FORMCENTRIC

Installation Manual

Version 4.5.2

Formcentric Analytics: Installation Manual

Breite Str. 61, 22767 Hamburg
Germany

Disclaimer

We reserve the right to alter the software and the contents of the manual without notice. We accept no liability for the accuracy of the contents of the manual, nor for any losses that may result from the use of this software.

Trademarks

In the course of this manual, references may be made to trademarks that are not explicitly marked as such. Even if such a mark is not given, the reader should not conclude that the name is free of third-party rights.

Please note: Printed copies are uncontrolled when printed.

Access to Documentation

You can always find the latest version of the Installation Manual in the Help centre help.formcentric.com. Older versions and additional information are available in the Formcentric Helpdesk helpdesk.formcentric.com.

1. Introduction

This manual describes how to install the Formcentric Analytics Backend and Reporting web app.

The first part describes the steps to install the Formcentric Analytics Backend web app and is then followed by the instructions for the Formcentric Analytics Reporting web app. You will also find out which property files need adjusting in order to configure the web applications to your requirements. Both the Backend and the Reporting web applications are provided to you as WAR files and Spring Boot JAR files. We recommend the use of the Spring Boot variant.

The following figure illustrates the overall Formcentric Analytics system and offers an initial introduction to the architecture.

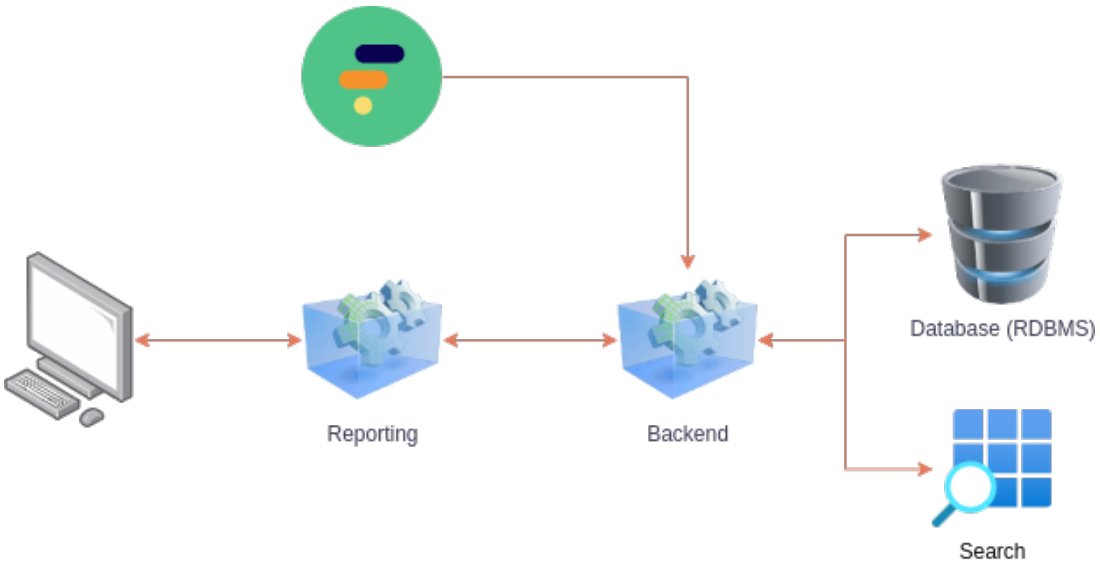

Figure 1.1. Analytics components

This section provides a brief description of the individual components.

Reporting: The Reporting component provides the user interface that you see in your browser. You use this application to manage Analytics and view the collected data.

Backend: The Backend component supplies the data from the database (RDBMS) and the search engine to Reporting. This component also receives and stores data from the Formcentric forms submitted.

Database: The Database component is used to store the submitted forms. The following database management systems (RDBMS) are supported: PostgreSQL, MySQL, MariaDB, Oracle, Microsoft SQL Server and DB2.

Search engine: Either *Solr* or *Elasticssearch* are used to handle all search queries. These search engines permit rapid and efficient searching of the data in Analytics.

2. Backend web app

This chapter provides you with everything you need to know about the setup, installation and administration of the Formcentric Analytics Backend web app. The system requirements for the component are covered as well as the various options available to you for installation. You will also receive insights into how to monitor and manage the application, as well as detailed explanations of the configuration parameters needed both for Formcentric Analytics itself and for the Solr enterprise search engine. This chapter also covers access from the content management system as well as the access privileges required for the database user.

2.1. System requirements

This section describes the requirements for operating the Formcentric Analytics Backend web app. Before an initial installation or update of Formcentric Analytics, you can consult the following table to identify the third-party component versions supported.

Not all of the components will be necessary for operation. As one example, RabbitMQ integration is an optional interface that can be added at any time, as required. Further information is available in the corresponding sections.

2.1.1. Databases

Formcentric Analytics supports the following databases.

Database	Status
PostgreSQL	
PostgreSQL 17	supported
PostgreSQL 16	supported
PostgreSQL 15	supported
PostgreSQL 14	supported
PostgreSQL 13	supported
PostgreSQL 12	deprecated
PostgreSQL 11	deprecated
PostgreSQL 10	deprecated
MySQL	
MySQL 8	supported
MariaDB	
MariaDB 11.x	supported

Database	Status
MariaDB 10.x	supported
Oracle	
Oracle Database 21c	supported
Oracle Database 19c	supported
Oracle Database 18c	supported
Microsoft SQL Server	
MSSQL Server 2022	supported
MSSQL Server 2019	supported
MSSQL Server 2017	supported
IBM DB2	
IBM DB2 11.5	supported

2.1.2. Java

Formcentric Analytics can be used with the following Java versions. Execution must take place inside a servlet container, version 5.0 or later (e.g. Apache Tomcat 10).

