Muse Identity Manager Install

September 8, 2022

Document Version 1.15
Muse 2.8.0.1

Notice

Disclaimer

Trademarks

museknowledge.com

260 Madison Avenue, 8th Floor
New York, NY 10016, U.S.A.
+1 (646) 740-4968 SUPPORT@MUSEKNOWLEDGE.COM

Table of Contents

1.0

Overview

Muse Identity Manager (Muse IDM) is a web application with processes for identifying, authenticating and authorizing individuals or groups of people to access configured applications by associating user rights and restrictions with established identities.

This application offers support for user registration with email validation, with or without administrator approval, centralized management including users visualization, searching and filtering, bulk updates, sending email, export/import as/from CSV.

2.0

Installation

The Muse IDM application can be installed on any platform on which a Java Virtual Machine can be installed.

2.1 Hardware Requirements

The following are the minimum and recommended hardware specifications for running the Muse IDM application.

Minimum specification:

- 120 GB hard disk space;
- 4 GB RAM;
- Intel Pentium IV equivalent or better processor, multicore (at least 2 cores);
- Processor clock speed of 2.4 GHz or better.

Recommended specification:

- 240 GB hard disk space;
- 8 GB RAM;
- Intel Pentium IV equivalent or better processor, multicore (at least 4 cores);
- Processor clock speed of 3.0 GHz or better.

2.2 Software Requirements

The following environment software products are necessary to operate the Muse IDM application.

All the environment software listed below has been tested with the Muse IDM application, and supports it fully. This does not mean that the Muse IDM application is not fully functional on other environments not listed here.

1 Operating System

The following operating systems have been tested with the Muse IDM application;

- Windows 10;
- Linux, various distributions: Ubuntu, CentOS, Debian;

2 Java Virtual Machine Software Development Kit (JDK)

It is recommended the installation of a Java Virtual Machine Software Development Kit (SDK, or also referred to as JDK) package because they contain tools not available in the Java Runtime Environment (JRE) package that can be used for monitoring and troubleshooting issues related to the Java Virtual Machine and the Muse IDM application. For example, the `jstat` tool available in a JDK package displays performance statistics for an instrumented HotSpot Java Virtual Machine (JVM). The Java Virtual Machine Software Development Kit contains the JRE.

The following virtual machines and platform combinations have been tested with the Muse IDM application:

- Intel x64/Linux, JDK8, OpenJDK 8, OpenJDK 11;
- Intel x64/Windows, JDK8, AdoptOpenJDK 8, JDK11, AdoptOpenJDK 11;

Note: Java 8 is the minimum and recommended software specification for running the IDM application.

Note: It is recommended to install the JDK or OpenJDK version that matches the architecture, e.g. if the architecture is on 64 bit then the 64 bit JDK or OpenJDK must be installed.

a. OpenJDK Installation on Windows Platforms

- Download the latest build of the Windows OpenJDK version 8 from:

  `https://adoptium.net/temurin/releases/?version=8`

  according to the architecture of the server, e.g. download the 64 bit version if the server's architecture is on 64 bit.

- Install the OpenJDK according to the installation instructions from here:

https://adoptium.net/installation/windows/

After the installation is performed, define the environment variable called `JAVA_HOME` pointing to the OpenJDK installation. Assuming the destination directory is `C:/OpenJDK1.8` set `JAVA_HOME=c:/OpenJDK1.8` .

Also it is recommended to add in the system `Path` environment variable the OpenJDK `bin` folder. For example, if your Java installation folder is `C:/OpenJDK1.8` then add the following string to the `Path` environment variable: `C:/OpenJDK1.8/bin` . Note that the separator between the entries from the `Path` variable is the semicolon (;) . Details for how to update the `Path` variable for some Windows distributions can be found here:

https://www.java.com/en/download/help/path.xml

b. OpenJDK Installation on Linux Platforms

- Download the latest build of the Linux OpenJDK version 8 from:

  https://adoptium.net/temurin/releases/?version=8

  according to the architecture of the server, e.g. download the 64 bit version if the server's architecture is on 64 bit.

  The OpenJDK setup kit can be of 2 types: a `rpm` package or a `tar.gz` archive, download the one which is appropriate.

- Install the OpenJDK according to the installation instructions from here:

  https://adoptium.net/installation/linux/

  After the installation is performed, define the environment variable called `JAVA_HOME` pointing to the JDK installation. Assuming the destination directory is `/usr/java/openjdk` run the following command :

```
export JAVA_HOME=/usr/java/openjdk
```

  Add the line above in the file `/etc/profile` to make this setting permanent on Linux. The variable becomes globally available after the next login.

  Also it is recommended to add in the system `PATH` environment variable the JDK `bin` folder. Details for how to update the PATH variable for Linux can be found here:

`http://www.java.com/en/download/help/path.xml`

3 Database Management System (DBMS)

A DBMS installation is needed by the Muse IDM application, such as MySQL, Microsoft SQL Server, Oracle, etc.

The MySQL DBMS was tested with the Muse IDM application and it is the one used and refered in this installation manual.

MySQL Community Edition is a freely downloadable version which is provided under the GPL License.

Because version 8.0 was tested with the Muse IDM application, all instructions from below and comments refer only to this version. Other MySQL versions may work as well, but they were not tested.

a. MySQL server installation on Windows Platforms

- Download MySQL Community Server Windows executable package, making sure to get the version that matches the hardware architecture (32 bit or 64):

  `https://dev.mysql.com/downloads/mysql/8.0.html#downloads`

- Install it according to the installation instructions from here:

  `https://dev.mysql.com/doc/refman/8.0/en/windows-installation.html`

b. MySQL server installation on Linux Platforms

- Download MySQL Community Server Linux , making sure to get the version that matches the hardware architecture (32 bit or 64):

  `https://dev.mysql.com/downloads/mysql/8.0.html#downloads`

- Install it according to the installation instructions from here:

  `https://dev.mysql.com/doc/refman/8.0/en/linux-installation.html`

Note: If the installed Linux distribution has a package manager (e.g. `Apt` on Debian/Ubuntu, `Yum` for RedHat) then install MySQL server using the package manager by following the instructions from the above link.

4 Web server implementing Java Servlet and JavaServer Pages technologies

The Muse IDM application is a web application which runs inside a servlets container server such as Apache Tomcat, JBoss, WebLogic, GlassFish.

The Apache Tomcat server is the web server tested with the Muse IDM application and this is further refered into documentation.

Apache Tomcat is developed in an open and participatory environment and released under the Apache License version 2 and it is the servlet container that is used in the official Reference Im plementation for the Java Servlet and JavaServer Pages technologies.

5 phpMyAdmin (optional)

phpMyAdmin is a free software tool written in PHP, intended to handle the administration of MySQL over the Web. phpMyAdmin is released under GNU General Public License, version 2.

It can be installed for providing an easier user interface oriented management over the MySQL database.

The installation of phpMyAdmin is not covered here, please refer to the phpMyAdmin website:

`https://www.phpmyadmin.net`

The phpMyAdmin tool requires a web server (such as Apache, IIS), so this must be installed too, prior to the installation of phpMyAdmin.

2.3 Installation Steps

The steps for installing the Muse IDM application are:

1. The installation of the software requirements. See the above `Software Requirements` chapter regarding the software requirements.
2. The installation of the Muse IDM application. This is covered in the current section.
3. Performing post-install configuration. This is covered in the next sections.

The overall steps for installing the Muse IDM application are the following:

1. Create the necessary database and users in the DBMS. This is described in the "DBMS Settings" section.

2. Install the Web Server. Apache Tomcat server is the servlets container server tested with the Muse IDM application, its installation is described in the "Apache Tomcat Installation" section.
3. Install the Muse IDM application. This is descried in the "Muse IDM Installation" section.

2.3.1 DBMS Settings

The Muse IDM application makes use of a DBMS (Database Management System) for storing data.

Because only MySQL DMBS was tested with the Muse IDM application, all instructions from this paragraph are for MySQL.

Note: The Muse IDM application was also tested with PostgreSQL 11.1 but doesn't work because of a table named `user` which is a reserved word based on documentation from here https://www.postgresql.org/docs/11/sql-keywords-appendix.html

Note: Starting with `2.1.0.8` version, `user` table was renamed to `users`, PostgreSQL 13.3 can be used with Muse IDM. Alo is required to rename `oauth2-schema-postgres.sql` file to `oauth2-schema.sql`.

Create the MySQL database and users for Muse IDM application:

In the `mysql` command line client, as the database `root` user execute the following commands:

```
mysql> CREATE DATABASE IDM CHARACTER SET utf8mb4 COLLATE utf8mb4_bin;
```

This command will create a new database with the name `IDM` which will be used by the Muse IDM application.

The responses to each command should look like:

```
Query OK, 1 row affected (0.00 sec)
```

Then create users for accessing the newly created database and specify their priviledges. This is done by running the following commands:

```
mysql> CREATE USER 'IDMUser'@'localhost' IDENTIFIED BY 'IDMUserPass';
mysql> GRANT ALL ON IDM.* TO 'IDMUser'@'localhost';
mysql> FLUSH PRIVILEGES;
```

where replace `IDMUserPass` with a more secure password.

The result of running this command is the creation of the user `IDMUser` with the provided password and with all priviledges on the `IDM` database.

2.3.2 Apache Tomcat Installation

Apache Tomcat version 8.5 or higher must be installed for running the Muse IDM application.

Tomcat 8.5 and 9 were also tested and they work fine, other Apache Tomcat versions were not tested, but they may work as well.

- Download the Apache Tomcat version 8.5 archive from:

  `https://tomcat.apache.org/download-80.cgi`

  according to the server's platform (e.g. if the server architecture is 64 bit then download the 64 bit archive)

- Extract the Apache Tomcat archive in the desired installation location. Recommended locations:
  - `C:` drive on Windows platforms.
  - `/opt/` folder on Linux platforms.

  Note: On Linux platforms make sure that the script files (`*.sh`) from the `bin` folder from the Apache Tomcat installation folder are executable.

2.3.3 Muse IDM Installation

The Muse IDM aplication can be installed by 2 methods:

- Independently of the Muse Federated Search Platform, as Web Application Archive (`.war` file).
- Part of the Muse Federated Search Platform, into the Embedded Apache Tomcat as external context.

Note: Before going further with the installation make sure the Hardware and Software requirements are met and you have a valid License Key File.

Note: On following sections and chapters you will see `IDM_HOME` very often and represents the location where Muse IDM application is installed, which can be `MUSE_HOME/idm/www` if is installed in the Embedded Apache Tomcat from the Muse Federated Search Platform or `CATALINA_HOME/webapps/idm` if is installed as Web Application Archive in

`CATALINA_HOME/webapps` folder.

Note: Any changes made in the following files:

1. `IDM_HOME/WEB-INF/classes/application.properties`
2. `IDM_HOME/WEB-INF/classes/database.properties`
3. `IDM_HOME/WEB-INF/classes/email.properties`
4. `IDM_HOME/WEB-INF/classes/users.properties`
5. `IDM_HOME/WEB-INF/classes/log4j2.xml`

Note: require a server restart. This rule is also applied for all XML files from the `IDM_HOME/WEB-INF/config` folder.

2.3.3.1 Installing the Muse IDM application as Web Application Archive

The Muse IDM application is distributed as a Web Application Archive (`.war` file) that is deployed inside a web server implementing the servlets specification.

The steps for installing the Muse IDM application into Apache Tomcat web server using the `war` archive are below:

1. Deploy into `CATALINA_HOME/webapps` the Muse IDM Web Application Archive `idm.war` file.
2. If Security Manager is enabled for the Apache Tomcat server then the following entry need to be added in `CATALINA_HOME/conf/catalina.policy` at the end of the file:

   ```
   grant codeBase "file:${catalina.base}/webapps/idm/-" {
                 permission java.security.AllPermission;
   };
   ```

3. Stop the Apache Tomcat server:

   ```
   CATALINA_HOME/bin/shutdown.{bat|sh}
   ```

   After the shutdown command was issued check the system's processes to make sure the Apache Tomcat process has really stopped, and only when the process has stopped issue the startup command.

4 Make sure that the following structure of properties files with configurations for the application, MySQL database, SMTP and users exists properly configured:

- Edit the `CATALINA_HOME/webapps/idm/WEB-INF/classes/application.properties` file as described at section "*application.properties config file*".
- Edit the `CATALINA_HOME/webapps/idm/WEB-INF/classes/database.properties` file as described at section "*database.properties config file*".
- Edit the `CATALINA_HOME/webapps/idm/WEB-INF/classes/email.properties` file as described at "*email.properties config file*".
- Edit the `CATALINA_HOME/webapps/idm/WEB-INF/classes/users.properties` file as described at "*users.properties config file*".

5 Start the Apache Tomcat server:

```
CATALINA_HOME/bin/startup.{bat|sh}
```

Upon startup the Muse IDM application archive is deployed, this should take a couple of seconds. Verbose information is written in the Apache Tomcat console.

6 To access the Muse IDM application login in a web browser at:

```
http://Installation_HOSTNAME:PORT/idm
```

where replace `Installation_HOSTNAME` with the actual server hostname and `PORT` with the actual server port.

As a good practice for stopping the IDM application follow the rules:

- Use only the shutdown/startup sequence via Tomcat's scripts, do not terminate the process by kill/end task actions;
- After the shutdown command was issued wait as long as it takes until the Apache Tomcat has stopped. This may take even a couple of minutes, to verify that it stopped check the system's processes to make sure the Apache Tomcat is gone.

