Lab - Adding push notification to your application

In this lab we are going to take existing Cordova/Ionic application which was instrumented for MobileFirst and add push notification support using the MobileFirst push notification SDK.

We are going to add the mfp-push-notification plugin and then modify the client side app to be able to register in order to receive push notifications from the server.

> 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.

Source code for labs

In order to get the latest code for the ionic application, run the following git command:

```
git clone https://github.com/eliranbi/MFPushNotificationLab.git
```

The command above will download the latest documentation and code snippets.

This lab will use the Cordova CLI to add the MobileFirst Cordova plugin. These are the standard steps required to add plugins to an application. Adding the MobileFirst plugin allows the application to use the MobileFirst platform features.

General Steps:

1. Within the IBM Employee application directory, run Cordova CLI command to add a plugin.
2. Add one or more device platforms.
3. Use the MobileFirst CLI to preview the application to ensure the plugin was successfully added.

Before we can start we need to follow the steps below to add the mfp plugins and OS environment.

Steps:

1. Change context into the MobileFirst project.

```
cd IBMEmployeeApp
```

2. **Add** the Android platform, run the following command cordova platform add android

```
cordova platform add android
```

```
Elirans-MacBook-Pro:IBMEmployeeApp eliran_pro$ cordova platform add android
Adding android project...
Creating Cordova project for the Android platform:
        Path: platforms/android
        Package: com.ionicframework.ibmemployeeapp420875
        Name: Employee
        Activity: MainActivity
        Android target: android-23
Android project created with cordova-android@5.1.1
Running command: /Users/eliran_pro/Documents/projects/NATechnicalAcademy2016/MFPushNotificationLab/IBMEmployeeApp/hooks/after_prepare/010_add_platform_class.js /Users/eliran_pro/Documents/projects/NATechnicalAcademy2016/MFPushNotificationLab/IBMEmployeeApp
add to body class: platform-android
Elirans-MacBook-Pro:IBMEmployeeApp eliran_pro$
```

3. **Run** the cordova plugin add cordova-plugin-mfp

```
cordova plugin add cordova-plugin-mfp
```

```
mongod | node | bash | node | bash | bash
Elirans-MacBook-Pro:IBMEmployeeApp eliran_pro$ cordova plugin add cordova-plugin-mfp
Fetching plugin "cordova-plugin-mfp" via npm
Installing "cordova-plugin-mfp" for ios
Dependent plugin "cordova-plugin-device" already installed on ios.
Fetching plugin "cordova-plugin-dialogs" via npm
Installing "cordova-plugin-dialogs" for ios
Fetching plugin "cordova-plugin-globalization" via npm
Installing "cordova-plugin-globalization" for ios
cp: no such file or directory: /Users/eliran_pro/Documents/projects/Madrid2016/IBMEmployeeApp/platforms/ios/Employee/main.m

If you made changes to your main.m file, manually merge main.m.bak with the main.m file that is provided with IBM MobileFirst Platform Foundation.
Elirans-MacBook-Pro:IBMEmployeeApp eliran_pro$
```

4. **Run** the cordova plugin add cordova-plugin-mfp-push

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

```
Elirans-MacBook-Pro:IBMEmployeeApp eliran_pro$ cordova plugin add cordova-plugin-mfp-push
Fetching plugin "cordova-plugin-mfp-push" via npm
Installing "cordova-plugin-mfp-push" for android
Dependent plugin "cordova-plugin-mfp" already installed on android.
Elirans-MacBook-Pro:IBMEmployeeApp eliran_pro$
```

> Note: To be able to easily debug your application and view your application console log, run the following command to add the cordova console plugin

```
cordova plugin add cordova-plugin-console
```

> **Note:** Running the add plugin command above will add all the required MFP plugin files from npm. This requires a network connection. For more information on the MFP Cordova Plugin, visit:

> https://www.npmjs.com/package/cordova-plugin-mfp

5. **Start** the mobile first server, navigate to the mobile first server installation folder and run the following commands:

   For Mac

   - Open a new terminal session

   ```
   cd ~/MobileFirst-8.0.0.0
   ```

   ```
   ./run.sh
   ```

   ```
   Elirans-MacBook-Pro:mfp-server-all-in-one eliran_pro$ ./run.sh
   Picked up JAVA_TOOL_OPTIONS: -DwlDevEnv=true
   objc[25473]: Class JavaLaunchHelper is implemented in both /Library/Java/JavaVirtualMa
   chines/jdk1.7.0_80.jdk/Contents/Home/jre/bin/java and /Library/Java/JavaVirtualMachine
   s/jdk1.7.0_80.jdk/Contents/Home/jre/lib/libinstrument.dylib. One of the two will be us
   ed. Which one is undefined.
   Listening for transport dt_socket at address: 10777
   Launching mfp (WebSphere Application Server 8.5.5.8/wlp-1.0.11.cl150820151201-1942) on
   Java HotSpot(TM) 64-Bit Server VM, version 1.7.0_80-b15 (en_US)
   [AUDIT   ] CWWKE0001I: The server mfp has been launched.
   ```

6. **Run** the mfpdev app preview

   ```
   mfpdev app preview
   ```

   ```
   Elirans-MacBook-Pro:IBMEmployeeApp eliran_pro$ mfpdev app preview
   Verifying server configuration...
   ? Select how to preview your app: browser: Simple browser rendering
   The CLI preview will remove the Content-Security-Policy meta tag during preview to allow the live reload to function in a browser. Please ignore the '
   No Content-Security-Policy' error messages from whitelist.js in a browser's debugger console.
   [BS] Access URLs:
    ------------------------------------
          Local: http://localhost:10081
       External: http://10.0.1.5:10081
    ------------------------------------
             UI: http://localhost:3001
    UI External: http://10.0.1.5:3001
    ------------------------------------
   [BS] Serving files from: ./platforms
   Launching browser.
   ```

7. The preview will prompt for using either the simple browser or the mobile browser simulator. Select the simple browser. This will launch the application using the default browser.

