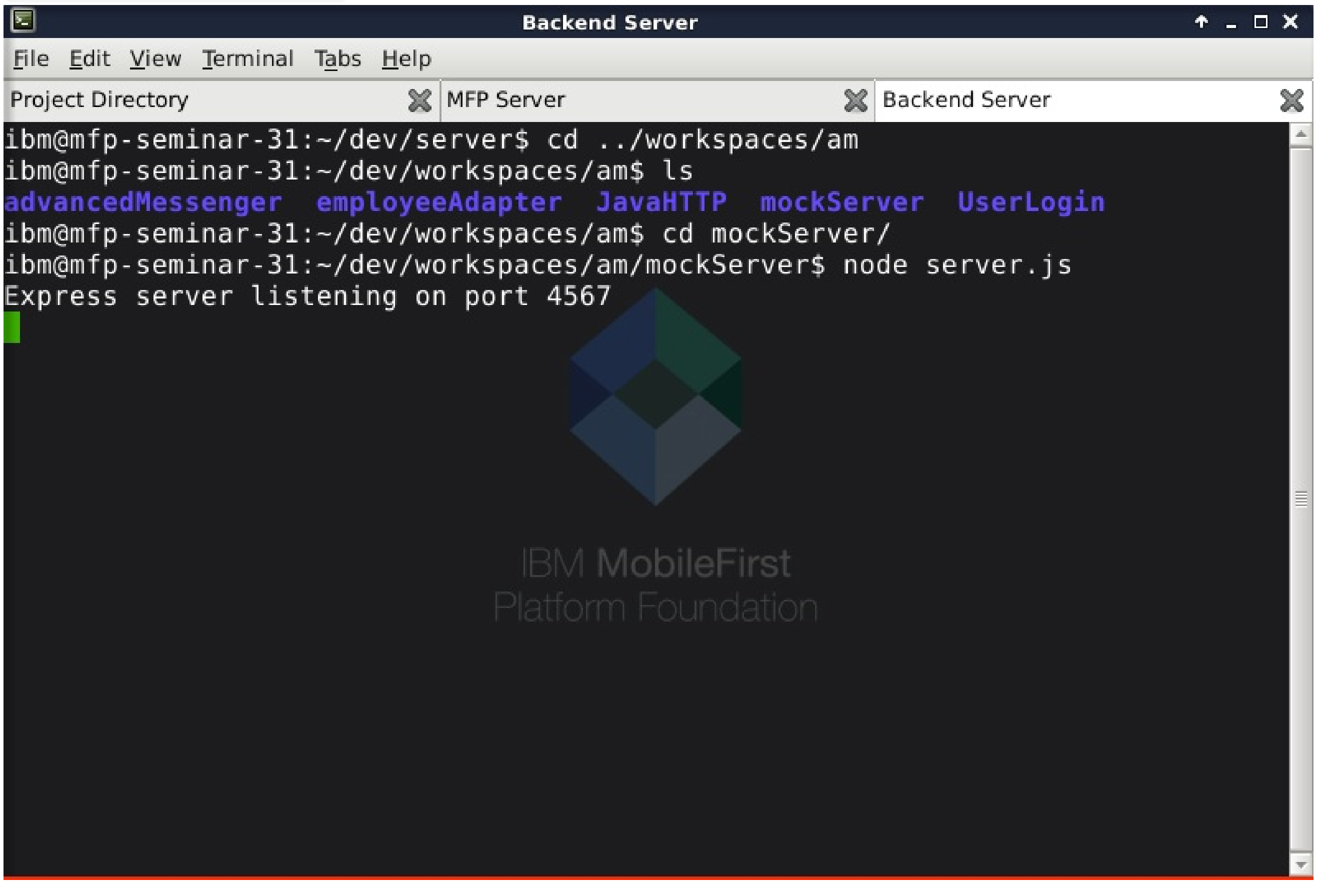
7. Click on the Project Directory terminal tab and enter the mobile app project directory like so:
   `cd advancedMessenger`
8. Install npm dependencies into the project by running the command:
   `npm install`
   You should see a tree of dependencies at the end and you may see a few warnings, but don't worry if you do.
9. Install the Android platform into the Cordova project by entering this command:
   `cordova platform add android`

10. Install the MobileFirst plugins by entering the following three commands.
    These plugins are for core MFP functions like security and analytics, offline secure storage, and push notifications, respectively.

```
cordova plugin add cordova-plugin-mfp
cordova plugin add cordova-plugin-mfp-jsonstore
cordova plugin add cordova-plugin-mfp-push
```

11. In the "Project Directory" terminal tab, type the command `gulp build` to rebuild the Ionic project.

12. Enter the following commands to register the app with the MobileFirst Server and propagate the changes to the app:
    ```
    mfpdev app register
    cordova prepare
    ```
13. Open a new tab in the terminal, set its title to **Backend Server**. Start the backend Node server project by entering the following commands:
    ```
    cd ~/dev/workspaces/am/mockServer
    node server.js
    ```


Backend Server
File Edit View Terminal Tabs Help
Project Directory
MFP Server
Backend Server
IBM MobileFirst Platform Foundation

```
ibm@mfp-seminar-31:~/dev/server$ cd ../workspaces/am
ibm@mfp-seminar-31:~/dev/workspaces/am$ ls
advancedMessenger  employeeAdapter  JavaHTTP  mockServer  UserLogin
ibm@mfp-seminar-31:~/dev/workspaces/am$ cd mockServer/
ibm@mfp-seminar-31:~/dev/workspaces/am/mockServer$ node server.js
Express server listening on port 4567
```

You can test that the mockserver is running by opening up an internet browser window and entering in the url :

localhost:4567/api

You should see the message displayed **Mock APIs are running**

14. Now we will need to build and deploy the adapters from lab 2. *Note: As per the prereqs, if you are not using the supplied lab image, then you must install Apache Maven*. From a terminal run the following commands to build and deploy the employee adapter:

    ```
    cd ~/dev/workspaces/am/employeeAdapter
    ```

    ```
    mfpdev adapter build
    ```

    On completion you should see the message "Successfully built adapter"

    ```
    mfpdev adapter deploy
    ```

    On completion you should see the message "Successfully deployed adapter"

15. We will repeat this process to deploy our HTTP Java (News) adapter. From your terminal run the following commands:

    ```
    cd ~/dev/workspaces/am/JavaHTTP
    ```

    ```
    mfpdev adapter build
    ```

    On completion you should see the message "Successfully built adapter"

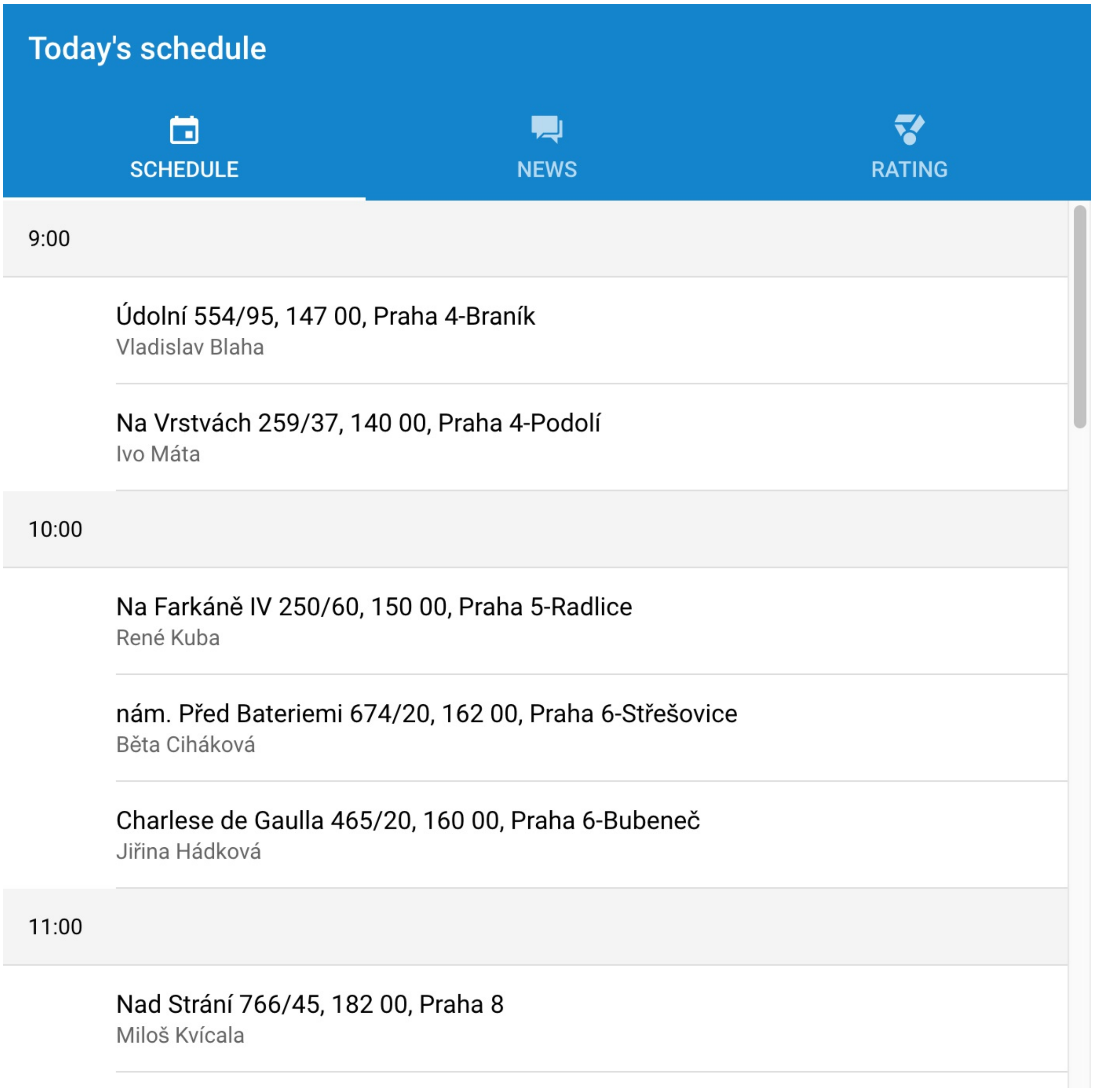
```
mfpdev adapter deploy
```

On completion you should see the message "Successfully deployed adapter"

16. Now you can preview your app by typing the following commands from the terminal:

```
cd ~/dev/workspaces/am/advancedMessenger
mfpdev app preview
```

Select **browser: Simple Browser Rendering** when prompted. This should bring up a preview of the app in your browser window



9:00

Údolní 554/95, 147 00, Praha 4-Braník
Vladislav Blaha

Na Vrstvách 259/37, 140 00, Praha 4-Podolí
Ivo Máta

10:00

Na Farkáně IV 250/60, 150 00, Praha 5-Radlice
René Kuba

nám. Před Bateriemi 674/20, 162 00, Praha 6-Střešovice
Běta Ciháková

Charlese de Gaulla 465/20, 160 00, Praha 6-Bubeneč
Jiřina Hádková

11:00

Nad Strání 766/45, 182 00, Praha 8
Miloš Kvícala

17. Finally we will watch for changes to our app using gulp. Open another terminal (or terminal tab) and enter the following commands:

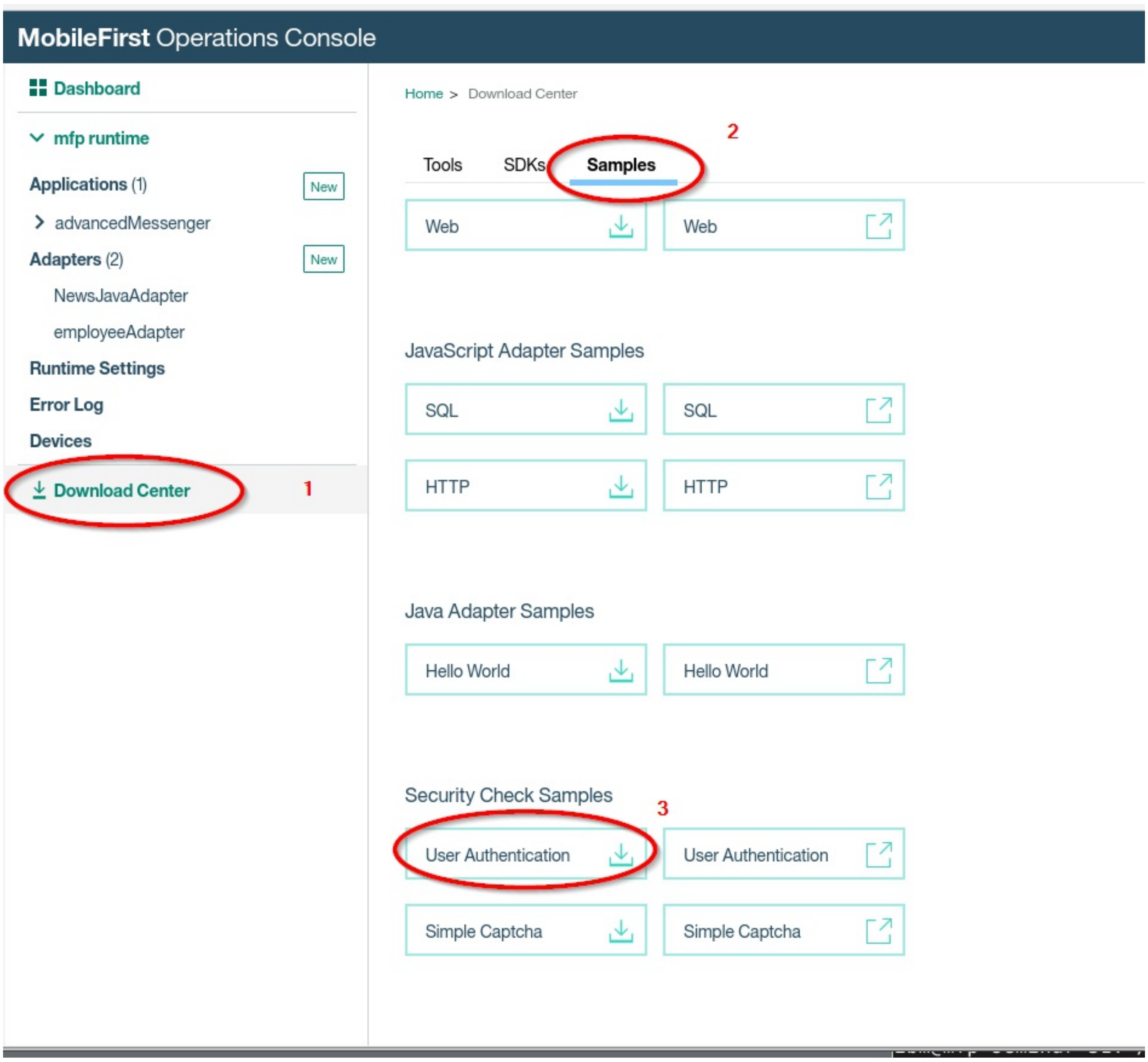
```
cd ~/dev/workspaces/am/advancedMessenger
gulp watch
```

