SAP Cloud Platform - Technical Workshop

Exercise 6 - Create a basic Java app in SAP Cloud Platform

You will learn

In this tutorial series you will start from zero and develop a fully operational weather application. In total there are 10 parts to the series, each building on top of its predecessor. The entire source code of both the final and all intermediate parts are available on GitHub.

Details

The constituent parts of this tutorial series cover the following:

- How to create a simple web application on HCP
- How to apply authentication and authorization
- How to expose business functionality as an external RESTful API
- How to add JPA-based persistence to your web app
- How to leverage the multi-tenancy features of SAP Cloud Platform
- How to use the connectivity service to consume external services
- How to add a mobile-friendly UI5-based user interface to the web application

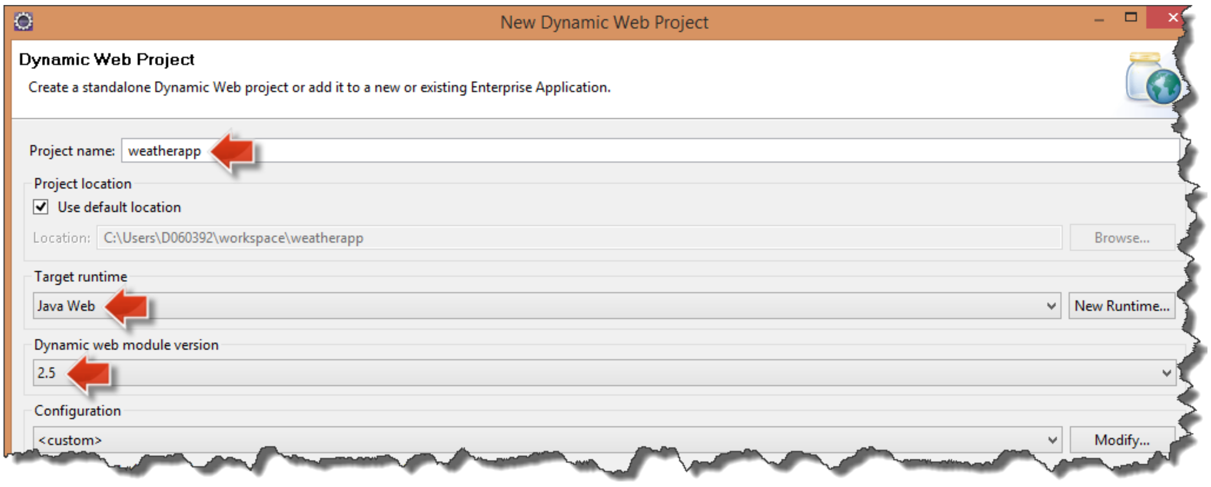
In Part 1, you will develop a basic Java app to ensure that both Eclipse IDE and the local SAP Cloud Platform (HCP) tooling have been properly installed and configured.

Step 1: Create a new dynamic web project

Create a new dynamic web project by selecting the **File > New > Dynamic Web Project** menu entry and enter the following information:

- **Name:** `weatherapp`
- **Target Runtime:** `Java Web`
- **Dynamic Web Module Version:** `2.5`

Click on **Next**


New Dynamic Web Project

Dynamic Web Project

Create a standalone Dynamic Web project or add it to a new or existing Enterprise Application.

Project name: weatherapp

Project location

☑ Use default location

Location: C:\Users\D060392\workspace\weatherapp

Target runtime

Java Web

Dynamic web module version

2.5

Configuration

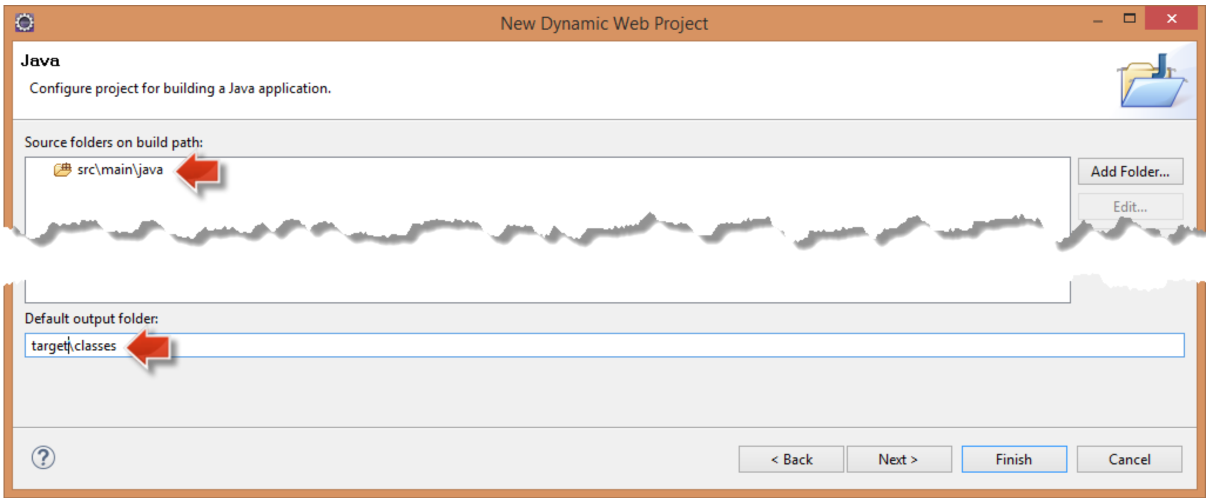
<custom>

Step 2: Add new folder

Remove the standard `src` Source folder and add a new one called `src/main/java` to create a project that adheres to the standard Maven Directory Layout.

Change the default output folder to `target/classes`

Click on **Next**.


New Dynamic Web Project

Java

Configure project for building a Java application.

Source folders on build path:

src\main\java

Default output folder:

target\classes

Step 3: Change the Content Directory

Change the Content Directory from `WebContent` to `src/main/webapp` (again, to adhere to Maven conventions)

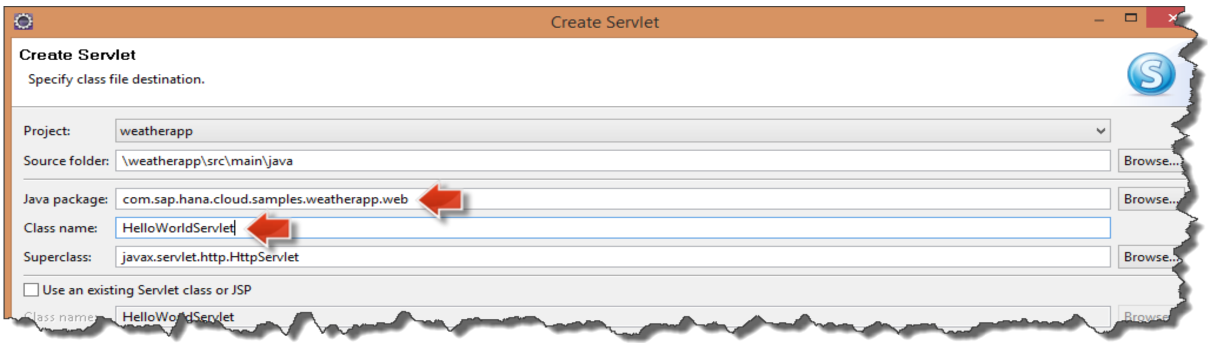
Click on **Finish**.

Step 4: Add details to weather app project

Make sure you have your `weatherapp` project folder selected, and then create a new Servlet by selecting the **File > New > Servlet** menu entry and enter the following information:

- **Package name:** `com.sap.hana.cloud.samples.weatherapp.web`
- **Class name:** `HelloWorldServlet`

Click on **Next**.


Create Servlet

Create Servlet

Specify class file destination.

Project: weatherapp

Source folder: \weatherapp\src\main\java

Java package: com.sap.hana.cloud.samples.weatherapp.web

Class name: HelloWorldServlet

Superclass: javax.servlet.http.HttpServlet

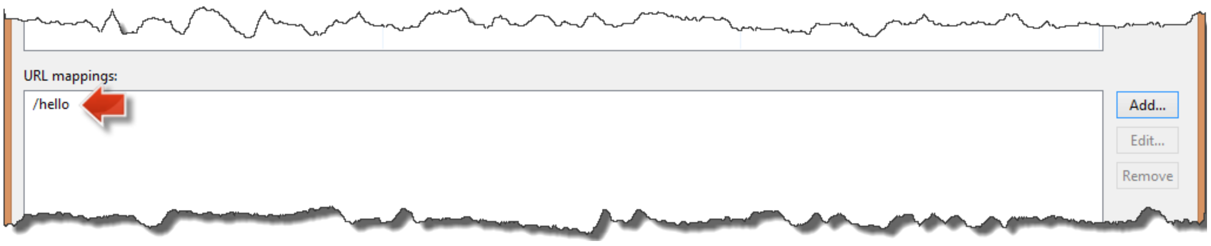
Use an existing Servlet class or JSP

Class name: HelloWorldServlet

Step 5: Change the URL Mapping

Change the URL Mapping from `/HelloWorldServlet` to `/hello` to make it a bit easier to memorize.

Click on **Finish** and the `HelloWorldServlet.java` file will open in the editor.

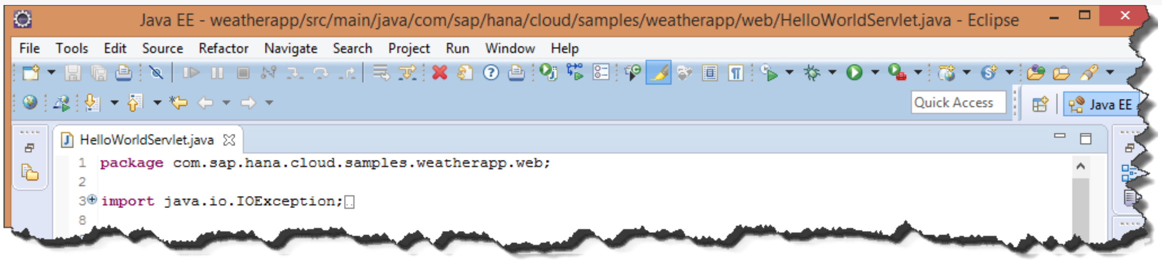

URL mappings:

/hello

Step 6: Replace comment with code

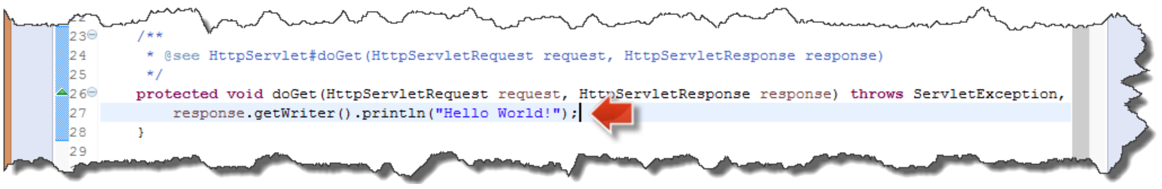
Now we need to do our first bit of coding. Navigate to the servlet's `doGet()` method and replace the `TODO` comment with the following line of code and save your changes:

```
response.getWriter().println("Hello World!");
```


Java EE - weatherapp/src/main/java/com/sap/hana/cloud/samples/weatherapp/web/HelloWorldServlet.java - Eclipse

File Tools Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
Java EE
HelloWorldServlet.java

```
package com.sap.hana.cloud.samples.weatherapp.web;

import java.io.IOException;
```


```
    /**
     * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
     */
    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException,
        response.getWriter().println("Hello World!");
    }
```

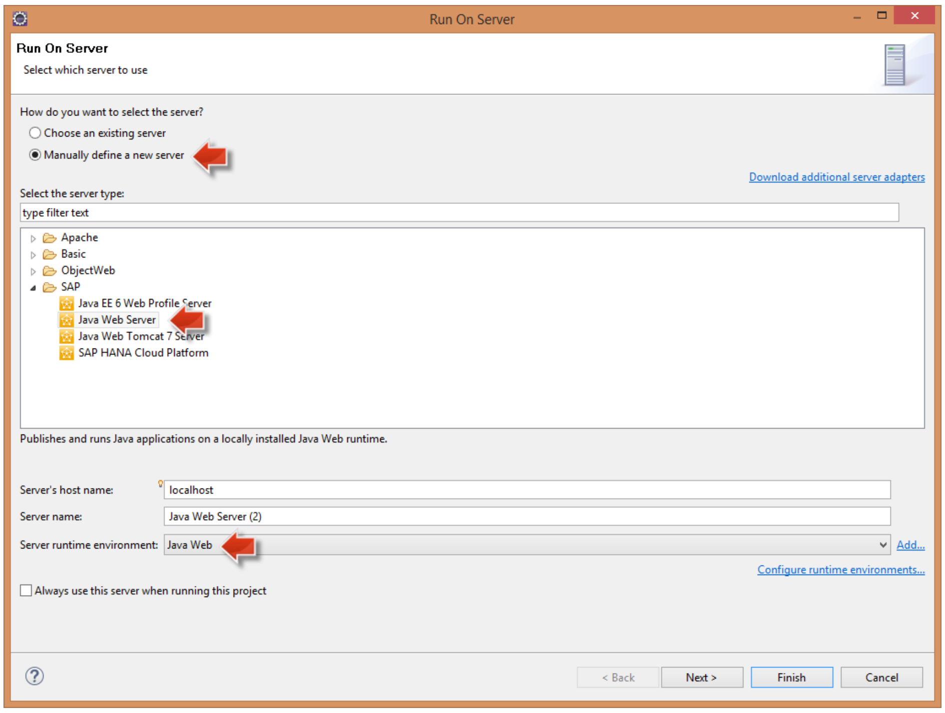
Step 7: Run on server

Deploy the application to your local server by using the **Run as > Run on Server** context menu of the `HelloWorldServlet` node in the Project Explorer view.