Java	Status
OpenJDK 21	supported
OpenJDK 17	supported

2.1.3. Solr

Formcentric Analytics requires the use of a search engine. If you decide to use Solr, further information can be found here Section 2.5.2, “Solr”.

Solr	Status
Apache Solr 9.0.x - 9.8.x	supported
Apache Solr 8.11.x	deprecated

2.1.4. Elasticsearch

Formcentric Analytics requires the use of a search engine. If you decide to use Elasticsearch, further information can be found here Section 2.5.1, “Elasticsearch”.

Elasticsearch	Status
Elasticsearch 8.17.x	supported
Elasticsearch 8.16.x	supported
Elasticsearch 8.15.x	supported

2.1.5. RabbitMQ/AMQP

RabbitMQ lets you integrate your own applications with Formcentric Analytics. This is an optional component.

RabbitMQ	Status
RabbitMQ 3.13	supported
RabbitMQ 3.12	supported
RabbitMQ 3.10	deprecated
RabbitMQ 3.9	deprecated

2.2. Installation

The Backend web app is supplied in two variants. You can run the Backend with a WAR in a servlet container or as a standalone Spring Boot JAR.

We recommend the use of the Spring Boot JAR. This offers various advantages compared with a WAR deployment. These include the following:

- Both the application and logging are easier to configure (see below)
- Simple updates to new versions, since only the JAR needs to be swapped out
- The JAR is simpler to run in a Docker container
- You do not need to select a compatible servlet container – a suitable Tomcat servlet container is included with the JAR.

Each variant is configured slightly differently. The configuration is described below.

2.2.1. General

Regardless of the variant used, the following configuration steps must be completed:

Set up a schema
In your database, create a schema that will be used by the Formcentric Analytics Backend. Ensure that both the schema and the database support UTF-8 encoding, so as to avoid potential losses of data and encoding errors.

If you are using an IBM DB2 database, please also ensure that you are using a table space with a page size of at least 16 kB.

Create a database user
Create a user for the schema. The user must have permission to execute all CRUD operations. All tables are created automatically when the application first starts and migrated during updates. Accordingly, the user needs permission to manage tables within the schema.

Full details of the access privileges required can be found in the chapter Section 2.7, “Access rights needed for the database user”.

Search engine
The Backend requires a *search engine*. You can choose between Solr and Elasticsearch. Please see the chapter Section 2.5.2, “Solr” for details of Solr setup and the chapter Section 2.5.1, “Elasticsearch” for details of Elasticsearch setup.

2.2.2. Spring Boot

You only need to read this section if you want to use the Spring Boot JAR, as recommended.

For a better understanding of the following sections, here is a diagram of the directories and files:

```
.
|-- formcentric-backend-webapp-boot-${project.version}.jar
|-- logback-spring.xml
|-- application.properties
\-- lib/
    \-- your-jdbc-driver-here.jar
```

JDBC driver
The Formcentric Analytics Backend web app does not ship with its own database drivers. To ensure a JDBC4 driver is found in the class path, place the driver in a */lib* folder next to the executable JAR.

Logging
The Spring Boot variant for the Backend uses Logback. For configuration, we recommended creating a file with the name *logback-spring.xml* alongside the executable JAR and referencing this file in the *application.properties* (see below).

You can find the standard configuration used in the JAR at */BOOT-INF/classes/logback-spring.xml*. We recommend that you use this as the starting-point for your configuration.

A general description of the XML configuration for Logback can be found at the following address:https://logback.qos.ch/manual/configuration.html.

Configuration
To configure the Backend, we recommend creating an *application.properties* in the same directory as the JAR. Alternatively, you can set the parameters with environment variables. For further Backend configuration details, please see Section 2.4, “Configuration”.

An example of the content in *application.properties* is shown below:

```
# active database and search profile:
spring.profiles.active=postgres,solr
```

```
#########################################################################
## Database settings
#########################################################################

schema.name=mwf_analytics
spring.datasource.username=mwf_analytics
spring.datasource.password=Monday123
spring.datasource.driver-class-name=org.postgresql.Driver
spring.datasource.url=jdbc:postgresql://analytics-database:5432/${schema.name}

### Search & Solr Settings
analytics.solr.url=http://analytics-solr:8983/solr/
analytics.solr.snippet.count=50
analytics.solr.collection=mwfanalytics

#########################################################################
## You can configure logging the 'Spring Boot way' (shown below) but
## we recommend to go with the logback-spring.xml configuration.
#########################################################################

### configuration with an external logback-config-file:
logging.config=file:logback-spring.xml

### logging the spring boot way:
#logging.level.root=INFO
#logging.level.com.formcentric.backend=DEBUG
```

Here are several examples of configuring the Backend with environment variables, which is a popular option for containers:

```
MANAGEMENT_SERVER_PORT=9090
MANAGEMENT_ENDPOINTS_WEB_EXPOSURE_INCLUDE=prometheus,health
ANALYTICS_ACTUATOR_SECURITY_ENABLED=false
```

Starting the application
The application can be started in the configuration as shown with *java -Dloader.path="lib" -jar formcentric-backend-webapp-boot-4.5.2.jar*.

2.2.3. Servlet container

You only need to read this section if you want to run the Backend in a servlet container.

JDBC driver
The Formcentric Analytics Backend web app does not ship with its own database drivers. You should either configure your servlet container so that it provides the drivers or you should copy your JDBC 4 driver into the directory *WEB-INF/lib*, so as to add it to the classpath.

Logging
You configure logging in the file *WEB-INF/classes/logback-spring.xml*.

A general description of the XML configuration for Logback can be found at the following address:https://logback.qos.ch/manual/configuration.html.

