Lab 5 - Implement the MFP adapter framework (Server Side)

If you have looked at the code in app.js in detail, you have noticed that the app is currently using the Employee service and Employee Details controllers. These services use angular $http services to get the application data, which is stored in .json files locally on the device. In the next section, you will create MFP adapters to get data from back-end services. MFP Adapters provide a way to retrieve and manage data for your mobile client app on the server side.

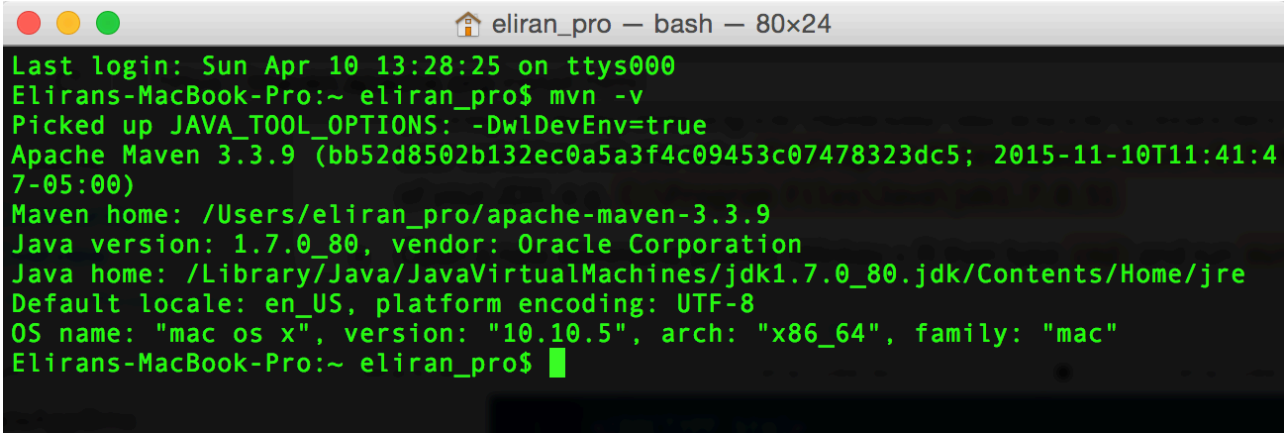
The MobileFirst Adapter framework provides support for developing adapters in Java or JavaScript to interface with various back-end architectures such as HTTP, SQL and SAP among others. The Adapter framework also automatically couples with the MFP security and analytics frameworks, enabling consistent security to back-end resources and ability to record, measure and compare the operational characteristics of the adapter traffic - volumes, servers, response times, etc...

For our lab, we will build a Java adapter to interact with a REST API provided by BlueMix service StrongLoop/API Connect (Node.js application) available using the follow url: http://employeenodeapp.mybluemix.net/

We are provided two REST end points **employees** and **details**. The **employees** end point takes no parameters and returns a JSON-formatted string of employees (our employee "list"), and **details** takes an employee ID parameter and returns the details for that employee.

Using imagination, the **REST Api** could be an REST APi provided by your organization which allow you to access your system of records.
The adapter framework allows you to easily adapt to changes in backend data formats or even completely replace with a different data source, without affecting the client app running on (dozens, hundreds, thousands or millions of) mobile devices. Changes in your back-end can be addressed in the adapter tier without forcing you to rebuild and redistribute the client app.

> 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.
>
> Note: Please make sure you have **Apache Maven** installed, and you add it to your path, you can confirm by running **mvn -v** in the terminal



Steps

Create the adapter

> Note: In previous versions of MFP, you had to create BackEnd project first, before you could create an adapter, Starting v8.0 you don't need to create a back-end project in order to create an adapter.