Step 8: Manually define a new Server

Choose the **Manually define a new Server** option and select the **SAP / Java Web Server** option from the server selection. Make sure to select **Java Web** as the server runtime environment.

Click on **Finish**. The internal browser is now started and displays the traditional message marking the first step into a new programmer's journey.



Run On Server

Run On Server
Select which server to use

How do you want to select the server?
○ Choose an existing server
● Manually define a new server

Download additional server adapters

Select the server type:
type filter text

- Apache
- Basic
- ObjectWeb
- SAP
  - Java EE 6 Web Profile Server
  - Java Web Server
  - Java Web Tomcat 7 Server
  - SAP HANA Cloud Platform

Publishes and runs Java applications on a locally installed Java Web runtime.

Server's host name: localhost
Server name: Java Web Server (2)
Server runtime environment: Java Web

Add...
Configure runtime environments...

☐ Always use this server when running this project

< Back | Next > | Finish | Cancel

Adding a simple web page to a Java app

You will learn

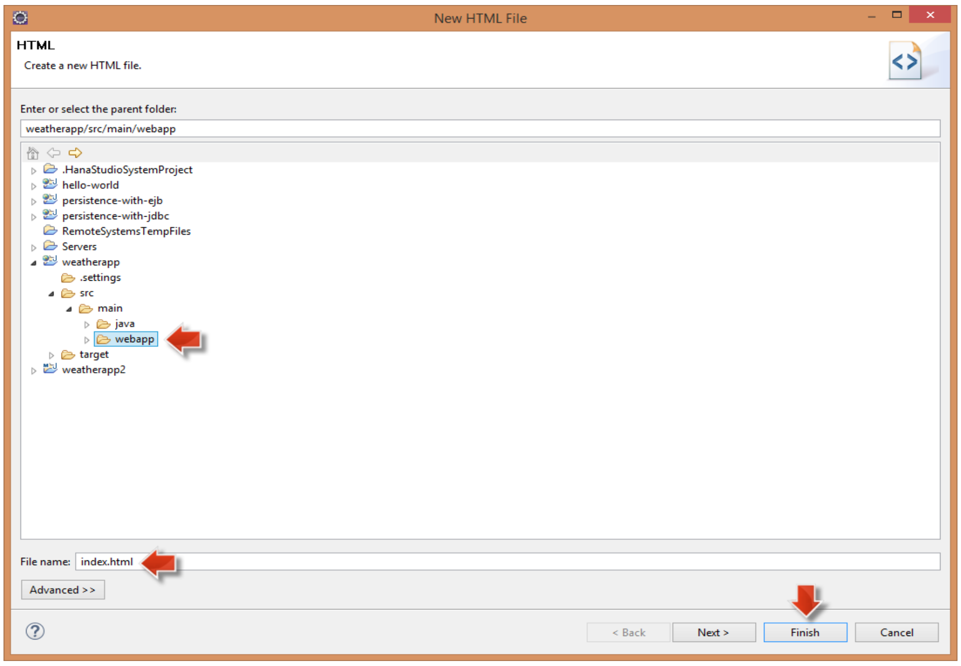
In this tutorial you will learn how to add a basic HTML page to a Java app.

Details

Step 1: Create a new HTML page

Navigate to the `webapp` node in the Project Explorer and create a new HTML page by selecting the respective context menu entry: **New > HTML File**.

Choose the name `index.html` and click on **Finish**.

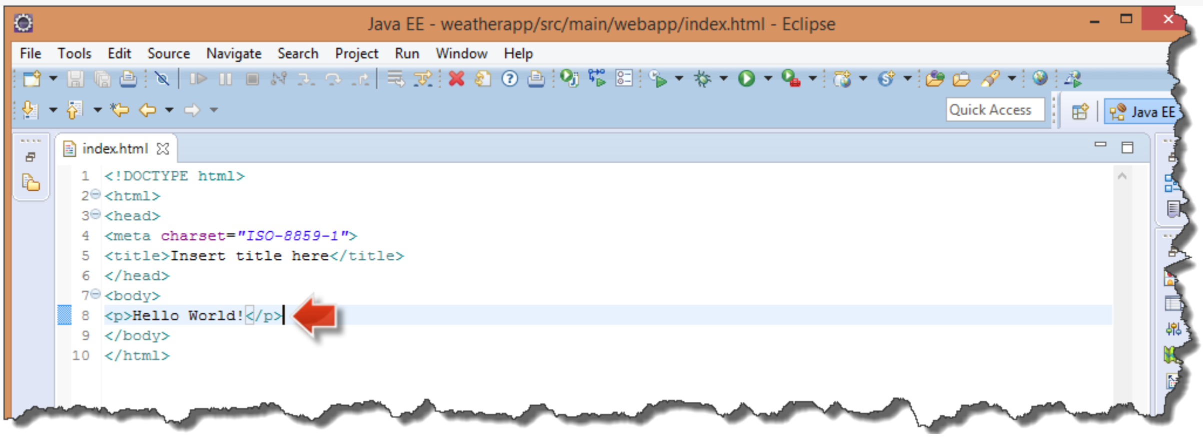


Step 2: Add body

Add the following line of code in between the opening and the closing tag:

Copy code

```
<p>Hello World!</p>
```


Java EE - weatherapp/src/main/webapp/index.html - Eclipse
File Tools Edit Source Navigate Search Project Run Window Help
Quick Access
Java EE
index.html

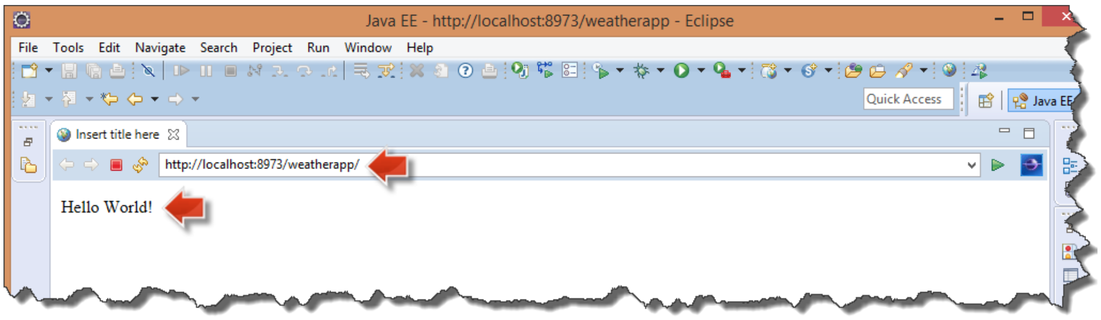
```
<!DOCTYPE html>
<html>
<head>
<meta charset="ISO-8859-1">
<title>Insert title here</title>
</head>
<body>
<p>Hello World!</p>
</body>
</html>
```

Step 3: Save and publish locally

Save your changes and publish the update project to the local server by selecting the respective context menu on the server in the Server view.

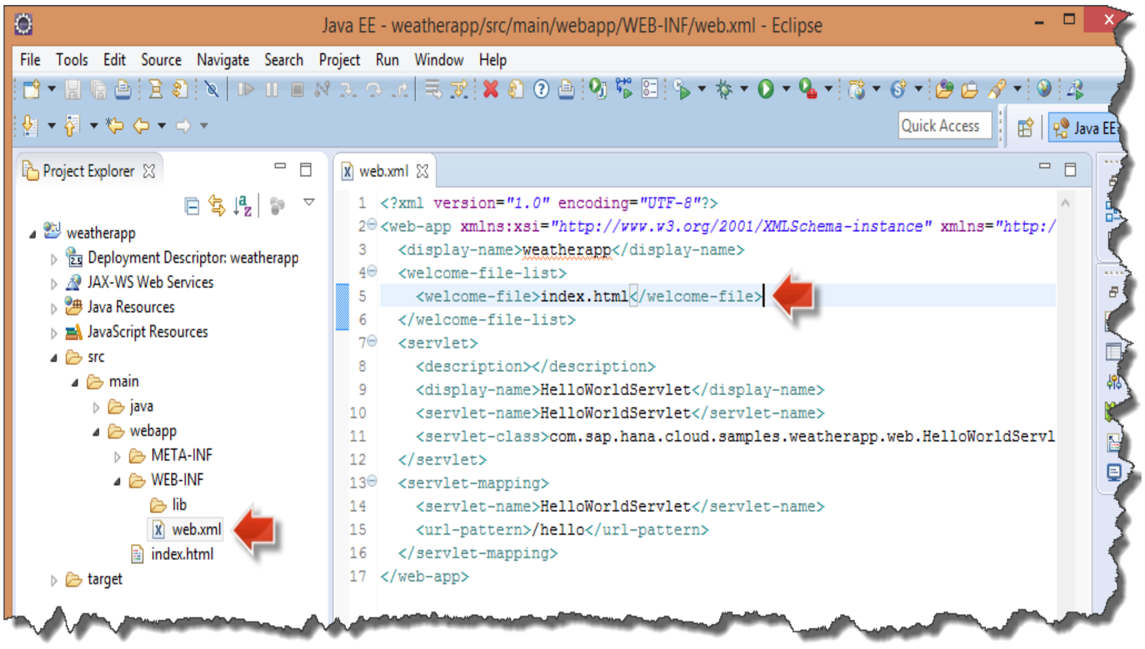
Step 4: View app

Once the publishing is done you can go back to the internal browser window and remove the `/hello` part of the URL and hit the **Enter** key. Now, you should see the same "Hello World!" message for the URL http://localhost:8080/weatherapp/ as well. The reason for this is the `<welcome-file-list>` in the `web.xml` (located in the `WEB-INF` directory underneath the `webapp` folder).


Java EE - http://localhost:8973/weatherapp - Eclipse
File Tools Edit Navigate Search Project Run Window Help
Quick Access
Java EE
Insert title here
http://localhost:8973/weatherapp/
Hello World!

Step 5: Clean up entries

For security reasons and for the sake of housekeeping you should remove all entries except for the `index.html` entry from the `web.xml` file.



```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http:/
  <display-name>weatherapp</display-name>
  <welcome-file-list>
    <welcome-file>index.html</welcome-file>
  </welcome-file-list>
  <servlet>
    <description></description>
    <display-name>HelloWorldServlet</display-name>
    <servlet-name>HelloWorldServlet</servlet-name>
    <servlet-class>com.sap.hana.cloud.samples.weatherapp.web.HelloWorldServl
  </servlet>
  <servlet-mapping>
    <servlet-name>HelloWorldServlet</servlet-name>
    <url-pattern>/hello</url-pattern>
  </servlet-mapping>
</web-app>
```

Adding authentication and authorization to your Java app

You will learn

In this tutorial you will learn how to add authentication and authorization to your Java app.