   ```
   Verifying server configuration...
   ? Select how to preview your app: (Use arrow keys)
   ❯ browser: Simple browser rendering
   mbs: Mobile Browser Simulator
   ```

> **Tip:** To change the default browser use the `mfpdev config` command.

Summary

So far we took basic Cordova/Ionic application and add the MobileFirst Plugins, we add to go through the porcess since its not a good practice to add plugins and environments to git repositories.

Next

Now that we add the necessaries plugins and environments we can start edit the code and the push support.

1. Open the **app.js** file and search for the wlCommonInit() method and add the MFPPush.initialize the block inside

```
//Adding push notification support
MFPPush.initialize(
   function(successResponse) {
         console.log(">> in MFPPush.initialize - successResponse ...");
    WL.Logger.debug("Successfully intialized");
    MFPPush.registerNotificationsCallback(notificationReceived);
}, function(failureResponse) {
      console.log(">> in MFPPush.initialize - failureResponse ...:" + failureResp
onse);
    alert("Failed to initialize");
});
```

Before

```
function wlCommonInit() {
  console.log(">> wlCommonInit() ..." );
  var serverUrl = WL.App.getServerUrl(function(success){
      console.log(">> wlCommonInit() - success: " + success);
  }, function(fail){
      console.log(">> wlCommonInit() - failed: " + fail);
  });

  //Calling to the MobileFirst Server
  WLAuthorizationManager.obtainAccessToken().then(
        function (accessToken) {
          console.log(">> Success - Connected to MobileFirst Server");
        },
        function (error) {
          console.log(">> Failed to connect to MobileFirst Server");
        }
  );
};
```

After

```
function wlCommonInit() {
  console.log(">> wlCommonInit() ..." );
  var serverUrl = WL.App.getServerUrl(function(success){
      console.log(">> wlCommonInit() - success: " + success);
  }, function(fail){
      console.log(">> wlCommonInit() - failed: " + fail);
  });

  //Calling to the MobileFirst Server
  WLAuthorizationManager.obtainAccessToken().then(
        function (accessToken) {
          console.log(">> Success - Connected to MobileFirst Server");
        },
        function (error) {
          console.log(">> Failed to connect to MobileFirst Server");
        }
  );

  //Adding push notification support
  MFPPush.initialize(
       function(successResponse) {
            console.log(">> in MFPPush.initialize - successResponse ...");
        WL.Logger.debug("Successfully intialized");
        MFPPush.registerNotificationsCallback(notificationReceived);
    }, function(failureResponse) {
          console.log(">> in MFPPush.initialize - failureResponse ...:" + failureResponse);
        alert("Failed to initialize");
    });
};
```

2. Next we are going to add the **notificationReceived** callback method to the bottom of the file.

```
// Adding push notification callback method
var notificationReceived = function(message) {
alert (JSON.stringify(message));
};
```

```
function wlCommonInit() {
  console.log(">> wlCommonInit() ..." );
  var serverUrl = WL.App.getServerUrl(function(success){
      console.log(">> wlCommonInit() - success: " + success);
  }, function(fail){
      console.log(">> wlCommonInit() - failed: " + fail);
  });

  //Calling to the MobileFirst Server
  WLAuthorizationManager.obtainAccessToken().then(
        function (accessToken) {
          console.log(">> Success - Connected to MobileFirst Server");
        },
        function (error) {
          console.log(">> Failed to connect to MobileFirst Server");
        }
  );

  //Adding push notification support
  MFPPush.initialize(
       function(successResponse) {
            console.log(">> in MFPPush.initialize - successResponse ...");
        WL.Logger.debug("Successfully intialized");
        MFPPush.registerNotificationsCallback(notificationReceived);
    }, function(failureResponse) {
          console.log(">> in MFPPush.initialize - failureResponse ...:" + failureResponse);
        alert("Failed to initialize");
    });
};

// Adding push notification callback method
var notificationReceived = function(message) {
    alert (JSON.stringify(message));
};
```

3. Lets save the changes and test our application on the simulator.

4. Before we can test the application on the simulator, lets register the application with the MobileFirst server, run the following command within the IBMEmployeeApp folder to register the application:

```
$ mfpdev app register
```

```
Elirans-MacBook-Pro:IBMEmployeeApp eliran_pro$ mfpdev app register
Verifying server configuration...
Registering to server:'http://172.31.99.103:9080' runtime:'mfp'
Updated config.xml file located at: /Users/eliran_pro/Documents/projects/NATechnicalAcademy2016/MFPushNotificationLab/IBMEmployeeApp/config.xml
Run 'cordova prepare' to propagate changes.
Registered app for platform: android
Registered app for platform: ios
Elirans-MacBook-Pro:IBMEmployeeApp eliran_pro$
```

5. Run the cordova prepare command to propagate the changes to the application project.

```
$ cordova prepare
```

```
Elirans-MacBook-Pro:IBMEmployeeApp eliran_pro$ cordova prepare
Running command: /Users/eliran_pro/Documents/projects/NATechnicalAcademy2016/MFPushNotificationLab/IBMEmployeeApp/hooks/after_prepare/010_add_platform_class.js /Users/eliran_pro/Documents/projects/NATechnicalAcademy2016/MFPushNotificationLab/IBMEmployeeApp
add to body class: platform-android
add to body class: platform-ios
Elirans-MacBook-Pro:IBMEmployeeApp eliran_pro$
```

6. To test the application on the emulator run the following command

```
$ cordova run android
```

7. To test the application on a real device run the following command where 'TA0050AUKC' is the id of the connected device.

```
$ cordova run android --target=TA0050AUKC
```

> Note: To get a list of the available devices you can run the following command : **$ ./adb devices** adb should be found under **/Users/YOUR_NAME/Library/Android/sdk/platform-tools**