Introduction

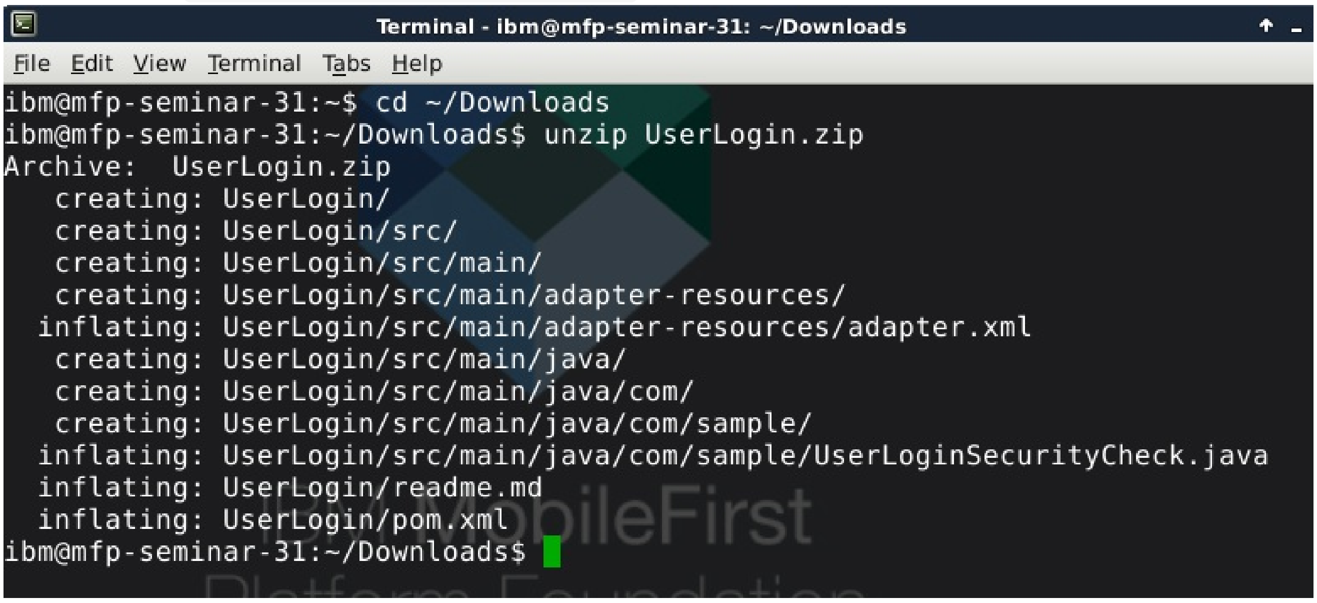
In this lab we will add some security to the adapters that we created and deployed in lab 2. In Part 1 of the lab, we will define a security test that is provided via a sample adapter. In Part 2 we will map the security test to backend resources that we want to protect. In Part 3 we will work with our application client-side code that needs to handle the security challenge that our newly-deployed server code returns. If authentication is successful, then the adapter call to our backend service will run.

Part 1 - Import the UserLogin Adapter Sample Code

1. Open a browser window and open the MobileFirst Operations Console (http://localhost:9080/mfpconsole). Sign in with the userid/password of admin/admin. From the lefthand navigation pane, navigate to **Download Center -> Samples** Scroll down to the bottom and underneath the **Security Check Samples** header download the **User Authentication** sample code (named UserLogin.zip).



2. From a terminal window, navigate to your ~/Downloads directory and type `unzip UserLogin.zip` file.


Terminal - ibm@mfp-seminar-31: ~/Downloads
File Edit View Terminal Tabs Help

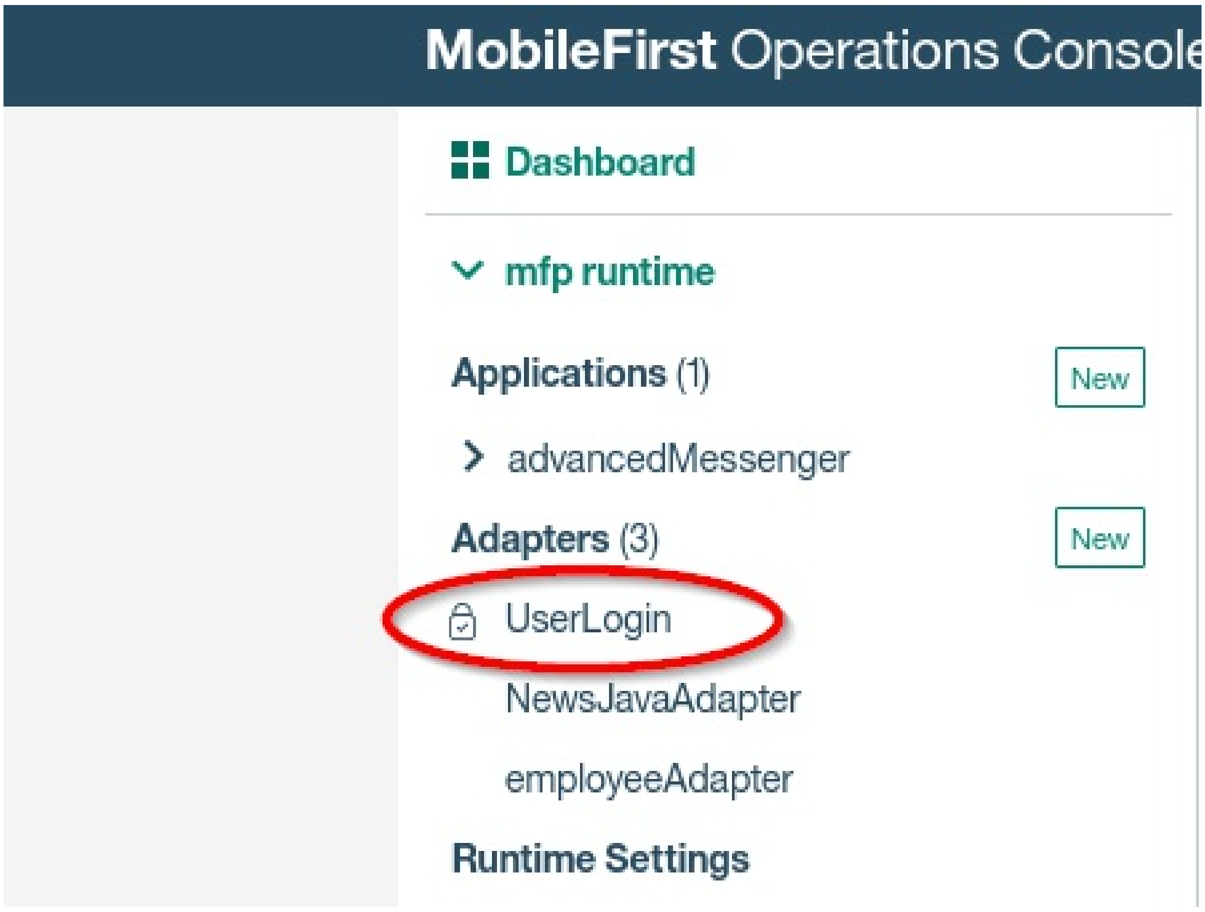
```
ibm@mfp-seminar-31:~$ cd ~/Downloads
ibm@mfp-seminar-31:~/Downloads$ unzip UserLogin.zip
Archive:  UserLogin.zip
   creating: UserLogin/
   creating: UserLogin/src/
   creating: UserLogin/src/main/
   creating: UserLogin/src/main/adapter-resources/
  inflating: UserLogin/src/main/adapter-resources/adapter.xml
   creating: UserLogin/src/main/java/
   creating: UserLogin/src/main/java/com/
   creating: UserLogin/src/main/java/com/sample/
  inflating: UserLogin/src/main/java/com/sample/UserLoginSecurityCheck.java
  inflating: UserLogin/readme.md
  inflating: UserLogin/pom.xml
ibm@mfp-seminar-31:~/Downloads$
```

3. From the terminal, move the sample code into our am workspace.

```
mv UserLogin ../dev/workspaces/am/
```

4. This sample code is an MFP adapter, which we will build and deploy on our MobileFirst Server. From a terminal run the following commands:
   ```
   cd ~/dev/workspaces/am/UserLogin
   ```
   ```
   mfpdev adapter build
   ```
   On completion you should see the message "Successfully built adapter"
5. Now lets deploy the adapter by running the command:
   ```
   mfpdev adapter deploy
   ```
   On completion you should see the message "Successfully deployed adapter"

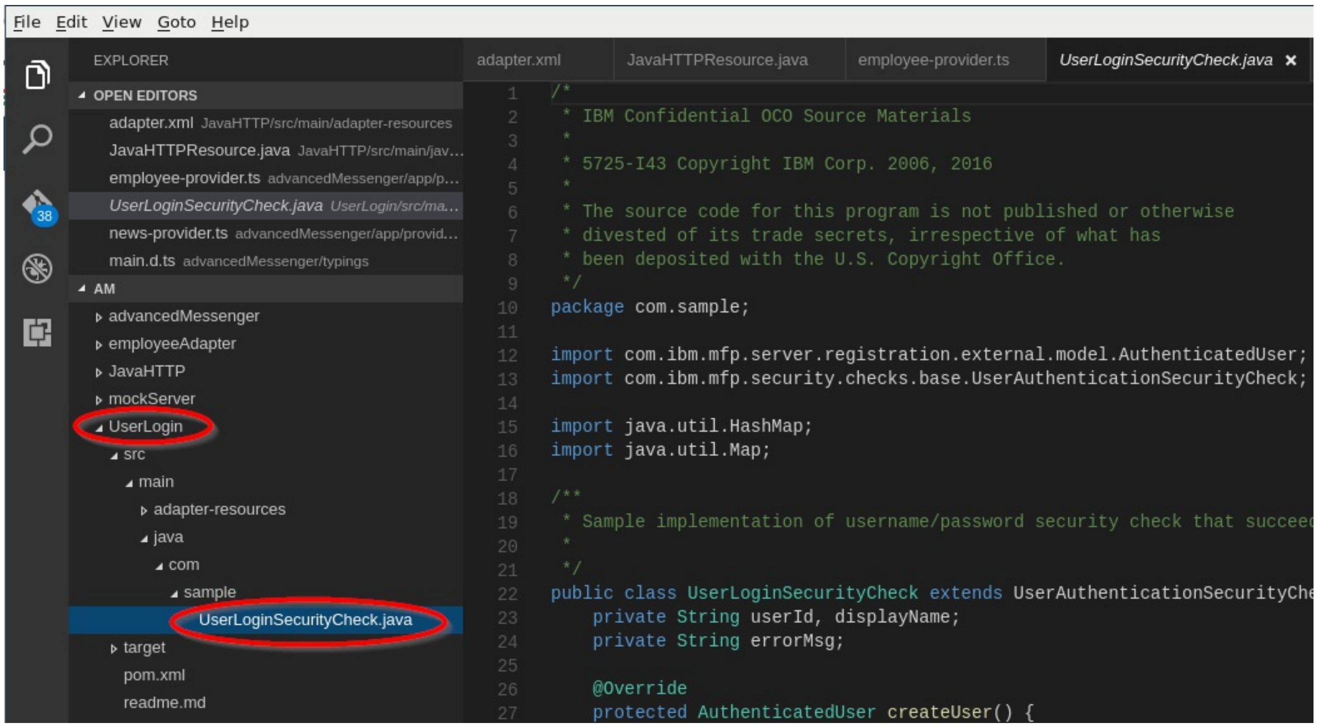
6. Now go back to the MobileFirst Operations Console in your browser and under Adapters you should now see the UserLogin adapter with a lock icon to the left of the adapter title. (Note, you may need to refresh your browser window to see the adapter).



We will take a closer look at this adapter and what it does in Part 2 of the lab.

Part 2 - Mapping the UserLogin Security check to adapter procedures

1. Now lets take a look at the UserLogin adapter code. Open your IDE (eg. VS code, Brackets, etc.) and open the ~/dev/workspaces/am folder. Now navigate to **UserLogin -> src -> main -> java -> com -> sample** and open the **UserLoginSecurityCheck.java** file.


