APPLICATION CONFIG FILE GUIDE

Table of Contents

Application Config File Guide

1. Introduction

This guide describes the application configuration file (*application.conf*) in Ambience/Repertoire software suite (hereby known as the software). This file is location in the "/etc" folder in the software.

The application configuration file is used to configure the parameters and initial settings for the software.

When upgrading the software suite, it is advisable to compare the new *application.conf* file with the old one to see whether any new items have been added.

The list of parameters that can be configured are listed in the table below.

The following sections uses Ambience as example and some sections are only applicable to Ambience only.

Refer to the following websites for more information:

- https://docs.elixirtech.com/Ambience/2022.1/index.html
- https://docs.elixirtech.com/Repertoire/2022.1/index.html
- www.elixirtech.com

2. Security

2.1. Configure and Test Mail Server

When identity is added, an email is sent with randomly generated password to the user. When a user wants to change the email or password, a verification is sent via email as well.

If you have not set up an email server, the default behaviour is to store the emails in the "/mail" folder within the softare. This is usually for diagnosis or debugging purposes. It is recommended to set up a mail server at the start.

Below are two examples of how to set up a mail server.

Example 1: Uses Gmail

1. Gmail allows only OAuth2 authentication without weakening security. Visit https://console.developers.google.com/apis/credentails to set up a "*clientID*" and "*clientSecret*". Use these to generate a "*refreshToken*".

2. In the software root folder, navigate to the "/etc" folder. Open the *application.conf* file using a text editor. In the `elixir.mail` section, edit the following with the information obtained earlier accordingly.

```
elixir.mail {
  smtp = "gmail"
  gmail {
    host = "smtp.gmail.com"
    port = 587
    debug = true
    oauth2 {
      userName = "xxx@gmail.com"
      clientId = "XXXX"
      clientSecret = "YYYY"
      refreshToken = "ZZZZ"
    }
  }
}
```

3. Save the above edits in the *application.conf* file and start the server. New users with valid email addresses can now be created in the Identities module.

Example 2: Uses AWS

1. In the software root folder, navigate to the "/etc" folder. Open the *application.conf* file using a text editor. In the "*elixir.mail*" section, edit the following:

```
elixir.mail {
  smtp = "aws"
  aws {
    from = "<user@example.com>"
    host = "<hostname>"
    dnsResolver = ""
    port = 465
    user = "XXXX"
    password = "YYYY"
    connectionTimeout = 30000
    tls = true
    ssl = true
    authMechanism = ""
    debug = false
  }
}
```

2. Save the above edits in the *application.conf* file and start the server. New users with valid email addresses can now be created in the Identities module.

2.2. Use GitLab As Authentication

The Identities module in Ambience software provides a simple mechanism for authentication (determining who is logging in). If you already have an authentication system, such as an SSO, LDAP or Active Directory, then it is possible to use that as the authentication mechanism. This identity management system is built upon OAuth2, which is what makes it possible to plug in alternate authentication providers.

If an external authentication system is used, the Identities module is not needed and should be removed to avoid confusion.

This section describes the steps to set up GitLab as the authentication method to log into the software.

The steps are as follows:

1. Create an account in GitLab.

2. In GitLab, add the software as an application under your user.

   Note: The URL callback should be http://hostname:1740/authclient for Ambience. Use port 1730 for Repertoire. This is consistent with the setting in the *application.conf* file.

3. Change the hostname on your machine to point to the proper endpoint (i.e., the added application).

4. Add the user into Users module with the same name that was created in the GitLab server.

5. Go to the software root directory and go to the "/etc" folder. Open the *application.conf* file using a text editor.

6. Make the following changes in the `elixir.sso.client` section.

```
elixir.sso.client {
  cookie-name = "elx-amb"

  cookie-same-site = "Lax"
  openid-field = "name"
  openid-scope = "openid email"
  service-definition {
    elxsso {
      authorization = "https://<gitlab-host>/oauth/authorize"
      token = "https://<gitlab-host>/oauth/token"
      userinfo = "https://<gitlab-host>/oauth/userinfo"
      logout = ${sso-server-baseurl}"/simple-sso/logout"
      debug = false
      client {
        id = "[Your Application ID]"
        secret = "[Your secret]"
        endpoint = ${sso-client-baseurl}"/authclient"
      }
    }
  }
}
```

7. Save the *application.conf* file.

8. Restart the software server. Open a browser and key in "Localhost 1740" in the address bar and hit the enter key.

   For Repertoire, ley in "localhost: 1730" in the address bar.