1. Create new folder called AdapterServices in parallel to your IBMEmployeeApp folder

   ```
   cd ..
   mkdir AdapterServices
   ```

2. Change context to AdapterServices

```
cd AdapterServices
```

```
Elirans-MacBook-Pro:IBMEmployeeApp eliran_pro$ mkdir AdapterServices
Elirans-MacBook-Pro:IBMEmployeeApp eliran_pro$ cd AdapterServices/
Elirans-MacBook-Pro:AdapterServices eliran_pro$
```

3. **Create** a Java-based adapter to your project

```
mfpdeve adapter create
```

1. When prompted, name your adapter `EmployeeAdapter`
2. For adapter type select : `Java`

bash | java

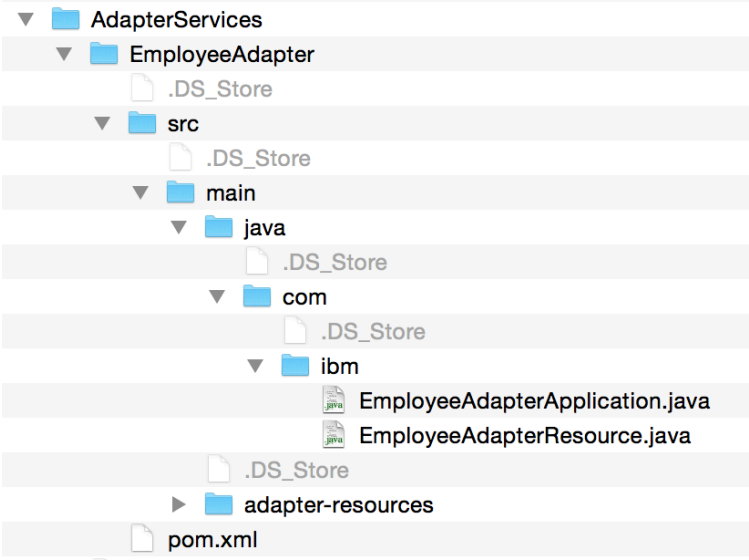
```
Elirans-MacBook-Pro:AdapterServices eliran_pro$ mfpdev adapter create
? Enter adapter name: EmployeeAdapter
? Select adapter type:
  HTTP
  SQL
❯ Java
```

3. For Java package enter : `com.ibm` and press `Enter`. You should get the following success message:
4. For group ID enter : `com.ibm` and press `Enter`. You should get the following success message:

bash | java

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

5. Looking at your file directory you should see the following structure/files



Implement the adapter procedures

The Java adapter implements the JAX-RS standard, allowing your adapter to also serve as a REST-ful endpoint. The procedures in the adapter are linked to HTTP verbs such as GET and POST. The adapter is created with sample procedures, which you can remove. In the next several steps, you will add code to implement two new methods:

- **list()**
- **details()**

1. Using your favorite IDE open the **EmployeeAdapterResource.java** file in the **EmployeeAdapter/src/main/java/com/ibm** directory.

2. Find the **ConfigurationAPI configApi;** statement around line 49 and remove all of the methods from the class. Your adapter should look like this:

```java
package com.ibm;

import io.swagger.annotations.Api;
import io.swagger.annotations.ApiOperation;
import io.swagger.annotations.ApiParam;
import io.swagger.annotations.ApiResponse;
import io.swagger.annotations.ApiResponses;

import java.util.HashMap;
import java.util.Map;
import java.util.logging.Logger;

import javax.ws.rs.FormParam;
import javax.ws.rs.GET;
import javax.ws.rs.HeaderParam;
import javax.ws.rs.POST;
import javax.ws.rs.Path;
import javax.ws.rs.PathParam;
import javax.ws.rs.Produces;
import javax.ws.rs.QueryParam;
import javax.ws.rs.core.Context;
import javax.ws.rs.core.MediaType;
import javax.ws.rs.core.Response;
import javax.ws.rs.core.Response.Status;

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

@Api(value = "Sample Adapter Resource")
@Path("/resource")
public class EmployeeAdapterResource {
    /*
     * For more info on JAX-RS see
     * https://jax-rs-spec.java.net/nonav/2.0-rev-a/apidocs/index.html
     */
    // Define logger (Standard java.util.Logger)
    static Logger logger = Logger.getLogger(EmployeeAdapter1Resource.class.getName());

    // Inject the MFP configuration API:
    @Context
    ConfigurationAPI configApi;
}