Details

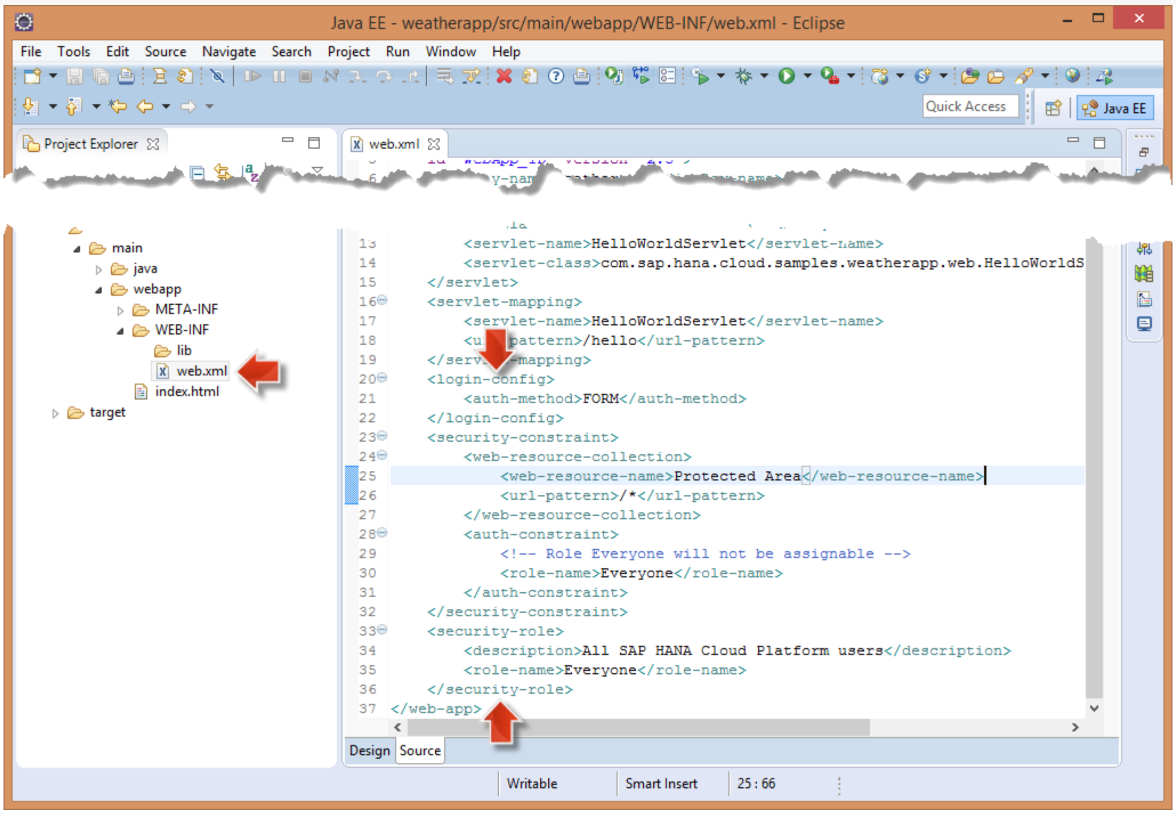
Please note that SAP Cloud Platform adheres to Java standards to manage authentication and authorization.

Step 1: Apply security settings

In order to activate authentication and establish authorization we have to apply the respective security settings in the `web.xml` configuration file. The full `web.xml` contents are below:

```
<?xml version="1.0" encoding="UTF-8"?>

<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd" id="WebApp_ID" version="2.5">

<display-name>cloud-weatherapp</display-name>

<welcome-file-list>

<welcome-file>index.html</welcome-file>

</welcome-file-list>

<servlet>

<display-name>HelloWorldServlet</display-name>

<servlet-name>HelloWorldServlet</servlet-name>

<servlet-class>

  com.sap.hana.cloud.samples.weatherapp.web.HelloWorldServlet

</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>HelloWorldServlet</servlet-name>

<url-pattern>/hello</url-pattern>

</servlet-mapping>

<login-config>
```

```
<auth-method>FORM</auth-method>

</login-config>

<security-constraint>

<web-resource-collection>

  <web-resource-name>Protected Area</web-resource-name>

  <url-pattern>/*</url-pattern>

</web-resource-collection>

<auth-constraint>

  <!-- Role Everyone will not be assignable -->

  <role-name>Everyone</role-name>

</auth-constraint>

</security-constraint>

<security-role>

<description>All SAP Cloud Platform users</description>

<role-name>Everyone</role-name>

</security-role>

</web-app>
```



```
        <servlet-name>HelloWorldServlet</servlet-name>
        <servlet-class>com.sap.hana.cloud.samples.weatherapp.web.HelloWorldS
    </servlet>
    <servlet-mapping>
        <servlet-name>HelloWorldServlet</servlet-name>
        <url-pattern>/hello</url-pattern>
    </servlet-mapping>
    <login-config>
        <auth-method>FORM</auth-method>
    </login-config>
    <security-constraint>
        <web-resource-collection>
            <web-resource-name>Protected Area</web-resource-name>
            <url-pattern>/*</url-pattern>
        </web-resource-collection>
        <auth-constraint>
            <!-- Role Everyone will not be assignable -->
            <role-name>Everyone</role-name>
        </auth-constraint>
    </security-constraint>
    <security-role>
        <description>All SAP HANA Cloud Platform users</description>
        <role-name>Everyone</role-name>
    </security-role>
</web-app>
```

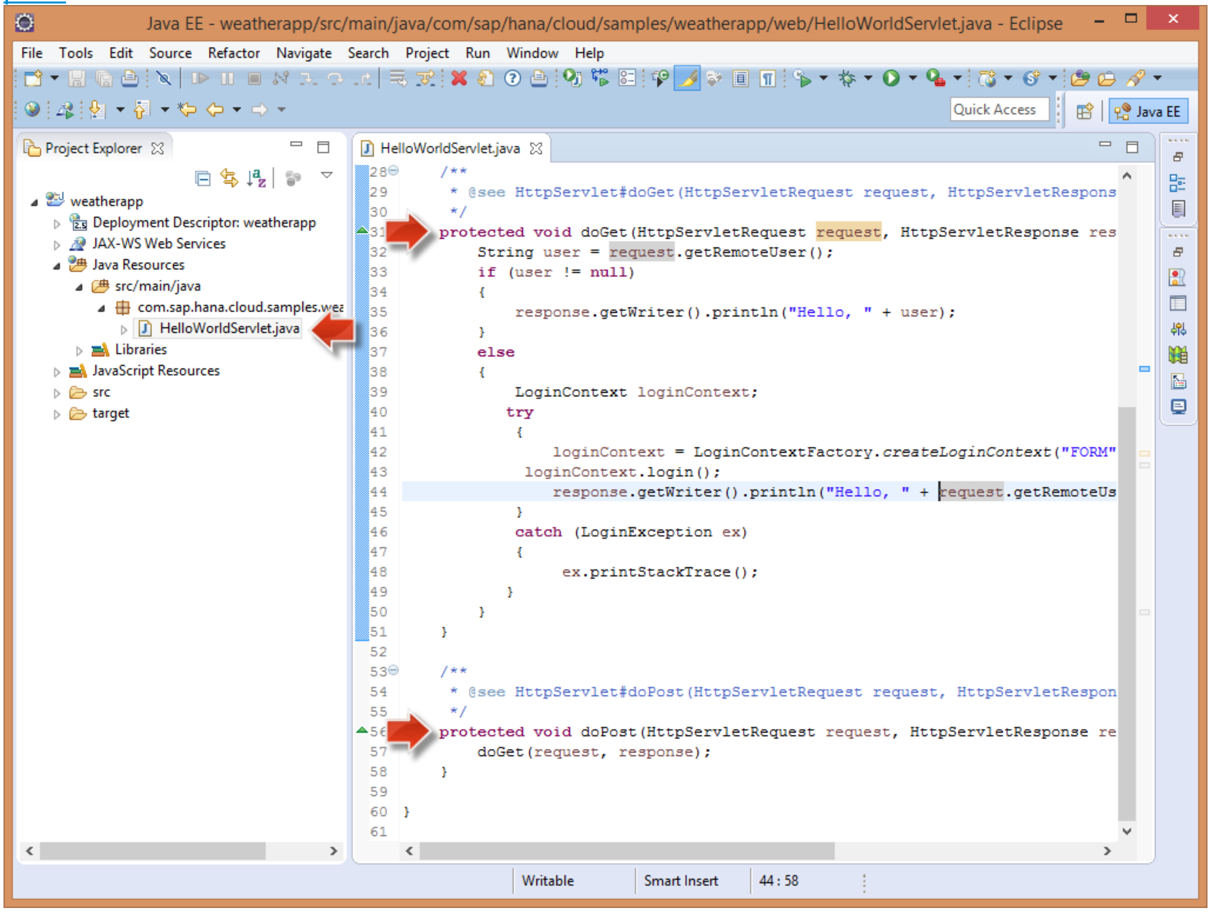
Step 2: Edit servlet

After successful authentication the application can access users' principal information using standard servlet APIs. To illustrate that, make the following changes to the `HelloWorldServlet`:

```
/**

* @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)

*/protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException

{

    String user = request.getRemoteUser();

    if (user != null)

    {

        response.getWriter().println("Hello, " + user);

    }

    else

    {

        LoginContext loginContext;

             try

        {

             loginContext = LoginContextFactory.createLoginContext("FORM");

                                 loginContext.login();

             response.getWriter().println("Hello, " +  request.getRemoteUser());

        }

        catch (LoginException ex)

        {

             ex.printStackTrace();

             }

    }

}

/**

* @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse
response)
```

```
*/

protected void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException

{

        doGet(request, response);

}
```

Note: The reason we also had to implement the `doPost()` method is related to specifics of the SAML 2.0 authentication process flow. For more information please refer to the respective parts of the SAP Cloud Platform online documentation.


Java EE - weatherapp/src/main/java/com/sap/hana/cloud/samples/weatherapp/web/HelloWorldServlet.java - Eclipse

```
    /**
     * @see HttpServlet#doGet(HttpServletRequest request, HttpServletRespons
     */
    protected void doGet(HttpServletRequest request, HttpServletResponse res
        String user = request.getRemoteUser();
        if (user != null)
        {
            response.getWriter().println("Hello, " + user);
        }
        else
        {
            LoginContext loginContext;
          try
           {
               loginContext = LoginContextFactory.createLoginContext("FORM"
            loginContext.login();
               response.getWriter().println("Hello, " + request.getRemoteUs
           }
           catch (LoginException ex)
           {
                ex.printStackTrace();
          }
        }
    }

    /**
     * @see HttpServlet#doPost(HttpServletRequest request, HttpServletRespon
     */
    protected void doPost(HttpServletRequest request, HttpServletResponse re
        doGet(request, response);
    }

}
```

Step 3: Organize imports

To remove the syntax errors, you need to organize import statements via the respective context menu **Source > Organize imports** of the main code editor window. Save your changes.

Step 4: Deploy the app

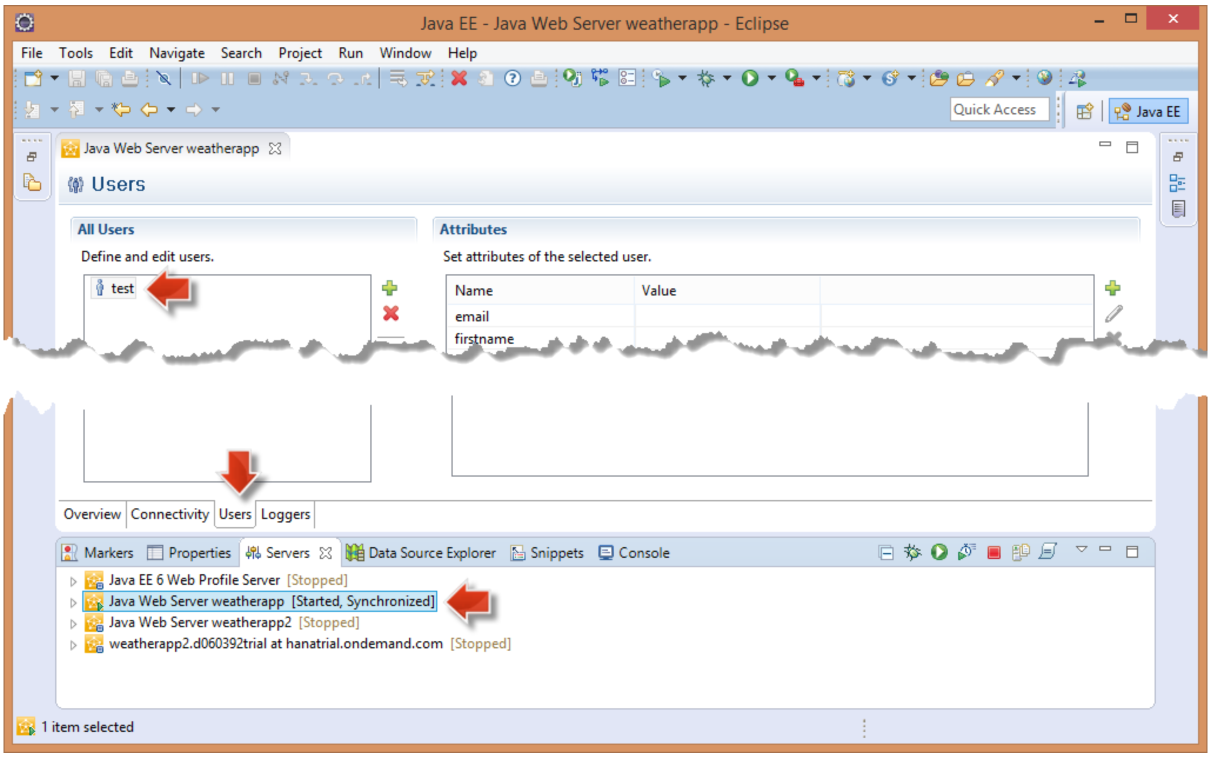
Deploy/publish the updated application (you should know the drill by now).

