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 0.7.1-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 0.7.1-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. Building Frontend Application

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.

2.2. 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.

2.3. 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.

2.4. Profiles

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

```
<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 2.3, “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 2.3, “Plugins”. They need to be consistent with real web address that VRM frontend application is deployed at, otherwise React router won't work properly.

2.5. 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 2.4, “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.

2.6. 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 3. Building Backend Application

3.1. Eurostat Maven Repository

In order to download ESTAT libraries you must configure ESTAT maven repository.

Create `settings.xml` file in `~/.m2` directory using the template:

Username to use starts with *em*. Encrypt your password as described here.

Do not leave your password in the shell history. Use prompting commands.

Example `settings.xml` file:

```
<?xml version="1.0" encoding="UTF-8"?>
<settings xmlns="http://maven.apache.org/SETTINGS/1.0.0"
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.0.0
 http://maven.apache.org/xsd/settings-1.0.0.xsd">
    <profiles>
        <profile>
            <id>estat</id>
            <properties>
                <project.build.sourceEncoding>UTF-8</
project.build.sourceEncoding>
            </properties>
            <repositories>
                <repository>
                    <id>eurostat</id>
                    <url>https://citnet.tech.ec.europa.eu/CITnet/nexus/
content/repositories/estat</url>
                    <releases>
                        <enabled>true</enabled>
                    </releases>
                    <snapshots>
                        <enabled>false</enabled>
                    </snapshots>
                </repository>
            </repositories>
        </profile>
    </profiles>

    <servers>
        <server>
            <id>eurostat</id>
            <username>Your username starting with - em*</username>
            <password>{Your encrypted password}</password>
        </server>
    </servers>

    <activeProfiles>
        <activeProfile>estat</activeProfile>
    </activeProfiles>
</settings>
```

3.2. Load Time Weaving

The application requires load time weaving to be enabled.

Download `spring-instrument-5.2.5.RELEASE.jar` and pass it as the java agent using VM option: *-javaagent:<path_to_jar_file>/spring-instrument-5.2.5.RELEASE.jar*

`spring-instrument-{version}.RELEASE.jar` is available in The Maven Central Repository.

Pass an additional *-noverify* VM option to the WebLogic server.

Chapter 4. Database Configuration

4.1. JNDI Lookup

For application to be versatile with multiple environments, i.e. WebLogic and Tomcat, it is required to perform a *jndi lookup* instead of - which is by default - using classic **.properties** file connection configuration.

Following entry in *.yaml* configuration file allows jndi-lookup.

```
spring:
  (...)
  datasource:
    (...)
    jndi-name: jdbc/vrm
  (...)
```

Spring performs jndi-lookup when the *jndi-name* parameter is set, otherwise it tries to use standard configuration properties provided in the file.

4.2. web.xml Descriptor

Another important thing to set up before any data source is the *web.xml* descriptor content. The file should have been placed in the **src/main/webapp/WEB-INF** directory.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<web-app xmlns="http://java.sun.com/xml/ns/javaee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
              http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
         version="3.0"
         metadata-complete="true">
    <resource-ref>
        <description>DB Connection</description>
        <res-ref-name>jdbc/vrm</res-ref-name>
        <res-type>javax.sql.DataSource</res-type>
        <res-auth>Container</res-auth>
    </resource-ref>
</web-app>
```

We specify resource reference for the web application, where *res-ref-name* references our *jndi* name acknowledged in the previous section.

4.3. SQL Dialect

By default SQL dialect variable `spring.jpa.properties.hibernate.dialect` is configured for Oracle database (*org.hibernate.dialect.Oracle12cDialect*). Please set the variable for the right database dialect used if necessary. Value for MySQL is *org.hibernate.dialect.MySQL8Dialect*.

4.4. Tomcat Server

4.4.1. JDBC connectors

Before we start any deployment it is very important to make sure that the Tomcat standalone server contains all the necessary JDBC Connector libraries required for making a connection to the database.

For MySQL see this Maven repository page.

For Oracle see this Maven repository page.

Verify that `/lib` directory contains these *jar* files.

4.4.2. server.xml

The connection resource is, by out assumption, configured in the Tomcat standalone `server.xml` file. Moving configuration from application's context to server's context allows us to make application completely independent to the way connection is made, and of course - which database configuration is used. In the configuration file `conf/server.xml` - adding following entry in the **GlobalNamingResources** enables the resource:

```
<!-- Driver has to be provided to the Tomcat standalone -->
<Resource name="jdbc/vrm"
          auth="Container"
          type="javax.sql.DataSource"
          username="vrm"
          password="vrm"
          driverClassName="com.mysql.cj.jdbc.Driver"
          factory="org.apache.tomcat.jdbc.pool.DataSourceFactory"
          url="jdbc:mysql://localhost:3306/vrm"
          removeAbandonedTimeout="60"
          testWhileIdle="true"
          timeBetweenEvictionRunsMillis="300000"

 connectionProperties="useUnicode=true;useJDBCCompliantTimezoneShift=true;

 useLegacyDatetimeCode=false;serverTimezone=UTC;useSSL=false;"
```

```
          initialSize="8"
          maxActive="8"
          minIdle="8"
          maxIdle="8"/>
```

Note that configuration above is specific for Tomcat + MySQL.