9. Log into the software using the GitLab account.

2.3. Two-factor Authentication

Ambience/Repertoire softwaresupports Time-based One-time Password (TOTP) Two-factor Authentication (2FA). By default, 2FA is disabled in the *application.conf* file. To enable 2FA, edit the *application.conf* file in two areas:

1. Under the *simple-server* section, change `show-totp = false` to `true`. This is to allow the login dialog to include 2FA.

```
simple-server {
  clients {
    ambience {
      secret = "171ccf22-670a-43c2-ac79-05c44bf305e3"
      redirect = ${sso-client-baseurl}"/authclient"
      #login-page = "" # set resource file here to use a custom
login page for this client
      landing-page = "http://"${host}":"${port}"/"
      name = "Elixir Ambience"
      show-totp = true
    }
  }
}
```

2. Add a new line in the *application.conf* file. This will allow User Settings module to include 2FA setup, in which users can set up their own 2FA.

```
ambience.user-settings.enable-panel.totp = true
```

2.4. Passwords

2.4.1. For First Login

When the user logs in with the randomly generated password (i.e., first login), they will be forced to change the password immediately. This can be disabled by editing the setting in the *application.conf* file. In the `elixir.identity` section, edit the `changePassword: true` to `false`.

```
elixir.identity {
    ...
    on-reset {
      changePassword: true
    }
}
```

2.4.2. Using Login Dialog

Any user can reset their own password using the login dialog. The new password will be sent to the user's email.

2.4.3. Password Policy

Strong password protects your system from hackers and malicious software. Some passwords may appear strong but are relatively easily guessed. Key aspects of a strong password are length (the longer the better), a mix of letters (upper and lower case), numbers and symbols, and no ties to your personal or corporate information.

The `elixir.identity.password-policy` section in *application.conf* file allows administrators to define the policy of the password to be used in the software suite.

```
elixir.identity {
    ...
    password-policy {
      minLength: 1
      maxLength: 0
      ...
      mustHaveSymbolSet: ""
      ...
      disallowedRegex: []
    }
}
```

The minimum and maximum length of the password, whether to use upper and/or lower case characters, whether to include digit or symbol, and disallow certain regular expressions, etc. can be defined in this section.

It may be advisable to disallow common passwords based on regular expressions, so that users are not able to set common passwords. A common password for one organisation may be very different from the next. Rather than a fixed list of regular expressions, administrators can decide in the list of regular expressions that should be disallowed.

The `disallowedRegex: []` field is empty by default. You can add your desired regular expression to disallow in the brackets.

Below is an example to disallow both "password" and "passw0rd" (with a zero instead if an o), as well as any password that starts with "e1" (a common elixir easily guessed example).

```
disallowedRegex: ["passw[o0]rd","e1.*"]
```

2.4.4. Customise Password Email

You can customise the content of the email to be sent to the user when the user resets his/her password or when new user is added.

In the *application.conf* file, add the following:

```
elixir.simple-identity {
  email (
    subject = "Elixir Password"
    add-account = ""<html><p>An account has been created for you
in toolTitle at
    <a href=`${landingPage]`>localhost:1740</a>.</p>
    <p>Your user name is ${name}, your password is ${password}</p>
    <p>Please login and change it.</p></html>""
    reset-account = ""<html><p>Your password has been reset in
toolTitle at
    <a href=`${landingPage]`>localhost:1740</a>.</p>
    <p>Your user name is ${name}, your password is ${password}</p>
    <p>Please login and change it.</p></html>""
  }
}
```

For Repertoire, change the port to 1730.

2.4.5. Customise Password Verification Email

You can customise the content of the email to be sent to the user when the user changes his/her password or email address.