Step 5: Add test user

Since we are working with a local server so far we need to provide a local user repository to authenticate against. For this purpose, double-click on the local server node in the **Servers** view to open the configuration window.

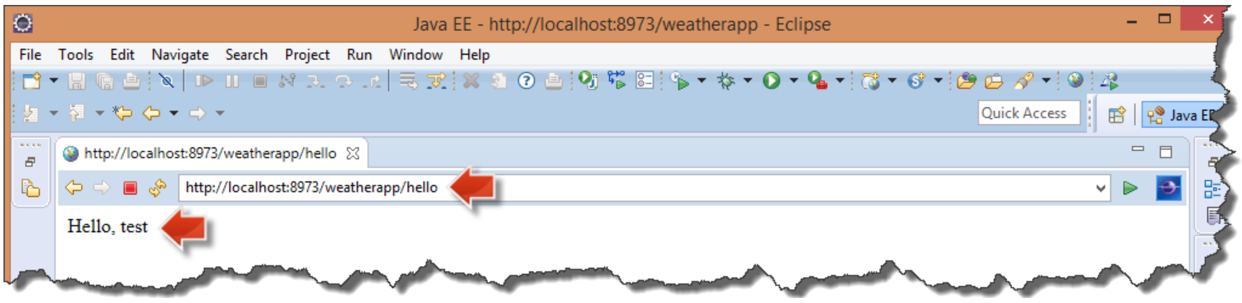
At the bottom of that window there are four tabs: **Overview**, **Connectivity**, **Users** and **Loggers**.

Within the Users tab you can manage local users. Let's create a simple test user with the user id "test" and a password of your choice. Save your changes.


Java EE - Java Web Server weatherapp - Eclipse
File Tools Edit Navigate Search Project Run Window Help
Quick Access
Java EE
Java Web Server weatherapp
Users
All Users
Define and edit users.
test
Attributes
Set attributes of the selected user.
Name
Value
email
firstname
Overview
Connectivity
Users
Loggers
Markers
Properties
Servers
Data Source Explorer
Snippets
Console
Java EE 6 Web Profile Server [Stopped]
Java Web Server weatherapp [Started, Synchronized]
Java Web Server weatherapp2 [Stopped]
weatherapp2.d060392trial at hanatrial.ondemand.com [Stopped]
1 item selected

Step 6: Enter credentials

Now, when you navigate to the `HelloWorldServlet` with the URL http://localhost:8080/weatherapp/hello you'll first be prompted to enter your user credentials before you are forwarded to the requested servlet. If the authentication was successful you should now see a personalized welcome message instead of the dull "Hello World!" we saw earlier.


Java EE - http://localhost:8973/weatherapp - Eclipse
File Tools Edit Navigate Search Project Run Window Help
Quick Access
Java EE
http://localhost:8973/weatherapp/hello
http://localhost:8973/weatherapp/hello

Hello, test

Convert your app to a Maven-based project

You will learn

In this tutorial you will learn how to convert your basic Java app into a Maven-based project which is useful to prepare for the complexities of dependency management in larger apps.

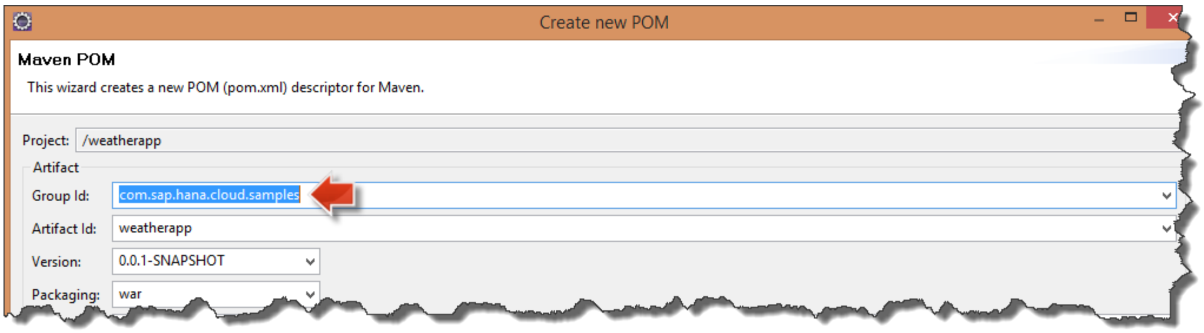
Details

Step 1: Open configuration menu

Select the `weatherapp` node in the project explorer and open the context menu. Select the **Configure > Convert to Maven Project** option.

Step 2: Change group ID

Change the group ID from `weatherapp` to `com.sap.hana.cloud.samples` and click on **Finish**. The most noticeable change will be that a `pom.xml` file will be created in the root folder of the`weatherapp` project.



This wizard creates a new POM (pom.xml) descriptor for Maven.

Project: /weatherapp

Artifact

Group Id: com.sap.hana.cloud.samples

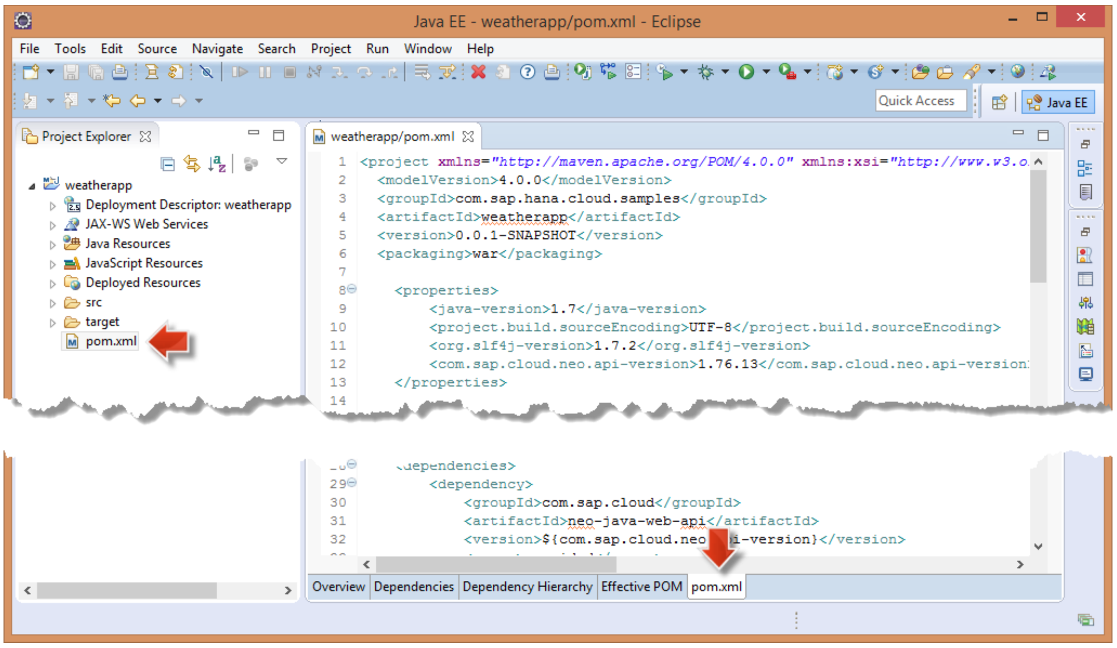
Artifact Id: weatherapp

Version: 0.0.1-SNAPSHOT

Packaging: war

Step 3: Replace existing content

Copy the entire content from the `pom.xml` file from GitHub and use it to replace the existing content in your project.



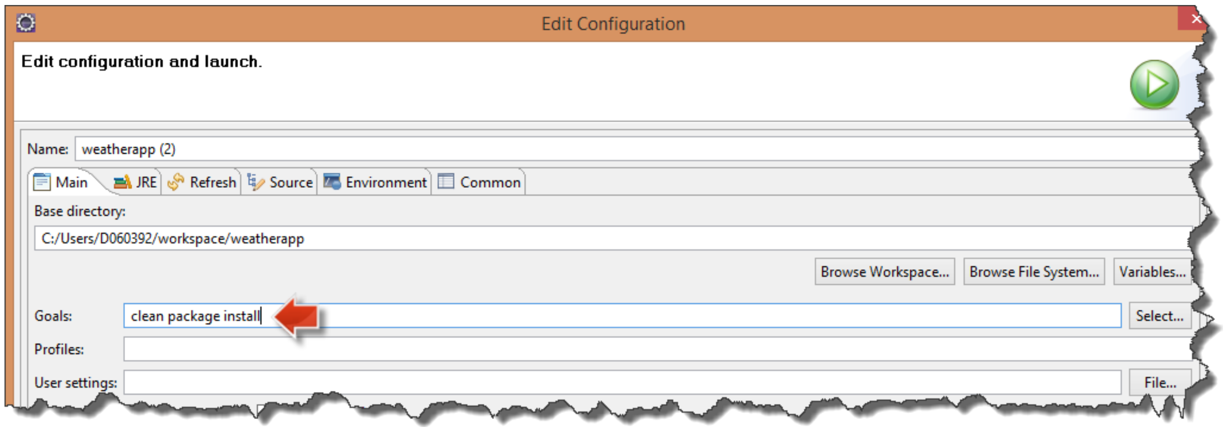
```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.o
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.sap.hana.cloud.samples</groupId>
  <artifactId>weatherapp</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>war</packaging>

    <properties>
        <java-version>1.7</java-version>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <org.slf4j-version>1.7.2</org.slf4j-version>
        <com.sap.cloud.neo.api-version>1.76.13</com.sap.cloud.neo.api-version
    </properties>

    <dependencies>
        <dependency>
            <groupId>com.sap.cloud</groupId>
            <artifactId>neo-java-web-api</artifactId>
            <version>${com.sap.cloud.neo i-version}</version>
```

Step 4: Update project

Open the context menu on the `weatherapp` project in the Project Explorer and select the menu entry **Maven > Update Project…** (The first time you do this can take a bit longer, as Maven will download all the required build plugins and dependencies specified in the `pom.xml` file). Make sure your project is selected then click on **OK**.

Step 5: Build

Select the **Run as > Maven build…** context menu of the `weatherapp` project and enter the following in the **Goals** field: `clean package install`.



Edit configuration and launch.

Name: weatherapp (2)

Base directory: C:/Users/D060392/workspace/weatherapp

Goals: clean package install

Profiles:

User settings:

Step 6: Run the project

Click on **Run** and the project should build successfully.

Adding RESTful services to your app

You will learn

In this tutorial you will learn how to expose RESTful services using a library called Apache CXF, which is one of the most often used implementations of the JAX-RS standard.