2.3.3.2 Installing the Muse IDM application into Muse Embedded Apache Tomcat as external context

The Muse IDM application is also distributed with the Muse Federated Search Platform, as external context for the Embedded Apache Tomcat version 8.5.

The steps are:

1. Make sure that the Muse IDM application is installed on machine under `MUSE_HOME/idm/www` folder.

2. Stop the Embedded Apache Tomcat server:

   ```
   CATALINA_HOME/bin/shutdown.{bat|sh}
   ```

   After the shutdown command was issued check the system's processes to make sure the Apache Tomcat process has really stopped, and only when the process has stopped issue the startup command.

3. If Security Manager is enabled for Apache Tomcat, then the following entry must be added in `MUSE_HOME/aas/java.tomcat.policy` at the end of file:

   ```
   grant codeBase "file:${MUSE_HOME}/idm/www/-"{
            permission java.security.AllPermission;
   };
   ```

4. Add following entry in `CATALINA_HOME/conf/server.xml` configuration file:

   ```
   <Context path="/idm" docBase="${MUSE_HOME}/idm/www"
   crossContext="true" xmlNamespaceAware="false" xmlValidation="false"/>
   ```
   inside the `Host` tag.

5. Make sure that the following structure of properties files with configurations for the application, MySQL database, SMTP and users exists properly configured:

   - Edit the `MUSE_HOME/idm/www/WEB-INF/classes/application.properties` file as described at section "*application.properties config file*".

   - Edit the `MUSE_HOME/idm/www/WEB-INF/classes/database.properties` file as described at section "*database.properties config file*".

   - Edit the `MUSE_HOME/idm/www/WEB-INF/classes/email.properties` file as described at section "*email.properties config file*".

   - Edit the `MUSE_HOME/idm/www/WEB-INF/classes/users.properties` file as described at "*users.properties config file*".

6 Start the Apache Tomcat server:

```
CATALINA_HOME/bin/startup.{bat|sh}
```

Upon startup the Muse IDM application is initialized, this should take a couple of seconds. Verbose information is written in the Embedded Apache Tomcat console.

7 To access the Muse IDM application login in a web browser at:

```
http://Installation_HOSTNAME:PORT/idm
```

where replace `Installation_HOSTNAME` with the actual server hostname and `PORT` with the actual server port.

2.3.3.3 License Key File Installation

When logging for the first time into the Muse IDM application with the default administrative account, as configured in the `application.properties` file, you will be redirected to a page where to upload and install the License Key File.

3.0

Advanced Configurations

This section describes advanced configurations that can be done for the Muse IDM application.

3.1 Changing Default Ports for Apache Tomcat

By default the port on which the Apache Tomcat server binds is 8080. To change it to the standard HTTP port 80 (or another port value) follow the steps:

1. Stop the Apache Tomcat server:

   ```
   CATALINA_HOME/bin/shutdown.{bat|sh}
   ```

2. Edit the file `CATALINA_HOME/conf/server.xml` file and locate the `Connector` XML node having the attribute `port="8080"`. Below is an example:

   ```
   <Connector port="8080" protocol="HTTP/1.1" connectionTimeout="20000"
   redirectPort="8443" />
   ```

3. Change the 8080 value to 80, e.g. the entry should look like below:

   ```
   <Connector port="80" protocol="HTTP/1.1" connectionTimeout="20000"
   redirectPort="8443" />
   ```

4. Edit the Muse IDM application configuration files to reflect the new port value:
   - Edit the file `IDM_HOME/WEB-INF/classes/application.properties` and change the value of the `application.url` property to reflect the port changes.

5. Start the Apache Tomcat server:

   ```
   CATALINA_HOME/bin/startup.{bat|sh}
   ```

3.2 Configuring SMTP

The Muse IDM application has email functionalities, meaning that it sends emails to the registered users. Sending of emails is done using a SMTP server.

The SMTP configuration for the Muse IDM application is done as following:

1. Stop the Apache Tomcat server:

   ```
   CATALINA_HOME/bin/shutdown.{bat|sh}
   ```

2. Edit the `IDM_HOME/WEB-INF/classes/email.properties` text file by using the in structions from the "*email.properties config file*" section.

   Make sure the correct SMTP values are configured in the `email.properties` file according with the instructions from the Appendix.

3. If the SMTP server is configured SSL or TLS authentication and if the certificate used by the server is not signed by a well known authority, then the server's certificate must be added into the truststore file used by the Muse IDM application, which is the one from JDK - `$JAVA_HOME/jre/lib/security/cacerts`.

   Below are the steps for doing this:

   - Obtain the server's certificate and store it into a `*.crt` or `*.cer` file. This can be done using a keystores/certificates manager tool such as CERTivity KeyStores Manager , or it can be retrieved from the email client, or by using advanced shell commands with the `openssl` tool.

   - Import the certificate into java's trustore using the keytool utility that comes with the JDK package:

     ```
     keytool -importcert -file your_certificate_file -keystore
     "$JAVA_HOME/jre/lib/security/cacerts" -alias "alias_name"
     ```

     where replace `your_certificate_file` with the actual name of the certificate file from the previous step and as `alias_name` a proper name for identifying the certificate inside the truststore (for example it can be the server's hostname).

Note: The `JAVA_HOME` variable must be defined and pointing to the current JDK installation. If not then replace it with the exact path pointing to the JDK installation folder.

Note: When asked for password by the `keytool` utility use `changeit`.

4 Start the Apache Tomcat server:

```
CATALINA_HOME/bin/startup.{bat|sh}
```

3.3 Configuring DBMS

The Muse IDM application use DBMS to store users and their details.

The database configuration for the Muse IDM application is done as following:

1 Stop the Apache Tomcat server:

```
CATALINA_HOME/bin/shutdown.{bat|sh}
```

2 Edit the `IDM_HOME/WEB-INF/classes/database.properties` text file by using the in structions from the "*database.properties config file*" section.

Make sure the correct database values are configured in the `database.properties` file according with the instructions from the Appendix.

3 Start the Apache Tomcat server:

```
CATALINA_HOME/bin/startup.{bat|sh}
```

3.4 Configuring Email Templates

The Muse IDM application has support to send emails to users and administrators of the system on different actions such as validation of user email on registration process, to notify administrators about to expire accounts or to notify users that their accounts was modified by an administrator etc. These templates are written using FreeMarker Template Engine and are located by default under

`IDM_HOME/WEB-INF/classes/emailTemplates` folder.

Note: There is no need to restart the Apache Tomcat server in order that changes made in an email template to be reloaded.

The following metavariables are available to be used in the email templates:

1. `${user.fullname}` to access user full name.
2. `${user.email}` to access user email.
3. `${user.locked?c}` to access user locked state which also is a boolean value.
4. `${user.status}` to access user status.
5. `${user.enabled?c}` to access user enabled state which also is a boolean value.
6. `${user.expiryAt}` to access the expiration date of user account in the following format: YYYY-MM-DD (for example: 2019-01-30).
7. `${user.createdAt}` to access the date when user account was created in the following format: YYYY-MM-DD (for example: 2019-01-30).

Note: Also in emails you can access all configuration properties from all configuration files using variables as follows: `${applicationConfig['application.name']}` where `application.name` represent a property from `IDM_HOME/WEB-INF/classes/application.properties` con figuration file.

Note: If you want to display a boolean property as string then you need to append `?c` at the end of variable as follows: `${applicationConfig['email.starttls.enable']?c}` where `email.starttls.enable` is a key from `IDM_HOME/WEB-INF/classes/users.properties` file.

Also the `selfUrl` property is set by default in all email templates and is accessd with `${selfUrl}` variable and represent the value of `application.url` defined in `IDM_HOME/WEB-INF/classes/application.properties` configuration file.

3.5 Configuring reCAPTCHA

reCAPTCHA was introduced to protect the login and registration forms from fraud and abuse.

All configurations related with reCAPTCHA can be found in the `WEB-INF/classes/application.properties` configuration file and are represented as follow:

```
#START CAPTCHA CONFIGURATION
# The site key used by Google reCAPTCHA v2.
captcha.site.key=
# The secret key used by Google reCAPTCHA v2.
captcha.secret.key=
# The verification URL where the Google reCAPTCHA v2 whill be server side
validated.
captcha.verification.url =
https://www.google.com/recaptcha/api/siteverify?secret=%s&response=%s&remot
eip=%s
#END CAPTCHA CONFIGURATION

#START CAPTCHA ATTEMPT CONFIGURATION
# Default value is false.
captcha.attempt.enabled = true
# Default value is 3.
captcha.attempt.maxAttempts = 3
# Default value is 8192.
captcha.attempt.cacheMaximumSize = 8192
# This value is parsed as java.time.Duration. Default value is 5 minutes.
captcha.attempt.expireAfterWrite = PT5M
# This flag enable or disable captcha on login page. Default value is
false.
captcha.login.enabled = false
#END CAPTCHA ATTEMPT CONFIGURATION
```

The most important reCAPTCHA configurations are:

- `captcha.site.key` – the public key used in interface to generate the token.
- `captcha.secret.key` – the private key used for server side validation of generated token.
- `captcha.login.enabled` – flag configuration which enable or disable the reCAPTCHA on login page.

The reCAPTCHA keys are configured through the Google Administrator Console and is available here.

3.6 Configuring Themes

The Muse IDM application comes with a set of predefined interface themes to choose from, the following features are available with regard to themes:

- A default theme can be configured, by editing the `IDM_HOME/WEB-INF/classes/application.properties` configuration file and setting the theme's identifier for the `theme.defaultThemeName` property. The following themes are available:

  Amelia (ID `amelia`), Brown (ID `brown`), Dark (ID `dark`), Dark Blue (ID `darkblue`), Gray (ID `gray`), Purple (ID `purple`), White (ID `white`).

- The theme can be set/changed by passing the theme identifier as value for the `&theme` parameter in the requests.
- A `Theme` menu item is available from which the end-users can change the theme. The `Theme` menu can be disabled by setting the `menu.themes.enabled` property to `false` in the `IDM_HOME/WEB-INF/classes/application.properties` configuration file.
- New themes can be developed and added in the list of themes, to add a new theme the next steps must be followed:

  1. Create a new properties file `IDM_HOME/WEB-INF/classes/theme/THEME_ID.properties` with following structure:

     ```
     #The name of CSS file which will be loaded when THEME_ID theme
     will be selected by user.
     theme.css = mgb.css
     theme.adds.css = mgb_adds.css
     ```

     where `THEME_ID` represent the id of theme and also must be unique, the identifier of client should be used instead.

  2. Copy `mgb.css` and `mgb_ads.css` files in `IDM_HOME/WEB-INF/static-resources/css/themes` folder.
  3. You can choose to set this new theme as default by replacing the current value of `theme.defaultThemeName` property found in `IDM_HOME/WEB-INF/classes/application.properties` file as follow:

     ```
     #The default theme loaded if none was selected by user.
     theme.defaultThemeName=THEME_ID
     ```

  4. Also you can add this new theme on themes menu by editing `IDM_HOME/WEB-INF/jsp/include/header.jsp` file as follow:

a. Locate the `Themes` menu element.

b. Add a new element under `ul` element as follow:

```
<li><a href="<c:url value="?theme=THEME_ID" />">
THEME_NAME</a></li>
```

where `THEME_ID` is the id of defined theme in previous steps and `THEME_NAME` represent the name of theme which will be displayed for users in menu.

5 Restart Apache Tomcat server:

```
CATALINA_HOME/bin/shutdown.{bat|sh}
CATALINA_HOME/bin/startup.{bat|sh}
```

Note: More details about themes can be found here.

3.7 Configuring Tasks

At this moment there are only two tasks defined in the `IDM_HOME/WEB-INF/config/tasks-context.xml` configuration file:

1 Purge unavalidated emails after registration process.

After a user registration request is made, a validation email is sent to the user to validate that the registered email is indeed his. This email contains a link with a token which expires after 24 hours. The purging task runs at every 10 minutes but this can be changed by specifying another cron expression for `task.purgeUnvalidatedEmails.cron` property from `IDM_HOME/WEB-INF/classes/application.properties` file.

This task has role to delete all registered users entries who do not validate their emails in 24 hours in order to maintain a clean database.

Also the duration of a registration token can be changed from `user.token.expiry` property from `IDM_HOME/WEB-INF/classes/application.properties` file.

2 Notify administrators about accounts that are about to expire.

A registered user has associated an expiration date with his account and it represent the last day when he can authenticate in the application or in all applications associated with the Muse IDM database.

This task runs every day at 1AM and checks if there are user accounts which are about to expiry over 5 or 15 days. If such accounts are found then an email will be sent to all users wil `Administrator` role to notify about these accounts which are about to expire. Also a link is provided in the email which takes the administrator to the Muse IDM application on the users page displaying the accounts that are about to expire.

Also this task can be scheduled to run at a different time by changing the cron expression defined by `task.notifyAdministratorsAboutAccountsToExpire.cron` property from `IDM_HOME/WEB-INF/classes/application.properties` configuration file.