Configuration
To configure the Backend, you can either unpack the WAR file and complete your configuration in *WEB-INF/classes/application-backend.properties* or you can set the parameters with environment variables. For further Backend configuration details, please see Section 2.4, “Configuration”.

2.3. Monitoring and management

This section describes the options available to you for monitoring the application and how you configure the corresponding settings.

If the setting *analytics.actuator.security.enabled* (see also Section 2.4, “Configuration”) is active, then the client must authenticate itself in order to access the management endpoints. If the Backend is installed as a Spring Boot JAR (see Section 2.2.2, “Spring Boot”), then we recommend that you set this setting to *false* and instead use *management.server.port* to separate the management server port from the server port. This significantly simplifies the configuration of monitoring systems.

Take care to ensure that unauthorised parties cannot access the port you have set as the *management.server.port*. The best way to do this is to use an upstream proxy that only forwards traffic to the required port (*server.port*).

Analytics uses Spring Boot Actuator. Actuator is a mature management framework that offers a wide range of configuration options. The default Analytics settings are listed below:

management.endpoints.web.exposure.include: This specifies which endpoints are made available via *web*. Various endpoints can be published (see also https://docs.spring.io/spring-boot/docs/3.2.x/reference/html/actuator.html#actuator.endpoints). The default that Analytics uses out of the box (*info,health,metrics*) is described in the following table:

…/actuator/info	This accesses general information about the application, such as the following, for example: `{ "build": { "name": "formcentric-backend", "version": "${project.version}", "timestamp": "2023-01-19T09:09:04+0000" } }`
…/actuator/health	This accesses the health status of the application. This endpoint is typically used by monitoring applications. Normally, the output is as follows:

	`{ "status": "UP" }` If more details need to be shown, you can configure this (see https://docs.spring.io/spring-boot/docs/3.2.x/reference/html/actuator.html#actuator.endpoints.health).
…/actuator/metrics	This can be used to access various metrics for the application (see also Section 2.3.1, “Metrics”).

management.info.env.enabled: The default setting is *true*. The general information that is accessed via *…/actuator/info* (see above) is written to the properties during the build.

management.health.rabbit.enabled: The default setting is *${analytics.rabbitmq.enabled:false}*. The health status of RabbitMQ is only evaluated if RabbitMQ is *enabled*.

management.health.mail.enabled: The default setting is *${mail.sending.enabled:false}*. The health status of the mail server is evaluated only if email sending has been activated.

2.3.1. Metrics

Spring Boot Actuator provides (*…/actuator/metrics*) very detailed metrics about the application (see https://docs.spring.io/spring-boot/docs/3.2.x/reference/html/actuator.html#actuator.metrics.endpoint).

If you access the API without additional parameters, a list of available metrics is returned. If you want to monitor a particular metric, then you append the corresponding name to the URL. In the following example, the memory used metric is being accessed:

```
http://<host>/actuator/metrics/jvm.memory.used
```

Accessing individual values only is also possible, as shown in the following example:

```
http://<host>/actuator/metrics/jvm.memory.used?tag=area:nonheap
```

2.3.2. Prometheus

Prometheus is one of the most popular monitoring systems and is supported by Spring Boot Actuator.

The Prometheus (https://prometheus.io/) metrics API is not activated by default. Add the value *prometheus* to *management.endpoints.web.exposure.include* to activate it. For more configuration details, please see the following address: https://docs.spring.io/spring-boot/docs/3.2.x/reference/html/actuator.html#actuator.metrics.export.prometheus

2.4. Configuration

This section explains all of the configuration parameters used with Formcentric Analytics.

As a first step, you need to decide which database and search engine you will be using:

spring.profiles.active: You use this configuration parameter to specify the database configuration that you will use for the Backend. Possible values are: *postgres*, *mysql*, *mariadb*, *db2*, *mssql* or *oracle*. Next, you specify which search engine will be used by the Backend. Possible values here are: *solr* or *elastic*. For example, if you want to use an Oracle database with Elasticsearch, you would enter "*oracle,elastic*" here.

If you run the application within FirstSpirit, you can complete your configuration in Server Manager.

2.4.1. Database

The following parameters are required for connecting to the database.

spring.datasource.username: Here, enter the username for the user account created above.

spring.datasource.password: Here, enter the password for the user account created above.

spring.datasource.driver-class-name: Here, enter the class name for the JDBC driver that you are using.

spring.datasource.url: Here, enter the JDBC URL that is required by your database.

2.4.2. General configuration

This section includes general configuration parameters for Formcentric Analytics.

analytics.security.oauth2.clientSecret: This is a password that you must create and store yourself. Please note that the exact same secret must also be stored in the Reporting web app.

analytics.security.oauth2.analyticsClientSecret: This is a password that you must create and store yourself. This enables access to Formcentric Analytics from the CMS (see Section 2.6, “Access from the content management system”).

analytics.security.oauth2.apiTokenValidity: Optional: number of days for which a self-generated API token is valid. If –1 is entered for this validity period, API tokens do not expire. (Default: `365`)

If the OAuth2 client secrets *analytics.security.oauth2.clientSecret* and *analytics.security.oauth2.analyticsClientSecret* have not been configured, these are generated automatically when the application starts and logging is set to "INFO".

analytics.actuator.security.enabled: Optional: Here you can specify whether the Spring Boot Actuator endpoints are secured by OAuth2 or that no authentication is necessary when making a query. (Default: *true*)

analytics.registration.enabled: Use this property to activate an option for users to register their own Analytics account on the login page. (Default: *false*)

export.delimiter: Use this property to set the character used for separating lists in exported CSV or Excel files. (Default: *;*)

export.textQualifier: Use this property to set the text qualifier. This character is used to mark entries as text if they contain the delimiter. (Default: *"*)

automation.cron: A cron expression that schedules the automation jobs for Formcentric Analytics. The default configuration executes these jobs overnight at 01:00. The cron expression is interpreted in UTC.