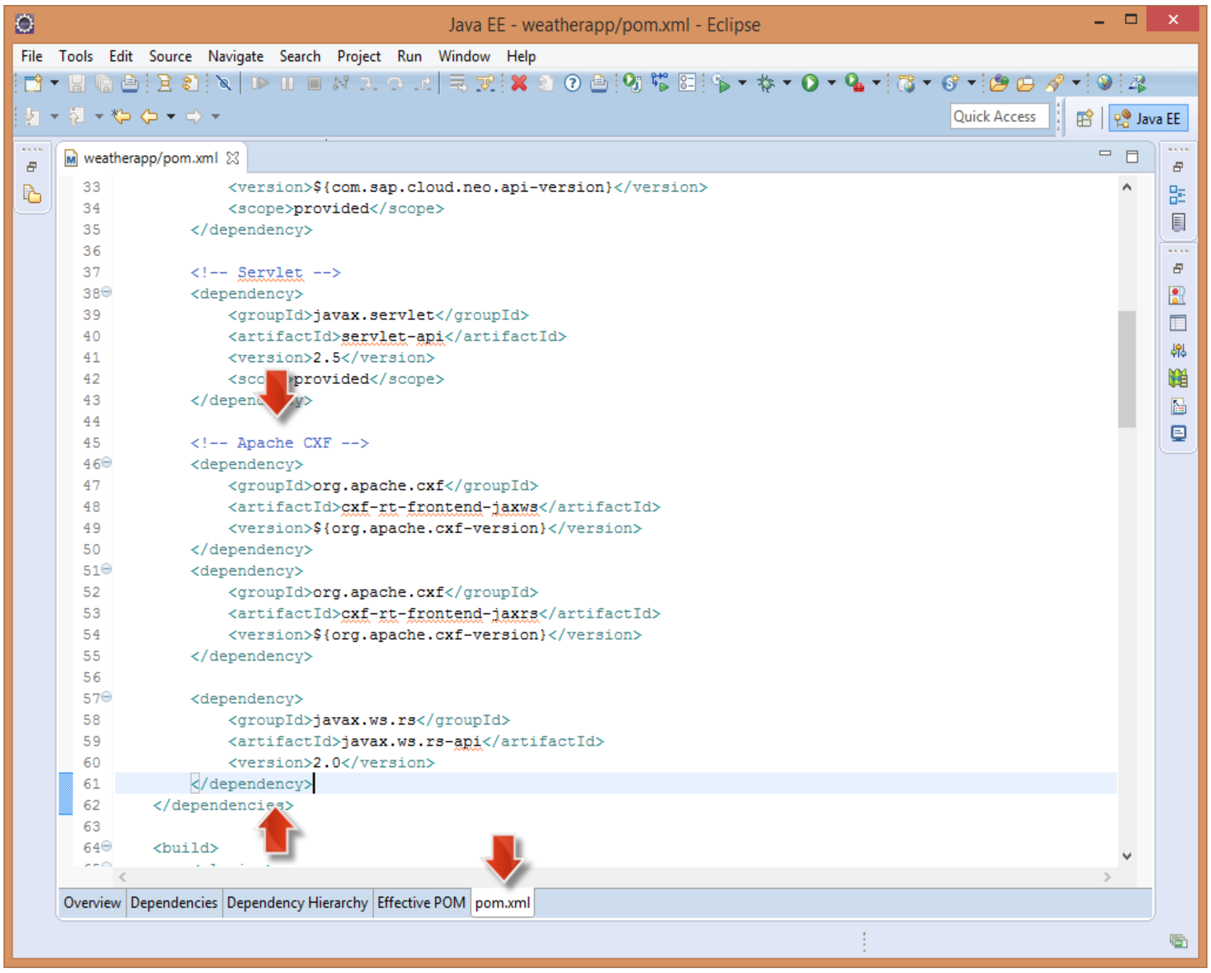
Details

Step 1: Add dependency references

First, we need to add the dependency references to Apache CXF in the `pom.xml` file. Insert the XML snippet below just below the **Servlet** dependency section.

```
<!-- Apache CXF -->

<dependency>

    <groupId>org.apache.cxf</groupId>

    <artifactId>cxf-rt-frontend-jaxws</artifactId>

    <version>${org.apache.cxf-version}</version>

</dependency>

<dependency>
```

```
    <groupId>org.apache.cxf</groupId>

    <artifactId>cxf-rt-frontend-jaxrs</artifactId>

    <version>${org.apache.cxf-version}</version>

</dependency>

<dependency>

    <groupId>javax.ws.rs</groupId>

    <artifactId>javax.ws.rs-api</artifactId>

    <version>2.0</version>

</dependency>
```

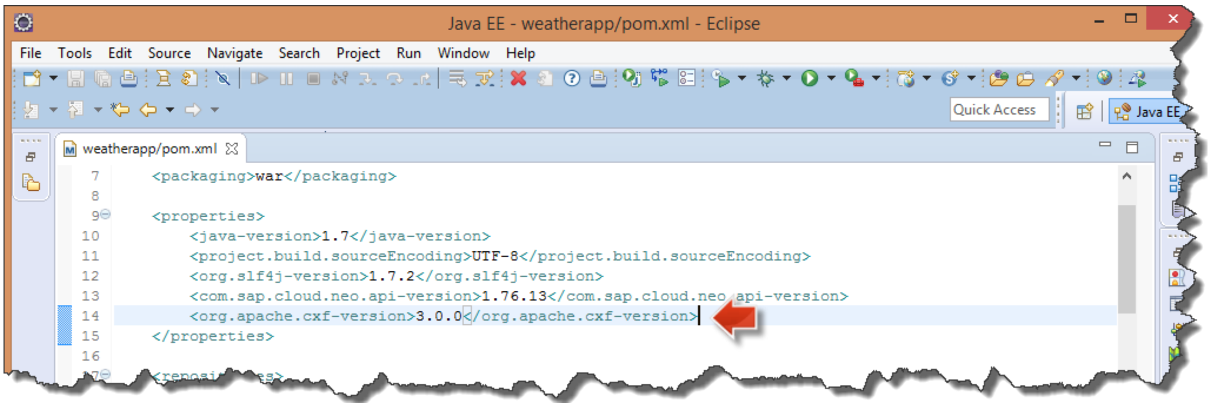

Java EE - weatherapp/pom.xml - Eclipse
File Tools Edit Source Navigate Search Project Run Window Help
Quick Access
Java EE
weatherapp/pom.xml
Overview
Dependencies
Dependency Hierarchy
Effective POM
pom.xml

```
            <version>${com.sap.cloud.neo.api-version}</version>
            <scope>provided</scope>
        </dependency>

        <!-- Servlet -->
        <dependency>
            <groupId>javax.servlet</groupId>
            <artifactId>servlet-api</artifactId>
            <version>2.5</version>
            <scope>provided</scope>
        </dependency>

        <!-- Apache CXF -->
        <dependency>
            <groupId>org.apache.cxf</groupId>
            <artifactId>cxf-rt-frontend-jaxws</artifactId>
            <version>${org.apache.cxf-version}</version>
        </dependency>
        <dependency>
            <groupId>org.apache.cxf</groupId>
            <artifactId>cxf-rt-frontend-jaxrs</artifactId>
            <version>${org.apache.cxf-version}</version>
        </dependency>

        <dependency>
            <groupId>javax.ws.rs</groupId>
            <artifactId>javax.ws.rs-api</artifactId>
            <version>2.0</version>
        </dependency>
    </dependencies>

    <build>
```

Step 2: Specify CXF version property

We also need to specify the corresponding CXF version property at the end of the `<properties>` tag in `pom.xml`. See the image below for where to insert this snippet.

```
<org.apache.cxf-version>3.0.0</org.apache.cxf-version>
```


Java EE - weatherapp/pom.xml - Eclipse
File Tools Edit Source Navigate Search Project Run Window Help
Quick Access
Java EE
weatherapp/pom.xml

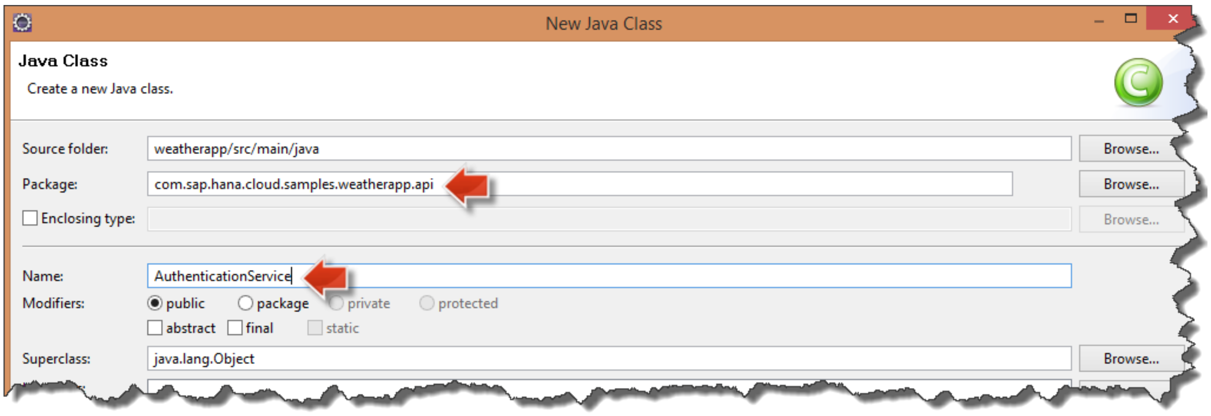
```
    <packaging>war</packaging>

    <properties>
        <java-version>1.7</java-version>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <org.slf4j-version>1.7.2</org.slf4j-version>
        <com.sap.cloud.neo.api-version>1.76.13</com.sap.cloud.neo.api-version>
        <org.apache.cxf-version>3.0.0</org.apache.cxf-version>
    </properties>
```

Step 3: Create a new class

Next, create a new Class via the context menu entry **New > Class** of the `weatherapp` node in the Project Explorer. Enter the following information:

- **Package name:** `com.sap.hana.cloud.samples.weatherapp.api`
- **Classname:** `AuthenticationService`

Click on **Finish**.



Source folder: weatherapp/src/main/java

Package: com.sap.hana.cloud.samples.weatherapp.api

Enclosing type:

Name: AuthenticationService

Modifiers: [✓ public] package private protected

abstract final static

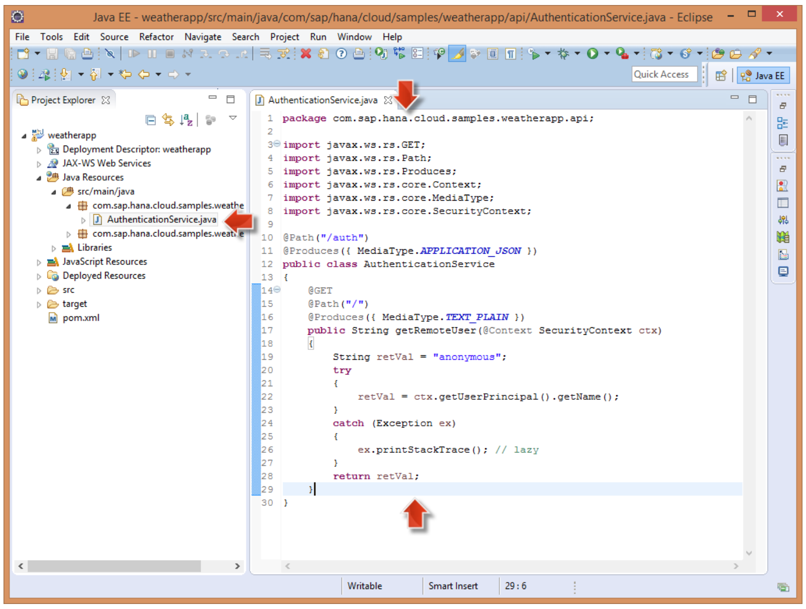
Superclass: java.lang.Object

Step 4: Replace code

Replace the contents of `AuthenticationService.java` with the following and save your changes.

```
package com.sap.hana.cloud.samples.weatherapp.api;


import javax.ws.rs.GET;

import javax.ws.rs.Path;

import javax.ws.rs.Produces;

import javax.ws.rs.core.Context;

import javax.ws.rs.core.MediaType;

import javax.ws.rs.core.SecurityContext;
```

```java
@Path("/auth")
@Produces({ MediaType.APPLICATION_JSON })
public class AuthenticationService
{
        @GET
        @Path("/")
        @Produces({ MediaType.TEXT_PLAIN })
        public String getRemoteUser(@Context SecurityContext ctx)
        {
                String retVal = "anonymous";
                try
                {
                        retVal = ctx.getUserPrincipal().getName();
                }
                catch (Exception ex)
                {       ex.printStackTrace(); // lazy
                }
                return retVal;
        }
}
```

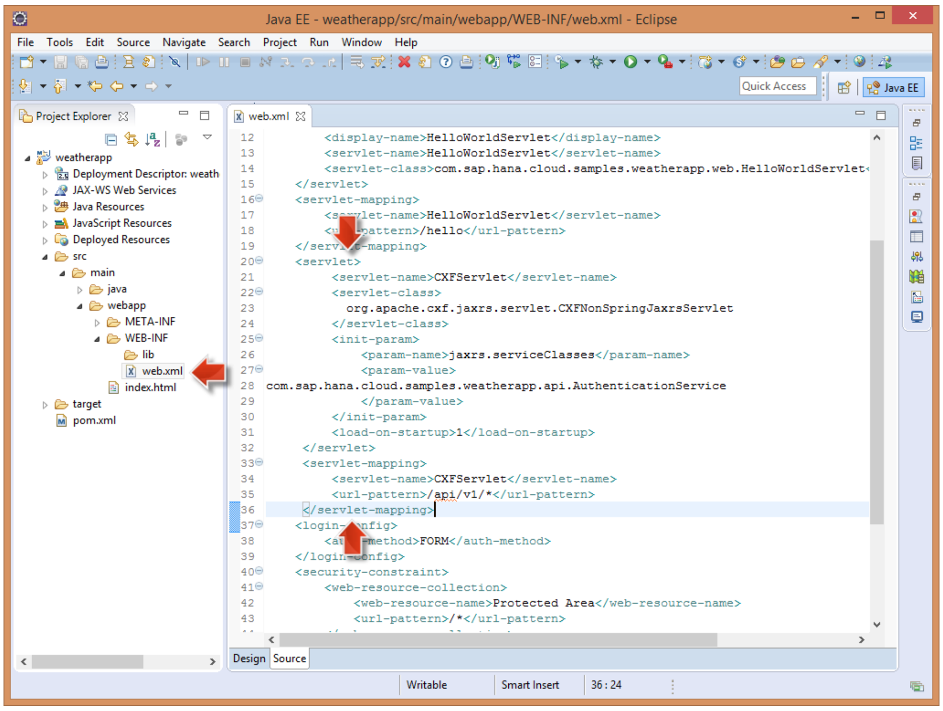

Java EE - weatherapp/src/main/java/com/sap/hana/cloud/samples/weatherapp/api/AuthenticationService.java - Eclipse
File Tools Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
Java EE
Project Explorer
weatherapp
Deployment Descriptor: weatherapp
JAX-WS Web Services
Java Resources
src/main/java
com.sap.hana.cloud.samples.weathe
AuthenticationService.java
com.sap.hana.cloud.samples.weat e
Libraries
JavaScript Resources
Deployed Resources
src
target
pom.xml
AuthenticationService.java
Writable
Smart Insert
29 : 6

```java
package com.sap.hana.cloud.samples.weatherapp.api;

import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.core.Context;
import javax.ws.rs.core.MediaType;
import javax.ws.rs.core.SecurityContext;

@Path("/auth")
@Produces({ MediaType.APPLICATION_JSON })
public class AuthenticationService
{
    @GET
    @Path("/")
    @Produces({ MediaType.TEXT_PLAIN })
    public String getRemoteUser(@Context SecurityContext ctx)
    {
        String retVal = "anonymous";
        try
        {
            retVal = ctx.getUserPrincipal().getName();
        }
        catch (Exception ex)
        {
            ex.printStackTrace(); // lazy
        }
        return retVal;
    }
}
```