8. If you are looking at the console logs, you can look at the console logs by running and see the successResponse of the MFPPush.initialize call.

```
$ cordova run android --target=TA0050AUKC
```

```
I/chromium(24082): [INFO:CONSOLE(82)] ">> wlCommonInit() ...", source: file:///android_asset/www/js/app.js (82)
I/chromium(24082): [INFO:CONSOLE(84)] ">> wlCommonInit() - success: http://172.31.99.103:9080/mfp/api", source: file:///android_asset/www/js/app.js (84)
W/PluginManager(24082): THREAD WARNING: exec() call to WLAuthorizationManagerPlugin.getClientIdOrInvokeRegistration blocked the main thread for 70ms. Plugin should use CordovaInterface.getThreadPool().
I/chromium(24082): [INFO:CONSOLE(102)] ">> in MFPPush.initialize - successResponse ...", source: file:///android_asset/www/js/app.js (102)
D/audio_hw_primary(  290): out_standby: enter: stream (0xb8ca7d40) usecase(1: low-latency-playback)
D/hardware_info(  290): hw_info_append_hw_type : device_name = speaker
I/MotoNetwCtrlr.MotoWifiHndlr( 1137): handleMessage: Entered msg.what=1
I/MotoNetwCtrlr.MotorolaWifiSignalController( 1137): isDirty: returns true
I/MotoNetwCtrlr.MotorolaWifiSignalController( 1137): notifyListeners: calling SB [0] setWifiIndicators (Motorola api): UseMotoUI=true Vis=true SBSig=0x7f02014c=stat_sys_wifi_signal_4_fully SBAct=0x7f020507=zz_moto_stat_sys_wifi_out_fully_wide DescSig="Wifi signal full." DescAct="wifi out" NetName=""Eliran's Wi-Fi Network""
I/MotoNetwCtrlr.MotorolaWifiSignalController( 1137): notifyListeners: calling SB [1] setWifiIndicators (Motorola api): UseMotoUI=true Vis=true SBSig=0x7f02014c=stat_sys_wifi_signal_4_fully SBAct=0x7f020507=zz_moto_stat_sys_wifi_out_fully_wide DescSig="Wifi signal full." DescAct="wifi out" NetName=""Eliran's Wi-Fi Network""
I/MotoNetwCtrlr.MotorolaWifiSignalController( 1137): notifyListeners: calling SB [2] setWifiIndicators (Motorola api): UseMotoUI=true Vis=true SBSig=0x7f02014c=stat_sys_wifi_signal_4_fully SBAct=0x7f020507=zz_moto_stat_sys_wifi_out_fully_wide DescSig="Wifi signal full." DescAct="wifi out" NetName=""Eliran's Wi-Fi Network""
I/MotoNetwCtrlr.MotoWifiHndlr( 1137): handleMessage: Completed
E/NEW_BHD (  3098): Battery Power Supply logging Daemon start!!!!!
```

9. Next we are going to implement the the **isPushSupported** method and **registerDevice** method, Open the **controllers.js** and find the **settingsCtrl** add modify the code to implement the **isPushSupported** and **registerDevice** methods

```
//MFPPush.isPushSupported method
  MFPPush.isPushSupported(
      function(successResponse) {
        alert("Push Supported: " + successResponse);
      }, function(failureResponse) {
        alert("Failed to get push support status");
  });

//MFPPush.registerDevice method
    var options = {"phoneNumber":""};
    MFPPush.registerDevice(options,
       function(successResponse) {
            console.log(">>> registerDevice - successResponse :" + successRespons
e);
            alert("Successfully registered");
    }, function(failureResponse) {
            console.log(">>> registerDevice - failureResponse :" + failureRespons
e);
            alert("Failed to register");
});
```

Before

```
20  ibmApp.controller('settingsCtrl', ['$scope', function ($scope) {
21          console.log(">> in settingsCtrl ... ");
22
23          $scope.isPushSupported = function(){
24            console.log(">> in isPushSupported ... ");
25          };
26
27          $scope.registerDevice = function(){
28            console.log(">> in registerDevice ... ");
29          }
30
31  }])
```

After

```
ibmApp.controller('settingsCtrl', ['$scope', function ($scope) {
        console.log(">> in settingsCtrl ... ");

        $scope.isPushSupported = function(){
          console.log(">> in isPushSupported ... ");
          //MFPPush.isPushSupported method
            MFPPush.isPushSupported(
            function(successResponse) {
              alert("Push Supported: " + successResponse);
            }, function(failureResponse) {
              alert("Failed to get push support status");
            });
        };

        $scope.registerDevice = function(){
          console.log(">> in registerDevice ... ");
          //MFPPush.registerDevice method
          var options = {"phoneNumber":""};
          MFPPush.registerDevice(options,
             function(successResponse) {
              console.log(">>> registerDevice - successResponse :" + successResponse);
              alert("Successfully registered");
          }, function(failureResponse) {
              console.log(">>> registerDevice - failureResponse :" + failureResponse);
              alert("Failed to register");
          });
        }

}])
```

Next we will need to configure the MFF server for push and we will need to create GCM server API Key and sender id

1. Open new browser window and navigate to: https://developers.google.com/mobile/add?platform=android&cntapi=gcm&cnturl=https:%2F%2Fdevelopers.google.com%2Fcloud-messaging%2Fandroid%2Fclient&cntlbl=Continue%20Adding%20GCM%20Support&%3Fconfigured%3Dtrue