Default values 5 and 15 days for expiration date can be changed from `IDM_HOME/WEB-INF/classes/application.properties` file as in following example:

```
# Need to be configured with values which can be parsed by
java.util.Period.parse(CharSequence text) method as in following doc
umentation
#
https://docs.oracle.com/javase/8/docs/api/java/time/Period.html#parse
-java.lang.CharSequence-
# Specify accounts who are about to expire over 10 days.
task.notifyAdministratorsAboutAccountsToExpire.firstNotification =
P10D
# Specify accounts who are about to expire over 30 days.
task.notifyAdministratorsAboutAccountsToExpire.lastNotification =
P30D
#If P0D is provided as value for one of previous two configurations
then that property will be skipped.
```

3 Purge oldest N password history entries for all users

Starting with version `2.1.0.0`, a registered user has associated a password history. This history represents a list of password changes made by the user or by the administrator of the system and this history of password changes is kept in order to make sure that the user doesn't use the same old password when he decide to change it.

Also this task can be scheduled to run at a different time by changing the cron expression defined by `task.purgeOldestNPasswordHistoryEntriesForAllUsers.cron` property from `IDM_HOME/WEB-INF/classes/application.properties` configuration file defined bellow:

```
#START TASK WHICH PURGE LAST N PASSWORD HISTORY ENTRIES
# N is specified by password.history.max.entries.per.user con
figuration.
# Run every day at 1PM.
task.purgeOldestNPasswordHistoryEntriesForAllUsers.cron = 0 0 1 * * ?
#END TASK WHICH PURGE LAST N PASSWORD HISTORY ENTRIES
```

4.0

Muse IDM as SAML 2.0 Identity Provider

Starting with version `2.1.0.0`, Muse IDM can be configured as `Identity Provider` in SAML 2.0 authentication.

The SAML IDP is enabled by default, new Service Providers can configured in the `WEB-INF/config/saml-idp-context.xml` file.

A new Service Provider entry can be configured by adding a new `External Service Provider Configuration` bean under `providers` property list, located under `identityProvider` bean as follow:

```
<bean
class="org.springframework.security.saml.provider.identity.config.ExternalS
erviceProviderConfiguration">
  <property name="alias" value="SERVICE PROVIDER ALIAS" />
  <property name="linktext" value="SERVICE PROVIDER ALIAS" />
  <property name="skipSslValidation" value="true"/>
  <property name="metadataTrustCheck" value="false"/>
  <property name="metadata" value="URL TO SERVICE PROVIDER METADATA"/>
</bean>
```

The `metadata` property can be configured in three ways as following:

1. By providing a HTTP URL to the metadata of the Service Provider:

   ```
   <property name="metadata"
   value="https://service.provider:port/sp/metadata-location"/>
   ```

2. By providing the content of metadata as text in a `CDATA` tag as follow:

   ```
   <property name="metadata">
     <value>
       <![CDATA[
   ```

```
        Add here the content of Service Provider metadata file
      ]]>
    </value>
  </property>
```

3. By coping the metadata file in `WEB-INF/classes/saml/metadata` directory and refer it in configuration file as follow:

   ```
   <property name="metadata"
   value="https://localhost:port/saml/sp/metadata/NAME_OF_COPIED_METADAT
   A_FILE" />
   ```

   This way to configure the Service Provider metadata file is useful when there is no way to access the metadata using a URL, for example when URL is not accessible or the SSL certificate is invalid.

The others properties of the `External Service Provider Configuration` bean can be changed accordingly with Service Provider requirements.

Note: The username of authenticated user can be extracted at Service Provider level from `name` SAML attribute.

Note: Any modification done in `WEB-INF/config/saml-idp-context.xml` configuration file requires a server restart.

4.1 SAML Attribute Filter

For security reasons, in Muse IDM was added support to filter the user attributes exposed to configured SAML Service Providers.

This filtering is done through `WEB-INF/classes/saml_attribute_filter.json` configuration file which is defined as follow:

```
{
   "serviceProviderEntityID" : [
      "email",
      "affiliation",
      "userName",
      "fullName",
      "lastLogin",
```

```
      "status",
      "roles",
      "customAttribute_.*"
   ],
   "serviceProviderEntityID2" : [
      "email",
      "lastLogin",
      "customAttribute_phoneNumber"
   ],
   ...
}
```

Through this configuration file, the administrator of the system can decide which user attributes to expose to which Service Provider based on their `Entity ID's`.

In previous example, for the SAML Service Provider with `serviceProviderEntityID2` as Entity ID, only the `email`, `lastLogin` and `customAttribute_phoneNumber` details of authenticated user will be send to the `serviceProviderEntityID2` Service Provider in the SAML response as following:

```
<saml2:Attribute Name="email"
NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:unspecified">
    <saml2:AttributeValue
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:type="xsd:string">user@domain.com</saml2:AttributeValue>
</saml2:Attribute>
<saml2:Attribute Name="lastLogin"
NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:unspecified">
    <saml2:AttributeValue
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:type="xsd:anyType">2020-07-03T11:19:09.664</saml2:AttributeValue>
</saml2:Attribute>
<saml2:Attribute Name="customAttribute_phoneNumber"
NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:unspecified">
    <saml2:AttributeValue
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:type="xsd:string">0123456789</saml2:AttributeValue>
</saml2:Attribute>
```

If no attribute filter is defined for a specific Service Provider, the SAML IDP of IDM will expose only the username(email) of the authenticated user in SAML response as follow:

```
<saml2:NameID Format="urn:oasis:names:tc:SAML:2.0:nameid-format:persistent"
SP
NameQualifier="https://rambo.museglobal.ro:9443/MKPF">username(email)</saml
2:NameID>
```

All supported user attributes which can be exposed to a specified Service Provider are:

- `email`
- `affiliation`
- `userName`
- `fullName`
- `lastLogin`
- `status`
- `roles`
- `customAttribute_CUSTOM_ATTRIBUTE_NAME`

  Where the `CUSTOM_ATTRIBUTE_NAME` represents the name of the user custom attribute configured in user attributes descriptor configuration file.

Note: Keep in mind that all these attribute names are represented as regular expressions. More details about regular expressions can be found here https://docs.oracle.com/javase/8/docs/api/java/util/regex/Pattern.html .

Note: For example, if `customAttribute_.*` is configured, then all custom attributes of the au thenticated user will be send to the Service Provider.

4.2 SAML Attribute Mapping

Beside attribute filtering, in Muse IDM was added support to map the user SAML attributes exposed to configured SAML Service Providers.

This mapping is done through `WEB-INF/classes/saml_attribute_mapping.json` configuration file which is defined as follow:

```
{
  "serviceProviderEntityID" : {
    "affiliation" : "department",
    "fullName" : "cn",
    "customAttribute_phoneNumber" : "phoneNumber"
  },
  "serviceProviderEntityID2" : {
    "email" : "username",
    "customAttribute_street" : "address"
  },
  ...
}
```

Through this configuration file, the administrator of the system can expose existing SAML attribute names to the Service Provider with another SAML attribute name.

In previous example, for the SAML Service Provider with `serviceProviderEntityID` as Entity ID, the `affiliation` attribute will be exposed as `department`, `fullName` as `cn` and `cus tomAttribute_phoneNumber` as `phoneNumber` in the SAML response.

```
<saml2:Attribute Name="affiliation"
NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:unspecified">
  <saml2:AttributeValue
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:type="xsd:string">Management</saml2:AttributeValue>
</saml2:Attribute>
<saml2:Attribute Name="fullName"
NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:unspecified">
  <saml2:AttributeValue
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:type="xsd:anyType">John Doe</saml2:AttributeValue>
</saml2:Attribute>
<saml2:Attribute Name="customAttribute_phoneNumber"
NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:unspecified">
  <saml2:AttributeValue
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:type="xsd:string">0123456789</saml2:AttributeValue>
</saml2:Attribute>
```

In that case, previous user SAML attributes, after filtering is applied, will be converted into following SAML attributes:

```
<saml2:Attribute Name="department"
NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:unspecified">
  <saml2:AttributeValue
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:type="xsd:string">Management</saml2:AttributeValue>
</saml2:Attribute>
<saml2:Attribute Name="cn"
NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:unspecified">
  <saml2:AttributeValue
```

```
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:type="xsd:anyType">John Doe</saml2:AttributeValue>
</saml2:Attribute>
<saml2:Attribute Name="phoneNumber"
NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-format:unspecified">
  <saml2:AttributeValue
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:type="xsd:string">0123456789</saml2:AttributeValue>
</saml2:Attribute>
```

Note: Mapping is applied after filtering process is done, so we can say that the filtering takes precedence and should be properly configured before configuring SAML attributes mapping.

Note: All supported user attributes which can be mapped to a specified Service Provider can be found on filtering SAML attributes section.

4.3 Muse Proxy configured as Service Provider

In order to add a Muse Proxy application as Service Provider in IDM SAML Identity Provider, the following steps must be performed:

- The bean configuration must be added in IDM in `WEB-INF/config/saml-idp-context.xml` configuration file under `identityProvider`:

  ```
  <bean
  class="org.springframework.security.saml.provider.identity.config.ExternalServiceProviderConfiguration">
    <property name="alias" value="MuseProxyAppID" />
    <property name="linktext" value="MuseProxyAppID" />
    <property name="skipSslValidation" value="true" />
    <property name="metadata"
  value="https://proxy.domain:port/ssoRWP/saml/metadata/alias/MuseProxyAppID"/>
  </bean>
  ```

  Make sure to replace `MuseProxyAppID` with the actual Muse Proxy application identifier.

  Note: A restart of the Tomcat server is needed.

- Create the Muse Proxy application with the identifier as used at step #1, using the Muse Proxy

Administrator Console;

- Generate the Service Provide (SP) metadata for the Muse Proxy application. For doing this follow the instructions from the Muse Proxy Administrator Console, `Configuration` -> `SAML Authentication` -> `Metadata administration` -> `Generate new service provide metadata` section;

  Switch the authentication method of the Muse Proxy application to SAML.

- Add the IDM SAML IDP into Muse Proxy, through the Muse Proxy Administrator Console, `Configuration` -> `SAML Authentication` -> `Metadata administration` -> `Add new identity provide metadata` section. Use the `By URL` method and provide the following URL for the IDM SAML Identity Provider metadata:

  ```
  https://idm.domain:port/saml/idp/metadata
  ```

  If the Muse IDM application is running in the same web server (Tomcat) as the MuseKnowledge Search application, the URL becomes:

  ```
  https://idm.domain:port/idm_webapp/saml/idp/metadata
  ```

  where replace `idm_webapp` with the correct value.

- In Muse IDM, review the attributes exposed by editing the `WEB-INF/classes/saml_attribute_filter.json` configuration file. Make sure the proper Service Provider Entity ID corresponding to the Muse Proxy application is used.

4.4 MuseKnowledge Search application configured as Service Provider

In order to add a MuseKnowledge Search application as Service Provider in IDM SAML Identity Provider, the following steps must be performed:

- In Muse IDM, the below bean configuration must be added in `WEB-INF/config/saml-idp-context.xml` configuration file under `identityProvider` bean:

  ```
  <bean
  class="org.springframework.security.saml.provider.identity.config.ExternalServiceProviderConfiguration">
    <property name="alias" value="MuseKnowledgeAppID" />
    <property name="linktext" value="MuseKnowledgeAppID" />
  ```

```
  <property name="skipSslValidation" value="true" />
  <property name="metadata"
value="https://muse.domain:port/muse/saml/metadata" />
</bean>
```

Note: A restart of the Tomcat server is needed.

- Follow the `Muse Web Bridge with SAML` manual, chapter `Setup a Muse Search Application to Authenticate using SAML` to configure the MuseKnowledge Search application with SAML authentication.
- Add the IDM SAML IDP into MuseKnowledge Search, through the administration console

  `https://search.domain:port/muse/saml/web/metadata/login`

  `Add new identity provider metadata`. Use the `By URL` method and provide the following URL for the IDM SAML Identity Provider metadata:

  `https://idm.domain:port/saml/idp/metadata`

  If the IDM application is running in the same web server (Tomcat) as the MuseKnowledge Search application, the URL becomes:

  `https://idm.domain:port/idm_webapp/saml/idp/metadata`

  where replace `idm_webapp` with the correct value.
- In Muse IDM, review the attributes exposed by editing the `WEB-INF/classes/saml_attribute_filter.json` configuration file. Make sure the proper Service Provider Entity ID corresponding to the MuseKnowledge Search application is used (default is https://search.domain:port/muse).

4.5 Rocket.Chat configured as Service Provider

In order to configure an instance of Rocket.Chat as Service Provider in IDM SAML Identity Provider, the following steps must be performed:

- In IDM, the below bean configuration must be added in `WEB-INF/config/saml-idp-context.xml` configuration file under `identityProvider` bean:

```
<bean
```

```
class="org.springframework.security.saml.provider.identity.config.Exte
rnalServiceProviderConfiguration">
  <property name="alias" value="Rocket.Chat" />
  <property name="linktext" value="Rocket.Chat" />
  <property name="skipSslValidation" value="true" />
  <property name="metadata"
value="https://rocket.chat.domain:port/_saml/metadata/${CUSTOM_PROVIDE
R}" />
</bean>
```

Note: Every `${CUSTOM_PROVIDER}` variable found need to be replaced with the value configured in `Rocket.Chat` SAML administration console configured in `Custom Provider` input field.

Note: A restart of the Tomcat server is needed.

- Configure the IDM as Identity Provider into `Rocket.Chat`, through his SAML administration console as follows:

  The value configured in `Custom Provider` input field represent an identifier used by `Rocket.Chat` to generate the SP metadata and the value set here should replace every `${CUSTOM_PROVIDER}` variable found in all configuration URL's.