automation.enabled: Use this property to activate or deactivate the execution of automation jobs. (Default: *true*)

analytics.revisions.enabled: Use this property to activate or deactivate revisions to form entries. (Default: *true*)

2.4.3. Send as Email

The following parameters configure sending mail from Formcentric Analytics. As mail sending is restricted to functions related to user management (activation link, forgotten password), a mail server is not mandatory when using an external directory service and can therefore be deactivated.

mail.sending.enabled: Use this property to activate or deactivate the sending of email from Formcentric Analytics. (Default: *true*)

mail.user: The account used by Formcentric Analytics to authenticate with the mail server.

mail.password: The password for the mail server user.

mail.host: The mail server host.

mail.port: The mail server port.

mail.properties.*: All other JavaMail properties can be configured by using the *mail.properties* prefix.

mail.properties.transport.protocol: The mail server transport protocol.

mail.properties.smtp.auth: Use this property to activate/deactivate use of the AUTH command by the mail user.

mail.properties.smtp.starttls.enable: Set this property if STARTTLS must be used.

mail.properties.smtp.socketFactory.class: Java class to set up SMTP sockets. Remove this property if plaintext connections are used.

mail.properties.debug: Use this property to activate/deactivate extra debugging output while mail is being sent.

mail.from.address: Here, you can configure the email address that Formcentric Analytics uses to send its email.

mail.from.name: This property is used to set the name that is shown as the sender for all mail sent by Formcentric Analytics.

mail.reporting.url: Here, enter the URL to the Reporting component within Formcentric Analytics.

mail.language: Configure this property to set the language used for sending email.

2.4.4. LDAP and Active Directory integration (optional)

The parameters listed below are used to integrate a Lightweight Directory Access Protocol (LDAP) or Active Directory server with Formcentric Analytics for user authentication. The rest of this documentation may use just "LDAP" but this always means "and Active Directory" if not explicitly excluded.

For external users to be able to log in, the user must be in at least one LDAP group that is known to Formcentric Analytics. Administrators therefore have the option of synchronising LDAP groups as part of system management. If all login attempts fail, this is a common source of errors and should be repeated. In group management, you can assign permissions and roles to an LDAP group, so as to make all members of a group administrators, for example.

If you have deployed LDAP over SSL (LDAPs), make sure that you add the LDAP certificate to your TrustStore.

Configuration parameters for the server: The server parameters use the prefix *analytics.ldap*. This prefix is also used for the Active Directory server.

spring.profiles.include: Extend this parameter by specifying a Spring profile if you want to use an external user directory (e.g. `spring.profiles.include=db2,activeDirectory`). To do so, select one of the following profiles: *ldap*, *activeDirectory*.

analytics.ldap.userDn: Username for access to the LDAP server.

analytics.ldap.password: Password for access to the LDAP server.

analytics.ldap.url: URL of the LDAP server.

analytics.ldap.baseDn: The base DN of the user directory from which all users and groups can be reached (e.g. `dc=mydomain,dc=com`).

analytics.ldap.subdomainDNs[0..n]: Optional: other subdomain DNs of the user directory from which all users and groups can be reached (e.g. `dc=subdomain1,dc=mydomain,dc=com`). For each additional subdomain, a new entry with the fully qualified DN of the subdomain must be added (e.g. `analytics.ldap.subdomainDNs[1]=dc=subdomain2,dc=mydomain,dc=com`)

analytics.ldap.domainLabel: Human-readable name of the domain used as a default for the user during registration.

analytics.ldap.userSearchBase: Specifies an object in the directory tree under which the user search is executed (e.g. *ou=people* for LDAP or *cn=users* for Active Directory).

analytics.ldap.userSearchFilter: Specifies an LDAP search filter to apply to the user search (which is run using the specified search base) (e.g. *(uid={0})* for LDAP or *(samaccountname={0})* for Active Directory).

The placeholder *{0}* is replaced with the username entered before executing the search.

A general description of the search filter syntax is available from the following link:http://www.faqs.org/rfcs/rfc2254.html.

analytics.ldap.userDnPattern: Pattern that is used to generate the distinguished name (DN) for a user. You only need to specify this parameter if you have selected the *LDAP* directory type (e.g. *uid={0},ou=people*).

analytics.ldap.mailAttribute: Name of the user object attribute field in which the email address can be found. The email address is saved in the Analytics database.

analytics.ldap.groupSearchBase: Specifies an object in the directory tree under which the group search is executed (e.g. *DC=company,DC=com* for LDAP or *cn=users* for Active Directory).

analytics.ldap.groupSearchFilter: Specifies an LDAP search filter to apply to the group search that is run using the specified search base (e.g. *(objectclass=groupOfUniqueNames)* for LDAP or *(objectclass=group)* for Active Directory).

analytics.ldap.groupsImportFilter: All of the groups for which specific permissions will be granted within Formcentric Analytics must first be imported into the internal user directory used by Formcentric Analytics. You can use the *group import filter* field to specify an LDAP search filter that will be applied in order to select the LDAP groups to import from the list of available groups (e.g. *(cn=*AnalyticsUsers*)*). If you do not enter anything into this field, then all of the groups identified by the LDAP search filter field *search filter for groups* will be imported.

analytics.ldap.userGroupsSearchFilter: This LDAP search filter is used to identify the groups in which a user is a member. You only need to specify this parameter if you have selected the *LDAP* directory type (e.g. *(memberUid={0})*). The placeholder *{0}* is replaced with the username entered before executing the search.

analytics.ldap.removeExternalUsers.enabled: Here you can activate (*true*) or deactivate (*false*) the automatic removal of LDAP/AD users from Analytics.

analytics.ldap.removeExternalUsers.cron: Cron expression that controls when LDAP/AD users are removed from Analytics. Users are removed only if they can no longer be found in LDAP/AD.

analytics.ldap.adBindDomainFromBaseDn: Only affects Active Directory: Here you can specify whether the user's *domain* is determined (*true*) or is not determined

(*false*) during login from the *analytics.ldap.baseDn* parameter. If you do not set the parameter, the default is *true*.

analytics.ldap.adBindDomain: Only affects Active Directory: Here you can specify a *domain* that is always to be used if *analytics.ldap.adBindDomainFromBaseDn* is set to *false*. If you do not set this parameter, the user must specify their domain themselves during login.