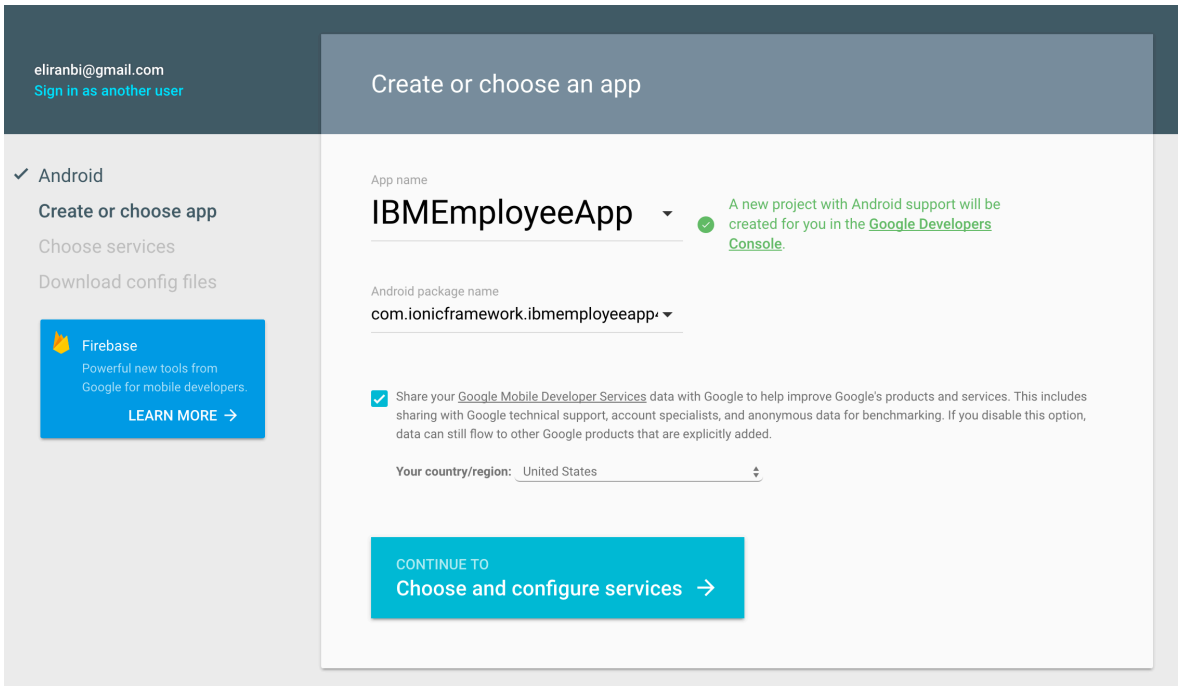
2. You will need to have a gmail account, go ahead and login with your email and password.

3. In the project name enter: **IBMEmployeeApp**

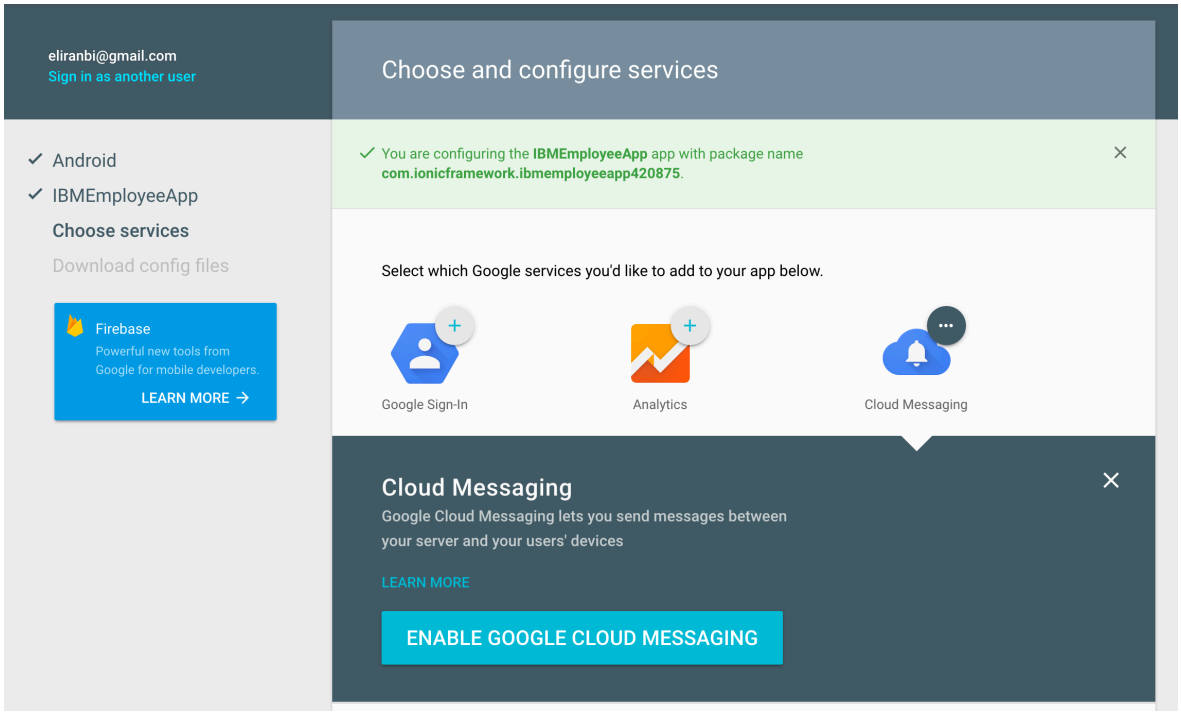
4. In the android package name enter : **com.ionicframework.ibmemployeeapp420875**