Step 5: Edit configuration file

Open the `web.xml` configuration file and copy and paste the following lines of code in between the closing `</servlet-mapping>` and the opening `<login-config>` tags:

```
<servlet>

        <servlet-name>CXFServlet</servlet-name>

        <servlet-class>

                org.apache.cxf.jaxrs.servlet.CXFNonSpringJaxrsServlet

        </servlet-class>

        <init-param>

                <param-name>jaxrs.serviceClasses</param-name>

                <param-value>
com.sap.hana.cloud.samples.weatherapp.api.AuthenticationService</param-value>

        </init-param>

        <load-on-startup>1</load-on-startup>

</servlet>

<servlet-mapping>

        <servlet-name>CXFServlet</servlet-name>

        <url-pattern>/api/v1/*</url-pattern>

</servlet-mapping>
```



```
        <display-name>HelloWorldServlet</display-name>
        <servlet-name>HelloWorldServlet</servlet-name>
        <servlet-class>com.sap.hana.cloud.samples.weatherapp.web.HelloWorldServlet<
    </servlet>
    <servlet-mapping>
        <servlet-name>HelloWorldServlet</servlet-name>
        <url-pattern>/hello</url-pattern>
    </servlet-mapping>
    <servlet>
         <servlet-name>CXFServlet</servlet-name>
         <servlet-class>
           org.apache.cxf.jaxrs.servlet.CXFNonSpringJaxrsServlet
         </servlet-class>
         <init-param>
             <param-name>jaxrs.serviceClasses</param-name>
             <param-value>
com.sap.hana.cloud.samples.weatherapp.api.AuthenticationService
             </param-value>
         </init-param>
         <load-on-startup>1</load-on-startup>
     </servlet>
     <servlet-mapping>
         <servlet-name>CXFServlet</servlet-name>
         <url-pattern>/api/v1/*</url-pattern>
     </servlet-mapping>
    <login-config>
        <auth-method>FORM</auth-method>
    </login-config>
    <security-constraint>
        <web-resource-collection>
            <web-resource-name>Protected Area</web-resource-name>
            <url-pattern>/*</url-pattern>
```

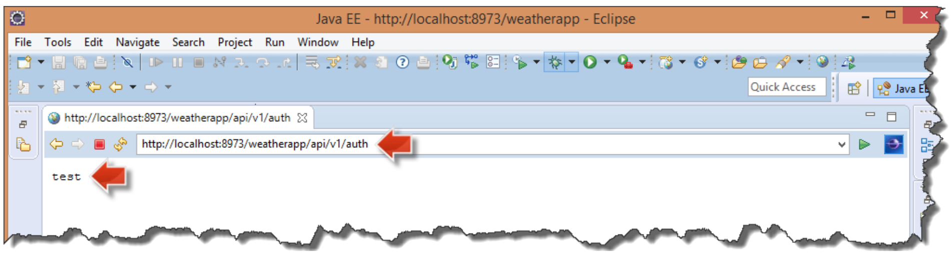
Step 6: Save and publish changes

With this, we have registered (Apache) CXF as a Servlet that listens to incoming requests using the URL-pattern: `/api/v1/*`. Furthermore, we registered our `AuthenticationService` class as one of the RESTful services. During start-up, CXF will introspect the class and use the provided JAX-RS annotations to properly configure our service.

Save your changes and publish/deploy your application.

Step 7: View the app

Navigate to the following URL: http://localhost:8080/weatherapp/api/v1/auth. After successful authentication you should see your username.


Java EE - http://localhost:8973/weatherapp - Eclipse

http://localhost:8973/weatherapp/api/v1/auth

```
test
```

Adding persistence to your app using JPA (Java Persistence API)

You will learn

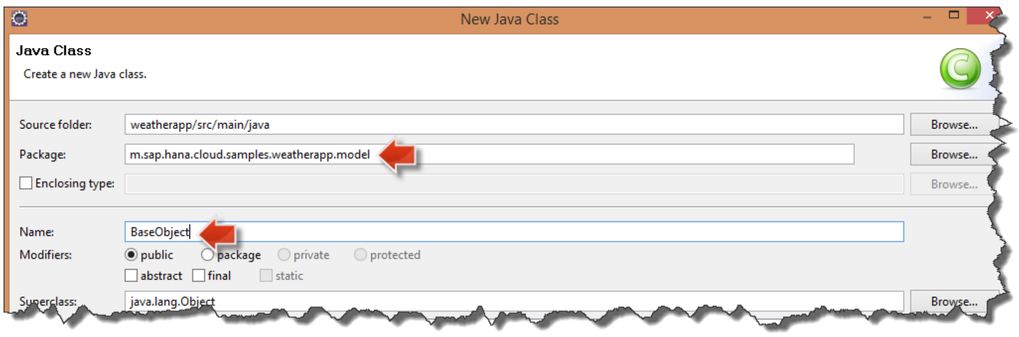
In this tutorial you will implement a simple domain model and implement the corresponding persistence layer using JPA (Java Persistence API). The domain model only features a single class: `FavoriteCity`, so that we can bookmark or favorite our favorite cities.

Details

Step 1: Create a base class

Let's create a base class for our domain model first. That's usually considered best practices as it provides us with a central place to add common functionality to be shared across all domain model objects on later. For that purpose, select New > Class from the context menu entry on the `weatherapp` project and provide the following details:

- **Package name:** `com.sap.hana.cloud.samples.weatherapp.model`
- **Classname:** `BaseObject`


New Java Class

Java Class
Create a new Java class.

Source folder:	weatherapp/src/main/java
Package:	m.sap.hana.cloud.samples.weatherapp.model
Enclosing type:	
Name:	BaseObject
Modifiers:	[✓ public] package private protected; abstract final static
Superclass:	java.lang.Object

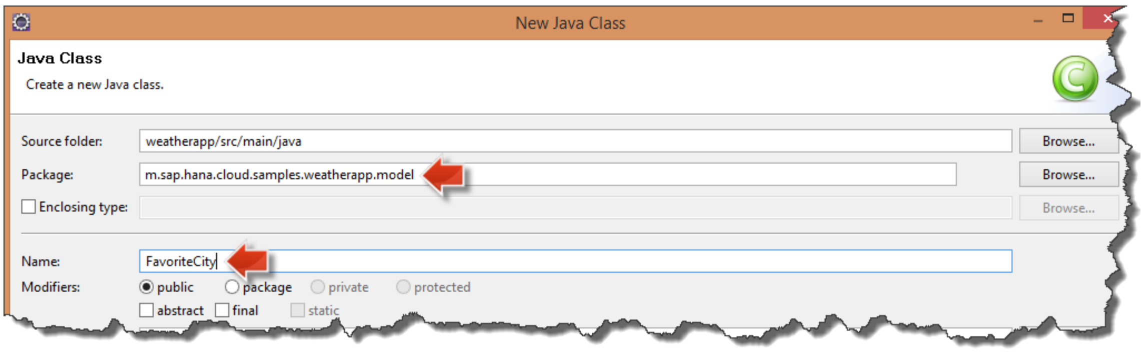
Step 2: Replace code

Replace the contents of the `BaseObject.java` file with this code from GitHub and save your changes.

Step 3: Create another Java class

Next, create another Java class (`FavoriteCity.java`) using the same procedure:

- **Package name:** `com.sap.hana.cloud.samples.weatherapp.model`
- **Classname:** `FavoriteCity`



New Java Class

Java Class

Create a new Java class.

Source folder: weatherapp/src/main/java

Package: m.sap.hana.cloud.samples.weatherapp.model

☐ Enclosing type:

Name: FavoriteCity

Modifiers: [✓ public] package private protected

☐ abstract ☐ final ☐ static

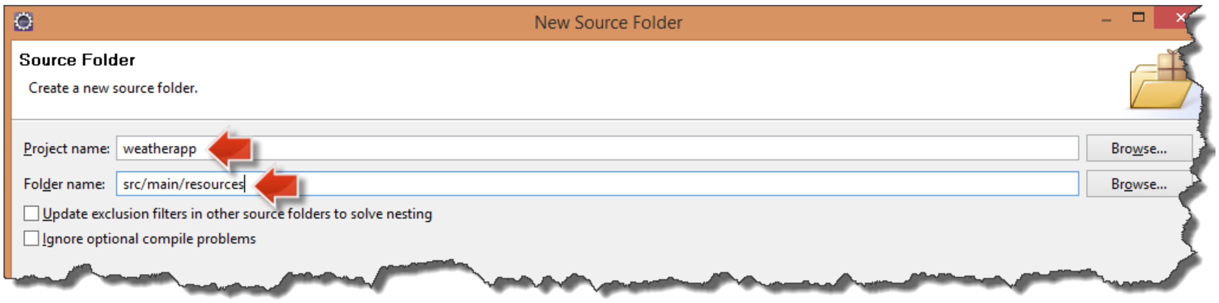
Step 4: Replace the code

Replace the contents of the `FavoriteCity.java` file with this code from GitHub and save your changes.

Step 5: Create a configuration file

Next, we need to create a configuration file for our persistence layer. By Maven conventions, these non-source code artifacts should be located in a separate source code folder called: `src/main/resources`. Hence, let's create that source folder via the corresponding context menu entry on the **Java Resources** node in the Project Explorer: **New > Source Folder**. Provide the following information:

- **Project name:** `weatherapp`
- **Folder name:** `src/main/resources`



New Source Folder

Source Folder

Create a new source folder.

Project name: weatherapp

Folder name: src/main/resources

☐ Update exclusion filters in other source folders to solve nesting

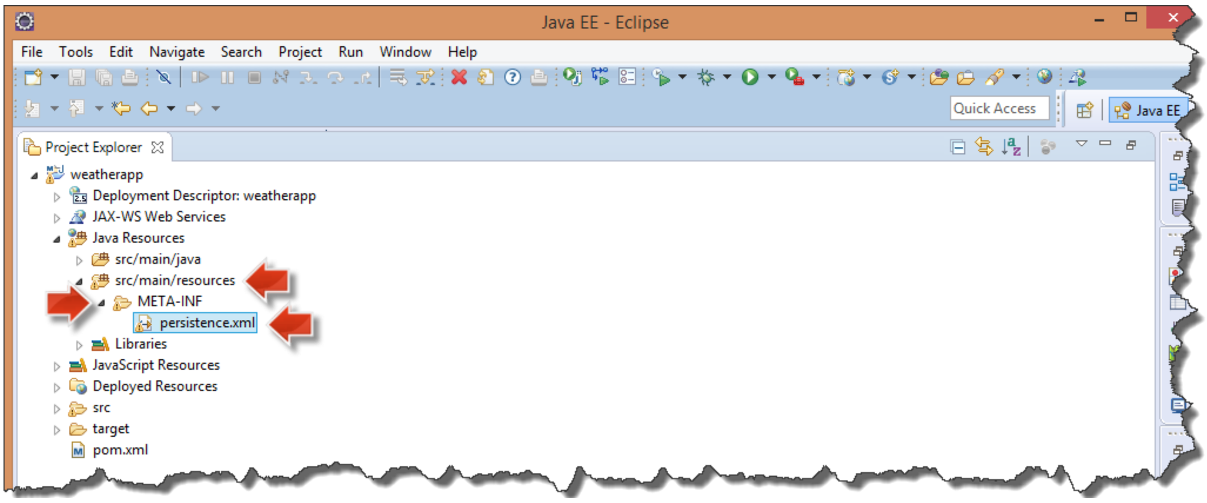
☐ Ignore optional compile problems

Step 6: Create new folder

Open the context menu of this newly created source folder and choose the **New > Other** option and then select the **Folder** option. Name the new folder `META-INF` (all capitals!) and click on **Finish**.