2.4.5. SSO/OpenID Connect integration (optional)

This section explains how to configure OIDC (OpenID Connect). Using OIDC gives you a way to implement single sign-on (SSO).

Please note the following changes to Analytics that result from the use of OIDC:

- You cannot use OIDC at the same time as LDAP and Active Directory.
- Using OIDC deactivates user management functionality in Analytics. All user information is provided instead by the JWT (JSON web token).
- Analytics no longer issues any OAuth2 tokens of its own. Your tokens are provided solely by your OIDC server. API access management is also deactivated. Any access with a valid JWT receives corresponding rights to the Analytics Backend interface.
- Analytics creates groups automatically if they are included in the JWT and identified as an Analytics group. If you delete groups, these are created again automatically once a user from this group is identified.
- You can only assign form rights to groups and can no longer assign rights to individual users.
- Menu items in the Reporting user interface that are not relevant for OIDC (such as user management) are not shown.

Most of the settings you need to configure are for the Backend. Settings for Reporting can be found under: Section 3.4.2, “SSO/OpenID Connect integration (optional)”.

Spring

spring.profiles.include: To activate OIDC, you need to add the *oidc* Spring profile here (e.g. `spring.profiles.include=db2,solr,oidc`).

spring.security.oauth2.resourceserver.jwt.issuer-uri: Enter the URI of the authorisation server that you have included in your JWTs under *iss*. Your authorisation server must supply a provider configuration endpoint (see https://openid.net/specs/openid-connect-discovery-1_0.html#ProviderConfig), also known as an authorisation server metadata endpoint (see https://datatracker.ietf.org/doc/html/rfc8414#section-3). If this is not the case, please contact our support team. We can help you complete your configuration.

While parsing the JWT, Analytics only looks at the *claims* (the payload) and ignores the header.

All settings that describe a path are a JSON path expression. All path settings are JSON path expressions. You can separate multiple expressions with commas. You can omit the default leading *$* in the Analytics configuration. Under these paths, Analytics expects to find either string arrays or individual strings – other parameter types are ignored.

As a simplified approach to configuration, we recommend decoding a JWT from your server (e.g. with https://jwt.io/) and then entering the payload into an online JSON path testing tool (e.g. https://jsonpath.com/).

Groups

The settings below are used by Analytics to determine the groups in which the token owner is a member:

analytics.security.oidc.groups-claim-paths: Define one or more JSON path expressions here that Analytics can use to find groups in the JWT.

analytics.security.oidc.group-filter-prefixes: This is an optional setting for filtering the groups found by prefix. If you define "analytics_" as a prefix, for example, then groups are only considered if their name starts with *analytics_*.

analytics.security.oidc.admin-group-name: Use this setting to specify the name of the admin group. Analytics requires this information to determine which users are administrators. This is achieved by using the *admin* role, which is assigned to the configured group automatically. All user accounts in this group receive the administrator role. Once Analytics has identified and filtered the groups, the prefix is then removed from the group name. If the remainder of the group name matches the name as specified here, then the user account is added to the admin group. For example:

```
# 1. Groups found via `...oidc.groups-claim-paths`:
#    ['ana_admins', 'ana_foobar']
# 2. Groups filtered using `...oidc.group-filter-prefixes=ana_`:
#    ['admins', 'foobar']
analytics.security.oidc.admin-group-name=admins
# Analytics will create a group named `admins`,
# which will be assigned the `admin` role.
```

analytics.security.oidc.user-group-name: Use this to specify the name of the user group. This group receives the *user* role automatically and all user accounts in this group are also assigned the user role. Without this role, a user cannot perform any actions in Analytics (unless the user is an administrator). Once Analytics has identified and filtered the groups, the prefix is then removed from the group name. If the remainder of the group name matches the name as specified here, then the user account is added to the user group. This functionality is essentially the same as that provided by *analytics.security.oidc.admin-group-name*.

Roles

Analytics works with roles. These roles are assigned to a user in much the same way as assignment to groups. Analytics recognises the following roles:

role *admin***:** Users with this role are administrators and have comprehensive rights.

role *user***:** Users with this role are normal users with limited rights.

role *api_consumer***:** This role is intended as a machine role. Formcentric is an example of this kind of "consumer". However, you are also free to develop your own application that uses this role to access the API.

We recommend using groups to configure access to the "admin" and "user" roles, and advise against direct assignment to these roles. Using groups to assign roles is significantly more flexible, as you can assign rights at group level, so that all members of these groups are granted the same set of rights. The creation of role mappings is optional. However, you must create a mapping for Formcentric, otherwise Formcentric will be unable to write any data into Analytics.