```

3. Modify the Path statement to root the adapter REST path at `/services` rather than `/resources`.

```java
@Api(value = "Sample Adapter Resource")
@Path("/resource")
public class EmployeeAdapterResource {
    /*
```

4. Add the **employees** method just before the final curly brace. This method implements the REST operation "/list", returning a list of all employees by calling **getHttp()** method with our back-end REST end point, we going to implement the **getHttp()** in the next few steps.

```
/*
 * Path for method:
 * "<server address>/mfp/api/adapters/EmployeeAdapter/services/list"
 */
@ApiOperation(value = "Get employee list", notes = "Return employee list")
@ApiResponses(value = { @ApiResponse(code = 200, message = "A constant string is
returned") })
@GET
@Path("/list")
@Produces(MediaType.TEXT_PLAIN)
@OAuthSecurity(enabled = false)
public String employees() {
    System.out.println(">> in employees() ...");
    logger.info(">> EmployeeAdapterResource: employees");
    String rsp = null;
    try {
        rsp =  getHttp("http://employeenodeapp.mybluemix.net/employees");
    } catch (ClientProtocolException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    } catch (IOException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
    return rsp;
}
```

5. Add the **getDetails** method just before the final curly brace. This method implements the REST operation "/details/{id}", returning the details of a given employee by calling into **getHttp()** method with the supplied employee id.

```
/*
 * Path for method:
 * "<server address>/mfp/api/adapters/EmployeeAdapter/services/details/{id}"
 */

@ApiOperation(value = "Employee Details by Id", notes = "Return the employee deti
als, by Id")
@ApiResponses(value = {
        @ApiResponse(code = 200, message = "Property value returned."),
        @ApiResponse(code = 404, message = "Property value not found.") })
@GET
@Path("/details/{id}")
@Produces(MediaType.TEXT_PLAIN)
public String getDetails(
        @ApiParam(value = "The name of the property to lookup", required = true)
@PathParam("id") String id) {
    // Get the value of the property:
    System.out.println(">> in getDetails() ...");
    System.out.println(">> id :[" + id + "]");
    String rsp = null;
    try {
        rsp =  getHttp("http://employeenodeapp.mybluemix.net/details?id=" + id);
    } catch (ClientProtocolException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    } catch (IOException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
    return rsp;

}
```

6. Add the **getHttp()** method just before the final curly brace. This method implements the REST http calls to our back-end and will be used internally by operation "/details/{id}" and "/list".

```
private final String USER_AGENT = "Mozilla/5.0";
public String getHttp(String url) throws ClientProtocolException, IOException{
    HttpClient client = HttpClientBuilder.create().build();
    HttpGet request = new HttpGet(url);
    // add request header
    request.addHeader("User-Agent", USER_AGENT);
    HttpResponse response = client.execute(request);
    System.out.println("Response Code : "
                + response.getStatusLine().getStatusCode());

    BufferedReader rd = new BufferedReader(
        new InputStreamReader(response.getEntity().getContent()));

    StringBuffer result = new StringBuffer();
    String line = "";
    while ((line = rd.readLine()) != null) {
        result.append(line);
    }
    return result.toString();
}
```

7. Add the import statement for our backend jar file after the other imports

```
/* Add org.apach.http*/
import org.apache.http.HttpResponse;
import org.apache.http.client.ClientProtocolException;
import org.apache.http.client.HttpClient;
import org.apache.http.client.methods.HttpGet;
import org.apache.http.impl.client.HttpClientBuilder;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
```

8. **Save** your changes.

