EUROPEAN COMMISSION
EUROSTAT

Directorate B: Methodology, corporate statistical and IT services
Unit B-3: IT for Statistical Production

Administrator Guide for the Validation Rule Manager

Administrator Guide

version 0.6.0-SNAPSHOT

Commission Européenne, 2920 Luxembourg, LUXEMBOURG - Tel. +352 43011

Administrator Guide: Administrator Guide for the Validation Rule Manager

by Commission Européenne, 2920 Luxembourg, LUXEMBOURG - Tel. +352 43011

version 0.6.0-SNAPSHOT

Table of Contents

Chapter 1. Getting started

These instructions were created to guide in the process of application build and deployment actions.

Chapter 2. Steps required

2.1. Prerequisites

Before starting any build process, we must first make sure that all the required software for the web application build to be successful or event start at all is installed in the system.

Please make sure that following software is present and working in your system:

- Java - version 1.8 at least,

- Maven - latest version,

- Node.js - latest version,

- NPM package manager - latest version.

Chapter 3. Build configuration

3.1. Properties

This section provides a step by step description of configuration entries and properties, which is required to understand the whole concept and how build overall works.

We'll go step by step through `pom.xml` (**Maven** build configuration) file explaining the whole internal composition of configuration and its effect on the build process.

Let's look at the properties section first:

```
<properties>
    ...
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <maven.compiler.source>1.8</maven.compiler.source>
    <maven.compiler.target>1.8</maven.compiler.target>

    <npm.output.directory>build</npm.output.directory>
    ...
</properties>
```

These properties contain basic configuration for the application.

Java version is set to `1.8` – as required for **Maven** to work.

Then there is one important property — `npm.output.directory` — that defines where the compiled / transpiled project code will land after the build.

3.2. Plugins

Following build (plugins) configuration:

```
<build>
  <finalName>${project.artifactId}-${project.version}</finalName>
  <plugins>
    ...
    <plugin>
      <groupId>org.codehaus.mojo</groupId>
      <artifactId>exec-maven-plugin</artifactId>
      <version>${maven.exec.plugin.version}</version>
      <executions>
        <execution>
          <id>npm run build</id>
          <goals>
            <goal>exec</goal>
          </goals>
          <phase>compile</phase>
          <configuration>
```

```
          <executable>npm</executable>
          <arguments>
            <argument>run</argument>
            <argument>build</argument>
          </arguments>
        </configuration>
      </execution>
    </executions>
    <configuration>
      <environmentVariables>
        <PUBLIC_URL>${deploy.host}:${deploy.port}/${deploy.path}</
PUBLIC_URL>
        <REACT_APP_ROUTER_BASE>/${deploy.path}</REACT_APP_ROUTER_BASE>
        <GENERATE_SOURCEMAP>${deploy.sourcemap}</GENERATE_SOURCEMAP>
      </environmentVariables>
    </configuration>
  </plugin>

  <plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-war-plugin</artifactId>
    <version>${maven.war.plugin.version}</version>
    <configuration>
      <nonFilteredFileExtensions>
        <nonFilteredFileExtension>pdf</nonFilteredFileExtension>
      </nonFilteredFileExtensions>
      <webResources>
        <resource>
          <directory>${npm.output.directory}</directory>
        </resource>
      </webResources>
      <webXml>${basedir}/web.xml</webXml>
    </configuration>
  </plugin>
  ...
  </plugins>
</build>
```

Contains four important parts, as listed below:

- **finalName**

  This property allows us to set the outcome filename, which is by default created from artifact id (vrm-frontend) and current version number.

- **exec-maven-plugin configuration**

  Exec plugin is configured to run the right npm build configuration – in this case

  ```
  npm run build
  ```

  which builds a production ready application.

  Three **environment variables** are set in the plugin configuration:

  - **PUBLIC_URL** – sets the public access URL. This way React application knows its full address.

  - **REACT_APP_ROUTER_BASE** – property used for routing purposes (application context) if necessary.

- **GENERATE_SOURCEMAP** – with this property prevent inclusion of Javascript source maps in the application package by Webpack. The source maps are files that contain mapping between source files and compressed versions. They allow to debug the code and track errors, but in production environment shouldn't be deployed to the web server nor available to normal users. You can read more about source map modes in [Webpack documentation](#).

- **maven-war-plugin configuration**

Maven `WAR` plugin is configured to copy web.xml descriptor, found in project root folder, to the WEB-INF folder in the build directory.