The settings below are used by Analytics to determine the roles to assign to the token owner:

analytics.security.oidc.role-claim-paths: As when searching for groups (see *analytics.security.oidc.groups-claim-paths*), Analytics uses one or more JSON path expressions in the JWT to look for roles.

analytics.security.oidc.role-filter-prefixes: As with groups, you can use a prefix to filter the roles. If you choose the prefix "analytics_", for example, you can ensure that roles are considered only if their name starts with *analytics_*. This configuration is optional.

analytics.security.oidc.role-mappings.<role_name>: This maps the role name to the Analytics role. Analytics first identifies the roles from the JWT and then uses the prefix to filter the names found. The prefix is then removed from the name. For example:

```
# 1. Roles found via `...oidc.role-claim-paths`:
#    ['ana_role_administrator', 'ana_role_consumer', 'somthing_different']
# 2. Roles after filtering (`...oidc.role-mappings.admin=ana_role`):
#    ['administrator', 'consumer']
# 3. Role mapping: analytics.security.oidc.role-mappings.<role_name>
analytics.security.oidc.role-mappings.administrator=admin
analytics.security.oidc.role-mappings.consumer=api_consumer
```

In this example, two role mappings are created:

1. The *administrator* role from the JWT is mapped to the *admin* role from Analytics.
2. The *consumer* role from the JWT is mapped to the *api_consumer* role from Analytics.

Case differences in the spelling of the Analytics role name are ignored. A leading *role_* can also be specified. Accordingly, the following role designations are all considered

identical: *role_api_consumer*, *api_consumer*, *Api_Consumer*, *API_CONSUMER* and *ROLE_API_CONSUMER*.

The Analytics Backend supplies the user data to Reporting. The following optional settings can be used to ensure that the username and the email address are displayed correctly in Reporting. If these settings are not configured, the corresponding data items are not displayed in Reporting.

analytics.security.oidc.preferred-user-name-path: This JSON path is used to search for the username in the JWT. If a string array is found, then Reporting displays the first string as the username.

analytics.security.oidc.email-path: This JSON path is used to search for the user's email address in the JWT. If a string array is found, then Reporting displays the first string as the email address.

2.4.6. RabbitMQ/AMQP (optional)

RabbitMQ lets you integrate your own applications with Formcentric Analytics.

analytics.rabbitmq.enabled: Use this property to activate (*true*) or deactivate (*false*) RabbitMQ integration.

analytics.rabbitmq.topic.enabled: Use this property to activate (*true*) or deactivate (*false*) the topic exchange type.

analytics.rabbitmq.headers.enabled: Use this property to activate (*true*) or deactivate (*false*) the headers exchange type.

spring.rabbitmq.host: Use this property to configure the RabbitMQ host. All of the configuration parameters supported by Spring RabbitMQ are also available here.

spring.rabbitmq.port: Use this property to configure the RabbitMQ port. All of the configuration parameters supported by Spring RabbitMQ are also available here.

spring.rabbitmq.username: Use this property to configure the RabbitMQ username. All of the configuration parameters supported by Spring RabbitMQ are also available here.

spring.rabbitmq.password: Use this property to configure the RabbitMQ password. All of the configuration parameters supported by Spring RabbitMQ are also available here.

2.4.7. Request ID

Formcentric Analytics can accept a request ID from the http request headers and output this to the log. This enables the tracking of a request across multiple log entries. To do this, Formcentric Analytics sets a variable *REQ-ID* in the mapped diagnostic context (MDC) of the logging framework used (see Section 2.2.2, "Spring Boot" and Section 2.2.3, "Servlet container").

You use the following parameters to configure this functionality.

analytics.http.requestId.enabled: You use this parameter to activate (*true*) or deactivate (*false*) this function. This functionality is activated by default.

analytics.http.requestId.headerName: You use this parameter to configure the name of the http header from which Formcentric Analytics accepts the request ID. The default value here is *x-request-id*.

analytics.http.requestId.internalRequestIdPrefix: If the request does not contain a request ID, Formcentric Analytics generates a request ID automatically. You use this parameter to generate the prefix for this automatically generated ID, which lets you distinguish IDs in the log more easily.

analytics.http.requestId.externalRequestIdPrefix: Formcentric Analytics can set a prefix in front of an external request ID (the ID from the request header) so as to make it clear in the log that this ID was not generated by Formcentric Analytics.

2.4.8. Encrypting configuration parameters

In the default configuration, sensitive data in configuration files is stored in plaintext. In the event of a security breach, attackers would gain access to valid login credentials. For this reason, you are given the option of storing passwords in an encrypted format. In this case, passwords are decrypted only when the application starts, using the stored encryption password. The password to be used for encryption must be stored in an environment variable before the application starts. The default environment variable used for encryption is *MWF_ENCRYPTION_PASSWORD*.

To encrypt or decrypt values, a command line tool is available, which can be accessed with the command

```
mvn dependency:copy -Dartifact=com.formcentric:encryption-cli:2.0:jar
      -DoutputDirectory=.
```

. The tool either encrypts or decrypts a value when executed.

```
java -jar encryption-cli-1.0.jar -p 'encryptionPassword'
      -e 'toEncrypt'
```

If run *without* parameters, the tool prompts the user for the password and the values to be encrypted/decrypted. Please note that the parameters must be entered in single quotation marks. The tool can be run with the following parameters:

Parameter	Description
-p or --encryptionPassword	Specifies the password to be used for encryption or decryption.
-d or --decrypt	Decrypts the value given after this parameter
-e or --encrypt	Encrypts the value given after this parameter
-? or --help	Shows the help file for the tool.

2.5. Search engine

A search engine is required to handle search operations. You can choose one of the following two search engines.

2.5.1. Elasticsearch

The following section explains how you run Elasticsearch.

Elasticsearch installation

Elasticsearch runs as a standalone application. For information about installing and configuring Elasticsearch, please see the product page https://www.elastic.co/guide/en/elasticsearch/reference/current/index.html.

Analytics creates the index in Elasticsearch automatically if this index is not yet present. Most index configurations can also be adjusted after the index has been created.

As an alternative to automatic index creation, you can also create the index manually beforehand. In this case, Analytics identifies the existing index when it starts and uses this index for all operations.