In the *application.conf* file, add the following:

```
ambience.user-settings.email.password-change {
  subject = "Customised Password Change Verification"
  body = ""<h2>Elixir Ambience Password Change</h2>
    <p>A change of password has been requested by ${name}.</p>
    <p>Please enter this approval code on the change password
page:</p>
    <p>${code}</p>
    <p>If you are an Elixir user and did not request this change,
please contact your administrator.</p>
    <p>If you take no further action, the password change will be
rejected.</p>""
}
ambience.user-settings.email.password-change {
  subject = "Customised Address Verification"
  body = ""<h2>Elixir Ambience Email Change</h2>
    <p>A change of email address has been requested by
${name}.</p>
    <p>Changed from ${oldEmail} to ${newEmail}.</p>
    <p>Please enter this approval code on the change emai
page:</p>
    <p>${code}</p>
    <p>If you are an Elixir user and did not request this change,
please contact your administrator.</p>
    <p>If you take no further action, the password change will be
rejected.</p>""
}
```

2.4.6. Customise Password Reset Email

You can customise the content of the email to be sent to the user when the user resets his/her password.

In the *application.conf* file, add the following:

```
elixir.sso.server {
  email.password-reset {
    subject = "Elixir Password Reset Request: ${clientTitle} user:
${name}"
   body = """<html><p>${clientTitle} Password Rest Request for user
${name}</p>
    <p>If you have requested a password reset, please click the
following link:</p>
    <p><a href=`${resetPage}`>${resetPage}</a></p>
    <p>If you have not requested a password reset, you can choose
to ignore this mail, no change has been made yet,
    or report to your administrator as someone has requested a
reset for an account with this email address.</p></html>"""
  }
}
```

2.4.7. Localise Password Reset Request Information

The password reset request information by default is in English. This information can be localised to the language of the user.

To do so, add the i18n localisation file onto the "etc/i18n/" folder.

The i18n localisation file has the filename *simple-sso_XX.properties*. where "*Xx*" is the abbreviation of the language.

Below is a sample of the i18n localisation file *simple-sso_zh.properties*.

```
PasswordResetRequestAccepted = 密码重置已提交，请查阅邮件。

CannotResetPassword = 所提供信息不足重置密码。请联系系统管理员。

PasswordResetEmail = 您的密码已重置。请查阅邮件。

InvalidConfirmationCode = 确认码无效。

CookieMismatch = 不匹配，请再试一次。

RequestDenied = 请求拒绝，请联系您的系统管理员。
```

Restart the server and change the browser default language to the language desired. When the user resets his/her password, if the new password does not conform to the password policy, the appropriate error message will appear.

2.4.8. Localise Password Policy Message

To localise the language of the password policy messages, perform the following:

1. Add the i18n localised file onto the "etc/i18n/" folder. The i18n file should have a filename *user-settings_XX.properties*, where *XX* is the abbreviation of the language. Below is an example of the content of a i18n file.

```
PasswordPolicy.6001 = DEPasswordPolicy DatabaseError

PasswordPolicy.6020 = DEPasswordPolicy passwort muss mindestens
${minLength} Zeichen enthalten

PasswordPolicy.6021 = DEPasswordPolicy passwort darf nicht mehr
als ${maxLength} Zeichen enthanlten

PasswordPolicy.6022 = DEPasswordPolicy passwort enthält ein
unzulässiges Symbol: ${mustNotHaveSymbolSet}

PasswordPolicy.6023 = DEPasswordPolicy passwort muss eine
Ziffer enthalten

PasswordPolicy.6024 = DEPasswordPolicy passwort muss einen
Großbuchstaben enthalten

PasswordPolicy.6025 = DEPasswordPolicy passwort muss einen
Kleinbuchstaben enthalten

PasswordPolicy.6026 = DEPasswordPolicy passwort muss ein Symbol
aus enthalten: ${mustHaveSymbolSet}

PasswordPolicy.6027 = DEPasswordPolicy passwörter dürfen nicht
mit dem Benutzernamen identisch sein

PasswordPolicy.6028 = DEPasswordPolicy Dieses Passwort wurde
bereits verwendet

PasswordPolicy.6029 = DEPasswordPolicy Dieses Passwort gilt als
unsicher
```

2. In the *application.conf* file, edit the following to match the conditions above:

```
elixir.identity {
  user-collection: "Identities"
  password-policy {
    minLength:3
    maxLength:8
    notSameAsLogon:true
    maxPasswordExpiresDays:0
    differentPasswordCount:0
    mustHaveDigit:true
    mustHaveUpperCase:true
    mustHaveLowerCase:true
    mustHaveSymbolSet:""
    mustNotHaveSymbolSet:""
    retryAttemptLockoutCount:0
    disallowedRegex:["passw[o0]rd","e1.*"]
  }
  on-add {
    changePassword:false
  }
  on-reset {
    changePassword:false
  }
}
```

3. Restart the server and change the language settings to the desired language in the browser.

4. To test the localisation, login to the server and change the password that conflict one of the conditions in the i18n file. A message should appear.