5. Press on the "**CONTINUE TO Choose and configure service**" button

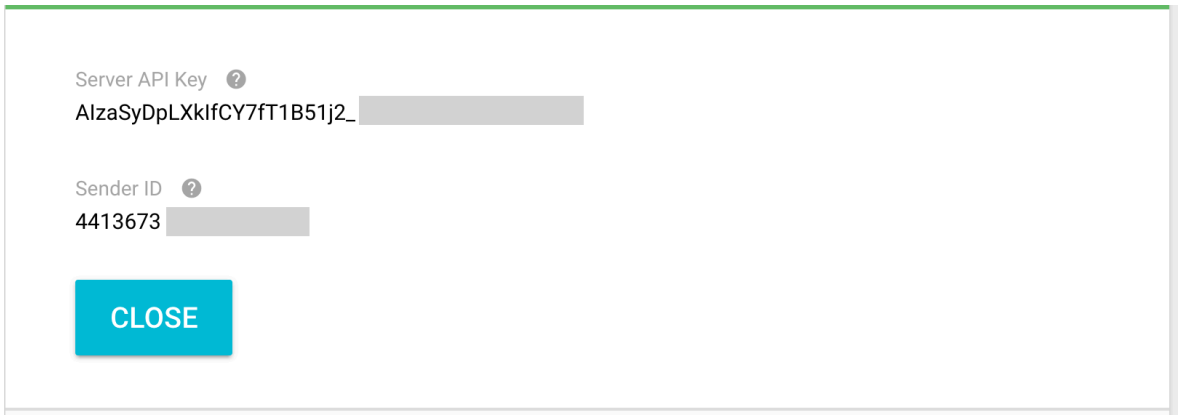
---



6. Press on the “**Enable Google Cloud Messaging**” button



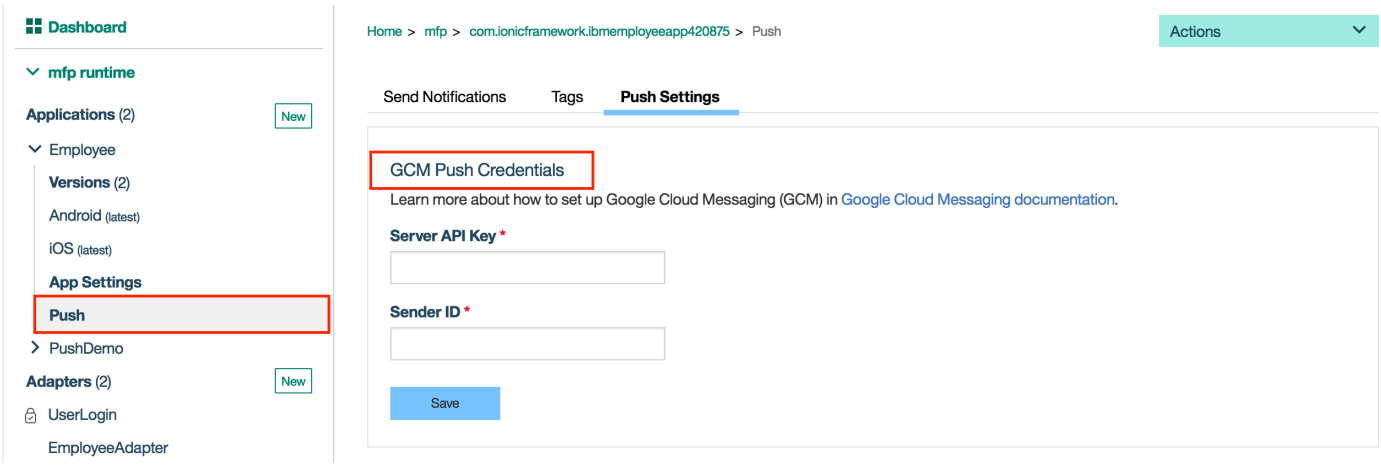
7. Copy your application Server API Key and Sender ID



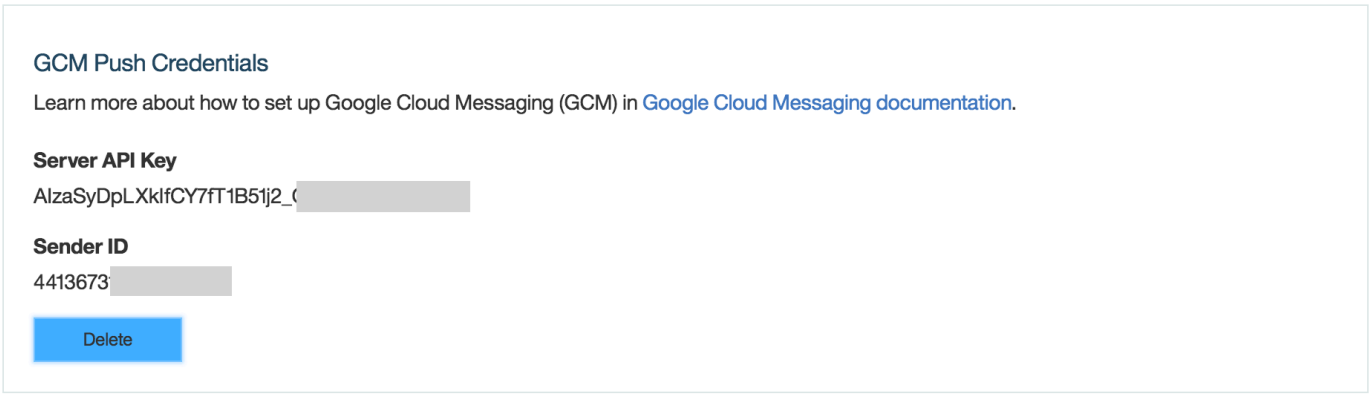
8. Now that we have the GCM Server API key and Sender ID, lets launch the MFF web console, run the following command:

```
$ mfpdev server console
```