  The following value need to be configured for `Custom Entry Point` input field:

  ```
  https://idm.domain:port/idm_webapp/saml/idp/SSO/alias/${IDM_IDP_ENTITY
  _ID}
  ```

  The following value need to be configured for `IDP SLO Redirect URL` input field:

  ```
  https:/idm.domain:port/idm_webapp/saml/idp/logout/alias/${IDM_IDP_ENTI
  TY_ID}
  ```

  Note: The `${IDM_IDP_ENTITY_ID}` need to be replaced with the value configured for `entityId` property of `identityProvider` bean defined in `WEB-INF/config/saml-idp-context.xml` configuration file.

  The following value need to be configured for `Custom Issuer` input field:

  ```
  https://rocket.chat.domain:port/_saml/metadata/${CUSTOM_PROVIDER}
  ```

- In Muse IDM, review the attributes exposed by editing the `WEB-INF/classes/saml_attribute_filter.json` configuration file.

This configuration file must be configured to expose the email and fullname of authenticated user in SAML attributes as follow:

```
{
  "https://rocket.chat.domain:port/_saml/metadata/${CUSTOM_PROVIDER}":
[
                "email",
        "fullName"
  ]
}
```

- In Muse IDM, review the mapped attributes exposed by editing the `WEB-INF/classes/saml_attribute_mapping.json` configuration file.

This configuration file should map the `fullName` SAML attribute defined in IDM to `cn` SAML attribute name which is accepted by `Rocket.Chat` SP.

```
{
  "https://rocket.chat.domain:port/_saml/metadata/${CUSTOM_PROVIDER}":
{
    "email": "email",
    "fullName": "cn"
  }
}
```

Note: The value of `${CUSTOM_PROVIDER}` need to be replaced with the value configured in `Rocket.Chat` SAML administration console, defined in `Custom Provider` input field.

Note: If the Muse IDM application is running in a Servlet Container and the context path is set for example to `idm`, then `idm_webapp` should be replaced by `idm`.

Note: More details about SAML configuration in `Rocket.Chat` administration console can be found here.

5.0

Muse IDM as OAuth2 Authorization Server

Starting with version `2.1.0.0`, Muse IDM has support to act as `OAuth2 Authorization Server` for OAuth2 authentication.

A new menu item named `OAuth2` was added in the Muse IDM application where helpful links can be found which also helps the administrators manage OAuth2 client details.

Also on `Clients` page can be found the `Authorization`, `Token` and `User Info` endpoints which need to be configured at client level.

These configuration links are:

- `Authorization Endpoint` : https://idm.domain:port/oauth/authorize
- `Token Endpoint` : https://idm.domain:port/oauth/token
- `User Info Endpoint` : https://idm.domain:port/oauth/userinfo

Note: Keep in mind that the Client Details secret is kept hashed in database and at client level the original value must be configured. There is no way to recover the original value, if the original value is lost, then the Client Secret can be updated from Edit Client Details page with a new value.

Note: IDM was successfully integrated and tested as `OAuth2 Authorization Server` with `Muse Proxy`, `miniOrange`(an OAuth2 client for WordPress) and `Moodle 3.7.1`.

Note: The username can be extracted from `name` attribute of `userinfo` response.

6.0

Muse IDM with HMAC authentication

Muse IDM can be configured to authenticate `Muse Proxy` and `Muse Knowledge` products by redirecting the authenticated users to these products with a generated HMAC link. Also another type of web applications can be configured with this type of authentication if these recognize and can validate the received HMAC link.

The pages where the HMAC can be configured for `Muse Proxy` and `Muse Knowledge` products can be found in `WEB-INF/jsp/access` directory. There exists two JSP pages as follow:

1. `muse.jsp` web page which is used to generate the HMAC link recognized by a `Muse Knowledge` application.
2. `proxy.jsp` web page which is used to generate the HMAC link recogized by a `Muse Proxy` application.

Under the same directory, also exists the `client.jsp` web page which contains a simple HMAC integration and can be easily changed accordingly with client requirements.

In both pages is necessary to configure the required authentication details such as `WEB APPLICATION URL`, `algorithm`, `secret` and `parameters` which also need to be a perfect match with details configured at application level where the IDM will redirect the successfully authenticated user.

Note: On `WEB-INF/config/security-context.xml` configuration file, under `redirectUrls` property, from `customLogoutSuccessHandler` bean, can be configured a map with allowed redirect URL used by IDM to redirect the authenticated user to those URL's based on `clientID` parameter name on a logout request with `clientID` parameter.

Note: An example of logout request is: `https://idm.domain:port/logout?clientID=IDMHMAC`, in that case, after accessing previous link, the authenticated user will be redirected to the URL identified by `IDMHMAC` key from `re directUrls` map.

7.0

Internationalization support

Starting with version`2.1.0.0`, Muse IDM is fully internationalized through messages files located under `WEB-INF/classes/i18n` directory.

The structure of `WEB-INF/classes/i18n` internationalization directory can be represented as following:

- `core` - there are all messages which are read by Server and are send as response for different requests such as user deletion, successful registration etc. .
- `security` - there are all messages related with authentication failure such as Invalid username, Account blocked etc. .
- `web` - these are all messages refered in JSP files for titles, descriptions, labels or table headers etc. .

In order to add a new locale the following steps are required:

- Make a copy of english `messages.properties` from `WEB-INF/classes/i18n` directory and append the locale at the end of properties file name as follow: `messages_es.properties` for spanish language.
- Make a copy of english email templates from `WEB-INF/classes/emailTemplates` directory and append the locale at the end of email template file name as follow: `ac countApproved_es.ftl` for spanish language.
- Translate the content of copied files in required language(to follow the examples, then the files need to be translated in spanish in that case). For `messages.properties` files, only the value need to be translated, the key must not change, but the entire content of email templates need to be translated.

All configurations related with internationalization can be found in `WEB-INF/classes/application.properties` configuration file. These configurations are:

```
#START LOCALIZATION CONFIGURATION
# The locale parameter name used to change the locale. For example if is
required to change the language to spanish, a request with languageCode=es
parameter will achieve this.
locale.paramName = languageCode
# Used to specify the default locale.
locale.default = en
# With which encoding the localization files will be read.
locale.defaultEncoding = UTF-8
locale.useCodeAsDefaultMessage = true
locale.cacheSeconds = 0
#END LOCALIZATION CONFIGURATION

# List with all supported locales:
en,ro,ar,el,es,fr,ja,nl,de,tr,zh_CN,zh_TW
# Empty locale list will no display any entry on menu.
locale.enabled =  en,ro,ar,el,es,fr,ja,nl,de,tr,zh_CN,zh_TW
```

8.0

Theme Support

The look and feel of the Muse IDM application is controlled by a theme, several predefined theme are available and can be set as the default theme.

Configurations related to the application's theme can be found in `WEB-INF/classes/application.properties` configuration file and are represented by following con figuration entries:

```
#START THEME CONFIGURATION
# The name of parameter used to change the theme. For example if is
required to change the theme to gray, a request with theme=gray parameter
will achieve this.
theme.paramName = theme
# Used to speciofy the default theme.
theme.defaultThemeName = default
# The name of directory from WEB-INF/classes directory where all themes are
conbfigured in properties files.
theme.baseNamePrefix = theme/
theme.cookieName = theme
#END THEME CONFIGURATION

# Flag configuration used to display or not the Themes menu entry on web
interface
themes.menu.enabled = true
```

In order to add a new theme the following steps are required:

- Add a new properties file in `WEB-INF/classes/theme` directory with the name of the theme represented by `THEME_NAME` variable with following content:

  ```
  theme.css = THEME_NAME.css
  theme.adds.css = THEME_NAME_adds.css
  theme.custom.css = THEME_NAME_custom.css
  ```

- Make sure that in the `WEB-INF/static-resources/css/themes` directory exists the following files: `THEME_NAME.css`, `THEME_NAME_adds.css` and `THEME_NAME_custom.css` whiche were already provided in the theme configuration file defined on previous step.
- Add the new configured theme under `Themes` header entry from `WEB-INF/jsp/include/header.jsp` file.

Note: The theme name is represented by the name of a properties files from `WEB-INF/classes/theme` directory.

Note: Supported themes out of the box are: `amelia`, `brown`, `dark`, `darkblue`, `default`, `gray`, `purple`, and `white`.

9.0

User Custom Attributes

Starting with version 2.1.0.0, was added support for custom attributes for all users which register within the Muse IDM application.

These custom attributes are configured in the user attributes descriptor configuration file found on WEB-INF/classes/user_attributes_descriptor.json JSON configuration file.

This user attributes descriptor file can be represented as following:

```
{
   "descriptors":[
      {
         "name":"department",
         "required":true,
         "defaultValue":null,
         "dataType":"SELECT",
         "adminOnly":true,
         "values":{
            "Production":"Production",
            "Purchasing":"Purchasing",
            "Human Resource Management":"Human Resource Management",
            "Accounting and Finance":"Accounting and Finance",
            "Research and Development":"Research and Development"
         },
         "properties":{
            "iconClassName":"fa fa-list fa-fw"
         },
         "errorCode":"custom_attributes.department.error",
         "pattern":null
      },
      {
         "name":"position",
         "required":true,
         "defaultValue":null,
         "dataType":"TEXT",
            "adminOnly":false,
            "properties":{
               "iconClassName":"fa fa-clipboard fa-fw"
            },
            "errorCode":"custom_attributes.position.error",
            "pattern":null
        },
        {
            "name":"mobileNumber",
            "required":true,
            "defaultValue":null,
            "dataType":"NUMBER",
            "properties":{
               "iconClassName":"fa fa-phone fa-fw"
            },
            "errorCode":"custom_attributes.mobileNumber.error",
"pattern":"^(\\+\\d{1,2}\\s)?\\(?\\d{3}\\)?[\\s.-]?\\d{3}[\\s.-]?\\d{4}$"
        },
        {
            "name":"accessReason",
            "required":true,
            "defaultValue":null,
            "dataType":"TEXT_AREA",
            "properties":{
               "iconClassName":"fa fa-universal-access fa-fw"
            },
            "errorCode":"custom_attributes.accessReason.error",
            "pattern":null
        }
    ]
}
```

One important configuration of a custom field is `adminOnly` which is a boolean configuration value used to specify that the custom attribute can be edited only by the users with `ADMIN` and `SUPER_ADMIN` roles. Default value of this configuration is `false`.

Internationalization of these custom attributes is done through `WEB-INF/classess/i18n/web/messages.properties` internationalization file as follow:

```
custom_attributes.mobileNumber.label = Mobile number
custom_attributes.mobileNumber.placeholder = Your mobile number
custom_attributes.mobileNumber.hint = Following formats are accepted:
0123456789, 012-3456-789 or 012 345 6789.
custom_attributes.mobileNumber.error = Provided mobile number is invalid.
```

As you can see the attribute name `mobileNumber` is prefixed by `custom_attributes` and a label, placeholder, hint and error message is provided for this attribute.

For each attribute defined in descriptor file, it must exist defined at least the `label`, `placeholder`, `error` and `errorCode` messages.

Note: The attribute name must be unique and not null.

Note: All changes from user attributes descriptor file doesn't require a server restart.

Note: The `errorCode` defined for an attribute descriptor need to exists in the `messages.properties` file. This error message will be displayed when the value cannot be validated against the configured pattern.

Note: If the user custom attributes are not necessary for a client, the content of user attributes descriptor file need to be replaced with following content:

```
{
	"descriptors": []
}
```

10.0 Limit Login Attempts

Limit login attempts is a new feature which was introduced in `2.1.0.0` version and which limits the number of retry attempts when a user tries to log in (this is based on client IP address and username).

This new feature blocks attackers which try to brute force the login form by submitting different combinations of username and password in order to guess the access details of a real user from Muse IDM. In these type of attacks, the attacker aims to access the protected resources of the application.

The login limits feature can be configured through the following entries from `WEB-INF/classes/application.properties` configuration file:

```
#START LOGIN ATTEMPT CONFIGURATION
# Flag configuration which enable or disable the login attempt feature.
Default value is false.
login.attempt.enabled = true
# This configuration specifies the maximum number of failed login attempts
after which the login process will be blocked for user who pass this value.
Default value is set to 3.
login.attempt.maxAttempts = 3
# The size of the In-Memory Cache which will hold all login attempts. After
this size is exceeded, the oldest entries will be removed. Default value is
8192.
login.attempt.cacheMaximumSize = 8192
# This configuration represents the amount of time the user will be
blocked, elapsed from the last failed login attempt submitted in the same
period of time.
# For example if this configuration is set to 5 minutes and a user will
send invalid credentials 3 times or more(the number of maximum attempts set
for "login.attempt.maxAttempts" configuration) in 5 minutes, then the user
will be blocked for 5 minutes after last failed login.
# This value is parsed as java.time.Duration. Default value is 5 minutes.
login.attempt.expireAfterWrite = PT5M
#END LOGIN ATTEMPT CONFIGURATION
```

11.0

Session Management

The session management feature was introduced in `2.1.1.0` version and is used to limit the maximum number of concurrent sessions per user. Support for realtime statistics is also available, such as: current active sessions or most active users based on number of sessions and many more.

Besides global configurations, each user has associated a maximum concurrent sessions configuration field which is visible only for the administrators of the system.