2.4.9. Password Reset Confirmation URL Expiry Time

The password confirmation URL expiry time can be customised to suit each system's requirements. The expiry time can be customised in the *application.conf* file.There are three scenarios:

- Request password reset
- Change password
- Forget password

In the *application.conf* file, added the line in bold in the `elixir.sso.sever` session:

```
elixir.sso.sever {
  cookie-name = "elxsso"
  ...
  reset-link-expiry = 10 minutes
  ..
}
```

This sets the expiry time of the reset link. Change the value to the desired value.

To customise reset code expiry time, add the following in the *application.conf* file.

```
ambience.user-settings-reset-code-expiry = 10 minutes
```

This will set the expiry code sent to expire in 10 minutes.

For the "Forgot password" scenario, both are used.

2.5. User Name Case Sensitivity

The user names created in the Users module are by default case sensitive. This is embedded in the `elixir.sso.client` section in the *application.conf* file as below:

```
elixir.sso.client {
  ...
  openid-case = ""
  ...
}
```

The default value for `openid-case` is empty (`""`), which indicates case sensitive.

The other vales that can be used are `"UPPER"` (for upper case) and `"lower"` (for lower case)

For example, if the usernames are uppercase, then change the `openid-case` value to `"UPPER"`, and any MixedCase strings from the Identity module will become MIXEDCASE.

2.6. Deploy Server With Customised Initial User

When a server is deployed, a customised initial user may be setup. This can be done using either one of the following methods.

2.6.1. Hard-coding Initial User

This method "hard-code" the initial user in the application.conf file. To do so, perform the following:

1. Ensure the server does not have any existing Users and Identities.

2. In the *application.conf* file, add the following:

```
ambience.initial-user {
  name = "not-admin"
  password = "not-sa"
}
```

Use "bin/ambience-cli" to encrypt the above if desired.

3. Then start the server and login with the above credentials.

2.6.2. Parameterised Initial User

This method uses parameters in a docker to customise the initial user. To do so, perform the following:

1. Ensure the server does not have any existing Users and Identities.

2. In the *application.conf* file, add the following:

```
ambience.initial-user {
  name = $[?AMBIENCE_INITIAL_USERNAME]
  password = $[?AMBIENCE_INITIAL_PASSWORD]
}
```

3. In the "docker-compose.yml" file, add the following environment values under the Ambience portion:

```
- AMBIENCE_NITIAL_USERNAME=not-admin
- AMBIENCE_NITIAL_PASSWORD=not-sa
```

4. Then restart the docker and login with the above credentials.

2.7. Enforce Single Session

Some systems may prefer to allow only single session log in.

To do so, perform the following to enforce single session login.

1. In the *application.conf* file, add the line in bold in the `ambience.web` session:

```
ambience.web {
  ...
  session-timeout = 15 minutes
  enforce-single-session = true
  ...
}
```

2. In the `elixir.sso.server` session, change the `session-length` to a short value.

```
elixir.sso.server {
  ...
  # this controls how long the sso cookie remains active – it should
  # be short if enforce-single-session is enabled, else the expired
  # embience session will be immediately and silently re-approved,
  # circumventing enforce-single-session
  session-length = 1 seconds
  ...
}
```

3. Restart the server for the single session to take effect.

2.8. Configure Server With Client Certificate

The following steps allows you to configure the server with a HTTPS client certificate.

1. To generate a client certificate, open a terminal window and issue the following commands:

```
openssl req -newkey rsa:4096 -keyout qa_key.perm -out qa-csr.perm -
nodes -days 3650 -subj "/CN=QA"
openssl x509 -req -in qa_csr.perm -signkey qa_key.perm -out
qa_cert.perm -days 3650
openssl pkcs12 -export -in qa_cert.perm -inkey qa_key.perm -out
qa.pkcs12
```

2. Copy the *qa.pkcs12* file and add it into the browser. Copy and rename as the same file as *trruststore.pkcs12* and place it in the Ambience "etc/https" folder.