File Edit View Goto Help
EXPLORER
OPEN EDITORS
adapter.xml JavaHTTP/src/main/adapter-resources
JavaHTTPResource.java JavaHTTP/src/main/jav...
employee-provider.ts advancedMessenger/app/p...
UserLoginSecurityCheck.java UserLogin/src/ma...
news-provider.ts advancedMessenger/app/provid...
main.d.ts advancedMessenger/typings
AM
advancedMessenger
employeeAdapter
JavaHTTP
mockServer
UserLogin
src
main
adapter-resources
java
com
sample
UserLoginSecurityCheck.java
target
pom.xml
readme.md
38
adapter.xml
JavaHTTPResource.java
employee-provider.ts
UserLoginSecurityCheck.java

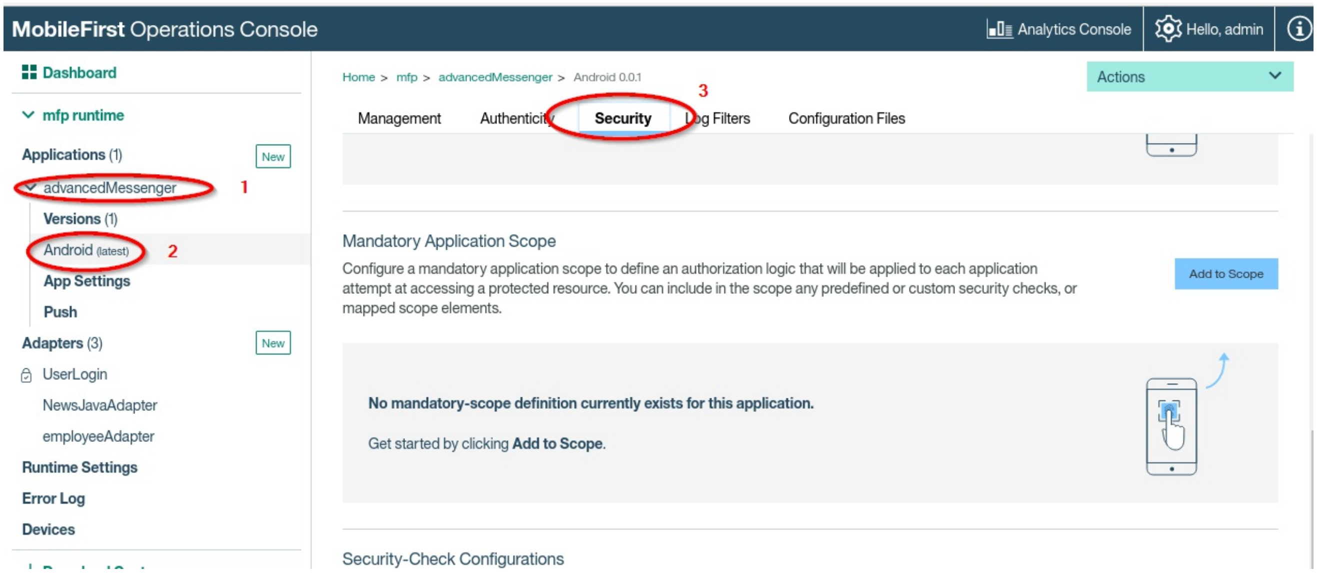
```
/*
 * IBM Confidential OCO Source Materials
 *
 * 5725-I43 Copyright IBM Corp. 2006, 2016
 *
 * The source code for this program is not published or otherwise
 * divested of its trade secrets, irrespective of what has
 * been deposited with the U.S. Copyright Office.
 */
package com.sample;

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 succeed
 *
 */
public class UserLoginSecurityCheck extends UserAuthenticationSecurityChe
    private String userId, displayName;
    private String errorMsg;

    @Override
    protected AuthenticatedUser createUser() {
```

2. Scroll down and look at the **validateCredentials** method at approximately line number 38. The code checks to make sure that both a username and password are supplied as parameters. It then checks if the username value equals the password value, then it sets the userId and displayName to the username value and then it returns a true value, otherwise an error message is set and false is returned. We will apply this security check to "secure" our backend calls.

```
if(username.equals(password)) {
    userId = username;
    displayName = username;
    return true
    } else ...
```

3. We can set our application settings to perform security checks when the app attempts to access a protected resource. From the MobileFirst Operations Console in our browser navigate to **Applications -> advancedMessenger -> Android** Now click the **Security** tab. Scroll down to see the **Scope-Elements Mapping** and **Mandatory Application Scope** sections.

   The MobileFirst Foundation platform can protect resources by defining a scope that specifies the required permissions for accessing the resource. For example you can assign a scope to a Javascript adapter procedure along with the required permissions to invoke that procedure. You can assign one or more security checks to a scope. Learn more about scopes here: Authentication and Security.

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

   No mandatory-scope definition currently exists for this application.

   Get started by clicking **Add to Scope**.

4. Now we will assign a scope to an adapter procedure that we want to protect. From your IDE navigate to the **am -> employeeAdapter -> src -> main -> adapter-resources ->**

folder and open the **adapter.xml** file. Now add the scope attribute to the **procedure** tag (at approximately line 26) so it looks like this:

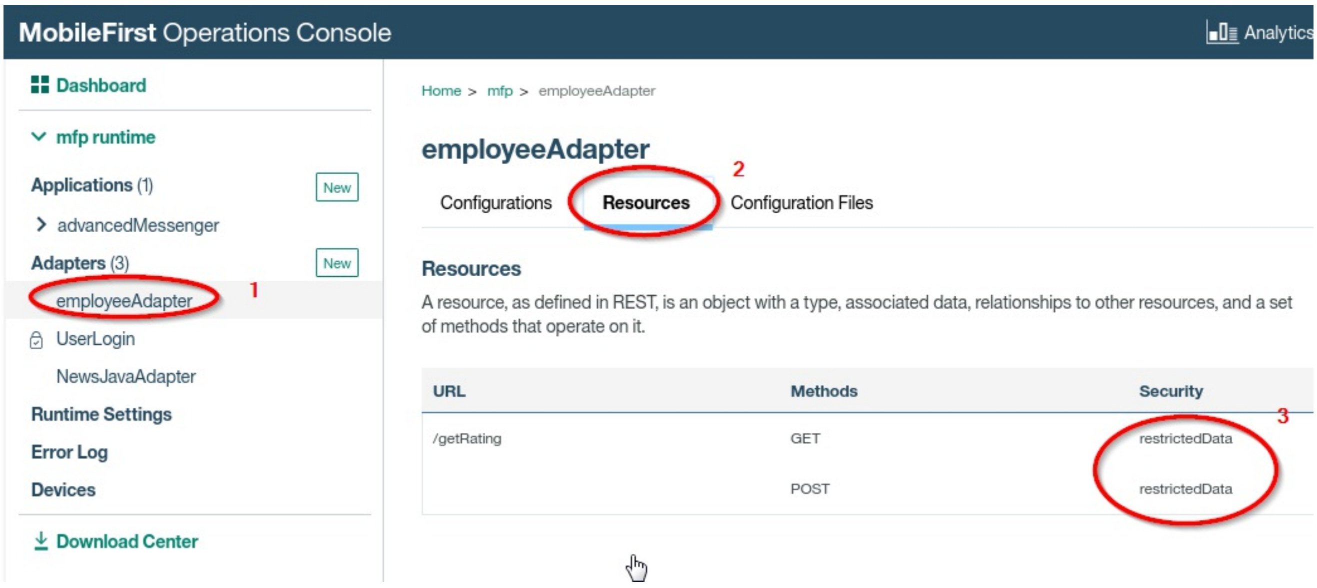
```
      <procedure name="getRating" scope="restrictedData"/>
```

Save your changes to the adapter.xml file.

5. Now we need to build and deploy our adapter to from a terminal navigate to the **~/dev/workspaces/am/employeeAdapter** directory. Run the command:
   `mfpdev adapter build`
   On completion you should see the message "Successfully built adapter"

6. Now we will redeploy the adapter by running the command:
   `mfpdev adapter deploy`
   On completion you should see the message "Successfully deployed adapter"

7. Now if we go back to our MobileFirst Operations Console, if we navigate to **adapters -> employeeAdapter** and look at the **Resources** tab, we now see resources secured using the newly defined **restrictedData** scope.

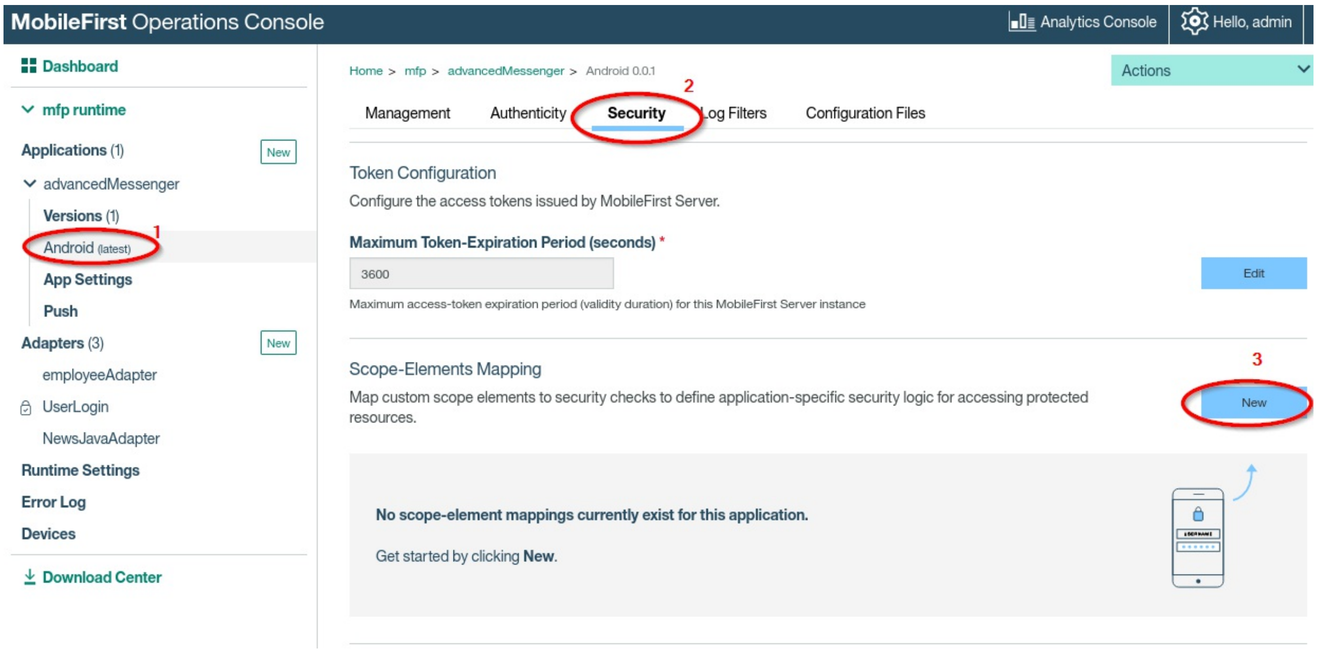


Resources

A resource, as defined in REST, is an object with a type, associated data, relationships to other resources, and a set of methods that operate on it.

URL	Methods	Security
/getRating	GET	restrictedData
	POST	restrictedData

8. In order to use our newly created restrictedData scope, we need to perform a mapping from the securityData scope to the UserLogin security check. From the MobileFirst Operations Console, left navigation pane, navigate to **Applications -> advancedMessenger -> Android** and then click on the **Security** tab. Navigate to the **Scope-Elements Mapping** section and click on the **New** button.



Token Configuration

Configure the access tokens issued by MobileFirst Server.

Maximum Token-Expiration Period (seconds) *

3600

Maximum access-token expiration period (validity duration) for this MobileFirst Server instance

Scope-Elements Mapping

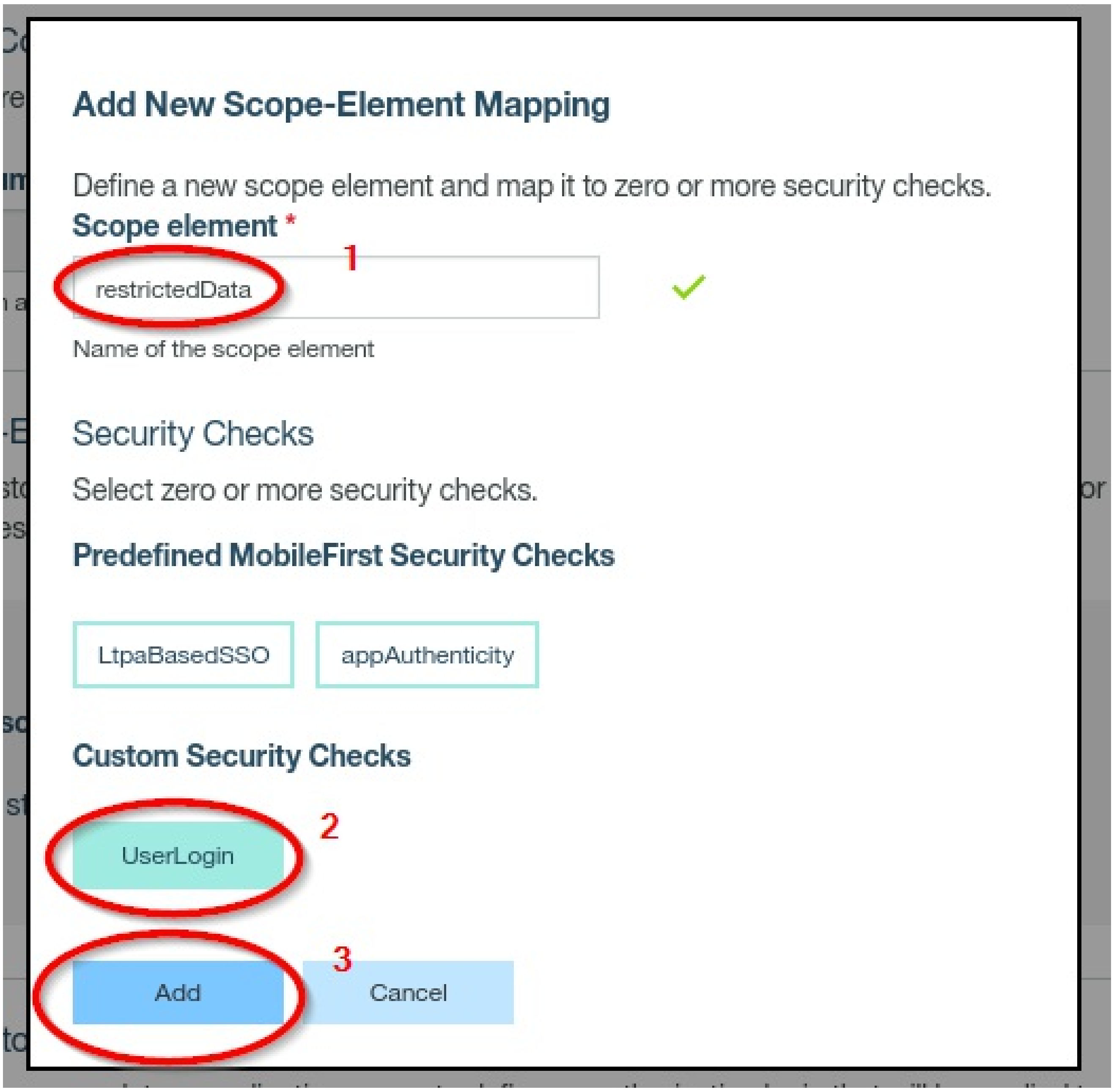
Map custom scope elements to security checks to define application-specific security logic for accessing protected resources.

No scope-element mappings currently exist for this application.

Get started by clicking **New**.

9. In the **Add New Scope-Element Mapping** panel that opens up, set the **Scope element** input field to **restrictedData**. Click the **UserLogin** button under the **Custom Security Checks** heading and click the *Add** button. This will map the custom UserLogin to the restrictedData scope that we just defined and applied to our

Javascript adapter procedure.


Add New Scope-Element Mapping

Define a new scope element and map it to zero or more security checks.

Scope element *

restrictedData

Name of the scope element

Security Checks

Select zero or more security checks.

