Administrator deployment guide

Getting started

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

Steps required

Prerequisites

Before starting any build process, we must first make sure that all the required software for the web application's 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.

Build configuration

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>
  <eurostat.address>http://ec.europa.eu/vrm</eurostat.address>
</properties>
```

These properties contain basic configuration for the application.

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

Then there are two important properties:

`npm.output.directory`, which defines where the compiled / transpiled project's code will land after the build and `eurostat.address` – property defining a real address of the Eurostat production server. The property – of course – has to be set to the value used in the actual production environment.

Plugins

Following build (plugins) configuration:

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

    <plugin>
      <groupId>org.codehaus.mojo</groupId>
      <artifactId>exec-maven-plugin</artifactId>
      <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>${app.address}:${app.port}/${project.artifactId}</PUBLIC_URL>
          <REACT_APP_ROUTER_BASE>/${project.artifactId}</REACT_APP_ROUTER_BASE>
        </environmentVariables>
      </configuration>
    </plugin>
  </plugins>
</build>
```

Contains four important parts, as listed below:

- **File name**

  Property allows us to set the outcome filename, which is by default the project's name, i.e. vtl-editor.

- **maven-war-plugin configuration**

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

- **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.

- **Environment variables**

  PUBLIC_URL – sets the public access URL – property that is required for React application to know the full application address, and REACT_APP_ROUTER_BASE – property prepared for routing purposes (application's context), if necessary.

Profiles

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

```
<profiles>
  <profile>
    <id>local-tomcat</id>
    <activation>
      <activeByDefault>true</activeByDefault>
    </activation>
    <properties>
```

```
      <app.port>8080</app.port>
      <app.address>http://127.0.0.1</app.address>
    </properties>
  </profile>

  <profile>
    <id>local-weblogic</id>
    <properties>
      <app.port>7001</app.port>
      <app.address>http://127.0.0.1</app.address>
    </properties>
  </profile>

  <profile>
    <id>prod-tomcat</id>
    <properties>
      <app.port>7001</app.port>
      <app.address>${eurostat.address}</app.address>
    </properties>
  </profile>

  <profile>
    <id>prod-weblogic</id>
    <properties>
      <app.port>7001</app.port>
      <app.address>${eurostat.address}</app.address>
    </properties>
  </profile>
</profiles>
```

There are four profiles configured for `.WAR` file build, described below:

- **local-weblogic** – application has `localhost / 127.0.0.1` address and `7001` port set – development purposes.
- **local-tomcat** – like above, except the port is set to `8080`.
- **prod-tomcat** and **prod-weblogic** working analogically (port-wise) with the only difference being the address set to aforementioned `eurostat.address's` property value.

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 **'Profile'** subsection under **'Build configuration'** section.

Run following command to execute build process:

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

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

The `WAR` file should be ready for deployment.

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's 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.