This can be configured through following configuration entries from the `WEB-INF/classes/application.properties` configuration file defined as follow:

```
#START SESSION MANAGEMENT CONFIGURATION
# Global configuration which sets the maximum number of concurrent sessions allowed for each user.
# Also each user can have configured a maximum number of concurrent sessions and this value will take precedence and will be used instead of this global value.
# This default value will be used if the value configured for maximum concurrent sessions for a user is set to 0.
sessionManagement.maximumConcurrentSessionsPerUser = -1
# Flag configuration used to specify if an exception will be trown when configured number of concurrent sessions is exceeded.
# If true, it will display an error message on login page and the user can't authenticate anymore.
# If false, the user will be authenticated and his associated oldest sessions will be invalidated.
sessionManagement.exceptionIfMaximumSessionsExceeded = true
# User session expiration expressed in seconds.
sessionManagement.maxInactiveIntervalInSeconds = 3600
# The page where an authenticated user will be redirected if his session expires.
sessionManagement.expiredSessionUrl = /expired-session
#END SESSION MANAGEMENT CONFIGURATION
```

12.0 Payment Feature

Starting with version `2.1.0.5`, in Muse Identity Manager was added support to configure a `Payment Platform`. This platform will be used by newly registered users to pay a subscription fee before accessing Muse IDM resources: `Client Applications` and `User Profile Page` and the external ap plications .

The configurations for the payment integrations are in the `WEB-INF/classess/application.properties` configuration file.

This configuration is defined as follow:

```
# The type of payment used after a successful registration. Supported
values are: paystack, stripe, verifone and none. If no value is set then
the default one will be "none".
registration.payment.type = none
```

In that case, before configuring one of `Paystack` or `Stripe` Payment Providers, the administrator will configure first the value of `registration.payment.type` configuration. For example if the client use `Stripe` as Payment Provider, the administrator will configure `registration.payment.type = stripe`. If no payment is required then the value of this configuration should be set to `none`.

The Payment flow is described through following diagram:

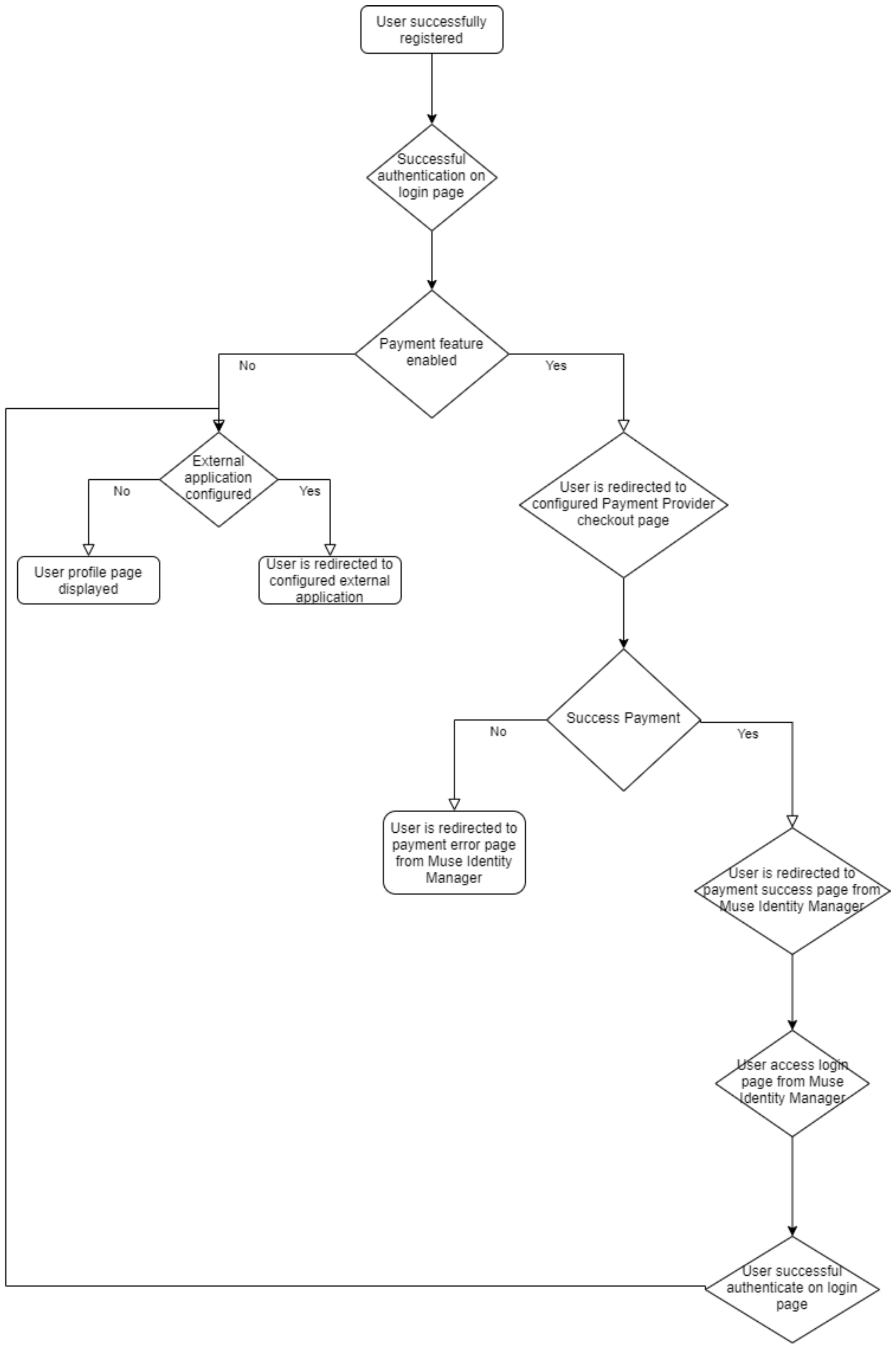
User successfully registered
Successful authentication on login page
Payment feature enabled
No
Yes
External application configured
No
Yes
User profile page displayed
User is redirected to configured external application
User is redirected to configured Payment Provider checkout page
Success Payment
No
Yes
User is redirected to payment error page from Muse Identity Manager
User is redirected to payment success page from Muse Identity Manager
User access login page from Muse Identity Manager
User successful authenticate on login page

Figure 1. Payment Flow Diagram

12.1 Stripe Integration

The payment feature was successfully tested with Stripe Payment Platform and can be configured through following configuration file: `WEB-INF/classess/payment_stripe.json`.

This configuration file is described as follow:

```
{
        "enabled": true,
        "publicKey": "pk_*",
        "secretKey": "sk_*",
        "webhookKey": "whsec_*",
        "domain": "https://idm.domain:port/idm_webapp/payment/stripe",
        "priceId": "price_*"
}
```

The `enabled` configuration entry is described as a flag configuration and is used to enable or disable the `Stripe Payment` feature. Default value for this configuration is `false` if no value is provided.

The `publicKey`, `secretKey`, `webhookKey` and `priceId` configurations can be found in Stripe Dashboard administration console.

The `priceId` is generated when a new product is added through Stripe Dashboard.

The `domain` configuration need to be properly configured based on current URL of Muse IDM and this configuration will be used to create the `Success URL` (the Muse IDM page where user will be redirected after a successful payment) and `Cancel URL` (the Muse IDM page where user will be redirected if the payment is cancelled).

Because Muse IDM is based on `Stripe Webhook Events` to update the payment status of newly registered users, in Stripe Dashboard, the administrator must configure a Stripe Webhook with the following URL:

```
https://idm.domain:port/idm_webapp/payment/stripe/webhook
```

where replace `idm_webapp` with the correct value.

and following event types:

- invoice.paid
- subscription_schedule.expiring

Note: All changes from `payment_stripe.json` configuration file don't require a server restart.

12.2 Paystack Integration

Starting with `2.1.0.7` version, in Muse Identity Manager was added support for Paystack Payment Platform and can be configured through the following configuration file `WEB-INF/classess/payment_paystack.json`.

This configuration file is described as follow:

```
{
        "enabled": false,
        "publicKey": "",
        "secretKey": "",
        "domain": "https://idm.domain:port/idm_webapp/payment/paystack",
        "oneTimePayment": {
                "amount": "10000", // this value is equivalent to 100 NGN.
                "userAccountExpiryAfter": "P2Y" // after a successfull
One-Time Payment the user account will expiry after two years.
        },
        "plans": [
                {
                        "code": "PLN_xxxxxxxxx",
                        "name": "Monthly Plan"
                },
                {
                        "code": "PLN_yyyyyyyyy",
                        "name": "Yearly Plan"
                }
        ]
}
```

The `enabled` configuration entry is a flag used to enable or disable the `Paystack Payment` feature. Default value for this configuration is `false` if no value is provided.

The `publicKey` and `secretKey` configurations can be found in Paystack Dashboard administration console.

The `domain` configuration need to be properly configured based on current URL of Muse IDM and this configuration will be used to create the `Success URL`(the Muse IDM page where user will be redirected after a successful payment) and `Cancel URL`(the Muse IDM page where user will be redirected if the user cancel his payment).

Because Muse IDM is based on `Paystack Webhook Events` to update the payment status of newly registered users, in Paystack Dashboard, the administrator must configure a `Paystack Webhook` with following URL:

```
https://idm.domain:port/idm_webapp/payment/paystack/webhook
```

Note: The `idm_webapp` should be replaced with the correct value if Muse IDM runs in Tomcat in `webapps` directory.

Following webhook event types are supported for this integration:

- `charge.success`
- `subscription.disable`

The Paystack Payment Integration can be configured in two ways:

1. `One-Time Payment` with a fixed amount.

   To enable this type of payment, in the configuration file must exists the `oneTimePayment` con figuration defined as follow:

   ```
   "oneTimePayment": {
           "amount": "10000", // this value is equivalent to 100 NGN.
           "userAccountExpiryAfter": "P2Y" // after a successfull One-
   Time Payment the user account will expiry after two years.
   }
   ```

   The value for `amount` configuration is represented as cents and configured amount must be multiplied by `100`, for example the `10000` value will be equivalent with `100 NGN`.

   The value of `userAccountExpiryAfter` is parsed as `java.time.Period` and describe the period of time until the user account will be enabled after a successful payment.

   Note: This type of payment will be used only if there is `no` value set for `plan` in configuration.

2 `Recurring Payment` using `Paystack Plans`.

For this integration, multiple plans can be configured, which can be created from Paystack Dashboard Plans dashboard page.

Any plan created will have a code with following form `PLN_xxxxxxxxx` and this value should be copied and configured under `plans` configuration from JSON configuration file.

One plan configuration is described as follows in configuration file under `plans` configuration entry:

```
{
        "code": "PLN_xxxxxxxxx",
        "name": "Yearly Plan"
}
```

The value for `code` represents the plan code generated in the `Paystack` dashboard plans page.

The value set for `name` is used only to easily distinguish between the plans.

To customize the messages from intermediary payment page, the page where a user can select a plan, the following entries from `WEB-INF/classes/i18n/web/messages.properties` localization file must be changed accordingly:

```
payment.paystack.plan.PLN_xxxxxxxxx.name = Daily Plan
payment.paystack.plan.PLN_xxxxxxxxx.description = The daily plan is an
automatically recurring payment. You will be billed once every day until
you wish to cancel.
payment.paystack.plan.PLN_xxxxxxxxx.amount = 100 NGN

payment.paystack.plan.PLN_yyyyyyyyy.name = Monthly Plan
payment.paystack.plan.PLN_yyyyyyyyy.description = The monthly plan is an
automatically recurring payment. You will be billed once every month until
you wish to cancel.
payment.paystack.plan.PLN_yyyyyyyyy.amount = 1.000 NGN
```

Previous messages are constructed based on configured plan codes, so in that case the `PLN_xxxxxxxxx` and `PLN_yyyyyyyyy` need to be modified accordingly with configured plan codes in `payment_paystack.json` configuration file.

If `One-Time Payment` is used, then following messages need to be changed accordingly, located in the same localization file as previous messages:

```
# Start Paystack Payment - One-Time Payment messages
payment.paystack.one_time_payment.name = One Time Payment
payment.paystack.one_time_payment.description = One Time Payment
payment.paystack.one_time_payment.amount = 2000 NGN
```

Note: The `One-Time Payment` and `Recurring Payments` configurations are exclusive, if both are properly configured then `Recurring Payments` takes precedence. In order to use only the `One-Time Payment` then the `plans` configuration entry need to be removed from configuration file.

Note: All changes from `payment_paystack.json` configuration file don't require a server restart.

12.3 Verifone Integration

Starting with `2.1.0.7` version, in Muse Identity Manager was added support for Verifone Payment Platform and can be configured through following configuration file `WEB-INF/classess/payment_verifone.json`.

This configuration file is described as follow:

```
{
        "enabled": false,
        "secretKey": "",
        "domain": "https://idm.domain:port/idm_webapp/payment/verifone",
        "testMode": false,
        "productId": "",
        "merchantCode": ""
}
```

The `enabled` configuration entry is described as a flag configuration and is used to enable or disable the `Verifone Payment` feature. Default value for this configuration is `false` if no value is provided.

The `testMode` configuration mode is described as a flag configuration and is used only for testing purposes.

The `secretKey`, `productId` and `merchantCode` configuration values can be found in Verifone Dashboard. The `productId` represent the product identifier from `Verifone Dashboard` platform and has the following form `12345678`(not to be confused with product code).