3. To enable the secure mode and client certification in Ambience, edit the following the *application.conf* file in the "bin" folder.

```
ambience {
  systemId: "System"
  bindAddress: "0.0.0.0"
  bindPort: ${interal-port}
  https {
    enabled = true
    keystore = "https/keystore"
    keystore-type = "JKS"
    keystore-password = "elixir"
    client-auth = "require"
    truststore = "https/truststore.pkcs12"
    truststore-type = "PKCS12"
    truststore-password = "elixir"
  } }
```

4. Place the *trsutstore.pkcs12* file and keystore file in the "etc/https/" folder.

5. Ensure the server name matches the certificates.

6. Restart the server and log in with HTTPS.

3. Database and File Size

3.1. Add Database

When MongoDB is installed, by default the database is named "eno". You can add another database into MongoDB and include it the *application.conf* file so that the Ambience software is able to access it.

Two areas need to be edited.

1. The first area allows user to read and write to the database via datasets and import/export modules. In the location indicated in bold, replace it with your new database name.

```
#Where datasets can be read from and written to via datasets and
import/export modules
ambience.datasets.databases = ["eno","NewDatabaseName"]
ambience.import.databases = ["eno","NewDatabaseName"]
ambience.export.databases = ["eno","NewDatabaseName"]
```

2. The other area allows the user to include the new database so that Ambience can access it in MongoDB. In the `elixir.data.mongodb` section, add a new line as indicated in bold, replacing the database name.

```
elixir.data.mongodb {
  ...
  default {
    connectionString = "mongodb://"${mongodb}":27017"
    ...
    database {
      eno = "eno"
      NewDatabaseName = "NewDatabaseName"
    }
  }
}
```

This section is for Ambience only and does not apply to Repertoire.

3.2. Session Timeout Duration

Ambience software has inactivity session timeout of 15 minutes. This timeout duration can be changed in the *application.conf* file. In the `ambience.web` section, edit the `session-timeout = 15 minutes` to the desired duration.

```
ambience.web {
  ...
  session-timeout = 15 minutes
  enforce-single-session = false
  ...
}
```

To disable the timeout, use the value `never`.

3.3. Session Cleaner

The software keeps the sessions for seven days by default. A session will typically expire due to inactivity of 15 minutes. The seven days will begin from the last activity. So you can still keep a session alive for a month, as long as the mouse is moved at least once every 15 minutes.

You can change the duration for the sessions to be kept in the system in the *application.conf* file. You can add following in the *application.conf* file.

```
ambience.session-cleaner {
  expiry: 2 days
}
```

The configuration above allows the software to remove sessions that are more than two days from the last activity.

3.4. BinaryStore

3.4.1. Enable BinaryStore

The Upload and Messages modules in the Ambience software suite supports BinaryStore.

By default, each module uses a specific collection under the Ambience database; namely `UploadDownloadChunk` and `MessageAttachments` respectively. The filesize is controlled in the *application.conf* file as follows:

```
ambience.modules.upload.max-content-length : 150m
```

To enable the use of BinaryStore for these modules, add the respective line in the *application.conf* file:

```
ambience.modules.upload.store = "default-large"
// For Upload module
ambience.modules.messages.attachmentsStore = "default-large"
// For Message module
```

If the line is missing or empty, then their respective module collection will be used.

The maximum filesize for BinaryStore is defined in the value `maxSize` in the *application.conf* file.

```
ambience.binary-store.lcation {
  default {
    type = "mongodb"
    database = "" # use default (typically "ambience") if empty
    collection = "BinaryStore"
    maxSize = 15MB
  }
  default-large {
    type = "mongodb-chunks"
    enable = true
    database = "" # use default (typically "ambience") if empty
    collection = "BinaryStoreChunks"
    maxSize = 500MB
  }
}
```

The setting for `max-content-length` will take precedence over `maxSize`. When `max-content-length` is empty or missing, then filesize is controlled by `maxSize`.

Any existing uploads and attachments in the Ambience specific collections, prior to the switch to BinaryStore, the system will still be able to find the previous items in their respective collections. Vice versa for switching from BinaryStore to the default Ambience collection.

3.4.2. Disk BinaryStore

By default, MongoDB is used for BinaryStore. This storage location can be changed to disk (i.e., a location in the local drive).

To do so, add the following lines into the *application.conf* file to configure the disk BinaryStore:

```
ambience.binary-store.location {
  <name> {
    type = "disk"
    path = "<pathname>"
    maxSize = 150MB
    enable = true
  }
}
```

The `<name>` value defines the name of the store and `<pathname>` defines the path to the desired location in the local drive.