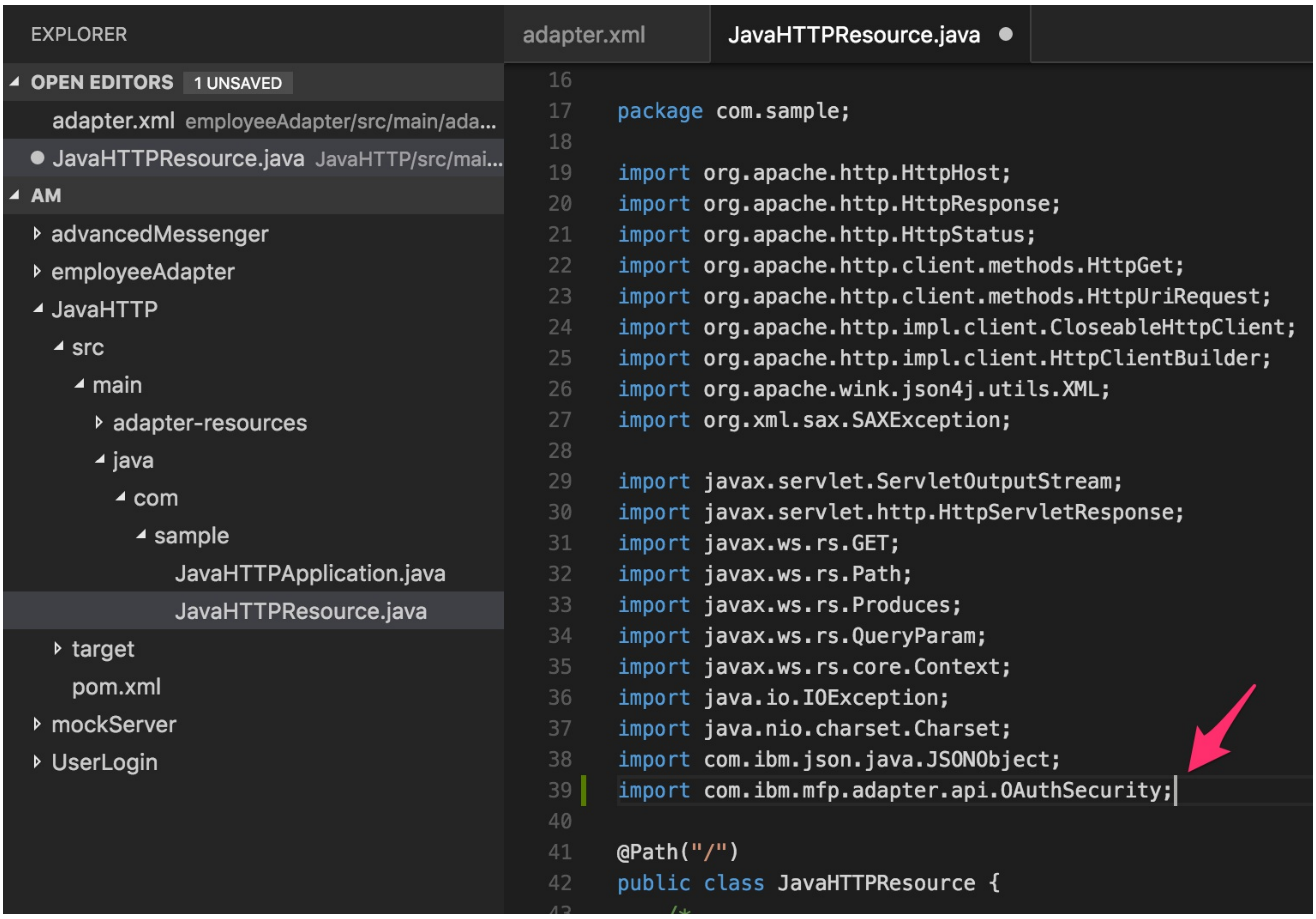
Predefined MobileFirst Security Checks

Custom Security Checks

10. Now we will take a similar approach to apply the UserLogin security check to a scope associated with a method in our "News" Java adapter using the OAuthSecurity annotation. From your IDE (eg. VS Code), navigate to **am -> JavaHTTP -> src -> main -> java -> com -> sample -> JavaHTTPResource.java**

11. First we need to add OAuthSecurity to our java imports. Add the following import statement to JavaHTTPResource.java.

```
import com.ibm.mfp.adapter.api.OAuthSecurity;
```



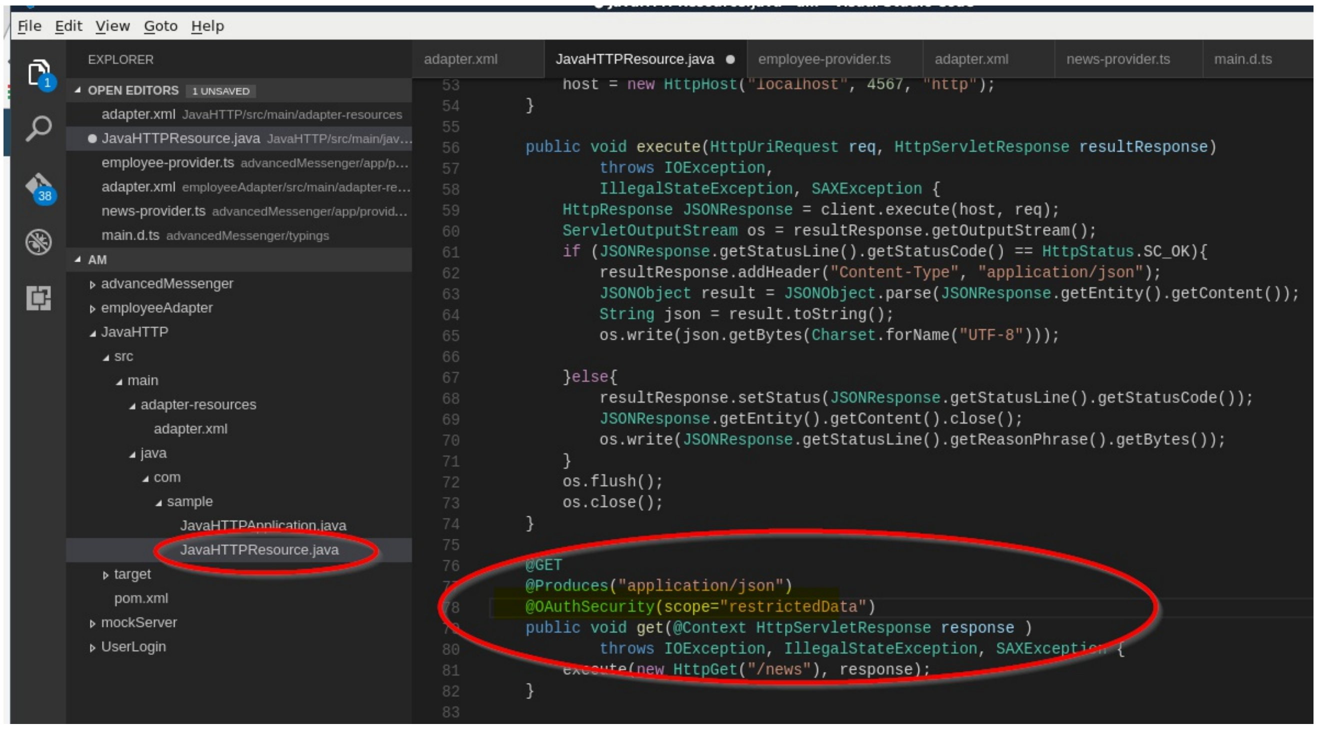
```
package com.sample;

import org.apache.http.HttpHost;
import org.apache.http.HttpResponse;
import org.apache.http.HttpStatus;
import org.apache.http.client.methods.HttpGet;
import org.apache.http.client.methods.HttpUriRequest;
import org.apache.http.impl.client.CloseableHttpClient;
import org.apache.http.impl.client.HttpClientBuilder;
import org.apache.wink.json4j.utils.XML;
import org.xml.sax.SAXException;

import javax.servlet.ServletOutputStream;
import javax.servlet.http.HttpServletResponse;
import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.QueryParam;
import javax.ws.rs.core.Context;
import java.io.IOException;
import java.nio.charset.Charset;
import com.ibm.json.java.JSONObject;
import com.ibm.mfp.adapter.api.OAuthSecurity;

@Path("/")
public class JavaHTTPResource {
```

12. Now we can use the OAuthSecurity annotation in the JavaHTTPResource class, scroll down to the **get** method (approximately line 78). Prior to the method declaration, add the OAuthSecurity annotation to associate the restrictedData scope to the get method:

```
@OAuthSecurity(scope="restrictedData")
```

Your code should look similar to this:

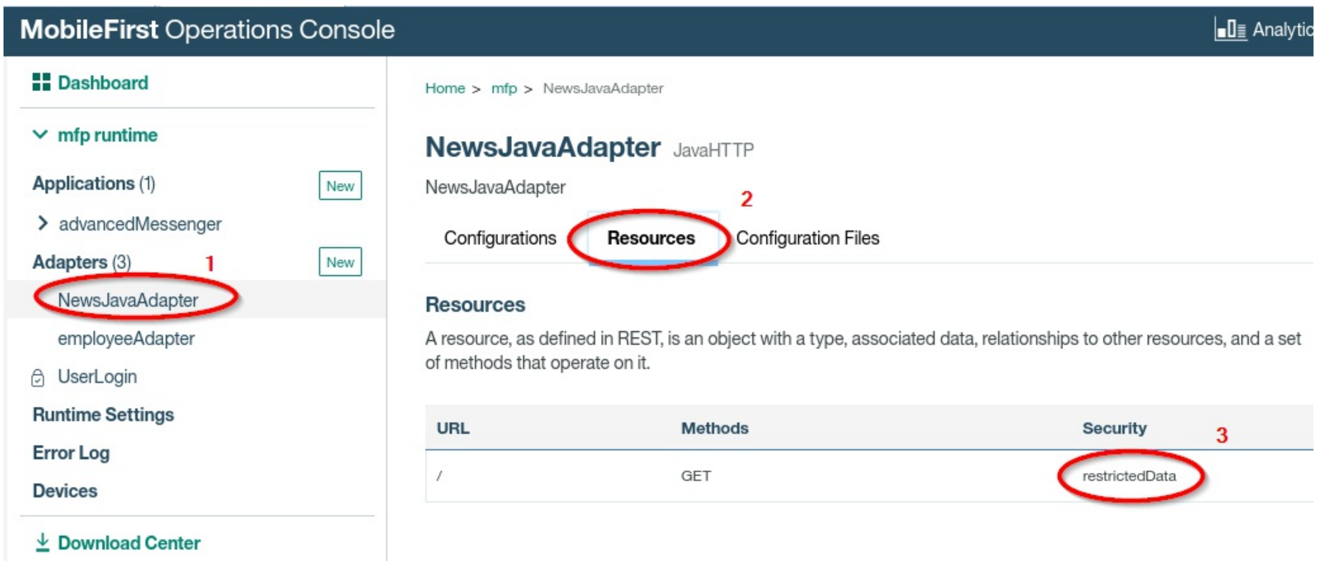
```
@GET
@Produces("application/json")
@OAuthSecurity(scope="restrictedData")
public void get(@Context HttpServletResponse response)
throws IOException, IllegalStateException, SAXException
```



```java
        host = new HttpHost("localhost", 4567, "http");
    }

    public void execute(HttpUriRequest req, HttpServletResponse resultResponse)
            throws IOException,
            IllegalStateException, SAXException {
        HttpResponse JSONResponse = client.execute(host, req);
        ServletOutputStream os = resultResponse.getOutputStream();
        if (JSONResponse.getStatusLine().getStatusCode() == HttpStatus.SC_OK){
            resultResponse.addHeader("Content-Type", "application/json");
            JSONObject result = JSONObject.parse(JSONResponse.getEntity().getContent());
            String json = result.toString();
            os.write(json.getBytes(Charset.forName("UTF-8")));

        }else{
            resultResponse.setStatus(JSONResponse.getStatusLine().getStatusCode());
            JSONResponse.getEntity().getContent().close();
            os.write(JSONResponse.getStatusLine().getReasonPhrase().getBytes());
        }
        os.flush();
        os.close();
    }

    @GET
    @Produces("application/json")
    @OAuthSecurity(scope="restrictedData")
    public void get(@Context HttpServletResponse response )
            throws IOException, IllegalStateException, SAXException {
        execute(new HttpGet("/news"), response);
    }

```

Save your changes.

13. Now we will build and deploy our modified java adapter. From a terminal, navigate to **~/dev/workspaces/am/JavaHTTP** and run the command:
    `mfpdev adapter build`
    You should see the message **Successfully built adapter** Now deploy the adapter by running the command:
    `mpfdev adapter deploy`
    You should see the message **Successfully deployed adapter**
14. Now from the MobileFirst Operations Console you can navigate to **Adapters -> NewsJavaAdapter** and click on the **Resources** tab. You should see the **restrictedData** scope associated with the **GET** method. *Note that we already have mapped the restrictedData scope to our UserLogin security check, so we won't have to repeat that step here.* We now have mapped our UserLogin security check to the restrictedData scope which is securing backend calls in both our Javascript and Java adapters.



Resources

A resource, as defined in REST, is an object with a type, associated data, relationships to other resources, and a set of methods that operate on it.