The `domain` configuration need to be properly configured based on current URL of Muse Identity Manager and this configuration will be used to create the `Success URL`(the Muse IDM page where user will be redirected after a successful payment) and `Cancel URL`(the Muse IDM page where user will be redirected if the user cancel his payment).

Note: Also the `idm_webapp` should be replaced with the correct value if Muse IDM runs in Tomcat in `webapps` directory.

Following changes must be done in `Verifone Dashboard` for this integration with Muse Identity Manager:

- Under `Setup` menu item, `Products` menu item can be found ant this link will redirect the ad ministrator to all products page where a new product must be added with required payment details.

  Generated product identifier will be configured in `payment_verifone.json` configuration file for this integration.

- In `Integrations` page, under `IPN settings` tab, following `IPN URL` should be configured.

  ```
  https://idm.domain:port/idm_webapp/payment/verifone/webhook
  ```

- In `Integrations` page, under `IPN settings` tab, following `Response tags` should be checked as well: `AVANGATE_CUSTOMER_REFERENCE` and `EXTERNAL_CUSTOMER_REFERENCE`.

Note: All changes from `payment_verifone.json` configuration file don't require a server restart.

13.0 User Password History

The Password History feature aims to keep a history of user password changes and is used to determine the number of unique passwords a user must use before they can use an old password again. This is an important setting because password reuse is a common issue – the more often the same (or similar) password is used, the greater chance that the password will be compromised in some way, for example through brute force.

This feature can be configured in `WEB-INF/classess/application.properties` configuration file as follow:

```
# The maximum password history which will be kept for an user represented
as integer. Default value is 5.
password.history.max.entries.per.user = 5
```

Note: Keep in mind that this feature is related with a database task from Configuring Tasks section which purge the latest N login history for all users.

14.0

User Password Expiration

Password Expiry is a mechanism used to request users to change their passwords after a specified period of time, for example 3 or 6 months. The default value for password expiration is 3 months.

This feature is implemented in Muse IDM by redirecting the user with an expired password to a page, with limited rights, where he can set a new password. After a successfull password changing, the new password will be saved in his password history. Also here, when the user tries to set a new password, he can't use an old password, because the rules defined by Password History feature are applied.

The user password lifetime can be configured in `WEB-INF/classess/application.properties` configuration file as follows:

```
# The configuration which set the user password life time. Is parsed as
java.util.Period and more details about the format of this configuration
can be found here:
https://docs.oracle.com/javase/8/docs/api/java/time/Period.html#parse-java.
lang.CharSequence-. Default value is P3M which represent 3 months.
user.password.life.time = P3M
```

Note: The user can't authenticate in Muse IDM until he changes his expired password.

15.0

Automatic Registration for Allowed Emails

This feature was introduced in version `2.1.0.7` and represents a new registration flow which is used to automatically register a user if the email he used for self-registration matches the configured allowed email domains.

When this registration flow is enabled, the registered emails which don't match the configured allowed domains, still can register but they need administrator approval.

This configuration takes precedence over `registration.admin.approval` configuration.

To enable this registration flow, the following configuration entry from `WEB-INF/classess/application.properties` file need to be modified as following:

```
# Flag used to automate the self registration to not require the admin
approval when registered email matches configured allowed emails under "re
gistration.allowed.email" configuration entry.
# At least one allowed email need to be configured when this configuration
is enabled, otherwise the application will not start.
# Default value is set to false.
registration.automatic.for.allowed.emails = true
```

Note: At least one allowed email domain need to be configured when this configuration is enabled, otherwise the application will not start.

This registration flow is described in following flow diagram:

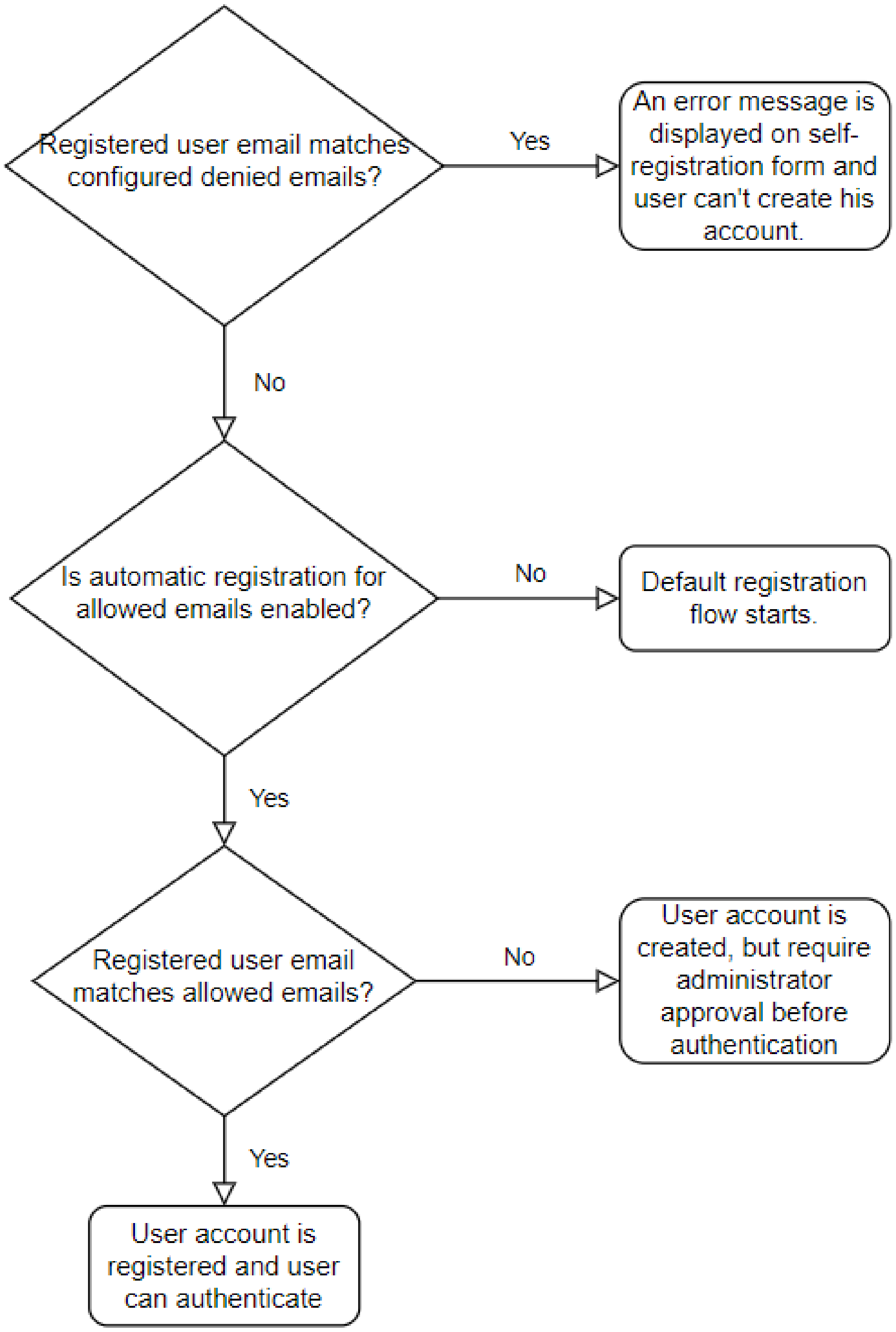
Registered user email matches configured denied emails?
Yes
An error message is displayed on self-registration form and user can't create his account.
No
Is automatic registration for allowed emails enabled?
No
Default registration flow starts.
Yes
Registered user email matches allowed emails?
No
User account is created, but require administrator approval before authentication
Yes
User account is registered and user can authenticate

Figure 2. Automatic Registration for Allowed Emails Flow Diagram

16.0

Registration Advanced Field Validation

Because some user fields requires advanced validation on registration forms, it was added support to configure regular expressions to validate these fields.

Currently it is supported to specify patterns (regular expressions) for the following fields:

- User Full Name
- User Password

These patterns can be configured in `WEB-INF/classess/application.properties` configuration file under the following configuration entries:

```
# The regular expression pattern used to validate the full name field from
self-registration and profile forms.
# If this configuration is not set then the full name should not be blank.
registration.fullName.pattern =
# The regular expression used to validate the password on registration
process.
password.pattern =
```

The `full name pattern` validation applies on the following forms:

- User profile
- Self registration

The `password pattern` validation applies on the following forms:

- User add and edit by administrator
- User profile

- Reset expired password
- Reset password
- Self registration

Note: When a pattern is created for any of these configurations make sure that are syntactically valid and properly escaped, otherwise the Muse IDM application will not start and an error will be logged.

17.0

Configure Shibboleth IDP with SQL authentication

Because the users are stored in a MySQL database and managed by the Muse IDM web application, you can configure (Shibboleth) as Identity Provider (IDP) for SAML authentication purpose.

Note: Because in a SAML authentication the time synchronization is crucial, before starting to install the necessary software you must to make sure that the time of current server is synchronized.

```
ntpdate -s time.nist.gov
```

17.1 Install Shibboleth on CentOS 7.6

The steps required to install Shibboleth on CentOS 7.6 are described below as follow:

17.1.1 Install latest JDK 8 available

In order to install the latest available JDK, follow the steps described below:

1. Remove OpenJDK if is already installed by using the following commands:

   ```
   java -version
   rpm -qa | grep openjdk
   yum remove -y java-
   1.7.0-openjdk-headless-1.7.0.91-2.6.2.1.el7_1.x86_64
   ```

2. Install the latest Java 1.8 JDK and set the `JAVA_HOME` environment variable. You can access this link for more details.

17.1.2 Install and configure Tomcat for Shibboleth

IDP

The following steps are required in order to install and configure the Tomcat Servlet Container:

1 Download the latest Apache Tomcat and install it under `opt/tomcat` directory.

2 Increase the JVM memory of Tomcat by creating the `/opt/tomcat/bin/setenv.sh` file with the following content:

```
CATALINA_OPTS="-Xmx2048m -XX:MaxPermSize=256m"
```

3 Make the newly created file executable by using the following command:

```
chmod +x /opt/tomcat/bin/setenv.sh
```

4 Make sure that 80 and 443 ports are not in use and edit `/opt/tomcat/conf/server.xml` and modify the Connector Ports as follow: (8080 -> 80 and 8443 -> 443).

5 Add the following SSL connector entry in the `/opt/tomcat/conf/server.xml` con figuration file:

```
<!-- Shibboleth -->
<Connector port="443"
protocol="org.apache.coyote.http11.Http11NioProtocol"
maxThreads="150" SSLEnabled="true" scheme="https"
    secure="true" clientAuth="want"
keystoreFile="/opt/shibboleth-idp/credentials/idp-backchannel.p12"
keystorePass="changeit" keystoreType="PKCS12"
    trustMan
agerClassName="net.shibboleth.utilities.ssl.TrustAnyCertificate" />
```

6 Replace `keystoreFile` and `keystorePass` attributes value according to your IDP install directory.

7 Download trust any SSL library and put it in the `/opt/tomcat/lib` directory:

```
cd /opt/tomcat/lib
wget
https://build.shibboleth.net/nexus/service/local/repositories/release
s/
content/
net/shibboleth/utilities/trustany-ssl/1.0.0/trustany-ssl-1.0.0.jar
```

17.1.2.1 Install Shiboleth IDP

The steps necessary to install Shibboleth IDP are:

1 Download the latest IDP Shibboleth by accessing this link in the `opt` folder as following (current version is `3.4.3`):

```
cd /opt

wget

http://shibboleth.net/downloads/identity-provider/latest/shibboleth-i
```

```
dentity-provider-3.4.3.tar.gz
tar xvfz shibboleth-identity-provider-3.4.3.tar.gz
cd shibboleth-identity-provider-3.4.3/bin/
```

2 Run `./opt/shibboleth-idp/bin/install.sh` script in order to obtain the war file which will be deployed in Tomcat.

3 The result of executing the previous script is:

```
Source (Distribution) Directory:
[/opt/shibboleth-identity-provider-3.4.3]

Installation Directory: [/opt/shibboleth-idp]

Hostname: [shibboleth.your-domain.ro]
SAML EntityID: [https://shibboleth.your-domain.ro/idp/shibboleth]

Attribute Scope: [your-domain.ro]

Backchannel PKCS12 Password:changeit
Re-enter password:changeit
Cookie Encryption Key Password:changeit
Re-enter password:changeit
Warning: /opt/shibboleth-idp/bin does not exist.
Warning: /opt/shibboleth-idp/dist does not exist.
Warning: /opt/shibboleth-idp/doc does not exist.
Warning: /opt/shibboleth-idp/system does not exist.
Warning: /opt/shibboleth-idp/webapp does not exist.
Generating Signing Key, CN = shibboleth.your-domain.ro URI =
https://shibboleth.your-domain.ro/idp/shibboleth ...
...done
Creating Encryption Key, CN = shibboleth.your-domain.ro URI =
https://shibboleth.your-domain.ro/idp/shibboleth ...
...done
Creating Backchannel keystore, CN = shibboleth.your-domain.ro URI =
https://shibboleth.your-domain.ro/idp/shibboleth ...
...done
Creating cookie encryption key files...
...done
Rebuilding /opt/shibboleth-idp/war/idp.war ...
...done

BUILD SUCCESSFUL
```

```
Total time: 2 minutes 1 second
```

17.2 Deploy Shibboleth in the Tomcat Servlet Container

The necessary steps to deploy the Shibboleth IDP in the Tomcat Servlet Container are:

1. Copy `/opt/shibboleth-idp/war/idp.war` file to `/opt/tomcat/webapps` directory.

   ```
   cp /opt/shibboleth-idp/war/idp.war /opt/tomcat/webapps
   ```

2. Start Tomcat by running the following script `./opt/tomcat/bin/startup.sh` and wait to deploy the `idp.war` web application.