The disk store then can be used as a store name for any configurable store. For example, the following line is added to the *application.conf* file to use the disk store for the Upload module.

```
ambience.modules.upload.store = "<name>"
```

Each entry has the metadata equivalent to what is stored in AWS S3, MongoDB, etc.

4. RML

4.1. RML Report Engine Left Panel

When generating a RML report, the report by default fills the entire area in the RML Report Engine page. You can enable the parameters on the left of the page. To do so, add the following line in the *application.conf* file.

```
ambience.rml-engine.ui-left-parameters = ["application/pdf"]
```
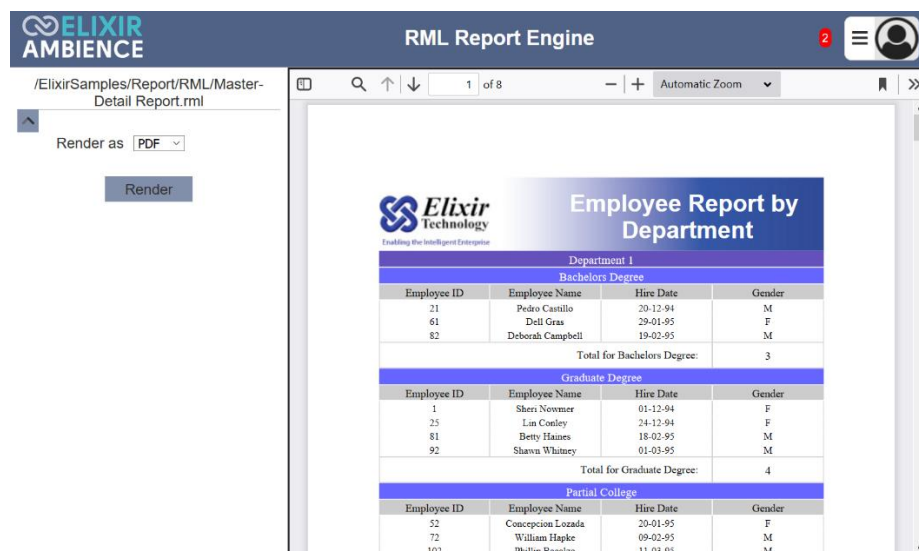

You can change the `pdf` to other types if your browser supports viewing them. You can toggle the left panel off by clicking on the ∧ icon.
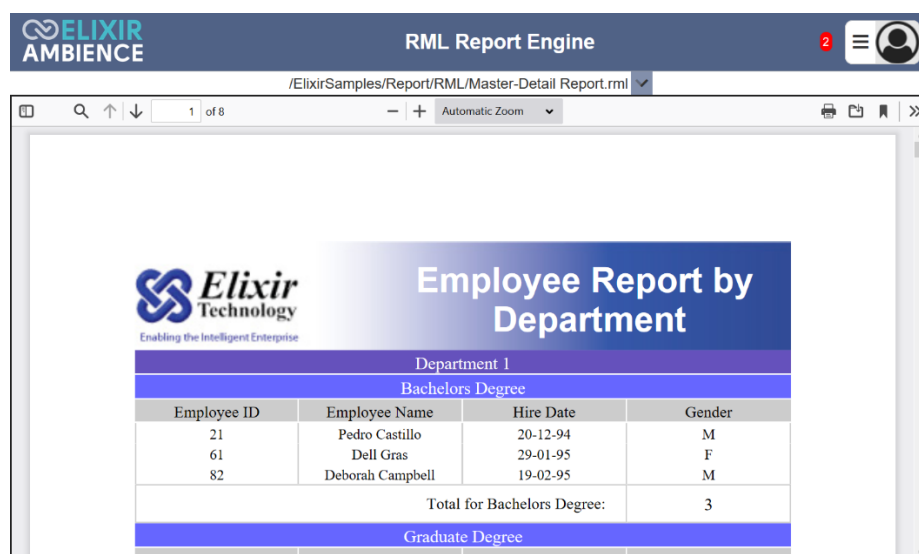
To toggle on, click on the ∨ icon on the top row.