URL	Methods	Security
/	GET	restrictedData

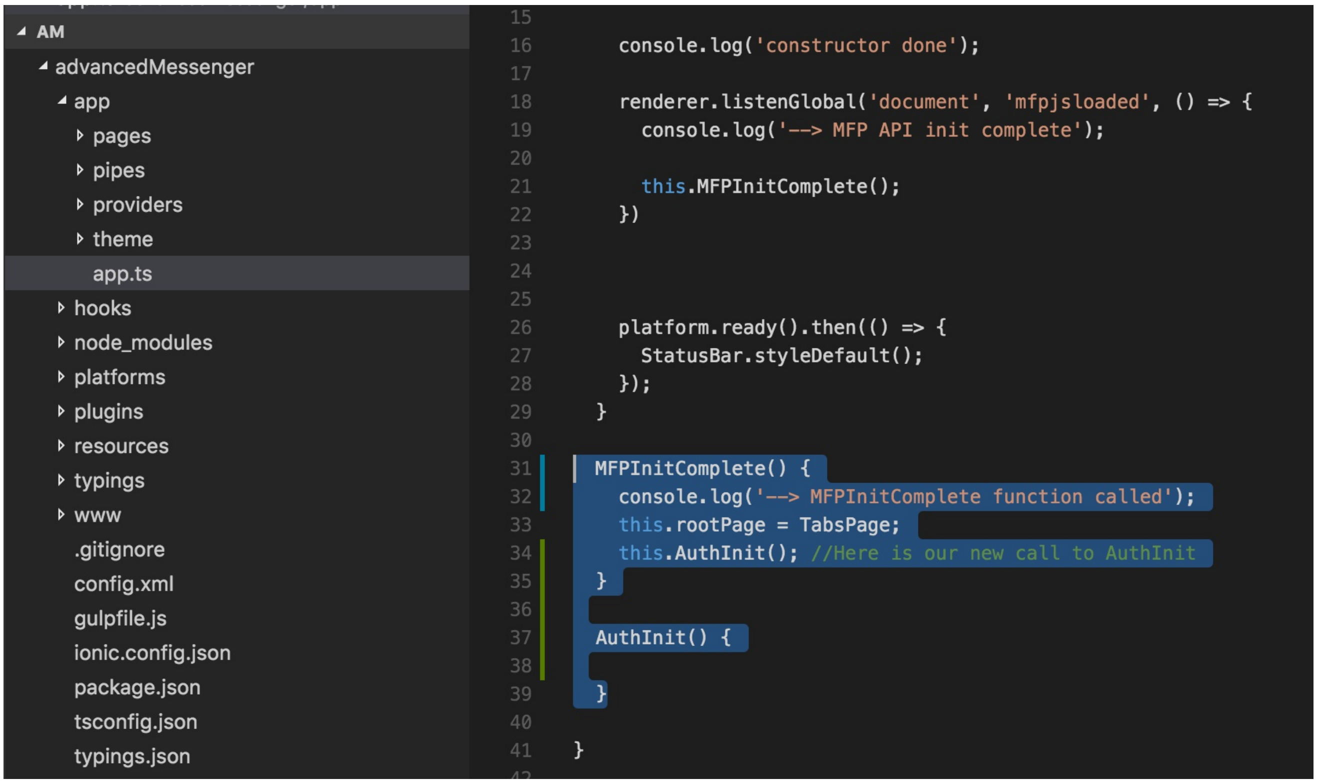
Part 3 - Working with the Client Side Code.

Now we need to write the client side challenge handler code which will handle the UserLogin security check, grabbing the credentials provided by the user and returning those to the server side security check adapter. If the user credentials are accepted then our initial call to our backend adapter will be executed; otherwise, the client side challenge handler will again prompt the user for credentials. Recall that the UserLogin adapter checks incoming credentials to see if the value of the username is equal to the value of the password. Code snippets will be provided to help with code editing. Let's get started with the client side code.

1. From our IDE navigate to **am -> advancedMessenger** and open **app.ts**. At the bottom of the **MyApp** class following the **MFPInitComplete()** function create a new function called **AuthInit()** and add a call to AuthInit from inside **MFPInitComplete**. (You can cut and paste snippet01). AuthInit will create our security challenge handler. Your modified code for the MFPInitComplete() function and for the AuthInit() function should look like this:

```
MFPInitComplete(){
console.log('--> MFPInitComplete function called');
```

```
  this.rootPage = TabsPage;
  this.AuthInit(); //Here is our new call to AuthInit
  }

  AuthInit(){

  }
```



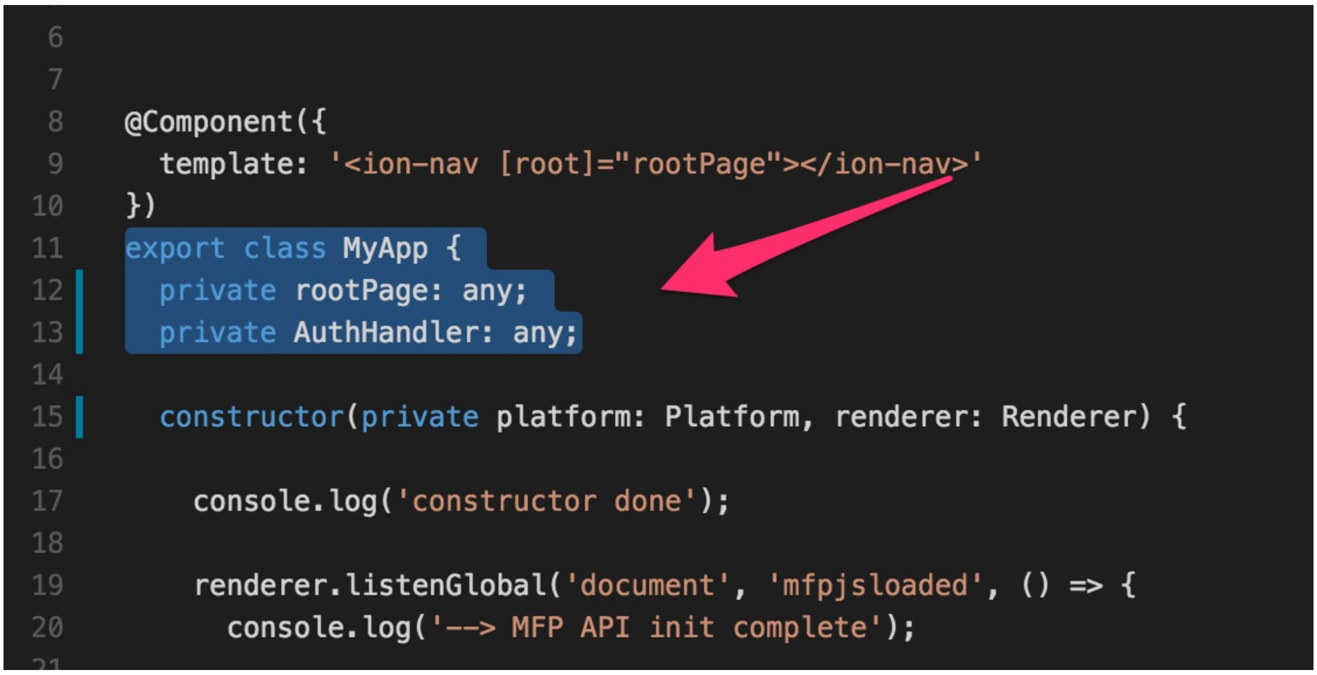
```
    console.log('constructor done');

    renderer.listenGlobal('document', 'mfpjsloaded', () => {
      console.log('--> MFP API init complete');

      this.MFPInitComplete();
    })



    platform.ready().then(() => {
      StatusBar.styleDefault();
    });
  }

  MFPInitComplete() {
    console.log('--> MFPInitComplete function called');
    this.rootPage = TabsPage;
    this.AuthInit(); //Here is our new call to AuthInit
  }

  AuthInit() {

  }

}
```

2. Now we need to declare the AuthHandler variable that we will use inside the MyApp Class. Underneath the **private rootPage:any;** declaration (approximately line 12, at the top of the class) add the following line:

```
private AuthHandler: any;
```

your code should now look like this:

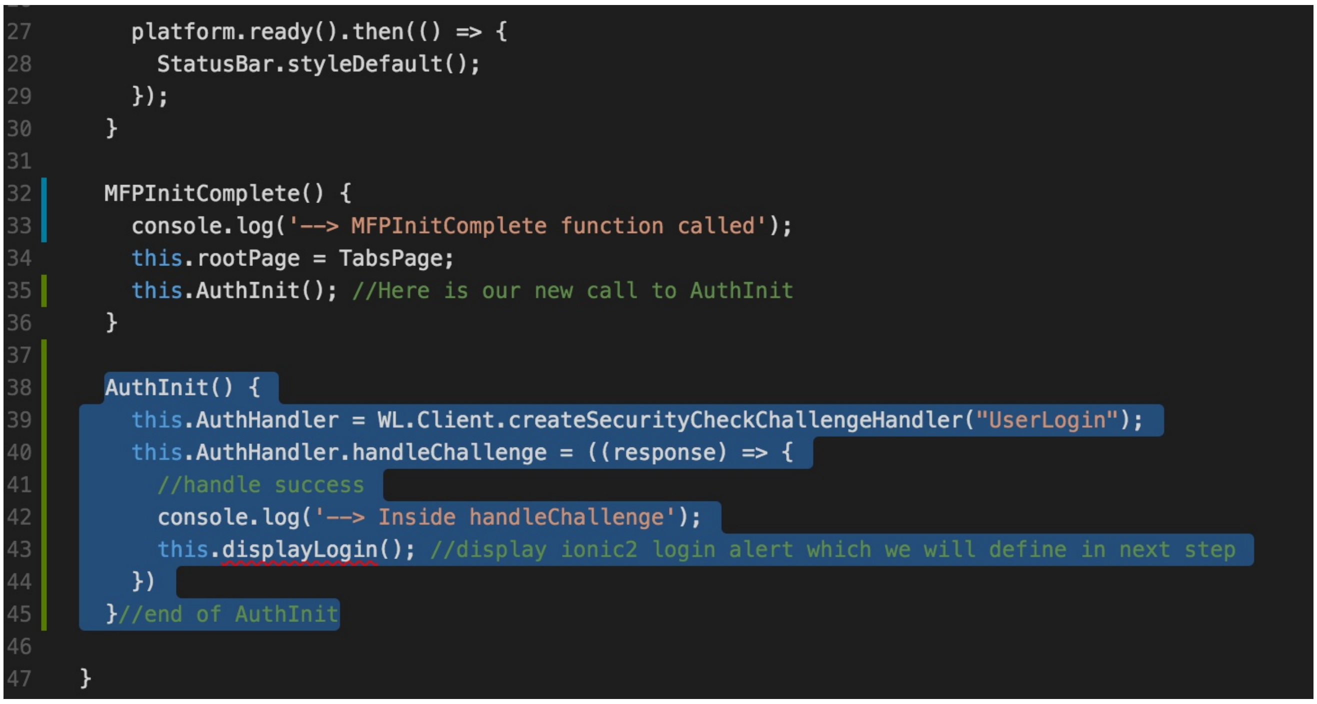
```
export class MyApp {
  private rootPage:any;
  private AuthHandler: any;
```



```
@Component({
  template: '<ion-nav [root]="rootPage"></ion-nav>'
})
export class MyApp {
  private rootPage: any;
  private AuthHandler: any;

  constructor(private platform: Platform, renderer: Renderer) {

    console.log('constructor done');

    renderer.listenGlobal('document', 'mfpjsloaded', () => {
      console.log('--> MFP API init complete');
```

3. Now we will set our newly created AuthHandler variable to our security challenge handler. In our newly created AuthInit function modify the function to look like this (snippet02):

```
AuthInit(){
  this.AuthHandler = WL.Client.createSecurityCheckChallen
  this.AuthHandler.handleChallenge = ((response) => {
    //handle success
    console.log('--> Inside handleChallenge');
    this.displayLogin(); //display Ionic2 login alert whi
  })
} //end of AuthInit
```

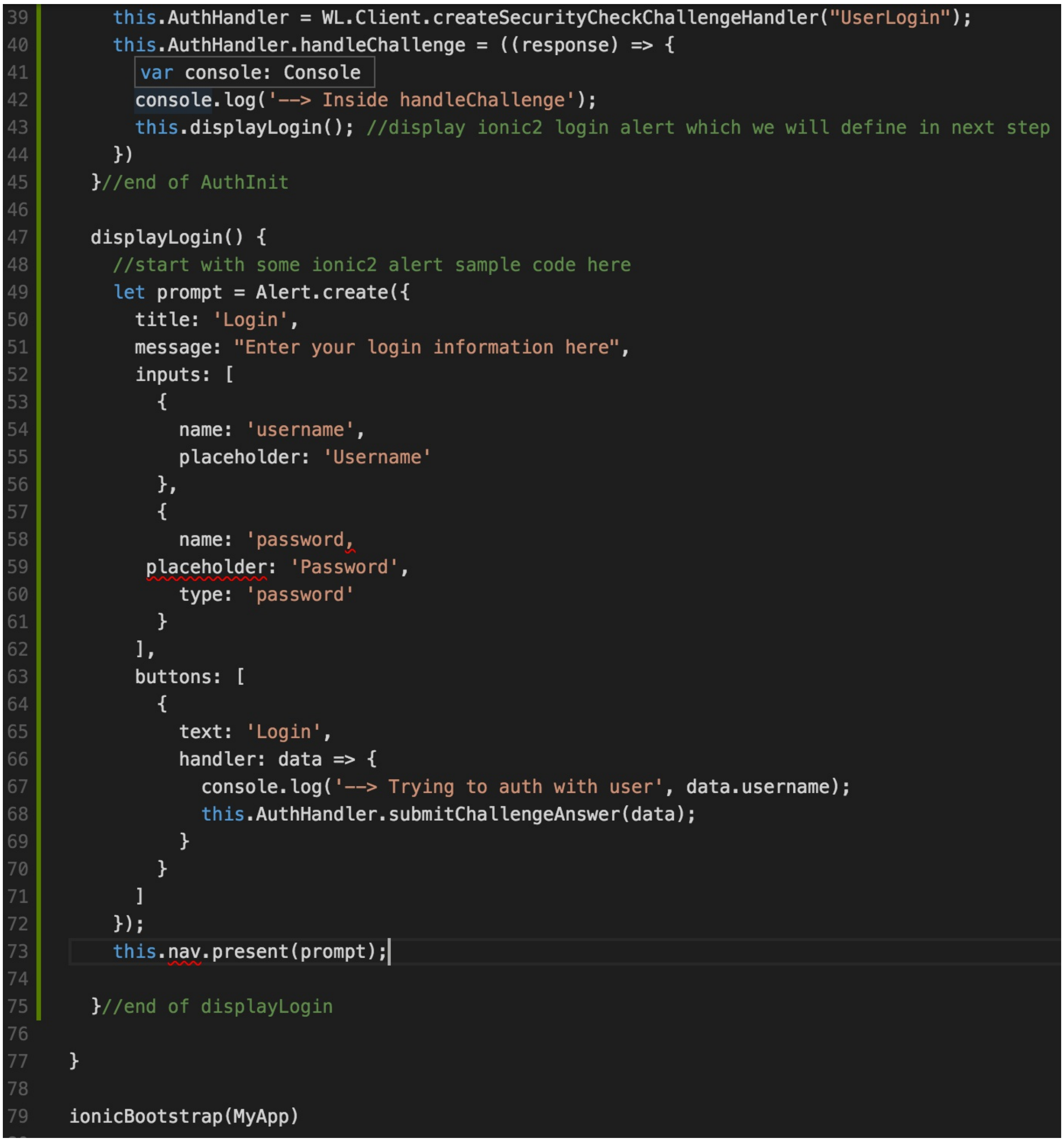


```
    platform.ready().then(() => {
      StatusBar.styleDefault();
    });
  }

  MFPInitComplete() {
    console.log('--> MFPInitComplete function called');
    this.rootPage = TabsPage;
    this.AuthInit(); //Here is our new call to AuthInit
  }

  AuthInit() {
    this.AuthHandler = WL.Client.createSecurityCheckChallengeHandler("UserLogin");
    this.AuthHandler.handleChallenge = ((response) => {
      //handle success
      console.log('--> Inside handleChallenge');
      this.displayLogin(); //display ionic2 login alert which we will define in next step
    })
  }//end of AuthInit

}
```

Notice that we used UserLogin, (our previously defined security check) when we create the AuthHandler.