You can use the following example as guidance when creating the index yourself:

```
PUT http://localhost:9200/analytics
Authorization: Basic elastic elastic
Content-Type: application/json

{
    "aliases": {},
    "mappings": {
        "dynamic": true,
        "dynamic_templates": [
            {
                "double-mapping-always": {
                    "match_mapping_type": [ "double" ],
                    "mapping": { "type": "double" }
                }
            }
        ],
        "properties": {
            "formId": { "type": "keyword" },
            "dtFieldValues": { "type": "keyword", "store": true },
            "attachments": { "type": "binary" },
            "numbers": {
                "type": "nested",
                "properties": {
                    "_class": { "type": "keyword", "doc_values": false,
                        "index": false },
                    "value": { "type": "double" },
                    "key": { "type": "keyword" }
                }
            },
```

```
            "dates": {
                "type": "nested",
                "properties": {
                    "_class": { "type": "keyword","doc_values": false,
                        "index": false },
                    "value": { "type": "long" },
                    "key": { "type": "keyword" }
                }
            },
            "fieldValues": { "type": "text", "store": true },
            "recordId": { "type": "keyword", "index": true },
            "versionId": { "type": "keyword" },
            "statusId": { "type": "keyword" },
            "form": {
                "type": "nested",
                "properties": {
                    "_class": { "type": "keyword", "doc_values": false,
                        "index": false },
                    "value": { "type": "wildcard" },
                    "key": { "type": "keyword" }
                }
            },
            "meta": {
                "type": "nested",
                "properties": {
                    "_class": { "type": "keyword", "doc_values": false,
                        "index": false },
                    "value": { "type": "wildcard" },
                    "key": { "type": "keyword" }
                }
            },
            "_class": { "type": "keyword", "doc_values": false,
                "index": false },
            "inputMap": { "type": "binary" },
            "tenant": { "type": "keyword" }
        }
    },
    "settings": {
        "index": {
            "number_of_shards": "1",
            "number_of_replicas": "0",
            "refresh_interval": "1s"
        }
    }
}
```

Analytics assumes that an existing index is correctly configured with the right name. The mapping shown above must therefore be applied exactly as presented.

Once Elasticsearch has been set up, you will typically need to configure the following:

analytics.elastic.index: You use this parameter to specify the name of the index in Elasticsearch. The default name is *analytics*.

spring.elasticsearch.uris: Comma-separated list of the Elasticsearch instances to use. The default is *localhost:9200*.

spring.elasticsearch.username: Username for authentication with Elasticsearch. The default username is *elastic*.

spring.elasticsearch.password: Password for authentication with Elasticsearch. The default password is *elastic*.

SSL

This section is only relevant if you want to set up SSL.

Analytics uses the Spring Boot integration for Elasticsearch. SSL is correspondingly configured with Spring Boot (see https://docs.spring.io/spring-boot/reference/features/ssl.html).

Many configuration options are available here. By way of example, here is a sample configuration for a *PKCS12* keystore:

```
spring.ssl.bundle.jks.mybundle.key.alias=application
spring.ssl.bundle.jks.mybundle.keystore.location=/path/to/your/keystore.p12
spring.ssl.bundle.jks.mybundle.keystore.password=secret
spring.ssl.bundle.jks.mybundle.keystore.type=PKCS12
```

You can set the name *mybundle* to any name you want. This name value must then be configured in the Elasticsearch integration, for example:

```
spring.elasticsearch.restclient.ssl.bundle=mybundle
```

2.5.2. Solr

The following section explains how you run Solr.

Solr installation

The Solr search engine runs as a standalone application. For information about installation and configuration, please see the official product page http://lucene.apache.org/solr/.

To use the external Solr instance with Formcentric Analytics, you first need to create a new core with the name *mwfanalytics*. You can also specify the name of the core with *analytics.solr.collection*.

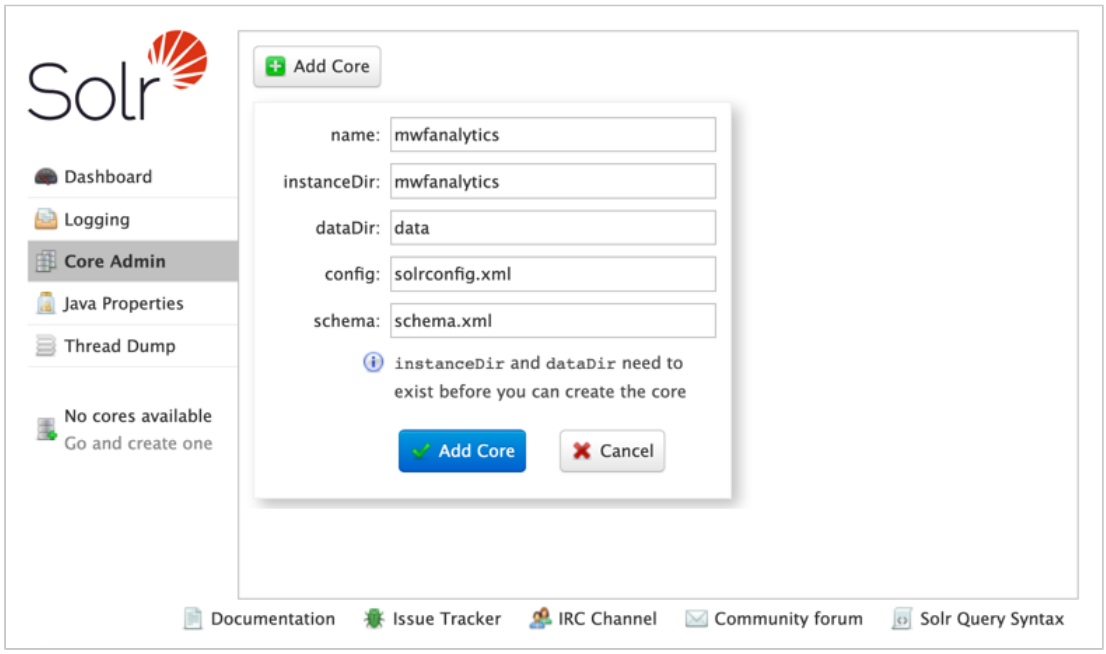

Figure 2.1. Core administration in the Solr administration interface

You can download the required configuration files *solrconfig.xml* and *schema.xml* from our Maven server.

```
mvn dependency:copy \ -Dartifact=com.formcentric.analytics:formcentric-backend-solr-config:4.5.2:t
```

Now enter the URL of the Solr server into the file *WEB-INF/classes/application-backend.properties* for the parameter *analytics.solr.url*.