Step 7: Create persistence XML file

Open the context menu of the newly created `META-INF` folder and select **New > File**. Name the new file `persistence.xml` and click on **Finish**.



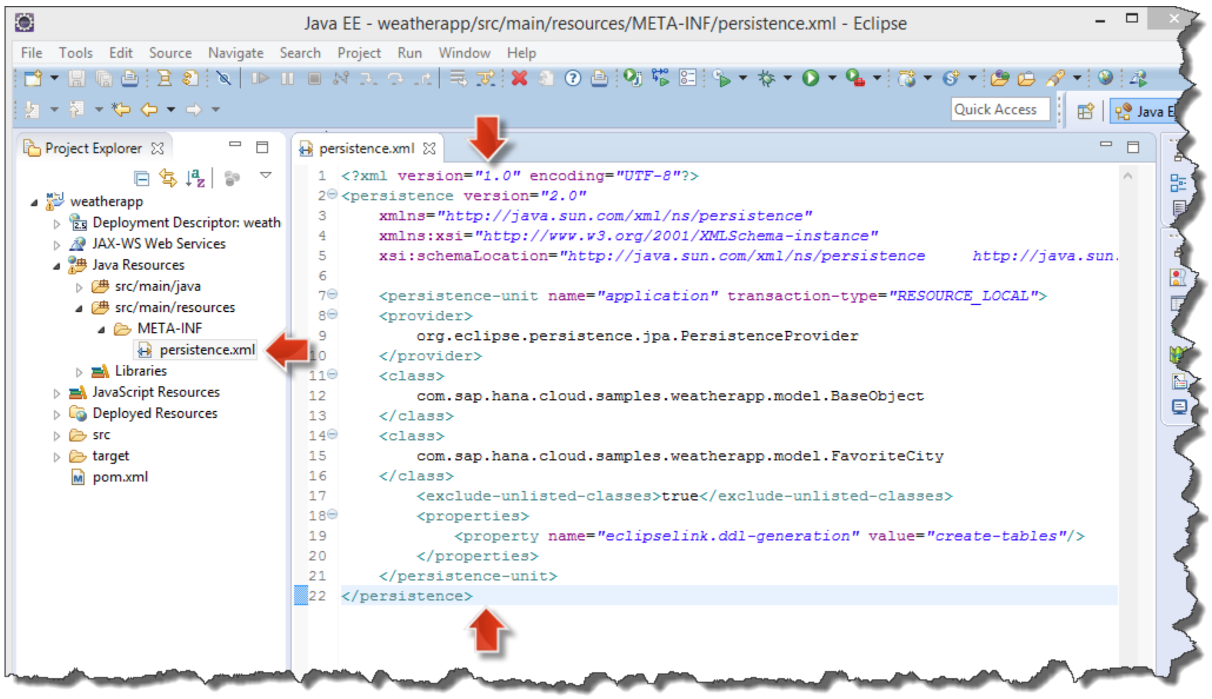
weatherapp
Deployment Descriptor: weatherapp
JAX-WS Web Services
Java Resources
src/main/java
src/main/resources
META-INF
persistence.xml
Libraries
JavaScript Resources
Deployed Resources
src
target
pom.xml

Step 8: Edit the file

Copy and paste the following XML content into the `persistence.xml` file:

```
<?xml version="1.0" encoding="UTF-8"?>

<persistence version="2.0"

xmlns="http://java.sun.com/xml/ns/persistence"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd">



<persistence-unit name="application" transaction-type="RESOURCE_LOCAL">

<provider>

    org.eclipse.persistence.jpa.PersistenceProvider

</provider>

<class>

    com.sap.hana.cloud.samples.weatherapp.model.BaseObject

</class>

<class>

    com.sap.hana.cloud.samples.weatherapp.model.FavoriteCity

</class>

    <exclude-unlisted-classes>true</exclude-unlisted-classes>

    <properties>

      <property name="eclipselink.ddl-generation" value="create-tables"/>

  </properties>
```

```
</persistence-unit>

</persistence>
```



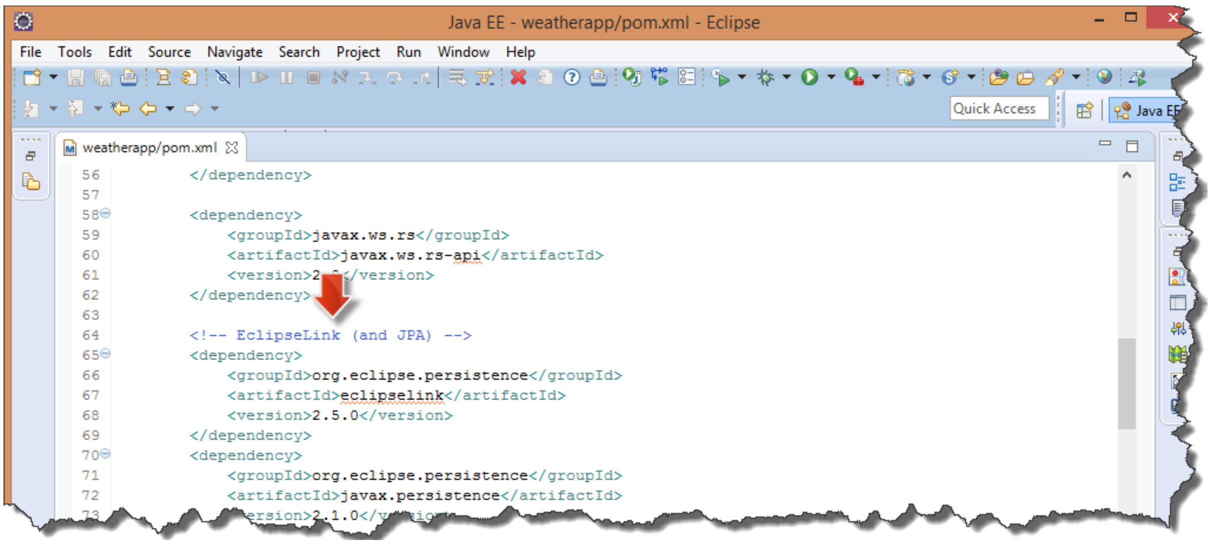
```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.0"
    xmlns="http://java.sun.com/xml/ns/persistence"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/persistence     http://java.sun.

    <persistence-unit name="application" transaction-type="RESOURCE_LOCAL">
    <provider>
        org.eclipse.persistence.jpa.PersistenceProvider
    </provider>
    <class>
        com.sap.hana.cloud.samples.weatherapp.model.BaseObject
    </class>
    <class>
        com.sap.hana.cloud.samples.weatherapp.model.FavoriteCity
    </class>
        <exclude-unlisted-classes>true</exclude-unlisted-classes>
        <properties>
            <property name="eclipselink.ddl-generation" value="create-tables"/>
        </properties>
    </persistence-unit>
</persistence>
```

Step 9: Add dependencies

Next, we need to add some more dependencies to our `pom.xml` file. In this case, the most important dependency is on EclipseLink (our JPA implementation of choice). However, we also need to declare dependencies for the Derby DB and Jackson (a serialization framework needed to convert data into JSON and vice versa.)

```
<!-- EclipseLink (and JPA) -->

<dependency>

  <groupId>org.eclipse.persistence</groupId>

  <artifactId>eclipselink</artifactId>

  <version>2.5.0</version>

  </dependency>

<dependency>

   <groupId>org.eclipse.persistence</groupId>

   <artifactId>javax.persistence</artifactId>

   <version>2.1.0</version>

 </dependency>

 <!-- Derby -->

<dependency>
```

```xml
<groupId>org.apache.derby</groupId>

  <artifactId>derbyclient</artifactId>

  <version>10.9.1.0</version>

</dependency>

<dependency>

<groupId>org.apache.derby</groupId>

<artifactId>derby</artifactId>

<version>10.9.1.0</version>

</dependency>


<dependency>

<groupId>javax.ws.rs</groupId>

<artifactId>javax.ws.rs-api</artifactId>

<version>2.0</version>

</dependency>


<dependency>

<groupId>org.codehaus.jackson</groupId>

<artifactId>jackson-jaxrs</artifactId>

<version>${org.codehaus.jackson-version}</version>

</dependency>
```

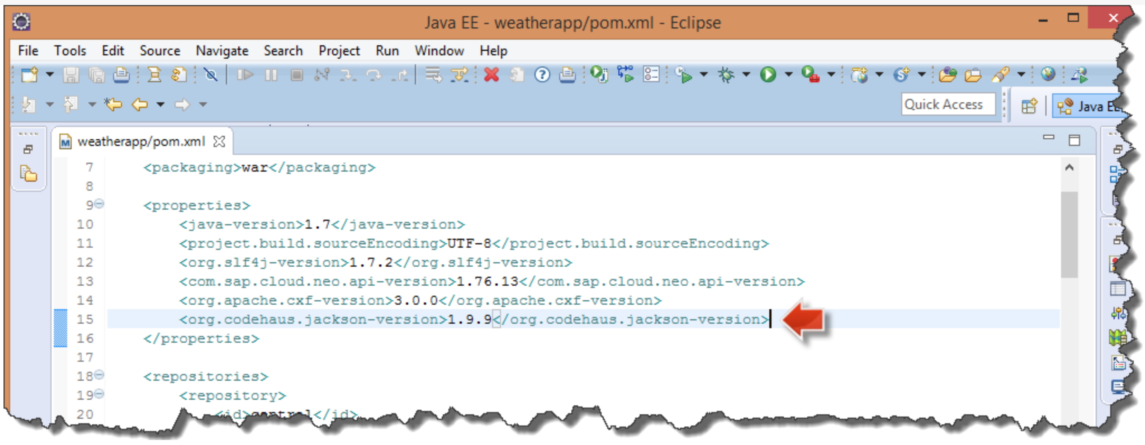

Java EE - weatherapp/pom.xml - Eclipse
File Tools Edit Source Navigate Search Project Run Window Help
Quick Access
Java EE
weatherapp/pom.xml

```xml
        </dependency>

        <dependency>
            <groupId>javax.ws.rs</groupId>
            <artifactId>javax.ws.rs-api</artifactId>
            <version>2.0</version>
        </dependency>

        <!-- EclipseLink (and JPA) -->
        <dependency>
            <groupId>org.eclipse.persistence</groupId>
            <artifactId>eclipselink</artifactId>
            <version>2.5.0</version>
        </dependency>
        <dependency>
            <groupId>org.eclipse.persistence</groupId>
            <artifactId>javax.persistence</artifactId>
             ersion>2.1.0</v
```

Step 10: Add Jackson version

We also need to add the Jackson version as a property in the properties section of the `pom.xml` file (as we have done in the previous section) and save your changes.

```
<org.codehaus.jackson-version>1.9.9</org.codehaus.jackson-version>
```

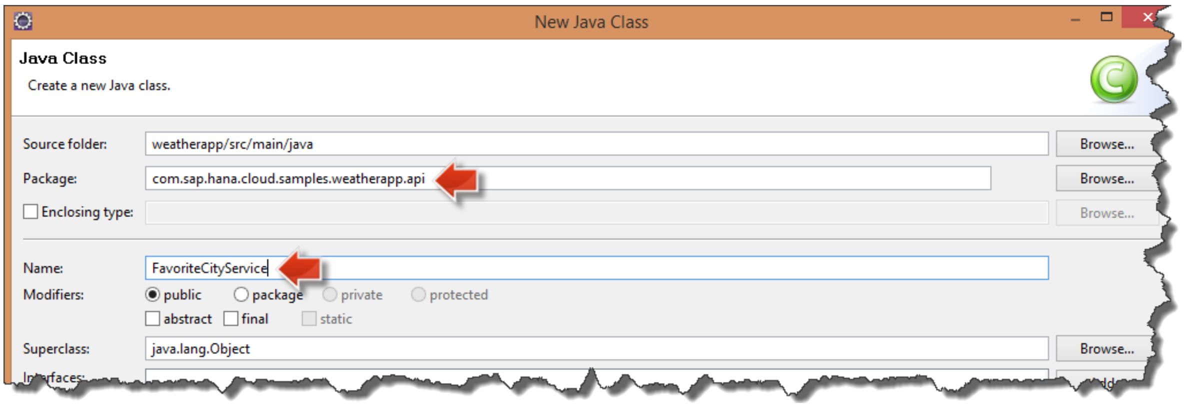

Java EE - weatherapp/pom.xml - Eclipse
File Tools Edit Source Navigate Search Project Run Window Help
Quick Access
Java EE
weatherapp/pom.xml

```
    <packaging>war</packaging>

    <properties>
        <java-version>1.7</java-version>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <org.slf4j-version>1.7.2</org.slf4j-version>
        <com.sap.cloud.neo.api-version>1.76.13</com.sap.cloud.neo.api-version>
        <org.apache.cxf-version>3.0.0</org.apache.cxf-version>
        <org.codehaus.jackson-version>1.9.9</org.codehaus.jackson-version>
    </properties>

    <repositories>
        <repository>
```

Step 11: Create CRUD service

Next step is to create the respective CRUD service. For that purpose, create a new class with the following details:

- **Package name:** `com.sap.hana.cloud.samples.weatherapp.api`
- **Classname:** `FavoriteCityService`



Java Class
Create a new Java class.