4. When we handle the security challenge we are calling the displayLogin function, which we have yet to define. Let's create that function. Underneath your newly created AuthInit function add our new displayLogin() function (snippet3):

```
displayLogin(){
  //start with some Ionic2 alert sample code here
  let prompt = Alert.create({
    title: 'Login',
    message: "Enter your login information here",
    inputs: [
      {
         name: 'username',
         placeholder: 'Username'
       },
       {
         name: 'password,
         placeholder: 'Password',
         type: 'password'
       }],
  buttons: [
   {
    text: 'Login',
    handler: data => {
     console.log('--> Trying to auth with user', da
     this.AuthHandler.submitChallengeAnswer(data);
    }
   }]
 });
}//end of displayLogin
```



```
    this.AuthHandler = WL.Client.createSecurityCheckChallengeHandler("UserLogin");
    this.AuthHandler.handleChallenge = ((response) => {
      var console: Console
      console.log('--> Inside handleChallenge');
      this.displayLogin(); //display ionic2 login alert which we will define in next step
    })
  }//end of AuthInit

  displayLogin() {
    //start with some ionic2 alert sample code here
    let prompt = Alert.create({
      title: 'Login',
      message: "Enter your login information here",
      inputs: [
        {
          name: 'username',
          placeholder: 'Username'
        },
        {
          name: 'password,
       placeholder: 'Password',
          type: 'password'
        }
      ],
      buttons: [
        {
          text: 'Login',
          handler: data => {
            console.log('--> Trying to auth with user', data.username);
            this.AuthHandler.submitChallengeAnswer(data);
          }
        }
      ]
    });
    this.nav.present(prompt);

  }//end of displayLogin

}

ionicBootstrap(MyApp)
```

In our code we are creating an Ionic2 alert with two input fields, a username input and a password input. We have one login (submit) button. The button code contais the handler that submits our credentials data.

5. The displayLogin function is using the Ionic2 Alerts component and assigns it to the prompt variable.

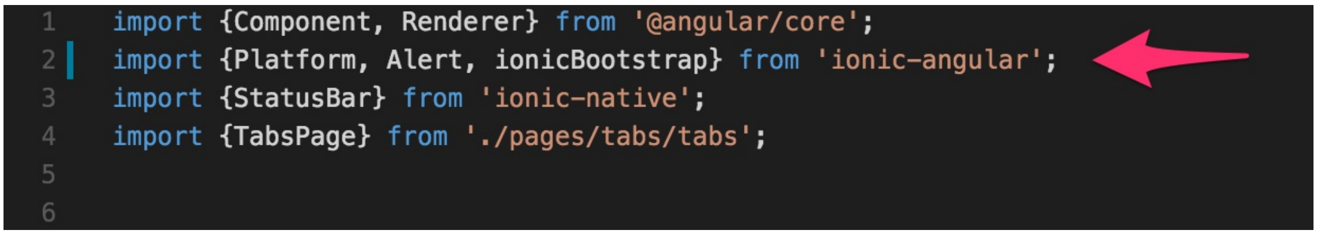
   An Alert component is similar to a popup screen that floats on top of an existing view. You can read more about Alerts here: http://Ionicframework.com/docs/v2/components/#alert.

   In order to use Ionic2 alerts we need to modify line 2 of the app.ts file and change the import statement from:

```
import {Platform, IonicBootstrap} from 'Ionic-Angular';
```

   to

```
import {Platform, Alert, IonicBootstrap} from 'Ionic-Angu
```



```
import {Component, Renderer} from '@angular/core';
import {Platform, Alert, ionicBootstrap} from 'ionic-angular';
import {StatusBar} from 'ionic-native';
import {TabsPage} from './pages/tabs/tabs';
```

6. In the DisplayLogin function we added the **this.nav.present(prompt)** to our app. We will need to add some additional code to work with **nav**. First we will add a reference to the Ionic2 **App** Utility Service which will be used to get the current Ionic nav. We will then assign the current Ionic nav to the **nav** variable. Start by modifying the import statement on line 2 of the app.ts file.

Change the line from:

```
import {Platform, Alert, IonicBootstrap} from 'Ionic-Angu
```

to

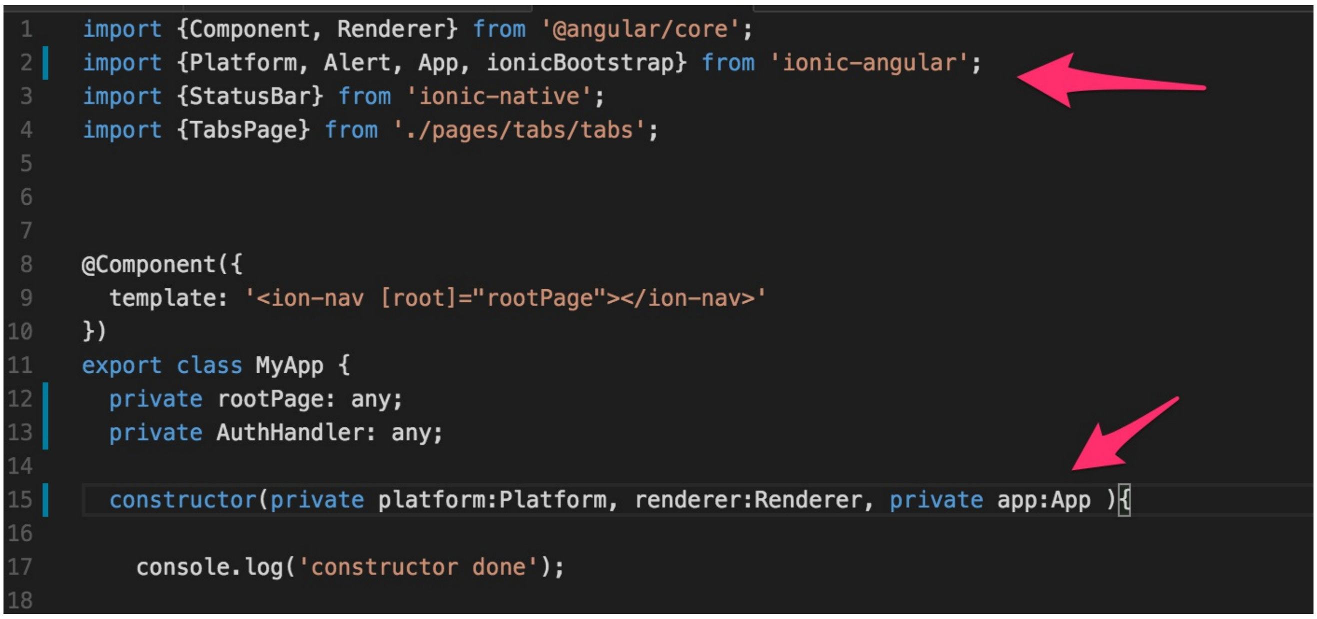
```
import {Platform, Alert, App, IonicBootstrap} from 'Ionic
```

7. Now add the **App** service to our class constructor (approximately line 15). Change the line from:

```
 constructor(private platform:Platform, renderer:Rendere
```

to

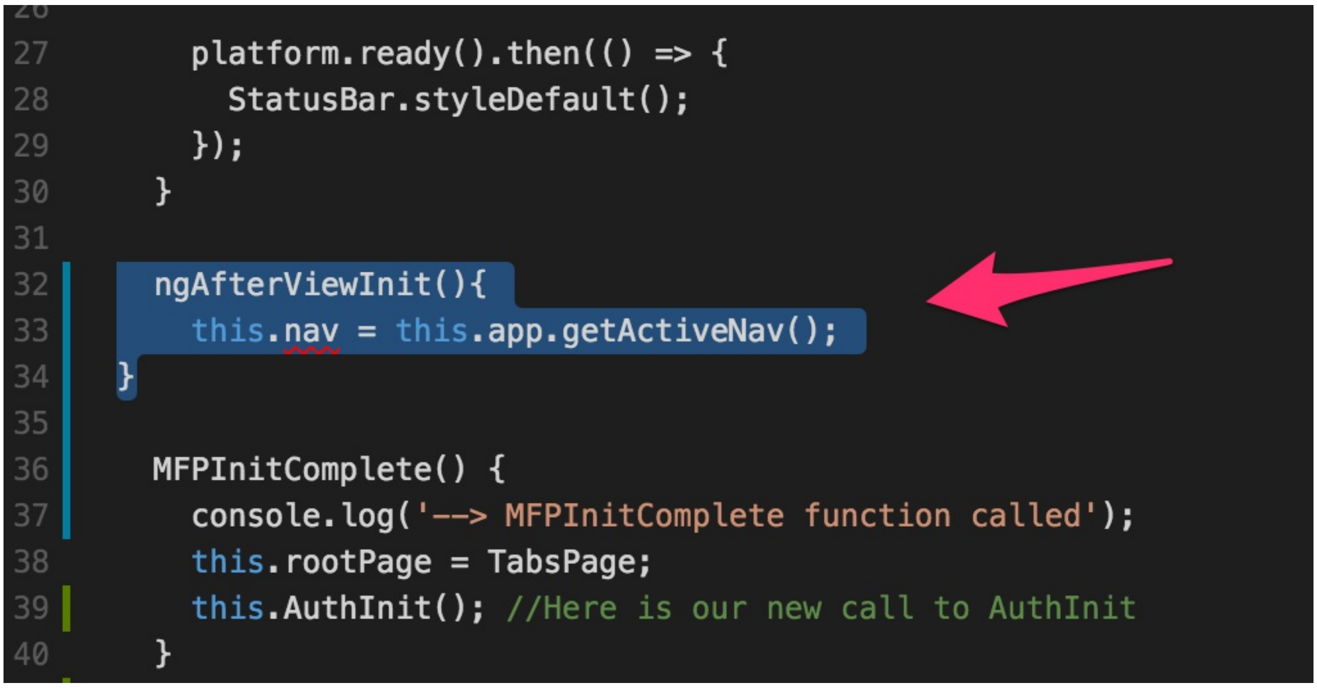
```
 constructor(private platform:Platform, renderer:Rendere
```



```
1   import {Component, Renderer} from '@angular/core';
2   import {Platform, Alert, App, ionicBootstrap} from 'ionic-angular';
3   import {StatusBar} from 'ionic-native';
4   import {TabsPage} from './pages/tabs/tabs';
5
6
7
8   @Component({
9     template: '<ion-nav [root]="rootPage"></ion-nav>'
10  })
11  export class MyApp {
12    private rootPage: any;
13    private AuthHandler: any;
14
15    constructor(private platform:Platform, renderer:Renderer, private app:App ){
16
17      console.log('constructor done');
18
```

8. Now we can use this Ionic **app** utility service to get the active Ionic nav. We will work with application lifecycle events to trigger when we get the active nav. The nav controller will be loaded after the view init event. Since this is an Ionic2/Angular2 app we will use the Angular2 lifecycle hook **ngAfterViewInit()**. Using this hook we set the **nav** variable to the active nav after our component view has been initialized. To do this, add the **ngAfterViewInit()** hook right above the **MFPInitComplete()** function and set this.nav to our active nav. It should look like this (snippet04):

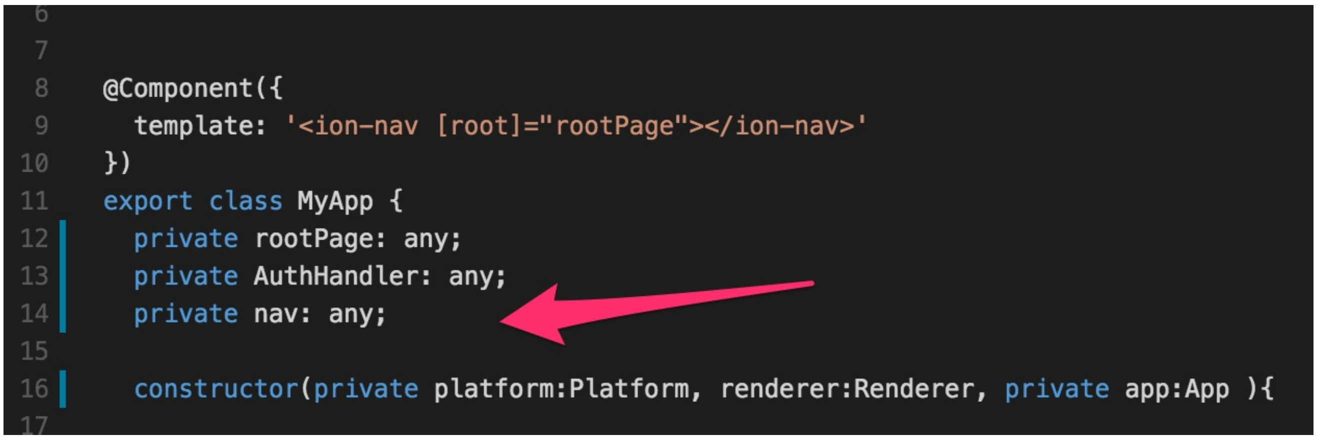
```
ngAfterViewInit(){
  this.nav = this.app.getActiveNav();
}
MFPInitComplete(){
```



```
      platform.ready().then(() => {
        StatusBar.styleDefault();
      });
    }

    ngAfterViewInit(){
      this.nav = this.app.getActiveNav();
    }

    MFPInitComplete() {
      console.log('--> MFPInitComplete function called');
      this.rootPage = TabsPage;
      this.AuthInit(); //Here is our new call to AuthInit
    }
```

9. Since we are referencing this.nav to contain our current Ionic nav we need to declare it as a variable. At the top of the **MyApp** class add the **nav** variable declaration so it looks like this:

```
export classMyApp {
  private rootPage:any;
  private AuthHandler: any;
  private nav: any;
```