9. Select the **Employee** application in the left side menu and press the **Push**

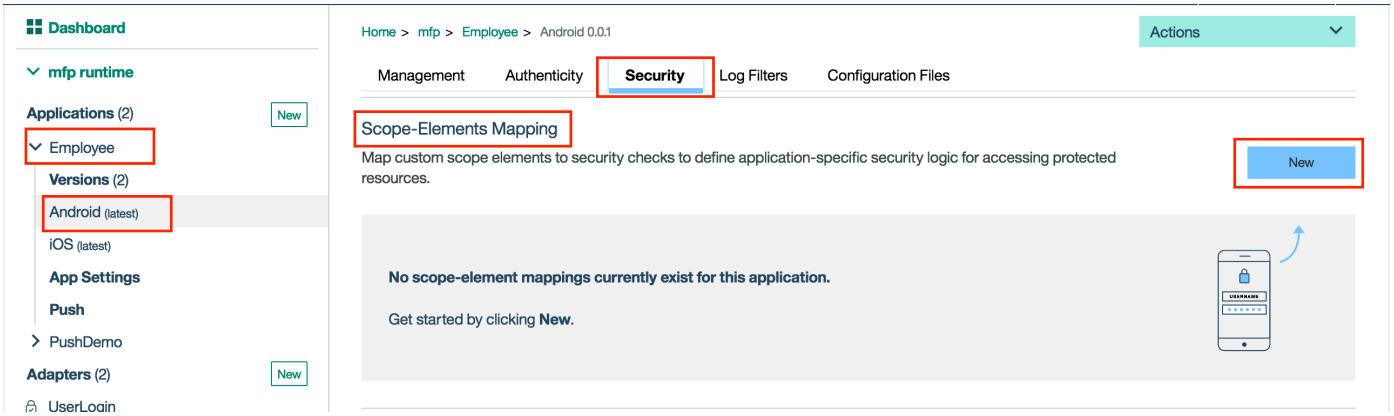


10. Enter the server API key and Sender ID and press the 'Save' button.

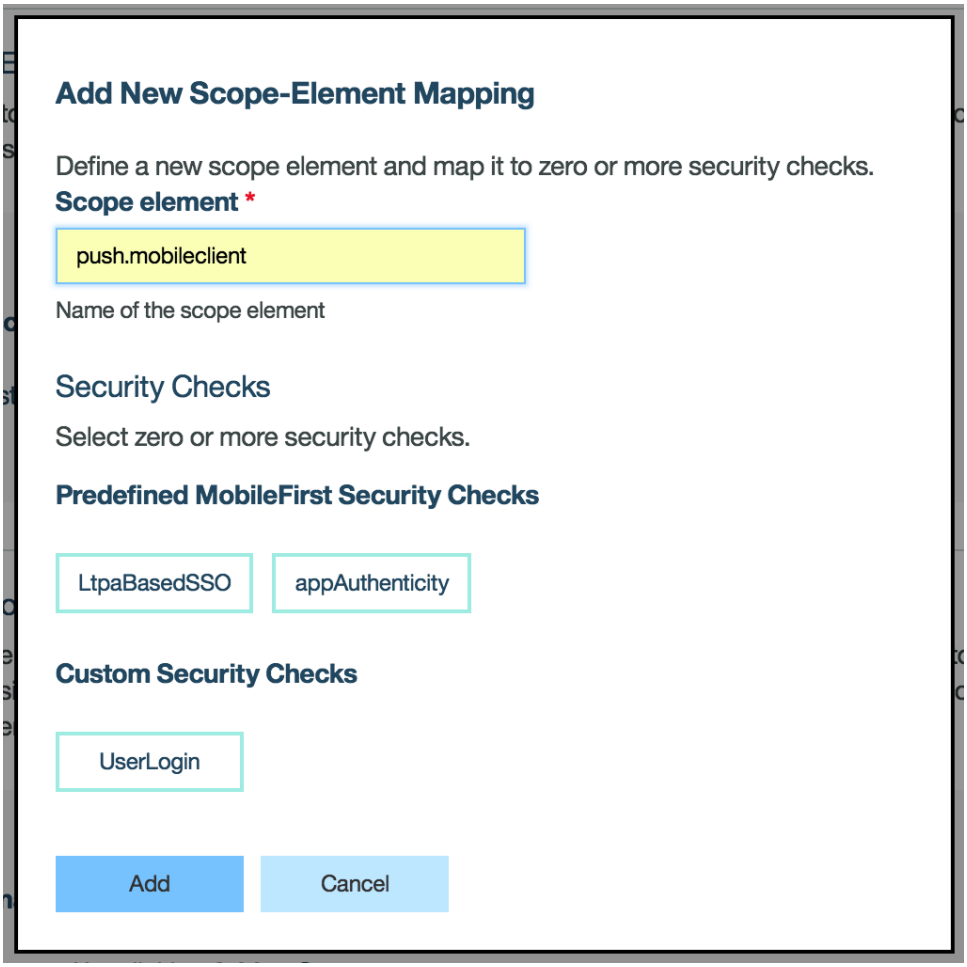


11. Next we need to set application security.

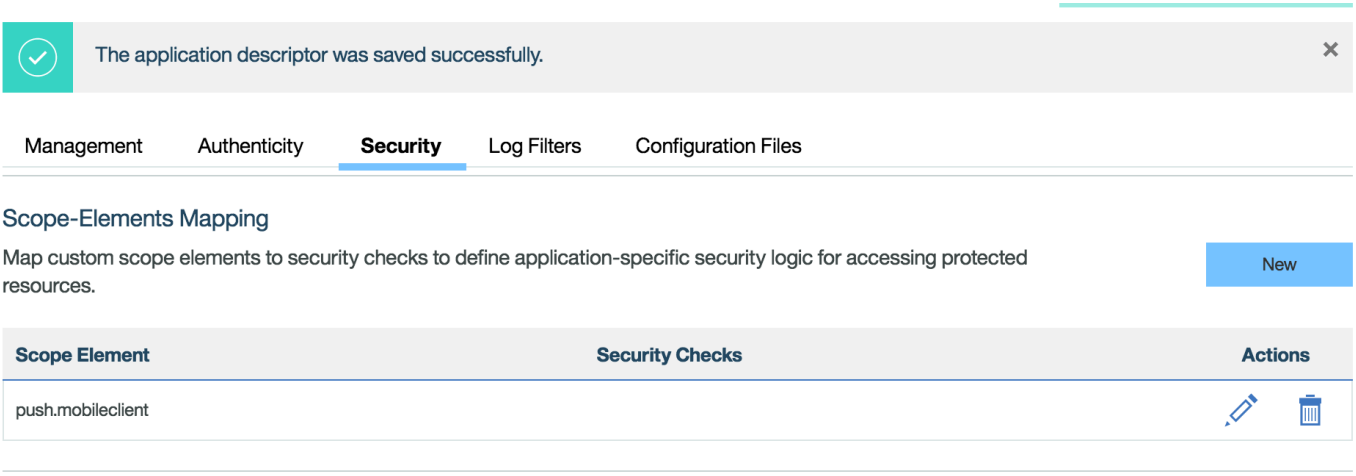
12. Under Applications select the **Employee -> Android -> Security** and press the 'New' button.