The resource configuration alone doesn't make it accessible throughout the application though.

4.4.3. context.xml

To share the resource in the application it has to be explicitly mapped in the `context.xml` file, so the application can find the right object during the jndi lookup.

During the deployment the `context.xml` file under `src/main/webapp/META-INF` directory is loaded by Tomcat standalone server.

Following content:

```
<?xml version='1.0' encoding='utf-8'?>
<Context reloadable="true" privileged="true">
    <WatchedResource>WEB-INF/web.xml</WatchedResource>
    <ResourceLink name="jdbc/vrm"
                  global="jdbc/vrm"
                  auth="Container"
                  type="javax.sql.DataSource"/>
</Context>
```

maps the global resource we've configured in the previous step to a local resource in the application.

4.5. WebLogic Server

Please note that following walk-through follows WebLogic + Oracle database pairing.

Go to **Services** (in the context menu, on the left) and select **Data sources**. Then on the toolbar of data sources table select the **New** dropdown menu and then click on **Generic Data Source**.

4.5.1. First Page

In the first configuration screen we only have to specify four fields:

- **Name** doesn't really matter, but it is a good practice to use a distinguishable name for future ease of use, e.g. *VRM_Oracle_DataSource*.
- For **Scope** we may select *Global*.
- For **JNDI Name** type *jdbc/vrm*, as specified previously, and the **database type** (in this case) *Oracle*.

4.5.2. Second Page

Please select the driver used for the specific database, for Oracle: **Database Driver: Oracle's Driver (Thin) for Service connections; Versions: Any**

4.5.3. Third Page

Following options must be selected: **Support Global Transactions** and **One-Phase Commit**.

4.5.4. Fourth Page

The database connection configuration screen - there you can set the properties required for actual database connection. Please bear in mind that the properties may vary depending on the use case.

Sample configuration:

```
Database Name: 'vrm'
Host Name: '127.0.0.1'
Port: '1521'
Database User Name: 'vrm'
Password: 'vrm'
```

4.5.5. Fifth Page

It is very important to make the connection available to the server. To do so - mark the checkbox next to **AdminServer** to make it as an active target.

Chapter 5. Security Configuration

This chapter describes how to configure OAuth2/OIDC authorization server in the VRM frontend and backend applications. It assumes the reader has basic knowledge about OAuth2 and OIDC protocols and authorization flows. Instructions how to configure VRM applications as clients in the authorization server aren't covered by the scope of this chapter.

The instructions in this chapter apply to version 0.8.0 or later of the VRM.

5.1. VRM Backend

The VRM backend application uses built-in libraries from Spring Security framework to support OAuth2 and OIDC protocols. The configuration parameters are defined in the *application.yml* configuration files in the *vrm-backend-app* and *vrm-backend-app* modules and automatically applied in the *WebSecurityConfiguration* class in the *vrm-configuration* module.

The VRM backend application has to be registered as both resource server (to validate incoming access tokens at the authorization server introspection URI) and client application (to query userinfo URI when adding newly authenticated user to the VRM database). All access tokens are treated as opaque and validated at authorization server, even if they are JWT tokens that can be validated locally.

> **Note**
>
> Roles for VRM users aren't defined in the authorization server, but in the VRM user database, and are assigned manually by the VRM administrator using administrative interface in the VRM frontend application. Since the VRM user database is empty after deployment, there isn't any user with administrator role. Such role is granted to the first user that authenticates in the VRM.

5.1.1. Production Build

The *application.yml* file for the production build can be found in the *src/main/resources* directory in the *vrm-backend-app* module. There are four sections related to security configuration: `spring.security.oauth2.client.registration`, `spring.security.oauth2.client.provider`,

`spring.security.oauth2.resourceserver.opaque-token`, and `vrm.security.registration-id`. The sections contain configuration parameters described below.

```
spring.security.oauth2.client.registration:
    <id>:
        client-id:
        client-secret:
```

Where:

- `id` — The identifier of specific OAuth2/OIDC provider (entered before colon) that uniquely identifies the client registration.
- `client-id` — The client identifier of the VRM backend application assigned by the authorization server.
- `client-secret` — The client secret of the VRM backend application assigned by the authorization server.

```
spring.security.oauth2.client.provider:
    <id>:
        issuer-uri:
        user-info-uri:
        user-name-attribute:
```

Where:

- `id` — The identifier of specific OAuth2/OIDC provider (entered before colon).
- `issuer-uri` — The issuer identifier URI for the OAuth2/OIDC provider.
- `user-info-uri` — The userinfo endpoint URI used to access the claims/ attributes of the authenticated end-user.
- `user-name-attribute` — The name of the attribute returned in the userinfo response that references the name or identifier of the end-user.