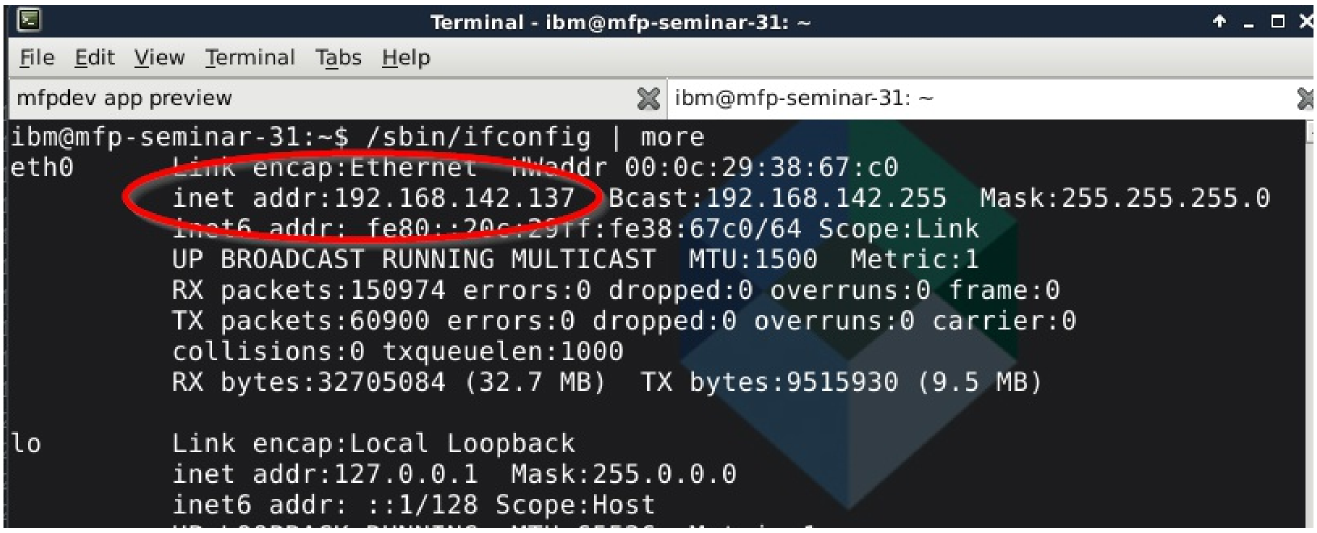
```
@Component({
  template: '<ion-nav [root]="rootPage"></ion-nav>'
})
export class MyApp {
  private rootPage: any;
  private AuthHandler: any;
  private nav: any;

  constructor(private platform:Platform, renderer:Renderer, private app:App ){
```

Save all your changes. We now should have a working security challenge handler!

10. Before testing our code, we still need to make another modification to our app. In our code, we currently are pointing to localhost in a sample code call to our mock service. In order for this to work properly on a device we need to edit this call to point to our machine's IP address. First we will need to get the primary IP address for your machine. Open a terminal window or tab and enter the following command to obtain and copy the primary ip address:

    ```
    /sbin/ifconfig | more
    ```

    and find the inet addr field for your primary (eg. eth0) adapter. (eg. 192.168.123.456) *Note: Your IP Address will likely be different*.


Terminal - ibm@mfp-seminar-31: ~
File Edit View Terminal Tabs Help
mfpdev app preview
ibm@mfp-seminar-31: ~

```
ibm@mfp-seminar-31:~$ /sbin/ifconfig | more
eth0      Link encap:Ethernet  HWaddr 00:0c:29:38:67:c0
          inet addr:192.168.142.137  Bcast:192.168.142.255  Mask:255.255.255.0
          inet6 addr: fe80::20c:29ff:fe38:67c0/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:150974 errors:0 dropped:0 overruns:0 frame:0
          TX packets:60900 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:32705084 (32.7 MB)  TX bytes:9515930 (9.5 MB)

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
```

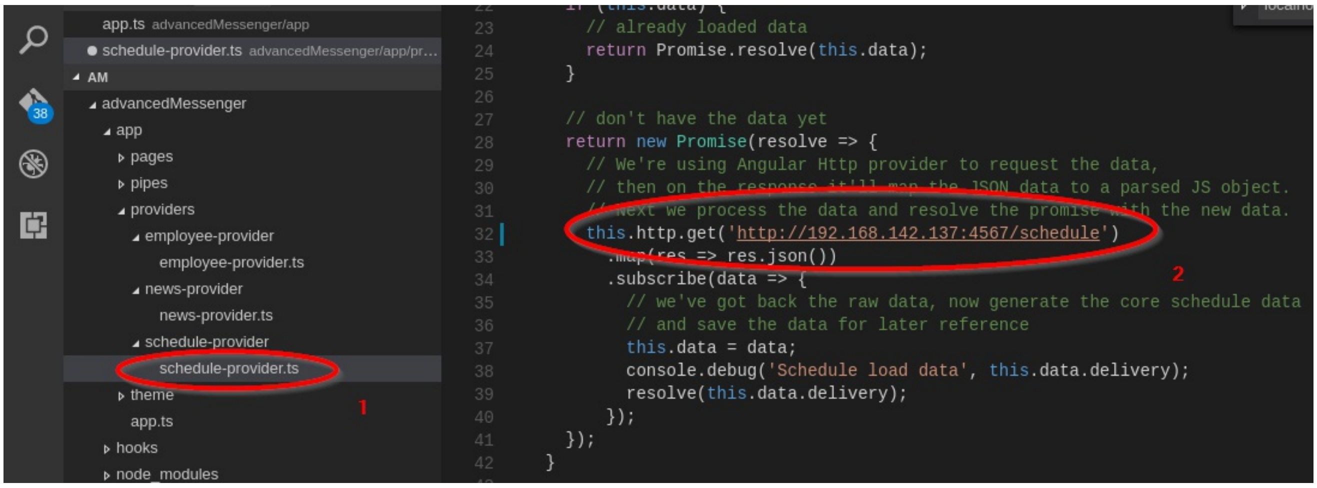
11. Now we will point the ScheduleProvider class to your ip, navigate to **am -> advancedMessenger -> app -> providers -> schedule-provider** and edit the **schedule-provider.ts** file. Navigate to approximately line 32 and change the line from:

```
this.http.get('http://localhost:4567/schedule')
```

to

```
this.http.get('http://yourIPAddress:4567/schedule)
```

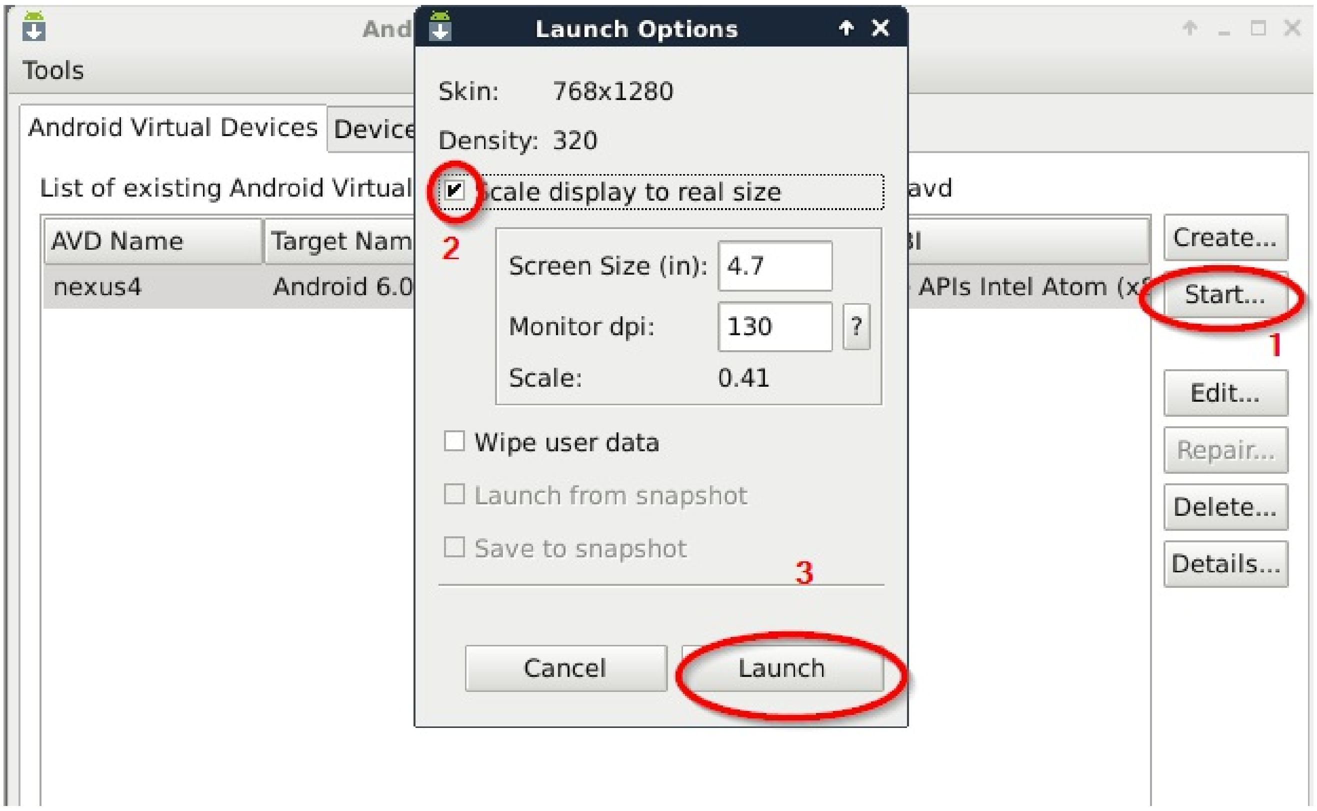
where **yourIPAddress** will be the ip address from the previous ifconfig command. **Save** your changes.



```
23        // already loaded data
24        return Promise.resolve(this.data);
25      }
26
27      // don't have the data yet
28      return new Promise(resolve => {
29        // We're using Angular Http provider to request the data,
30        // then on the response it'll map the JSON data to a parsed JS object.
31        // Next we process the data and resolve the promise with the new data.
32        this.http.get('http://192.168.142.137:4567/schedule')
33          .map(res => res.json())
34          .subscribe(data => {
35            // we've got back the raw data, now generate the core schedule data
36            // and save the data for later reference
37            this.data = data;
38            console.debug('Schedule load data', this.data.delivery);
39            resolve(this.data.delivery);
40          });
41      });
42    }
```

Part 4 - Testing our Changes

So far we have defined a security check using MobileFirst Platform Foundation adapters and we have added code to our client-side app to handle the security challenge. Now it's time to test our changes.

1. From a terminal window, navigate to **/dev/workspaces/am/advancedMessenger** and start an Android virtual device by typing the command: `android avd`
   Note: if you are using the lab provided virtual machine, then a Nexus 4 virtual device should be defined. Select this device and click **Start..** to start the virtual device. Select **Scale display to real size.** Take the rest of the default options and **launch** the virtual device.
   

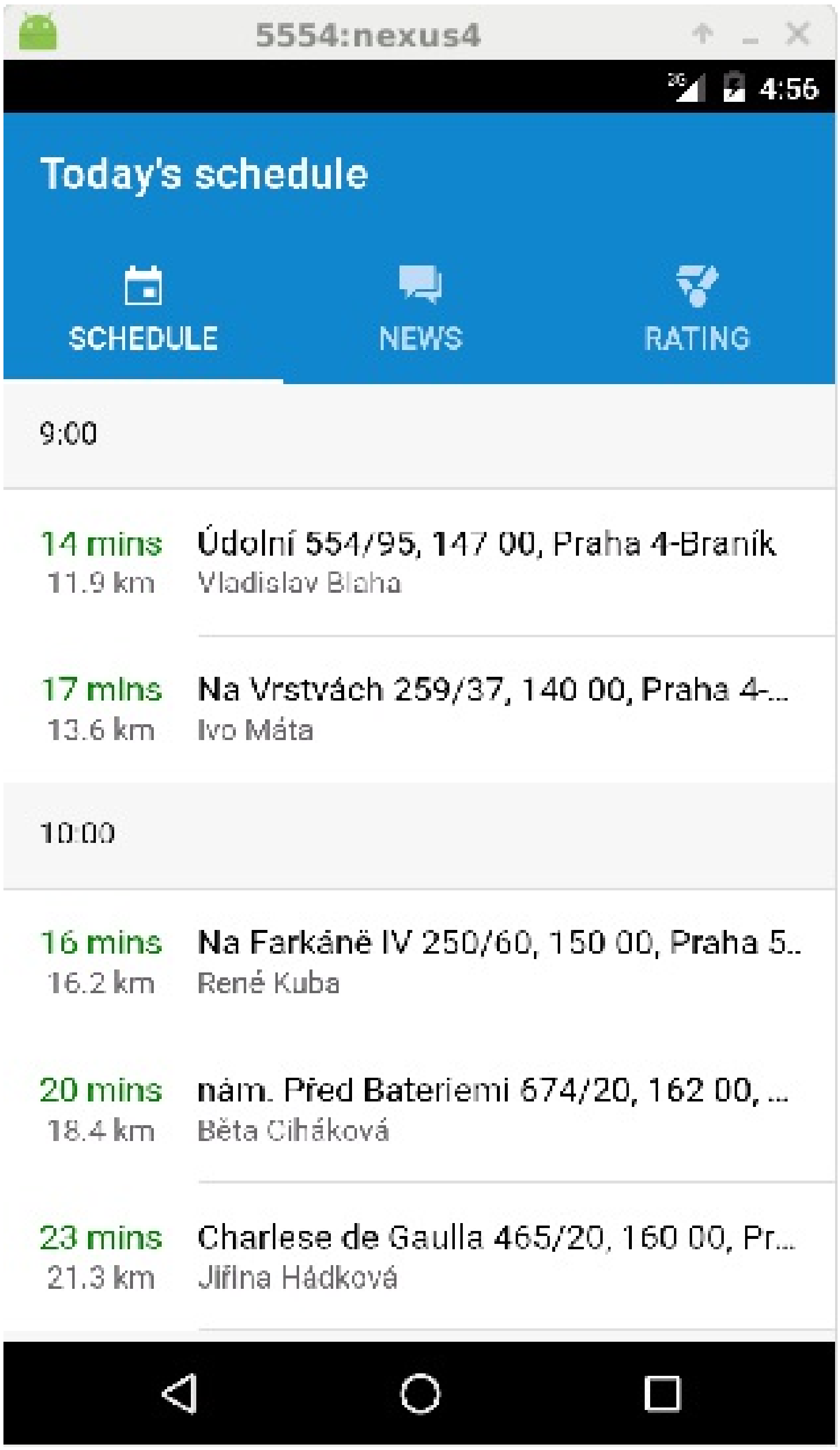
   Skin: 768x1280
   Density: 320
   [✓] Scale display to real size
   Screen Size (in): 4.7
   Monitor dpi: 130
   Scale: 0.41
   ☐ Wipe user data
   ☐ Launch from snapshot
   ☐ Save to snapshot

   *Note: **Android virtual machines may take a few minutes to start up, so this may be a good opportunity to grab a coffee :-)***

2. Now we are going to launch the app on the virtual android device. Open a new terminal or terminal tab and navigate to **/dev/workspaces/am/advancedMessenger**. From this directory enter the command: `cordova run android`.