13. In the Scope element text input filed enter **push.mobileclient** and press the 'Add' button.



14. Your application scope-elements mapping should look like this:

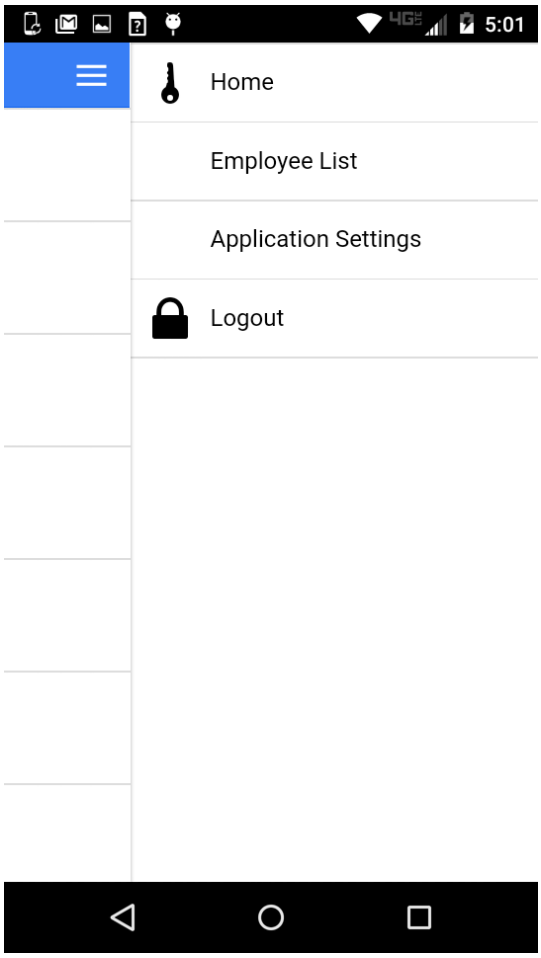


15. Run the application again on your device by running the following command:

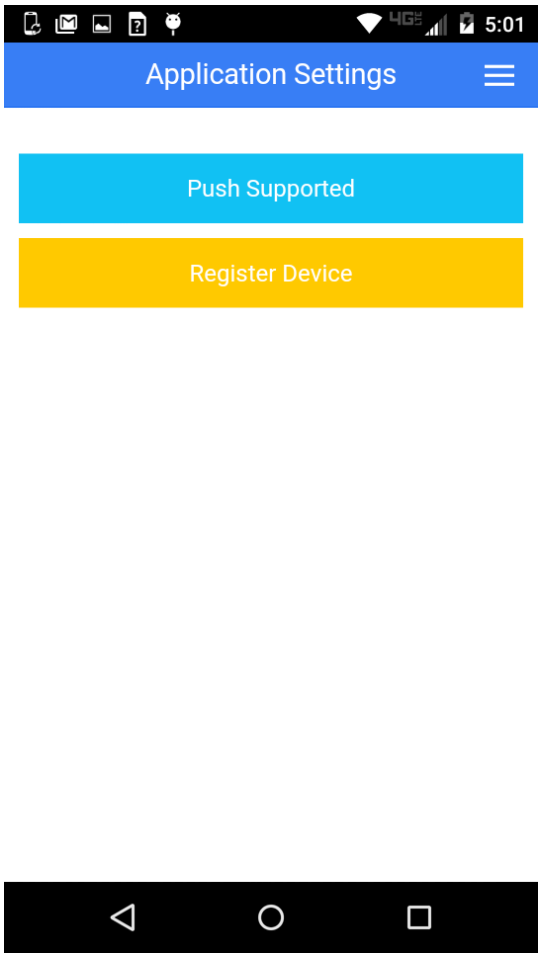
```
$ cordova run android --target=TA0050AUKC
```

16. Login to the application using demo/demo as the username and password.

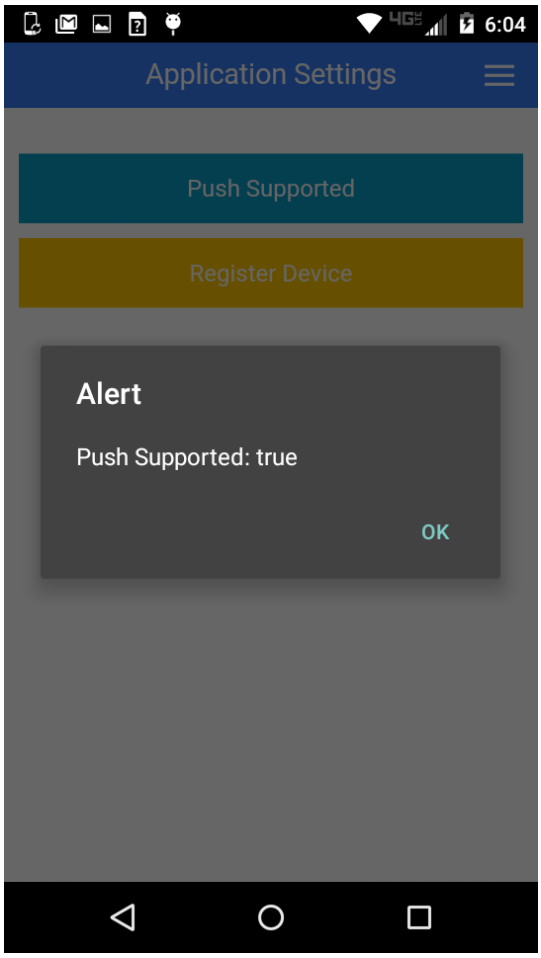
17. Press the Menu top right button.