3. Shutdown Tomcat by running the following script `./opt/tomcat/bin/shutdown.sh` and wait until server successfully shutdown.

4. Delete the `/opt/tomcat/webapps/idp.war` war file by using the following command:

   ```
   rm /opt/tomcat/webapps/idp.war
   ```

5. Edit `/opt/tomcat/webapps/idp/WEB-INF/web.xml` configuration file and add the following context parameter:

   ```
   <!-- #Added IDP location -->
   <context-param>
       <param-name>idp.home</param-name>
       <param-value>/opt/shibboleth-idp</param-value>
    </context-param>
   ```

6. In order to view the IDP status, add in the `/opt/shibboleth-idp/conf/access-control.xml` configuration file the allowed IPs as follow:

   ```
   <entry key="AccessByIPAddress">
       <bean parent="shibboleth.IPRangeAccessControl"
               p:allowedRanges="#{ {'127.0.0.1/32', '::1/128',
   'YOUR_IP_AS_CIDR' } }" />
   </entry>
   ```

7. Download JSTL tag library and put it into the IDP lib directory as follow:

   ```
   cd /opt/tomcat/webapps/idp/WEB-INF/lib/
   wget
   https://build.shibboleth.net/nexus/service/local/repositories/thirdpa
   rty/content/javax/servlet/jstl/1.2/jstl-1.2.jar
   ```

8. Start Tomcat and check if the status page works by accessing this link http://shibboleth.your-domain.ro/idp/status.

17.2.1 Configure Shibboleth to use the MySQL Database for authentication

The steps required to configure the MySQL authentication for the installed Shibboleth IDP are:

1. Download the MySQL Login module by accessing this link, or download the project from here, customize by your requirements and compile classes using `ant deploy` command inside of the project.
2. Copy the JAAS login module library under `/opt/tomcat/webapps/idp/WEB-INF/lib` directory.
3. Download the MySQL JDBC driver for your DBMS. For MySQL the latest available driver is `mysql-connector-java-8.0.17.jar` and can be downloaded from here.
4. Copy the driver library under `/opt/tomcat/webapps/idp/WEB-INF/lib` directory.

   Note: You can copy these two libraries `jaas_relational_login.jar` and `mysql-connector-java-8.0.15.jar` under `/opt/shibboleth-idp/edit-webapp/WEB-INF/lib` directory.

   Note: Run `/opt/shibboleth-idp/bin/build.sh` script to rebuild the IDP web application. Deploy the newly built application into `/opt/tomcat/webapps` directory by following the steps described in this section.

5. Create or replace the content of the `/opt/shibboleth-idp/conf/authn/jaas.config` configuration file with the following content:

```
ShibUserPassAuth {
    relationalLogin.DBLogin required debug=true
    dbDriver="com.mysql.cj.jdbc.Driver"
    userTable="USERS_TABLE_NAME"
    userColumn="USER_NAME_COLUMN_NAME"
    passColumn="PASSWORD_COLUMN_NAME"
    dbURL="jdbc:mysql://localhost:3306/IDM_DATABASE"
    dbUser="DATABASE_USER_NAME"
    dbPassword="DATABASE_USER_PASSWORD"
    hashAlgorithm="SHA-512"
    saltColumn=""
    errorMessage="Invalid password";
};
```

6. Edit previous `JAAS` configuration file as required including the relevant data according to your DBMS configuration.

   Note: If in your users table you have a salt stored in a column, set it in saltColumn, parameter, otherwise leave it blank, or omit it. The salted hash will be calculated as hash(password + salt).

   Note: In the case you wish to use cleartext password, leave the `hashAlgorithm` blank or omit it. This is not recommended and IDM application does not store the plain password.

   Note: Available hashing algorithms are DBMS native hashing functions such as SHA-1, SHA-256, SHA-384 and SHA-512.

Note: More details about the MySQL authentication configuration can be found at this link.

7. Open the `/opt/shibboleth-idp/conf/authn/password_authn_config.xml` con figuration file, uncomment `<import resource="jaas-authn-config.xml">` line and comment the others import tags.
8. Restart the Tomcat Servlet Container and try to authenticate to Shibboleth IDP with a user defined in the configured database.

17.3 Configure Shibboleth to extract and export the user attributes

In order to extract the necessary user attributes from database for authenticated user and pass them to the service provider as SAML2 attributes, follow the instructions as described below:

1. Replace the content of the `/opt/shibboleth-idp/conf/attribute-resolver.xml` configuration file with the following:

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
    This file is an EXAMPLE configuration file containing lots of
commented
    example attributes, encoders, and a couple of example data
connectors.

    Not all attribute definitions or data connectors are
demonstrated, but
    a variety of LDAP attributes, some common to Shibboleth
deployments and
    many not, are included.

    Deployers should refer to the Identity Provider 3 documentation

https://wiki.shibboleth.net/confluence/display/IDP30/AttributeResolve
rConfiguration

    for a complete list of components and their options.
-->
<AttributeResolver
    xmlns="urn:mace:shibboleth:2.0:resolver"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="urn:mace:shibboleth:2.0:resolver
http://shibboleth.net/schema/idp/shibboleth-attribute-resolver.xsd">

    <!-- ========================================== -->
    <!--      Attribute Definitions                 -->
    <!-- ========================================== -->

    <!-- Schema: Core schema attributes-->
    <AttributeDefinition xsi:type="Simple" id="uid">
        <InputDataConnector ref="mySIS" attributeNames="uid"/>
        <AttributeEncoder xsi:type="SAML1String"
name="urn:mace:dir:attribute-def:uid" encodeType="false" />
        <AttributeEncoder xsi:type="SAML2String"
name="urn:oid:0.9.2342.19200300.100.1.1" friendlyName="uid"
encodeType="false" />
    </AttributeDefinition>

    <AttributeDefinition xsi:type="Simple" id="mail">
        <InputDataConnector ref="mySIS" attributeNames="mail"/>
        <AttributeEncoder xsi:type="SAML1String"
name="urn:mace:dir:attribute-def:mail" encodeType="false" />
        <AttributeEncoder xsi:type="SAML2String"
name="urn:oid:0.9.2342.19200300.100.1.3" friendlyName="mail"
encodeType="false" />
    </AttributeDefinition>


    <AttributeDefinition xsi:type="Simple" id="displayName">
        <InputDataConnector ref="mySIS" at
tributeNames="displayName"/>
        <AttributeEncoder xsi:type="SAML1String"
name="urn:mace:dir:attribute-def:displayName" encodeType="false" />
        <AttributeEncoder xsi:type="SAML2String"
name="urn:oid:2.16.840.1.113730.3.1.241" friendlyName="displayName"
encodeType="false" />
    </AttributeDefinition>

        <AttributeDefinition xsi:type="Simple"
id="organizationalUnit">
        <InputDataConnector ref="mySIS" at
tributeNames="organizationalUnit"/>
```

```
        <AttributeEncoder xsi:type="SAML1String"
name="urn:mace:dir:attribute-def:ou" encodeType="false" />
        <AttributeEncoder xsi:type="SAML2String"
name="urn:oid:2.5.4.11" friendlyName="ou" encodeType="false" />
    </AttributeDefinition>

    <!-- ========================================== -->
    <!--      Data Connectors                       -->
    <!-- ========================================== -->

    <!-- Example Relational Database Connector -->
    <DataConnector id="mySIS" xsi:type="RelationalDatabase">
        <ApplicationManagedConnection
jdbcDriver="com.mysql.cj.jdbc.Driver"
jdbcURL="jdbc:mysql://DATABASE_HOST:DATABASE_PORT/DATABASE_NAME"
            jdbcUserName="DATABASE_USERNAME"
            jdbcPassword="DATABASE_PASSWORD" />
        <QueryTemplate>
            <![CDATA[
                SELECT * FROM user WHERE email =
'$resolutionContext.principal'
            ]]>
        </QueryTemplate>

        <Column columnName="email" attributeID="uid" />
                <Column columnName="email" attributeID="mail" />
        <Column columnName="full_name" attributeID="displayName" />
                <Column columnName="affiliation"
attributeID="organizationalUnit" />
    </DataConnector>

</AttributeResolver>
```

Update the `DATABASE_HOST`, `DATABASE_PORT`, `DATABASE_NAME`, `DATABASE_USERNAME` and `DATABASE_PASSWORD` configurations with the same values provided in the previously configured `JAAS` login module.

Note: The previous `ShibUserPassAuth` JAAS login module and the `DataConnector` con figuration entry from `/opt/shibboleth-idp/conf/attribute-resolver.xml` file should share the same database connection, that means they use the same database configuration such as the same username, password and url.

2 In the `/opt/shibboleth-idp/conf/attribute-filter.xml` configuration file, in order to provide rights to export the attributes configured in the `/opt/shibboleth-idp/conf/attribute-resolver.xml` configuration file, the following entry must be added:

```
<AttributeFilterPolicy id="sp_unique_identifier">
  <!-- Export these attributes to all configured Source Providers. -
->
  <PolicyRequirementRule xsi:type="ANY" />

  <!-- The list with configured attributes which will be accesible in
the Source Provider. -->
  <AttributeRule attributeID="uid" permitAny="true" />
  <AttributeRule attributeID="mail" permitAny="true" />
  <AttributeRule attributeID="displayName" permitAny="true" />
  <AttributeRule attributeID="organizationalUnit" permitAny="true" />
</AttributeFilterPolicy>
```

More informations can be found in the Shibboleth Identity Provider 3 documentation at this link.

3 Restart the Tomcat Servlet Container and try to authenticate to Shibboleth IDP with a user defined in configured database and check if the user attributes are exported by the IDP and are the same with ones defined in database.

1.0

application.properties config file

Below is the content of the `IDM_HOME/WEB-INF/classes/application.properties` file needed by the Muse IDM application to define the main application configuration.

Note: The value of `application.url` property must be set with current URL of the Muse IDM application. This value is used to create dynamic URL's for email templates.

```
# The name of current IDM application which will be displayed in web
application header.
application.name = MuseKnowledge&trade; Users Identity Manager

# The full URL of IDM installation.
application.url = https://idm.domain

# START HMAC CONFIGURATION INTEGRATION
# The HMAC entry point URL which will be added for Home link from header .
This link also is used to redirect the user after a successfull logout when
the value of clientID parameter of logout request matches the value of
"entryPoint.clientID" configuration entry.
entryPoint.url = https://idm.domain/access/muse
#The value of client ID parameter sent from HMAC integration which will
identify the entryPoint URL and redirect to that URL after successfull
login.
#Example of logout URL: https://idm.domain/logout?clientID=IDMHMAC. In that
case the user will be redirected to login page and after a successfull
login will be redirected again to configured entryPoint URL.
entryPoint.clientID = IDMHMAC
#END HMAC CONFIGURATION INTEGRATION

#START CONTACT US CONFIGURATION
#The email where the contact us messages from contact us page will be send.
contact.us.email = contactus@example.com
#END CONTACT US CONFIGURATION
```

```
#START LOGIN ATTEMPT CONFIGURATION
# Flag configuration which enable or disable the login attempt feature.
Default value is false.
login.attempt.enabled = true
# Specify here the maximum number of failed login attempts. Default value
is 3.
login.attempt.maxAttempts = 3
# The size of the inmemory cache which will hold the failed login attempts.
Default value is 8192.
login.attempt.cacheMaximumSize = 8192
# Represents how much time the user will be blocked whenever gets the wrong
username/password combination for at least 3 times. This value is parsed as
java.time.Duration. Default value is 30 seconds.
login.attempt.expireAfterWrite = PT30S
#END LOGIN ATTEMPT CONFIGURATION

#START REGISTRATION CONFIGURATION
# Flag used to enable or disable registration. The user will be redirected
on login page when he try to access the registration page.
registration.enabled = true
# Flag value to determine if registration is free or with administrator
approval.
registration.admin.approval = false
# Flag configuration used to skip the email validation step when a new user
register in the application. Default value is set to false.
registration.skip.email.validation = false
# Flag used to display or not the affiliation input field on registration
form.
registration.affiliation.input.show = true
#END REGISTRATION CONFIGURATION

#START CAPTCHA CONFIGURATION
# The site key used by Google reCAPTCHA v2.
captcha.site.key=6Ld0uYwUAAAAAP6Jtot0dTWI0jYva0vhNm0SrM42
# The secret key used by Google reCAPTCHA v2.
captcha.secret.key=6Ld0uYwUAAAAA0V2KyY0Rs2biatSU88ujIk32vnT
# The verification URL where the Google reCAPTCHA v2 whill be server side
validated.
captcha.verification.url =
https://www.google.com/recaptcha/api/siteverify?secret=%s&response=%s&remot
eip=%s
#END CAPTCHA CONFIGURATION
```