*Note: **This again may take several minutes, so this may be a good opportunity to grab a donut**.*

When the app comes up you should see the following screen:



Today's schedule

9:00

14 mins 11.9 km Údolní 554/95, 147 00, Praha 4-Braník
Vladislav Blaha

17 mins 13.6 km Na Vrstvách 259/37, 140 00, Praha 4-...
Ivo Máta

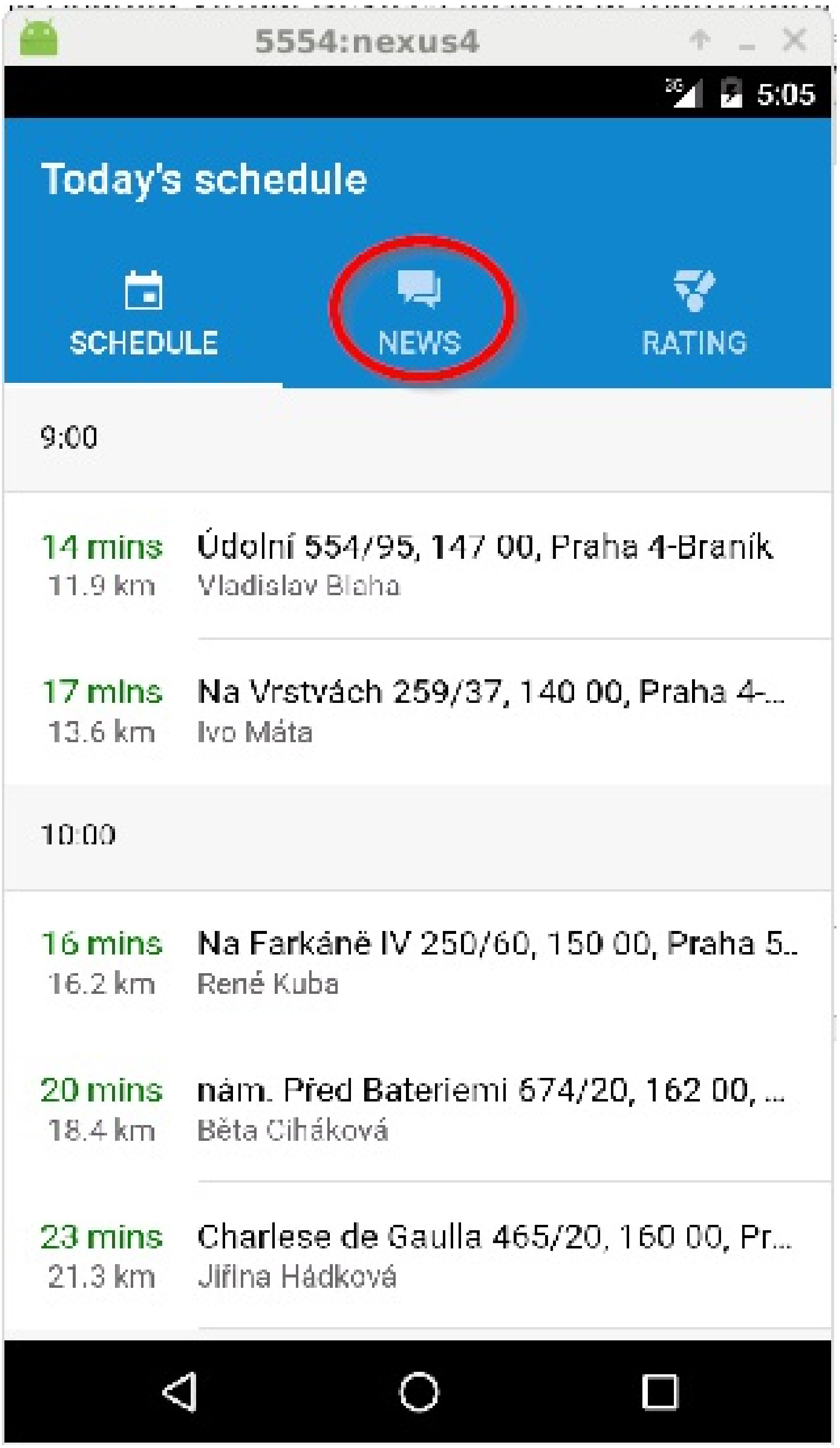
10:00

16 mins 16.2 km Na Farkáně IV 250/60, 150 00, Praha 5..
René Kuba

20 mins 18.4 km nám. Před Bateriemi 674/20, 162 00, ...
Běta Ciháková

23 mins 21.3 km Charlese de Gaulla 465/20, 160 00, Pr...
Jiřina Hádková

3. Now let's test our security challenge. From your app running on the Android virtual device, click on the **NEWS** tab.



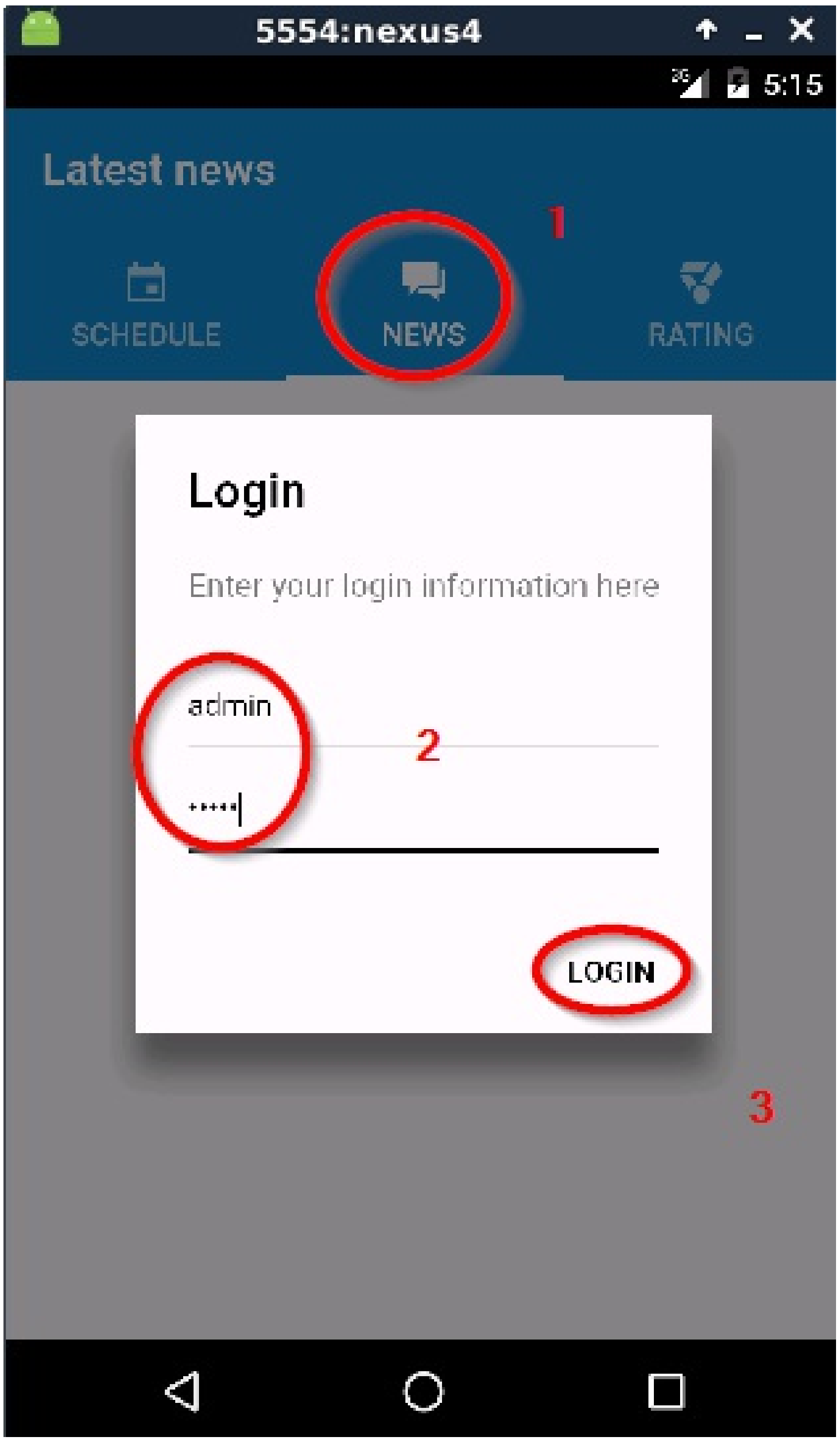
9:00

14 mins 11.9 km	Údolní 554/95, 147 00, Praha 4-Braník Vladislav Blaha
17 mins 13.6 km	Na Vrstvách 259/37, 140 00, Praha 4-... Ivo Máta

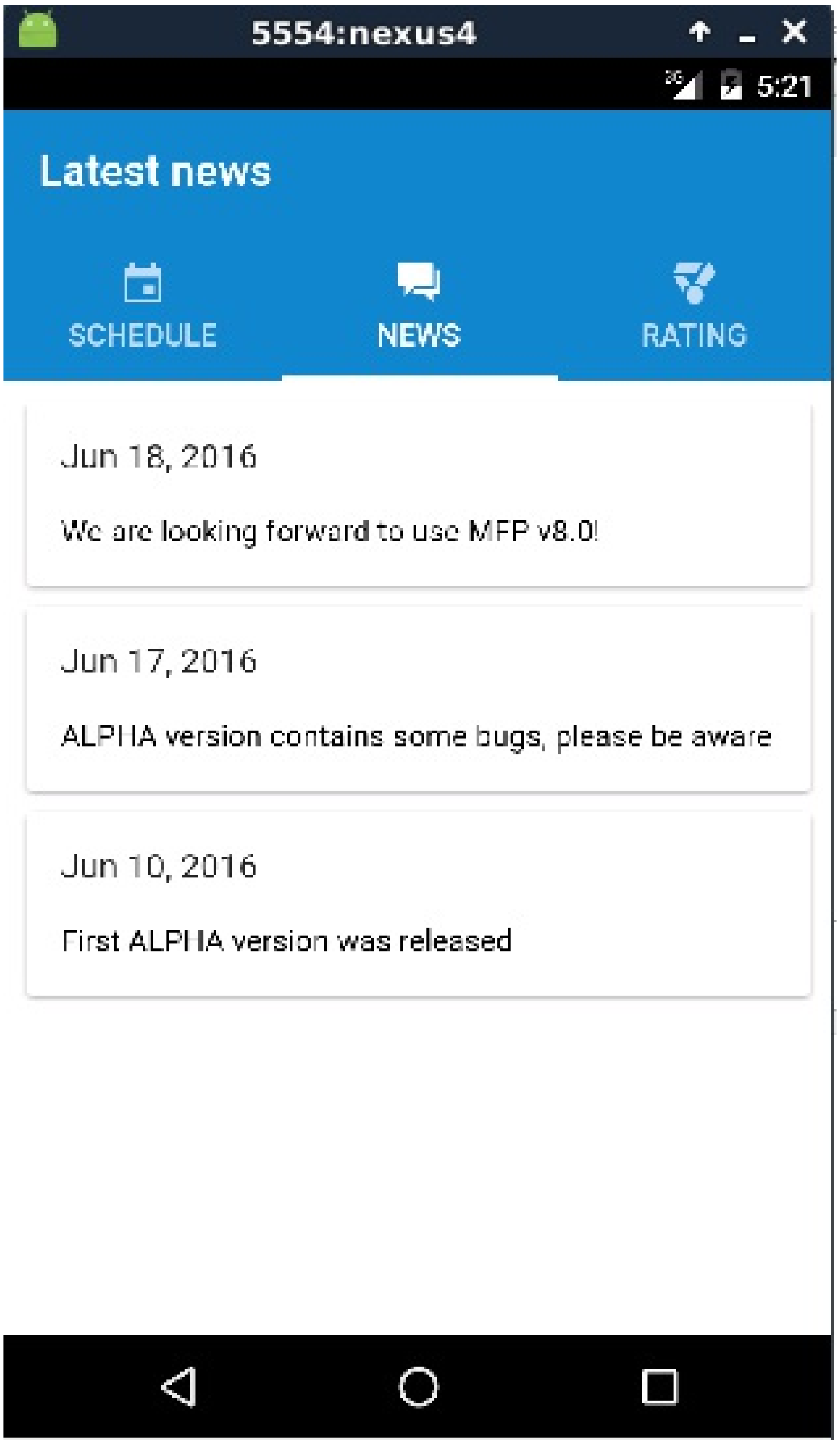
10:00

16 mins 16.2 km	Na Farkáně IV 250/60, 150 00, Praha 5... René Kuba
20 mins 18.4 km	nám. Před Bateriemi 674/20, 162 00, ... Běta Ciháková
23 mins 21.3 km	Charlese de Gaulla 465/20, 160 00, Pr... Jiřina Hádková

You should now see the security challenge handler that we defined. Type in a userid and password where both are set to the same value (eg. admin/admin) and click **LOGIN**.



4. You should now see the results from the call to our protected NEWS adapter. You can also test going to the RATING tab, where you also should now be prompted for a userid and password as part of our defined security challenge.



Jun 18, 2016

We are looking forward to use MFP v8.0!

Jun 17, 2016

ALPHA version contains some bugs, please be aware

Jun 10, 2016

First ALPHA version was released

Congratulations!, you have completed the Securing Backend Calls with MobileFirst lab. You can see how the resource security model is easy to use in MobileFirst. This extends into more sophisticated use cases such as social login, enterprise directory login, and step-up authentication scenarios.

For more tutorials on authentication and security, check out the MobileFirst Foundation Developer Center:

https://mobilefirstplatform.ibmcloud.com/tutorials/en/foundation/8.0/authentication-and-security/