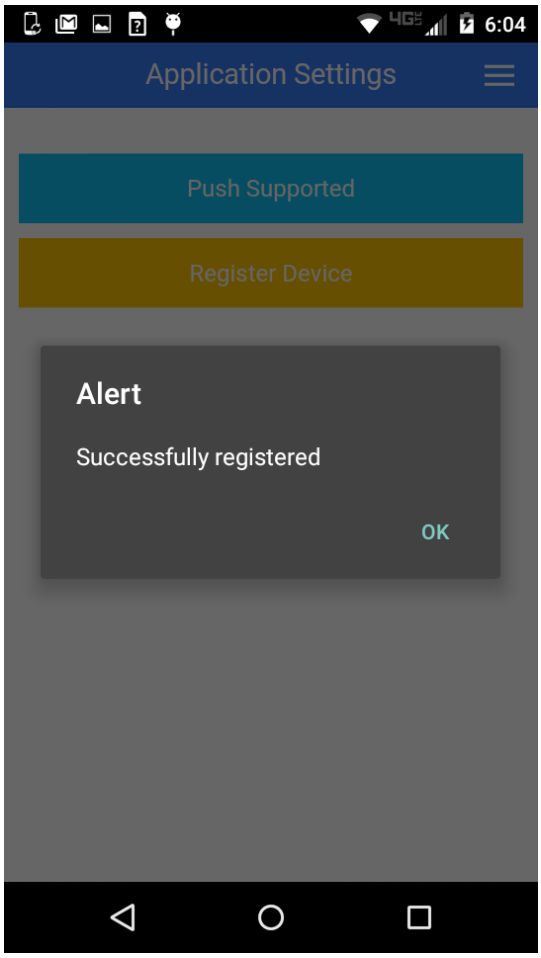
18. Press the Application Settings option.



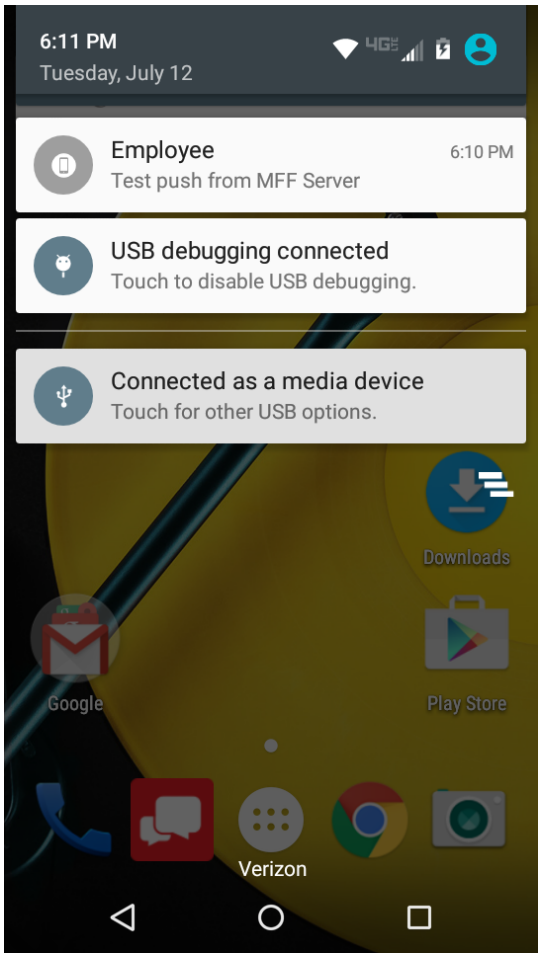
19. Press the "Push Supported" button you should see the following alert

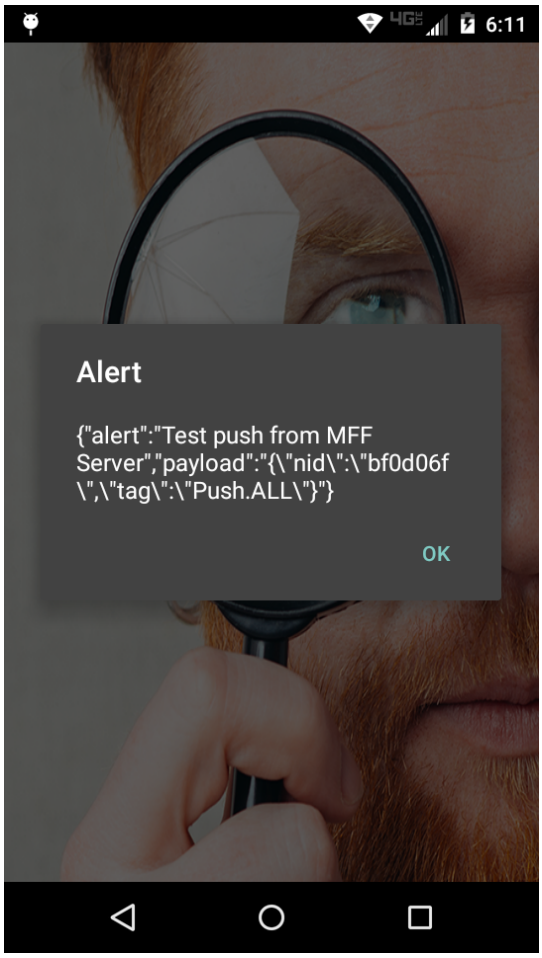


20. Press the "Register Device" button you should see the following alert.



21. Now lets use the MobileFirst console to send a push notification to the device.
22. Select the Application **Employee -> Push -> Send Notification** tab.
23. In the **SendTo** select box choose **All**
24. Enter any notification Text that you want "Test push from MFF Server"
25. Press the "Send" Button
26. Look at your device and you should see the push notification message coming:





Summary

In this lab, you used the MFF push notification SDK, and easily add support for push notification in your application.