Source folder: weatherapp/src/main/java
Package: com.sap.hana.cloud.samples.weatherapp.api
☐ Enclosing type:
Name: FavoriteCityService
Modifiers: [✓ public] package private protected
☐ abstract ☐ final ☐ static
Superclass: java.lang.Object

Step 12: Replace the code

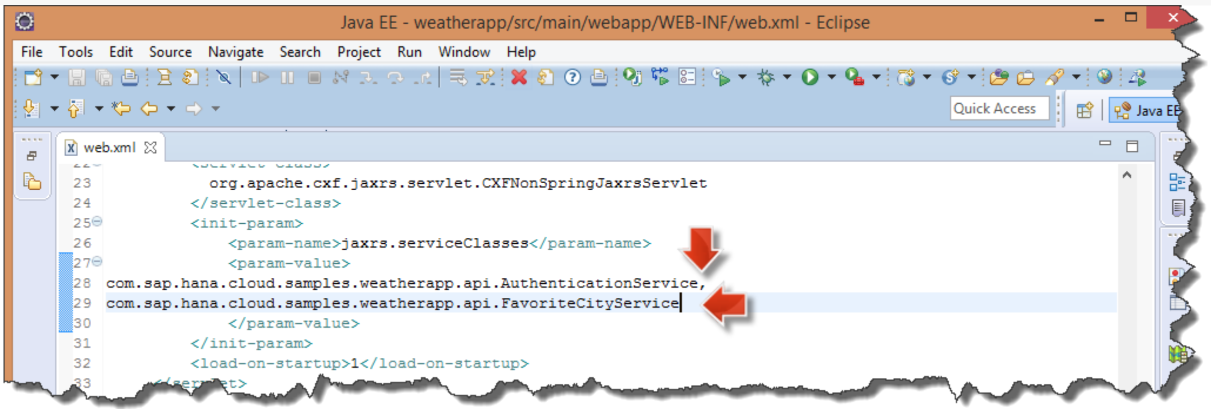
Replace the contents of the `FavoriteCityService.java` file with this code from GitHub and save your changes.

Step 13: Register REST service

To register our RESTful service implementation in the `web.xml` configuration file, add the fully qualified classname of our `FavoriteCityService` class to the *comma-separated* list of `jaxrs.serviceClasses`. See the snippet below for where to enter the fully qualified classname inside the `<param-value>` element (don't forget the comma at the end of the `AuthenticationService` line).

```
<init-param>

<param-name>jaxrs.serviceClasses</param-name>
```

```
<param-value>

com.sap.hana.cloud.samples.weatherapp.api.AuthenticationService,

com.sap.hana.cloud.samples.weatherapp.api.FavoriteCityService

</param-value>

</init-param>
```

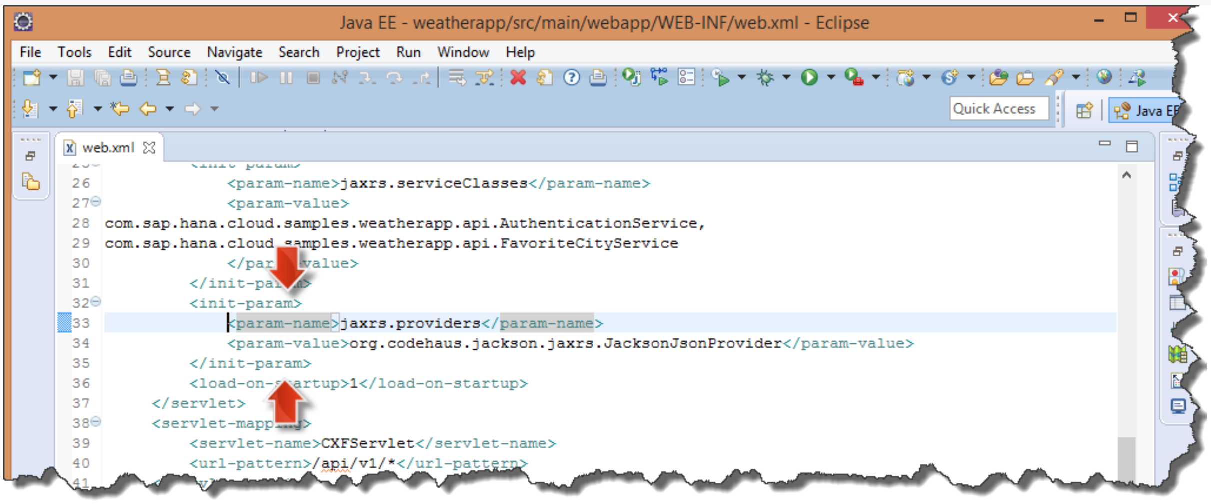

Java EE - weatherapp/src/main/webapp/WEB-INF/web.xml - Eclipse
File Tools Edit Source Navigate Search Project Run Window Help
Quick Access
Java EE
web.xml

```
            org.apache.cxf.jaxrs.servlet.CXFNonSpringJaxrsServlet
        </servlet-class>
        <init-param>
            <param-name>jaxrs.serviceClasses</param-name>
            <param-value>
com.sap.hana.cloud.samples.weatherapp.api.AuthenticationService,
com.sap.hana.cloud.samples.weatherapp.api.FavoriteCityService
            </param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>
```

Step 14: Add initialization parameter for JSON de-serialization

Just below the aforementioned `<init-param>` tag we need to add another `<init-param>` for JSON de-serialization as follows:

```
<init-param>

        <param-name>jaxrs.providers</param-name>

        <param-value>org.codehaus.jackson.jaxrs.JacksonJsonProvider</param-value>

</init-param>
```


Java EE - weatherapp/src/main/webapp/WEB-INF/web.xml - Eclipse
File Tools Edit Source Navigate Search Project Run Window Help
Quick Access
Java EE
web.xml

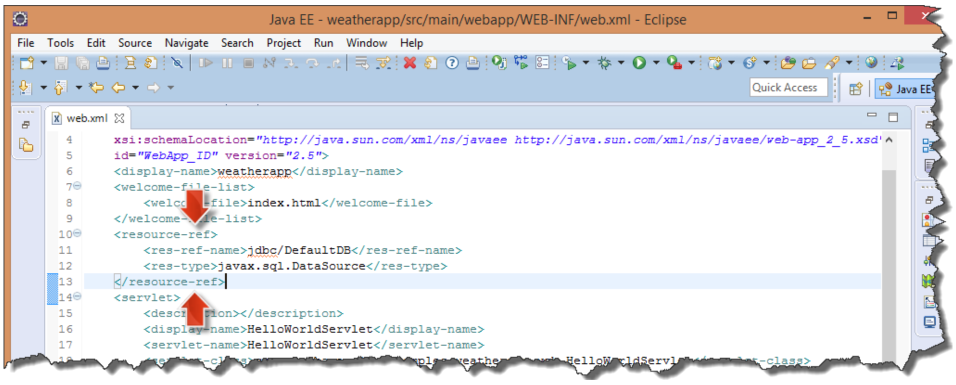
```
            <param-name>jaxrs.serviceClasses</param-name>
            <param-value>
com.sap.hana.cloud.samples.weatherapp.api.AuthenticationService,
com.sap.hana.cloud.samples.weatherapp.api.FavoriteCityService
            </param-value>
        </init-param>
        <init-param>
            <param-name>jaxrs.providers</param-name>
            <param-value>org.codehaus.jackson.jaxrs.JacksonJsonProvider</param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>CXFServlet</servlet-name>
        <url-pattern>/api/v1/*</url-pattern>
```

Step 15: Define a Data Source

The last change to make is to define a DataSource within the `web.xml` in order to connect to the underlying database. To do this, copy and paste the following XML snippet after the closing `</welcome-file-list>` tag:

Copy code

```
<resource-ref>

<res-ref-name>jdbc/DefaultDB</res-ref-name>

<res-type>javax.sql.DataSource</res-type>

</resource-ref>
```


Java EE - weatherapp/src/main/webapp/WEB-INF/web.xml - Eclipse
File Tools Edit Source Navigate Search Project Run Window Help
Quick Access
Java EE
web.xml

```
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
    id="WebApp_ID" version="2.5">
    <display-name>weatherapp</display-name>
    <welcome-file-list>
        <welcome-file>index.html</welcome-file>
    </welcome-file-list>
    <resource-ref>
        <res-ref-name>jdbc/DefaultDB</res-ref-name>
        <res-type>javax.sql.DataSource</res-type>
    </resource-ref>
    <servlet>
        <description></description>
        <display-name>HelloWorldServlet</display-name>
        <servlet-name>HelloWorldServlet</servlet-name>
```

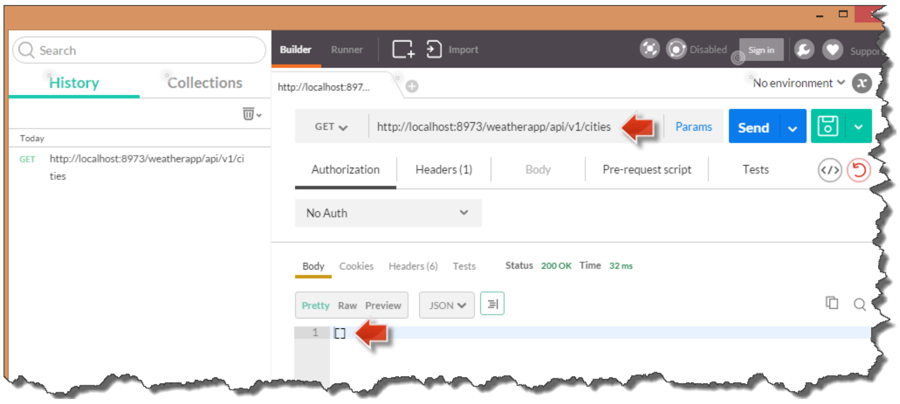
Step 16: Replace code

In order to properly test our RESTful service we need a REST tool (e.g. Postman) that allows you to execute HTTP calls in a convenient manner.

Step 17: Provide credentials

Within Postman, enter `http://localhost:8080/weatherapp/api/v1/cities` in the URL input field and make sure to provide your username/password as `Basic Auth` parameters in the **Authorization** tab.

Afterwards, make sure to update the request by pressing the respective **Update request** button. That will then add the "Authorization" parameter as an HTTP header parameter to your request.



GET http://localhost:8973/weatherapp/api/v1/cities

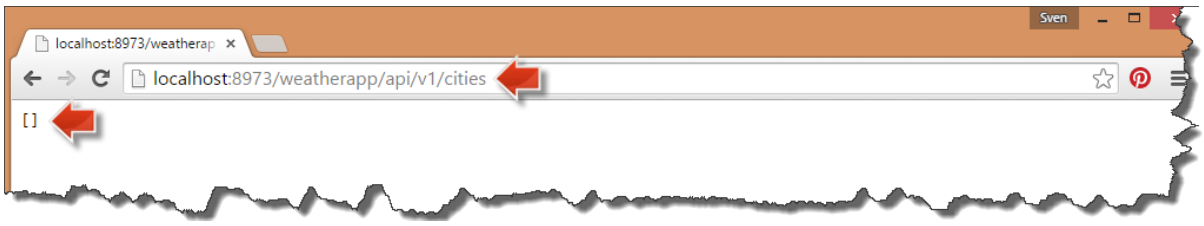
No Auth

Status 200 OK Time 32 ms

```
[]
```

Step 18: Execute the call

Once you execute the call, you'll see two empty brackets "[]" (indicating an empty array) after successful authentication. Don't worry, we haven't saved any cities as favorites yet, so that's just what we would expect.



localhost:8973/weatherapp/api/v1/cities

```
[]
```