Note

The attribute listed in the `user-name-attribute` setting must be present as a claim in the access token received from the authorization server by the VRM frontend application and must contain user login.

```
spring.security.oauth2.resourceserver.opaque-token:
    introspection-uri:

vrm.security.registration-id:
```

Where:

- `introspection-uri` — The Introspection Uri used to introspect the details of an opaque token.
- `vrm.security.registration-id` — The identifier of currently used OAuth2/OIDC provider used in `registrations` and `provider` sections above (eg. ecas or keycloak).

5.1.2. Development Build

The *application.yml* file for the production build can be found in the *src/main/resources* directory in the *vrm-backend-app-dev* module. The settings are the same as for the production build.

5.2. VRM Frontend

The VRM frontend application uses oidc-client library (wrapped in redux-oidc package), which provides OAuth2 and OpenID Connect (OIDC) protocol support for client-side, browser-based JavaScript client applications. Defined configuration parameters are applied in the *userManager.ts* file in the */src/utility* directory of the project.

The VRM frontend application has to be registered as public client (without a secret) in the authorization server. If the access token received from the authorization server doesn't contain claim listed in `user-name-attribute` setting of the VRM backend application with the username value, you may need to add related mapping in the authorization server configuration.

5.2.1. Production Build

Security configuration for the VRM frontend development build can be found in the *pom.xml* file in the root directory of the project. They are defined as profile properties that are converted to environment variables during Maven compile phase. The following security properties are in `<profiles>` section:

```
<deploy.authority.url>https://</deploy.authority.url>
<deploy.authority.clientId></deploy.authority.clientId>
<deploy.authority.response>code</deploy.authority.response>
<deploy.authority.scope>openid profile email</deploy.authority.scope>
```

- `deploy.authority.url` — The URL of the OAuth2/OIDC provider.
- `deploy.authority.clientId` — Client identifier of VRM frontend as registered with the OAuth2/OIDC provider.
- `deploy.authority.response` — The type of response expected from the OAuth2/OIDC provider. Default value is "`code`" which refers to Authorization Code Flow with PKCE. In case of ECAS authorization server this value should be "`id_token`", as ECAS requires public applications to use Implicit Flow with Proof-of-Possession Tokens.
- `deploy.authority.scope` — The scope being requested from the OAuth2/OIDC provider. Default value is "`openid profile email`".

The conversion to environment variables is performed by *exec-maven-plugin* configured in <build><plugins> section of the *pom.xml* file. The resulting variables are respectively:

- `REACT_APP_AUTHORITY_URL`
- `REACT_APP_AUTHORITY_CLIENT_ID`
- `REACT_APP_AUTHORITY_RESPONSE`
- `REACT_APP_AUTHORITY_SCOPE`

5.2.2. Development Build

Security configuration for the VRM frontend development build can be found in the *.env* file in the root directory of the project. Similar to production build, additional parameters are defined in the *userManager.ts* file in the */src/utility* directory of the project.

The configuration parameters are stored in the environment variables consistent with the ones from *pom.xml* file used for the production build. They are as follows:

- `REACT_APP_AUTHORITY_URL` — The URL of the OAuth2/OIDC provider.

- `REACT_APP_AUTHORITY_CLIENT_ID` — Client identifier of VRM frontend as registered with the OAuth2/OIDC provider.

- `REACT_APP_AUTHORITY_RESPONSE` — The type of response expected from the OAuth2/OIDC provider. Default value is "`code`" which refers to Authorization Code Flow with PKCE. In case of ECAS authorization server this value should be "`id_token`", as ECAS requires public applications to use Implicit Flow with Proof-of-Possession Tokens.

- `REACT_APP_AUTHORITY_SCOPE` — The scope being requested from the OAuth2/OIDC provider. Default value is "`openid profile email`".

  Note

  Both VRM backend and frontend must access the authorization server using the same URL, as per OAuth2 protocol requirement.

Chapter 6. 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.		
Revision 1.2	2020-10-16	Rafal Spryszynski
Updated. Added *Building Backend Application* and *Database Configuration* chapters.		
Revision 1.3	2021-05-04	Pawel Sobocinski
Updated. Added *Security Configuration* chapter.		