   Your adapter code should be like this now:

```java

package com.ibm;

import io.swagger.annotations.Api;
import io.swagger.annotations.ApiOperation;
import io.swagger.annotations.ApiParam;
import io.swagger.annotations.ApiResponse;
import io.swagger.annotations.ApiResponses;

import java.util.HashMap;
import java.util.Map;
import java.util.logging.Logger;

import javax.ws.rs.FormParam;
import javax.ws.rs.GET;
import javax.ws.rs.HeaderParam;
import javax.ws.rs.POST;
import javax.ws.rs.Path;
import javax.ws.rs.PathParam;
import javax.ws.rs.Produces;
import javax.ws.rs.QueryParam;
import javax.ws.rs.core.Context;
import javax.ws.rs.core.MediaType;
import javax.ws.rs.core.Response;
import javax.ws.rs.core.Response.Status;

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

/* Add org.apach.http*/
import org.apache.http.HttpResponse;
import org.apache.http.client.ClientProtocolException;
import org.apache.http.client.HttpClient;
import org.apache.http.client.methods.HttpGet;
import org.apache.http.impl.client.HttpClientBuilder;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;



@Api(value = "Sample Adapter Resource")
@Path("/services")
public class EmployeeAdapterResource {
    /*
     * For more info on JAX-RS see
     * https://jax-rs-spec.java.net/nonav/2.0-rev-a/apidocs/index.html
     */
    // Define logger (Standard java.util.Logger)
    static Logger logger = Logger.getLogger(EmployeeAdapter1Resource.class.getName());

```

```
57
58      // Inject the MFP configuration API:
59      @Context
60      ConfigurationAPI configApi;
61
62      /*
63       * Path for method:
64       * "<server address>/mfp/api/adapters/Employee/services/list"
65       */
66
67      @ApiOperation(value = "Get employee list", notes = "Return employee list")
68      @ApiResponses(value = { @ApiResponse(code = 200, message = "A constant string is returned") })
69      @GET
70      @Path("/list")
71      @Produces(MediaType.TEXT_PLAIN)
72      public String employees() {
73          System.out.println(">> in employees() ...");
74          logger.info(">> EmployeeAdapterResource: employees");
75          String rsp = null;
76          try {
77              rsp =  getHttp("http://employeenodeapp.mybluemix.net/employees");
78          } catch (ClientProtocolException e) {
79              // TODO Auto-generated catch block
80              e.printStackTrace();
81          } catch (IOException e) {
82              // TODO Auto-generated catch block
83              e.printStackTrace();
84          }
85          return rsp;
86      }
87
```

```
88      /*
89       * Path for method:
90       * "<server address>/mfp/api/adapters/Employee/services/details/{id}"
91       */
92
93      @ApiOperation(value = "Employee Details by Id", notes = "Return the employee detials, by Id")
94      @ApiResponses(value = {
95              @ApiResponse(code = 200, message = "Property value returned."),
96              @ApiResponse(code = 404, message = "Property value not found.") })
97      @GET
98      @Path("/details/{id}")
99      @Produces(MediaType.TEXT_PLAIN)
100     public String getDetails(
101             @ApiParam(value = "The name of the property to lookup", required = true) @PathParam("id")
                String id) {
102         // Get the value of the property:
103         System.out.println(">> in getDetails() ...");
104         System.out.println(">> id :[" + id + "]");
105         String rsp = null;
106         try {
107             rsp =  getHttp("http://employeenodeapp.mybluemix.net/details?id=" + id);
108         } catch (ClientProtocolException e) {
109             // TODO Auto-generated catch block
110             e.printStackTrace();
111         } catch (IOException e) {
112             // TODO Auto-generated catch block
113             e.printStackTrace();
114         }
115         return rsp;
116
117     }
118
119     private final String USER_AGENT = "Mozilla/5.0";
120     public String getHttp(String url) throws ClientProtocolException, IOException{
121         HttpClient client = HttpClientBuilder.create().build();
122         HttpGet request = new HttpGet(url);
123         // add request header
124         request.addHeader("User-Agent", USER_AGENT);
125         HttpResponse response = client.execute(request);
126         System.out.println("Response Code : "
127                     + response.getStatusLine().getStatusCode());
128
129         BufferedReader rd = new BufferedReader(
130             new InputStreamReader(response.getEntity().getContent()));
131         StringBuffer result = new StringBuffer();
132         String line = "";
133         while ((line = rd.readLine()) != null) {
134             result.append(line);
135         }
136         return result.toString();
137     }
138
139 }
140
```

Test your adapter

The MFP CLI provides the ability to test adapters using command line commands. This is not only helpful for manually testing your adapters during development, but it can be leveraged by automated test scripts as part of your DevOps process automation strategy.