```
#START CAPTCHA ATTEMPT CONFIGURATION
# Default value is false.
captcha.attempt.enabled = true
# Default value is 3.
captcha.attempt.maxAttempts = 3
# Default value is 8192.
captcha.attempt.cacheMaximumSize = 8192
# This value is parsed as java.time.Duration. Default value is 30 seconds.
captcha.attempt.expireAfterWrite = PT30M
# This flag enable or disable captcha on login page. Default value is
false.
captcha.login.enabled = false
#END CAPTCHA ATTEMPT CONFIGURATION

#START PASSWORD VALIDATION CONFIGURATION
# The minimum length of password allowed on registration process.
password.minLength = 8
# The maximum length of password allowed on registration process.
password.maxLength = 25
# The strength of password allowed on registration process.
password.strength = 2
#END PASSWORD VALIDATION CONFIGURATION

#START EMAIL VALIDATION CONFIGURATION
# Comma separated list with denied emails(or only domains) to validate the
user registration request. Example: yahoo.com, hotmail.com
registration.denied.email =
# Comma separated list with allowed emails(or only domains) to validate the
user registration request. Example: gmail.com, museglobal.ro
registration.allowed.email =
#END EMAIL VALIDATION CONFIGURATION

#START TASK SCHEDULER
# The pool size of task scheduler thread pool.
taskScheduler.poolSize = 100
#END TASK SCHEDULER

#START ACCOUNT CONFIGURATION
# The availability of the user registration token to validate his email
when he register. Parsed as java.util.Duration. Default value is P1D which
represent one day.
user.token.expiry = P1D
```

```
# The availability of a new user account created. Parsed as
java.util.Period. Default value is P1Y which represent one year.
user.account.expiry = P1Y
# The password life time. Is parsed as java.util.Period. Default value is
P3M which represent 3 months.
user.password.life.time = P3M
# The maximum password history which will be kept for an user represented
as integer. Default value is 5.
password.history.max.entries.per.user = 5
#END ACCOUNT CONFIGURATION

#START LOCALIZATION CONFIGURATION
# The parameter name of locale used to change the application locale.
locale.paramName = languageCode
# Default locale used by IDM application.
locale.default = en
locale.defaultEncoding = UTF-8
locale.useCodeAsDefaultMessage = true
locale.cacheSeconds = 0
#END LOCALIZATION CONFIGURATION

#START THEME CONFIGURATION
# The parameter name of theme used to change the application theme.
theme.paramName = theme
# Default theme name used by IDM application.
theme.defaultThemeName = default
# Directory where themes are defined.
theme.baseNamePrefix = theme/
# The cookie name where current selected theme name is saved.
theme.cookieName = theme
#END THEME CONFIGURATION

#START COOKIE AGREEMENT CONFIGURATION
# Flag configuration used to display or not the cookie agreement from web
interface. Default value is false.
cookieAgreement.enabled = false
#END COOKIE AGREEMENT CONFIGURATION

#START MENU CONFIGURATION
themes.menu.enabled = true
# List with all supported locales:
en,ro,ar,el,es,fr,ja,nl,de,tr,zh_CN,zh_TW
# Empty locale list will no display any entry on menu.
```

```
locale.enabled = en,ro,ar,el,es,fr,ja,nl,de,tr,zh_CN,zh_TW
#END MENU CONFIGURATION

#START STATIC RESOURCES CONFIGURATION
# The cache period in seconds for static resources loaded from /
WEB-INF/static-resources/ directory.
static.resources.cachePeriod = 31556926
#END STATIC RESOURCES CONFIGURATION

#START ADMIN USER CONFIGURATION
admin.email = admin
admin.password = admin
admin.fullName = Administrator
admin.affiliation = Affiliation
#END ADMIN USER CONFIGURATION

#START EXTERNAL HTTP CONNECTION TIMEOUT
http.connection.timeout = 6000
http.read.timeout = 6000
#END EXTERNAL HTTP CONNECTION TIMEOUT

#TASK EXECUTOR
#The keep alive for threads from task executor in seconds.
taskExecutor.keepAlive= 3600
#First number represent the pool size and second represent the maximum pool
size.
taskExecutor.poolSize = 100-300
#Queue capacity for the ThreadPoolTaskExecutor. If not specified, the
default will be Integer.MAX_VALUE.
taskExecutor.queueCapacity = 1000
#END TASK EXECUTOR

#START FILE UPLOAD CONFIGURATION
multipartResolver.maxUploadSize = 104857600
multipartResolver.maxUploadSizePerFile = 50485760
multipartResolver.maxInMemorySize = 10240
multipartResolver.defaultEncoding = UTF-8
multipartResolver.preserveFilename = true
#END FILE UPLOAD CONFIGURATION

#START DATABASE TASKS CONFIGURATION
# Run at every 10 minutes
task.purgeUnvalidatedEmails.cron = 0 */10 * ? * *
```

```
#START NOTIFY ADMINS ABOUT TO EXPIRY ACCOUNTS
# Run every day at 1PM
task.notifyAdministratorsAboutAccountsToExpire.cron = 0 0 1 * * ?
#task.notifyAdministratorsAboutAccountsToExpire.cron = 0 */1 * ? * *
# The period from current date to found about to expiry accounts.
# Need to be configured with values which can be parsed by
java.util.Period.parse(CharSequence text) method as in following doc
umentation
#
https://docs.oracle.com/javase/8/docs/api/java/time/Period.html#parse-java.
lang.CharSequence-
task.notifyAdministratorsAboutAccountsToExpire.firstNotification = P5D
task.notifyAdministratorsAboutAccountsToExpire.lastNotification = P15D
#END NOTIFY ADMINS ABOUT TO EXPIRY ACCOUNTS

#START TASK WHICH PURGE LAST N PASSWORD HISTORY ENTRIES
# N is specified by password.history.max.entries.per.user configuration.
# Run every day at 1PM.
task.purgeOldestNPasswordHistoryEntriesForAllUsers.cron = 0 0 1 * * ?
#END TASK WHICH PURGE LAST N PASSWORD HISTORY ENTRIES

#END DATABASE TASKS CONFIGURATION

#START REMEMBER ME CONFIGURATION
# The key used by Remember Me module to generate the remember me cookie
value.
rememberMe.key=XMsrgecwQFHQHwMsqyvCzxQUGzLsoMem
# Specify the validation of remember-me token in seconds.
rememberMe.tokenValiditySeconds=2592000
#END REMEMBER ME CONFIGURATION

#START LOGGING USER DETAILS ON AUTHENTICATION PROCESS
# Flag configuration which enables the logging of user and his associated
IP on login and logout actions.
authentication.user.log.enabled = false
#END LOGGING USER DETAILS ON AUTHENTICATION PROCESS
```

2.0

email.properties config file

Below is the content of the `IDM_HOME/WEB-INF/classes/email.properties` file needed by the Muse IDM application for email functionalities:

```
# EMAIL CONFIGURATION
#https://javaee.github.io/javamail/docs/api/com/sun/mail/imap/package-summa
ry.html
#https://javaee.github.io/javamail/docs/api/com/sun/mail/pop3/package-summa
ry.html
#https://javaee.github.io/javamail/docs/api/com/sun/mail/smtp/package-summa
ry.html
#More attributes to be set can be found on previous links and also these
need to be added on servlet-context.xml file under javaMailProperties.
email.from = EMAIL_FROM
email.replyTo = EMAIL_REPLY_TO
email.host = SMTP_HOST
email.port = SMTP_PORT
email.username = SMTP_USERNAME
email.password = SMTP_PASSWORD
email.debug= false
email.auth = true
email.starttls.enable = USE_TLS
email.smtp.ssl.enable = USE_SSL
email.charset = UTF-8
email.transport.protocol = smtp
email.template.location= classpath:/emailTemplates/
#Delay in milliseconds between emails when they are send bulk. 0 as value
means no delay.
email.bulk.delay.between.emails = 1000
# END EMAIL CONFIGURATION
```

where replace the following variables with the actual values:

- `SMTP_HOST`. Specify here the hostname of the SMTP server.
- `SMTP_PORT`. Specify here the port on which the SMTP server is listening. It may be 25 (default TCP prt), 587 (submission), 465 (SMPTPS), etc.
- `SMTP_USERNAME`. Specify here the username used for authenticating to the SMTP server if necessary. Leave blank if not using authentication.
- `SMTP_PASSWORD`. Specify here the password used for authenticating to the SMTP server if necessary. Leave blank if not using authentication.
- `USE_SSL/USE_TLS`. Specify `true` or `false` as values for each parameter, depending on the capabilities of the SMTP server that is configured.
- `EMAIL_FROM`. Specify here an email address that will be used in the FROM email field for all emails sent.
- `EMAIL_REPLY_TO`. Specify here an email address that will be used in the REPLY_TO email field for all emails sent.

3.0

database.properties config file

Below is the content of the `IDM_HOME/WEB-INF/classes/database.properties` file needed by the Muse IDM application for database functionalities:

Note: On a fresh install if IDM application the value of `database.hibernate.hbm2ddl.auto` property must be `update` and on an upgrade of IDM application the value must be `validate` instead of `update`.

```
# DATABASE CONFIGURATION
# More properties can be found here:
https://docs.jboss.org/hibernate/orm/3.3/reference/en/html/session-configur
ation.html
# Also these need to be added on database-context.xml file under
jpaPropertyMap.
database.driverClassName = com.mysql.cj.jdbc.Driver
database.url =
jdbc:mysql://HOST:PORT/DATABASE_NAME?autoReconnect=true&useSSL=false&allowP
ublicKeyRetrieval=true
database.username = DATABASE_USERNAME
database.password = DATABASE_USER_PASSWORD
# The following values are supported:
        #validate: validate the schema, makes no changes to the database.
        #update: update the schema.
        #create: creates the schema, destroying previous data.
        #create-drop: drop the schema when the SessionFactory is closed
explicitly, typically when the application is stopped.
database.hibernate.hbm2ddl.auto = update
database.hibernate.generate_statistics = false
database.hibernate.format_sql = false
database.hibernate.show_sql = false
database.hibernate.dialect = org.hibernate.dialect.MySQL8Dialect

# CONNECTION POOL CONFIGURATION
```

```
database.connectionPool.connectionTimeout = 30000
database.connectionPool.validationTimeout = 5000
database.connectionPool.idleTimeout = 600000
database.connectionPool.maxLifetime = 1800000
database.connectionPool.leakDetectionThreshold = 5000
database.connectionPool.maximumPoolSize = 100
database.connectionPool.minimumIdle = 5
database.connectionPool.connectionTestQuery = SELECT 1
database.connectionPool.poolName = IDMHikariConnectionPool
# END DATABASE CONFIGURATION
```

where replace the following variables with the actual values:

- `HOST`. Specify here the database server host.
- `PORT`. Specify here the database server port.
- `DATABASE_NAME`. Specify here the database name which based on previous configuration is `IDM`.
- `DATABASE_USERNAME`. Specify here the username used for authenticating to the database server if necessary. Leave blank if not using authentication.
- `DATABASE_USER_PASSWORD`. Specify here the password used for authenticating to the database server if necessary. Leave blank if not using authentication.

The connection pool configuration also can be changed according with requirements.

4.0

users.properties config file

Below is the content of the `IDM_HOME/WEB-INF/classes/users.properties` file needed by the Muse IDM application to define administrator users:

```
idm =
{bcrypt}$2a$10$7YfiR8540.EGB0feM2L6J0xC4v1q8klhQ5307XRQPcFGJ4MpxN2dC,SUPER_
ADMIN
```

The user represented by previous configuration entry has the username `idm`, the password is `idm` and the role is `SUPER_ADMIN`.

The key which in this case is `idm` represents the username of the username, and the value represents the password of the `idm` user with which he can authenticate as SUPER_ADMIN.

New entries can be added in this file and the password can be encoded from `Password Encoder` page from web interface located under `Utils` header entry.

Note: A server restart is required when this file is modified.

Note: Keep in mind that the `argon2` hashing algorithm cannot be used here.

5.0

log4j2.xml config file

Below is the content of the `IDM_HOME/WEB-INF/classes/log4j2.xml` file needed by the Muse IDM application for logging purposes:

```
<?xml version="1.0" encoding="UTF-8"?>
      <Configuration status="info">
      <Properties>
      <Property name="LOG_DIR">../logs</Property>
      </Properties>
      <Appenders>
      <RollingFile name="FileAppender" fileName="${LOG_DIR}/idm.log"
filePattern="${LOG_DIR}/idm.%d{dd-MM-yyyy}.%i.log.gz" ig
noreExceptions="false">
      <PatternLayout>
      <Pattern>%d{HH:mm:ss.SSS} [%t] %-5level %logger{36} -
%msg%n</Pattern>
      </PatternLayout>
      <Policies>
      <OnStartupTriggeringPolicy />
      <SizeBasedTriggeringPolicy size="100 MB" />
      <TimeBasedTriggeringPolicy />
      </Policies>
      <DefaultRolloverStrategy max="10" />
      </RollingFile>
      </Appenders>
      <Loggers>
      <Logger name="org.springframework.security.saml" level="debug"
additivity="false">
      <AppenderRef ref="FileAppender" />
      </Logger>
      <Logger name="org.springframework.security.oauth2" level="debug"
additivity="false">
      <AppenderRef ref="FileAppender" />
```

```
</Logger>
<Root level="info">
<AppenderRef ref="FileAppender" />
</Root>
</Loggers>
</Configuration>
```

You can access link in order to change the default configuration.