4.2. Fonts In RML Reports

You can use extra fonts (non-OS) in the RML reports. To do so, add an extra line in the *application.conf* file.

```
elixir.rml.fonts.path = "/<pathname>/Fonts"
```

5. Currency Symbol In XLSX Output

By default, the currency symbol used is (¤) is used when an XLSX output is generated. This may not be the desired symbol to use.

You can use the currency symbol "$" XLSX output for the RML reports and for dashboard export to XLSX.

To do so, add an extra line in the *application.conf* file:

- For RML reports

```
elixir.rml.xlsx.currency-symbol = "[$$-4809]"
```

- For dashboard export to XLSX format (for Ambience only)

```
ambience.dashboard.pivot { xlsx-export { currency-code: "[$$-409]" } }
```

6. ETL

6.1. ETL Logs

By default, the ETL job logs is created as the job runs. These logs allow you to observe the behaviour of the jobs. But these logs may not be useful for production systems as they may run the ETL jobs 24 hours a day, thus generating huge volumes of log. This may slow down the system.

You can turn off the job log by adding the following into the *application.conf* file.

```
ambience.etl.logging.write-to-server-log: false
ambience.etl.logging.write-to-job-log: false
```

6.2. Apose Extensions for ETL Steps

To enable the Apose extensions for ETL steps, you require a separate license. Add the following into the *application.conf* file.

```
ambience.docx-engine.aspose {
  enabled = true
  words-license = "/<pathname>/Aspose.Words.Java.lic"
}
```

6.3. Minimum XLSX Inflate Ratio

To allow ETL steps to be able to read lower zip ratio, you can customise the minimum XLSX inflate ratio to support Apache POI zip bomb detection.

In the *application.conf* file, add the following line:

```
elixir.poi.zip-secure-file.min-inflate-ratio = 0.006639
```

The inflate ratio used above is an example, it can be changed to suit your usage.

7. Config Editor

The Config Editor module in Ambience allows you to edit the configuration and save the changes onto MongoDB. This allows you to edit at one location and let other servers share the same configuration without the need to duplicate the *application.conf* file.

Before that can be done, the following need to be added into the *application.conf* file to allow the configuration in the Config Editor to be loaded.

```
ambience.config-loader.mongodb.enabled = true
```

This section is for Ambience only and does not apply to Repertoire.

8. Audit Log

Audit log will log any actions taken in the modules. Workflow module stores data either in `elxPublic` (server only) or `elxPrivate` (shared within browser). It does not know whether any data it is logging is sensitive. You can turn on or off the logging by adding the following into the *application.conf* file.

```
ambience.modules.workflow.audit {
  public: true
  private: false
}
```

The default will only show the public changes. Set `private: true` to include `elxPrivate` changes.

This section is for Ambience only and does not apply to Repertoire.

9. Redirect Using ETL

To use ETL chainset to redirect a URL, there needs to be an ETL user which has a role that is added to the appropriate chainset. This is needed as the user being redirected may not have permission to run ETL. So, this user acts as a proxy to run the chain the redirect uses.

In the *application.conf* file, add the following:

```
ambience.modules.redirect-edit.etl-user = "<username>"
```

This section is for Ambience only and does not apply to Repertoire.

10. Git Deploy

To use Git deploy, the application configuration file need to add some code to allow the Git Deploy module to know where to locate the repository, either local or remote.

In the *application.conf* file, add the following:

```
ambience {
  modules {
    git-deploy {
      repositories {
        ## remote key: value pairs where the value may be a remote git
          repository like:
        "remote-sample":"git@git.example.com:<path in Git>"
        ## or a local git repository (cloned earlier, or used git init to
          create)
        "local-sample":"<path of local git folder>"
      }
      collections: ["Ambience","Chainsets","Dashboards","DashboardViews",
      "Datasets","Forms","GISExplorer","Visualisations","Wev","WebStore",
      "Workflows","Privileges"]
    }
  }
}
```

The values support `{enc}` encryption.

Only the collections listed here will be stored by `git-push`.

Only the collections listed here and included in the it repository will be loaded by `git-pull`.

Any collections not listed here will not be used, even if they are in the Git repository.

This section is for Ambience only and does not apply to Repertoire.

---

11. Disabling Modules

11.1. Developer Module

The Developer module is useful in test/UAT. But it should be disabled in production system, this is to avoid other users from using it, accidentally altering data or any attack on the server.