1. To test your adapter using the MFP CLI, you must first build it and deploy it to the MFP Development server.

```
mfpdev adapter deploy
```

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

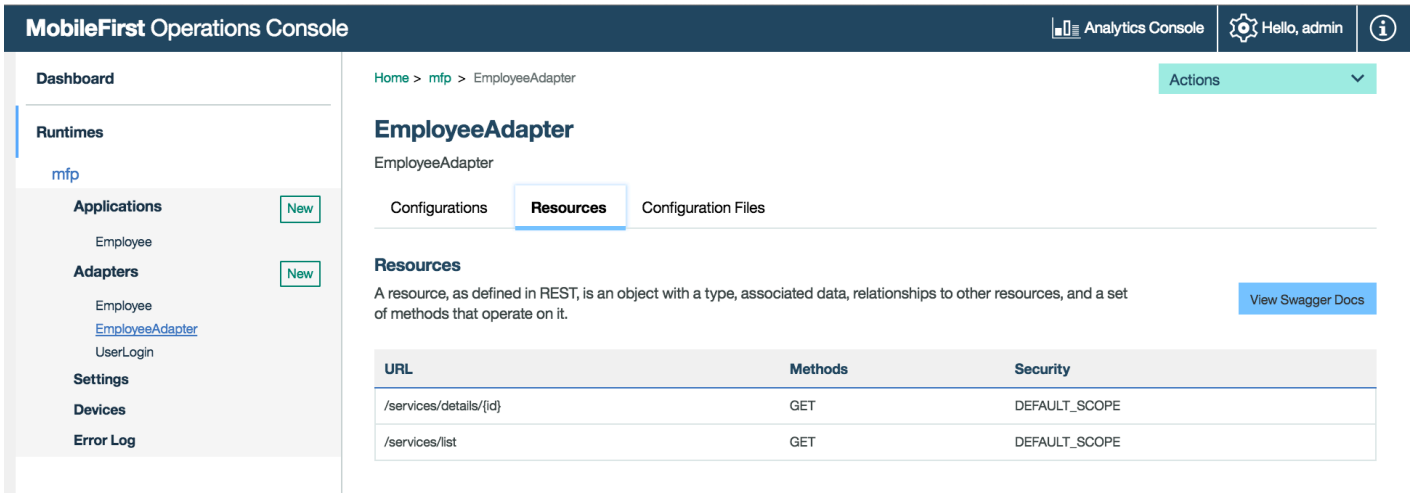
> **Note:** If you encounter compilation errors, you will need to correct them before moving forward. You can get compilation error listings by using the -d switch on the push command:

```
mfpdev adapter deploy -d
```

> Evaluate the results, edit your code and continue to push until your errors have been resolved.

2. Once your adapter builds correctly, open the **Operational Console**. You should see that your EmployeeServices adapter has been deployed

```
mfpdev server console
```



3. Close the **browser**.

4. Test the **list** procedure using the CLI

```
mfpdev adapter call
```

Use your keyboard arrow keys to highlight the adapter **EmployeeAdapter** and then press **Enter**. Then

use your keyboard arrow keys to highlight the endpoint **get:/EmployeeAdapter/services/list** and then press **Enter**. The adapter response object will be printed in the console:

```
Elirans-MacBook-Pro:EmployeeAdapter eliran_pro$ mfpdev adapter call
Verifying server configuration...
Fetching adapters from runtime 'mfp'
? Which adapter do you want to use? EmployeeAdapter
? Which endpoint do you want to use? get:EmployeeAdapter/services/list

Calling GET '/mfp/api/adapters/EmployeeAdapter/services/list'

Response:

[
  {
    "_id": "01800192",
    "first_name": "Mike",
    "last_name": "Chepesky",
    "img": "male1.png",
    "job_title": "Sales Associate"
  },
  {
    "_id": "01800193",
    "first_name": "Amy",
    "last_name": "Jones",
    "img": "female1.png",
    "job_title": "Sales Representative"
  },
  {
    "_id": "01800121",
    "first_name": "Eugene",
    "last_name": "Lee",
    "img": "male2.png",
    "job_title": "CFO"
  },
  {
    "_id": "01800114",
    "first_name": "Gary",
    "last_name": "Donovan",
    "img": "male3.png",
    "job_title": "Marketing Manager"
  },
```