3.3. Profiles

The last part of configuration contains probably the most important deployment-wise configuration.

```xml
<profiles>
  <profile>
    <id>dev-tomcat</id>
    <activation>
      <activeByDefault>true</activeByDefault>
    </activation>
    <properties>
      <deploy.host>http://127.0.0.1</deploy.host>
      <deploy.port>8080</deploy.port>
      <deploy.path>vrm</deploy.path>
      <deploy.sourcemap>true</deploy.sourcemap>
    </properties>
  </profile>

  <profile>
    <id>dev-weblogic</id>
    <properties>
      <deploy.host>http://127.0.0.1</deploy.host>
      <deploy.port>7001</deploy.port>
      <deploy.path>vrm</deploy.path>
      <deploy.sourcemap>true</deploy.sourcemap>
    </properties>
  </profile>

  <profile>
    <id>prod-tomcat</id>
    <properties>
      <deploy.host>http://ec.europa.eu</deploy.host>
      <deploy.port>8080</deploy.port>
      <deploy.path>vrm</deploy.path>
      <deploy.sourcemap>false</deploy.sourcemap>
    </properties>
  </profile>

  <profile>
    <id>prod-weblogic</id>
    <properties>
      <deploy.host>http://ec.europa.eu</deploy.host>
      <deploy.port>7001</deploy.port>
      <deploy.path>vrm</deploy.path>
      <deploy.sourcemap>false</deploy.sourcemap>
    </properties>
```

```
        </profile>
    </profiles>
```

There are four profiles configured for the `WAR` file build: **dev-weblogic**, **dev-tomcat**, **prod-tomcat**, and **prod-weblogic**. Two are for development builds and two for production builds, taking into account each type of target application server (Weblogic and Tomcat). The main difference between them are properties defining host and port of application server host, and context path where VRM frontend application should be deployed. The properties in each profile are:

- `deploy.host` – the web address of the application host. In *dev* environment default value is localhost `http://127.0.0.1`, while in *prod* environment it's Eurostat server `http://ec.europa.eu`. The property – of course – has to be set to the value used in the actual environment.

- `deploy.port` – the port that deployed application will run at. For *Weblogic* default value is `7001`, while for *Tomcat* it's `8080`.

- `deploy.path` – the base path that is the trailing part of the deployed application address. The default value is `vrm`.

- `deploy.sourcemap` – indicates if application package should contain Javascript source maps. For more information about source maps see [description of GENERATE_SOURCEMAP](#) environment variable in [Section 3.2, "Plugins"](#). The default value is `true` for *dev* environment (source maps are generated) and `false` for *prod* environment (no source maps resulting in smaller package).

If the values in your target environment are different than defaults mentioned above, replace them before starting the build process.

These values are also used in the environment variables **PUBLIC_URL** and **REACT_APP_ROUTER_BASE** described in [Section 3.2, "Plugins"](#). They need to be consistent with real web address that VRM frontend application is deployed at, otherwise React router won't work properly.

3.4. Build process

The build process is carried by [Maven](#), which requires Java to be installed in the system (preferred 1.8 minimum).

To build the application the right profile must be chosen, see [Section 3.3, "Profiles"](#).

Run following command to execute build process:

```
mvn clean install –P[profile_name]
```

Whereas `[profile_name]` refers to the chosen profile name.

The `WAR` file should be ready for deployment.

3.5. Deployment process

The deployment process may vary depending on the platform.

The general idea is that by default the application deploys `WAR` file under the context specified as the project name / war file name.

It is possible to change that behavior for the most of existing application servers that comply with the JEE standards, however in case of Tomcat – simple servlet container – it is not possible to do on the application side.

For simplicity reasons we have assumed that it is better to deploy application using context as it is – its name, and if the it must change – take required steps later or just change the deployment archive name.

Because the official documentation is enriched enough with descriptions, popular issues solutions, and updated on frequent base – in **Tomcat's**, as well as in **WebLogic's** case - we have decided to redirect to these sources.

For more information about **WebLogic** web application server-side deployment go here and for Tomcat here.

Chapter 4. Revision History

Revision History		
Revision 1.0	2020-05-28	Adrian Fijalkowski
Released. First version of the document		
Revision 1.1	2020-07-07	Pawel Sobocinski
Updated. Document updated to reflect the changes in the Maven configuration in pom.xml		