In the *application.conf* file, add the following:

```
ambience.developer.enabled = false
```

This will disable the Developer module regardless of the privileges given. The module will still be loaded but the output will appear as:

```
12:28:20.234 INFO ambience.developer.DeveloperModule – Developer
module enabled: false
```

This section is for Ambience only and does not apply to Repertoire.

11.2. Themes Module

The Themes module allows you to manage and edit the theme (colour, font, etc.) used in Ambience. To have a uniform and standard look on the software, this module may be disabled to avoid other users from changing the settings in the theme.

In the *application.conf* file, add the following:

```
ambience.theme.enabled = false
```

This will disable the Themes module regardless of the privileges given. The Themes module will still be loaded with a "(Disabled)" label beside the title. There will be no response to any selection done in the module.

This section is for Ambience only and does not apply to Repertoire.

12.External Vault

The Secrets module acts as a repository and manager for secrets. These secrets can be stored either internally (in Ambience) or externally (using Hashicorp vault).

By default, the secrets are stored internally in Ambience and external vault is disabled.

To use external vault (in this case, its Hashicorp), install the Hashicorp, then enable the external vault in the *application-conf* file.

Below is the procedure to install Hashicorp and enabling Hashicorp in Ambience.

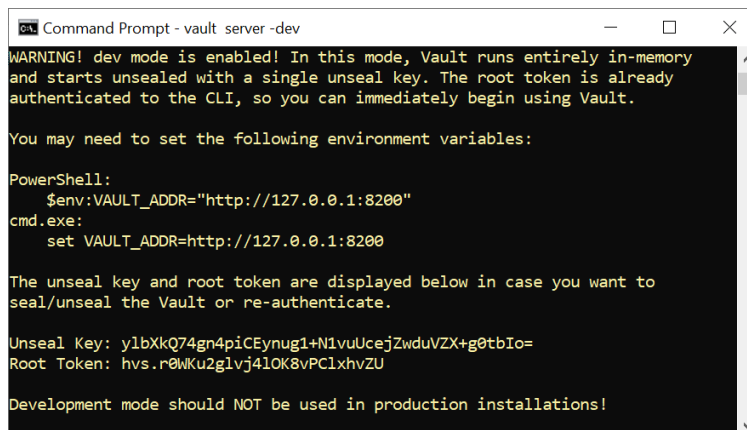1. The Hashicorp vault can be downloaded from the below link:

    https://developer.hashicorp.com/vault/downloads

    Select the correct version for your OS and follow the instructions in the webpage to install the vault.

2. There are two modes to run the vault:

    a. Developer mode – which is held in memory

    b. HCL configuration mode – starts server with an HCL configuration file

3. To start the server, open a terminal window and navigate to the location where the Hashicorp is installed.

    1. Use the following command to run the server in developer mode:

        ```
        vault server -dev
        ```

        The vault server will produce a token. Copy the token.

        ```
        Command Prompt - vault server -dev                       —    □    ×

        WARNING! dev mode is enabled! In this mode, Vault runs entirely in-memory
        and starts unsealed with a single unseal key. The root token is already
        authenticated to the CLI, so you can immediately begin using Vault.

        You may need to set the following environment variables:

        PowerShell:
            $env:VAULT_ADDR="http://127.0.0.1:8200"
        cmd.exe:
            set VAULT_ADDR=http://127.0.0.1:8200

        The unseal key and root token are displayed below in case you want to
        seal/unseal the Vault or re-authenticate.

        Unseal Key: ylbXkQ74gn4piCEynug1+N1vuUcejZwduVZX+g0tbIo=
        Root Token: hvs.r0WKu2glvj4lOK8vPClxhvZU

        Development mode should NOT be used in production installations!
        ```

    2. Use the following command to run the server with an HCL configuration file:

        ```
        vault server -config=<path>
        ```

        where `<path>` is the location of the HCL configuration file.

        Once the server is running, use the following command to generate a token:

        ```
        vault token create
        ```

        Copy the token.

4. Enable Hashicorp and add the token copied in step 3 using the *application.conf* file in the Ambience "bin" folder. Edit the fields in bold as follows:

```
ambience.secrets-store.location {
  default {
  ...
  }
  hashicorp {
    type = "hashicorp"
    enabled = true
    address = "http://127.0.0.1:8200"
    vault = secret"
    token = "<insert your value here>"
    is-addable = true
    is-linkable = true
  }
}
```

5. Restart the Ambience server for the Hashicorp vault to take effect.