Onboarding How-To: ONAP Portal SDK's Framework (FW) based Applications on ONAP Portal

This page discusses how to on-board Java-based web application to the ONAP Portal using (epsdk-fw-x.x.x.jar) library that is developed by the ONAP Portal development team (under repo - https://gerrit.onap.org/r/admin/repos/portal/sdk).

Get the dependency jar File

The following link will let you browse "final release versions" in the nexus repository: https://nexus.onap.org/content/repositories/releases/org/onap/portal/sdk/epsdk-fw/

Maven users will also need to add the following repositories to the pom.xml file:

```
        <repositories>
                <repository>
                        <id>onap-releases</id>
                        <name>ONAP - Release Repository</name>
                        <url>https://nexus.onap.org/content/repositories/releases</url>
                </repository>
        </repositories>
```

Maven users should use the following dependency in their project's pom.xml file (better with exclusions):

```
<dependency>
                        <groupId>org.onap.portal.sdk</groupId>
                        <artifactId>epsdk-fw</artifactId>
                        <version>2.6.0</version>
                        <exclusions>
                                <exclusion>
                                        <groupId>commons-logging</groupId>
                                        <artifactId>commons-logging</artifactId>
                                </exclusion>
                                <exclusion>
                                        <groupId>log4j</groupId>
                                        <artifactId>log4j</artifactId>
                                </exclusion>
                                <exclusion>
                                        <groupId>log4j</groupId>
                                        <artifactId>apache-log4j-extras</artifactId>
                                </exclusion>
                                <exclusion>
                                        <groupId>org.slf4j</groupId>
                                        <artifactId>slf4j-log4j12</artifactId>
                                </exclusion>
                                <exclusion>
                                        <groupId>junit</groupId>
                                        <artifactId>junit</artifactId>
                                </exclusion>
                                <exclusion>
                                        <groupId>commons-fileupload</groupId>
                                        <artifactId>commons-fileupload</artifactId>
                                </exclusion>
                                <exclusion>
                                        <groupId>commons-beanutils</groupId>
                                        <artifactId>commons-beanutils</artifactId>
                                </exclusion>
                                <!-- EELF omits "test" scope on this dependency -->
                                <exclusion>
                                        <groupId>org.powermock</groupId>
                                        <artifactId>powermock-module-junit4</artifactId>
                                </exclusion>
                                <!-- EELF omits "test" scope on this dependency -->
                                <exclusion>
                                        <groupId>org.powermock</groupId>
                                        <artifactId>powermock-api-mockito</artifactId>
                                </exclusion>
                        </exclusions>
                </dependency>
```

Configure the properties files

All applications that on-board to the ONAP Portal and use either the FW or SDK libraries must provide a configuration file called **portal.properties**. These properties configure the application for required network resources, including the Common Security Platform's Global Log On page, and the ONAP Portal's REST APIs. This file must be available on the Java classpath of a running application where it is found by the ONAP Framework (FW) library.

templates

Sample configuration

portal.properties

Configuration Key	Expected Value
`ecomp_rest_url`	Portal REST URL that is reachable by the application back-end. This is a value like https://portal.api.simpledemo.onap.org:30225/ONAPPORTAL/auxapi
`portal.api.impl.class`	Java class name. No default value. Value must be like org.oransc.ric.portal.dashboard.portalapi.PortalRestCentralServiceImpl

`ecomp_redirect_url`	Portal URL that is reachable by a user's browser. This is a value like `https://portal.api.simpledemo.onap.org:30225/ONAPPORTAL/login.htm`
role_access_centralized	Selector for role access. No default value. Value must be remote.
`ueb_app_key`	Unique key assigned by ONAP Portal to the onboarded application. No default value.

key.properties

The file `key.properties` must be provided on the Java classpath for the Spring-Boot application, as required by the EPSDK-FW library. The Helm chart for the application should mount this file appropriately.

The file must contain the following entries.

`cipher.enc.key` - Encryption key used by the EPSDK-FW library. No default value.

Implement the Users Management API

The application developers must provide a Java class that implements the interface org.onap.portalsdk.core.onboarding.crossapi.IPortalRestCentralService as documented below. The implementation should throw a PortalAPIException if anything fails; e.g., an unknown role or user.

- [ ] **public void editUser(String loginId, EcompUser user) throws PortalAPIException**. This method allows on-boarded applications to receive updates to a user profile. For example, to mark user as inactive. If any exception occurs, the application must throw PortalApiException. The FW library will catch the exception and send an appropriate response to Portal.

- [ ] **public void pushUser(EcompUser user) throws PortalAPIException:** The on-boarded application must implement this method to create the user provided in the argument. If any exception occurs (lets say user already exists), the onboarded application must throw PortalApiException. The FW library will catch the exception and send an appropriate response to Portal.

- [ ] **public String getUserId(HttpServletRequest request) throws PortalAPIException**: The on-boarded application must return the userId for the logged-in user. As a guideline for specific implementation, see the sample app - Sample code from RIC Dashboard app. Its a sample for on-boarded applications using EPSDK-FW. However, the apps can always choose to have their own implementation of this method if what's provided in the sample is not chosen. For Open-source implementation, for example, the app will have a totally different implementation for this method instead of the sample provided.

- [ ] **public Map<String, String> getAppCredentials() throws PortalAPIException**: Should return Map<String,String>., which includes username, password and appName of application . If any exception occurs, the application must throw PortalApiException. The FW library will catch the exception and send an appropriate response to Portal.

Onboard and test

Application Onboarding using UI in ONAP Portal.

When on-boarding the application to the ONAP Portal the administrator must supply the following information about the deployed instance:

- App URL that is reachable by a user's browser. The domain of this host name must match the Portal URL that is similarly reachable by a user's browser for cookie-based authentication to function as expected. This should be a value like `http://yourapp.simpledemo.onap.org:8080`
- App REST URL that is reachable by the Portal back-end server. This can be a host name or an IP address, because it does not use cookie-based authentication. This must be a URL with suffix "/api/v3" for example `http://yourapp.simpledemo.onap.org:8080/api/v3`