5. Test the **details** procedure using the CLI

```
mfpdev adapter call
```

Use your keyboard arrow keys to highlight the adapter **EmployeeAdapter** and then press **Enter**. Then use your keyboard arrow keys to highlight the endpoint **get:/EmployeeAdapter/services/details/{id}** and then press **Enter**. The adapter response object will be printed in the console:

```
Elirans-MacBook-Pro:EmployeeAdapter eliran_pro$ mfpdev adapter call
Verifying server configuration...
Fetching adapters from runtime 'mfp'
? Which adapter do you want to use? EmployeeAdapter
? Which endpoint do you want to use? (Use arrow keys)
❯ get:EmployeeAdapter/services/details/{id}
  get:EmployeeAdapter/services/list
```

When prompted for the path parameters, enter `/services/details/01800292`, then press `Enter`. This will retrieve the details record for employee Amy Jones.

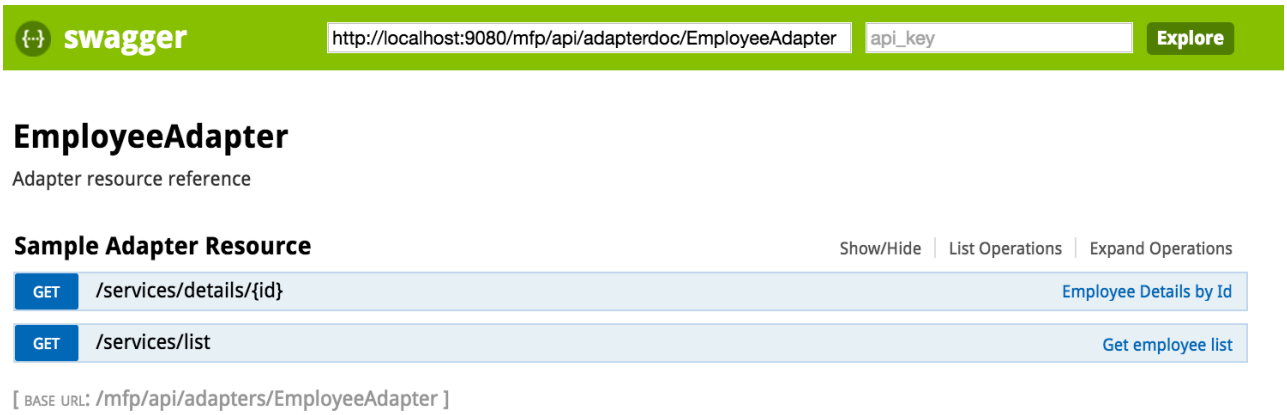
The adapter response object will be printed in the console:

```
Elirans-MacBook-Pro:EmployeeAdapter eliran_pro$ mfpdev adapter call
Verifying server configuration...
Fetching adapters from runtime 'mfp'
? Which adapter do you want to use? EmployeeAdapter
? Which endpoint do you want to use? get:EmployeeAdapter/services/details/{id}
? Enter path parameters in the form '/services/details/{id}': /services/details/01800292

Calling GET '/mfp/api/adapters/EmployeeAdapter/services/details/01800292'

Response:

{
  "_id": "01800292",
  "address": "121 5th Ave, New York, NY, 10010",
  "email": "Steven.Wells@us.ibm.com",
  "mobile": "347-002-9911",
  "fax": ""
}
Elirans-MacBook-Pro:EmployeeAdapter eliran_pro$
```

Summary

In this lab, you added a Java-based MobileFirst adapter to your project. You then edited the code to implement two procedures that will return a list of employees and employee details via REST interface calls from your mobile client. You then used the MFP CLI to invoke your adapter procedures manually to confirm they work as expected.

> Note: You can also test your adapters by using the built-in Swagger interface available through the console.



If you were unable to complete this lab, you can catch up by running this command:

```
git checkout -f step-5
```