Example: *analytics.solr.url=http://localhost:8983/solr/*

General settings

analytics.solr.user: You can specify the username for connecting to Solr here.

analytics.solr.pass: You can specify the password for connecting to Solr here.

analytics.solr.snippet.count: Maximum number of matches that should be determined for a search result.

SSL

You can use the following parameters to set up SSL:

analytics.solr.ssl.keyStoreType: Defines the keystore type. Possible values are (see also https://docs.oracle.com/javase/8/docs/technotes/guides/security/StandardNames.html#KeyStore): *jceks*, *jks*, *dks*, *pkcs11* or *pkcs12* (default)

analytics.solr.ssl.keyStore: Specifies the keystore file to use. We recommend using an absolute path here.

analytics.solr.ssl.keyStorePass: Specifies the password for the keystore. This setting is not needed if the keystore does not have a password.

2.5.3. Feeder

The feeder synchronises the database content with the search engine. Configure the following parameters for the feeder:

feeder.enabled: Activates or deactivates the feeder. The feeder is activated by default. This setting is useful if you are running several backends simultaneously but only one should be used to update the search engine.

feeder.id: Specify a unique ID for the feeder. The ID can be defined by using an arbitrary character string. If you are running multiple Backend applications, we recommend using unique Feeder IDs to avoid conflicts.

2.6. Access from the content management system

To access the Backend web app, an *access token* is required. The token is generated by using the *analytics.security.oauth2.analyticsClientSecret* (see Section 2.2, “Installation”).

Three methods are available for generating the token:

1. Backend web app

   The first method requires the use of a configured and operational Backend system. Only the value of the *access_token* is required from the server response.

   ```
   curl --user webforms-webapp-client:${WEBFORMS_CLIENT_SECRET}
         ${BACKEND_HOST}/oauth/token -d grant_type=client_credentials
   ```

   Result:

   ```
   {"access_token":"uFDXjCR+pWL+8iQIigJr9iFLcQm04G7NILrrY+bWcHg=", ...}
   ```

2. Reporting web app

   The second method requires the Backend web app and the Reporting web app. Administrators can configure the webforms-webapp-client in the Reporting web app system overview in the same way as for API accesses they have set up themselves, and request or invalidate tokens. For a detailed guide, please see the user manual for the Reporting web app.

3. Automatic generation in the CMS

   Instead of using a token generated at Backend web app runtime, you can also pass the *client secret* to the Formcentric CMS extension. In this case, the *BackendApiClient* automatically requests a token when first accessing Formcentric Analytics.

2.7. Access rights needed for the database user

The database user requires the permission to manage tables within the schema, since all tables are created automatically when the application first starts and also migrated during updates.

Alongside CRUD operations (*INSERT*, *SELECT*, *UPDATE*, *DELETE*), permissions for DDL operations are also required. The database user must be able to execute the following commands: *CREATE TABLE*, *ALTER TABLE*, *DROP TABLE*, *CREATE INDEX*, *DROP INDEX*, *CREATE PROCEDURE*, *DROP PROCEDURE*. The database user must also be authorised to execute database procedures.

Some databases work with sequences (DB2, Oracle, Postgres, MS SQL). For these databases, the database user must also be authorised to create and delete sequences (*CREATE SEQUENCE*, *DROP SEQUENCE*).

2.7.1. DB2

Tables must be reorganised in DB2 while the database schema is being migrated. To do this, the database user must be able to perform the following operation: *call SYSPROC.ADMIN_CMD('REORG TABLE <table>');*

2.7.2. Oracle

With Oracle, objects are placed in a recycle bin for some operations. This recycle bin is emptied with the following command: *purge recyclebin;*

During the schema migration, PL/SQL code must be executed. The database user must be able to execute code as shown below:

```
-- alter table MWF_FORM_SEARCH_NUMBER drop column ID if exist
DECLARE
    cnt integer;
BEGIN
    SELECT COUNT(*)
    INTO cnt
    FROM ALL_TAB_COLUMNS
    WHERE table_name = 'MWF_FORM_SEARCH_NUMBER'
      AND column_name = 'ID';

    IF (cnt = 1) THEN
        EXECUTE IMMEDIATE
            'alter table MWF_FORM_SEARCH_NUMBER drop column ID';
    END IF;
END;
```

2.7.3. MS-SQL

During the schema migration, Transact-SQL code must be executed. The database user must be able to execute code as shown below:

```
declare @var1 AS nvarchar(256);
```

```
select @var1 = TC.CONSTRAINT_NAME
from INFORMATION_SCHEMA.TABLE_CONSTRAINTS TC
inner join INFORMATION_SCHEMA.CONSTRAINT_COLUMN_USAGE CC on
  TC.CONSTRAINT_NAME = CC.CONSTRAINT_NAME
where CC.COLUMN_NAME = 'attachment_id'
  and CC.TABLE_NAME = 'mwf_form_attachments'
  and TC.CONSTRAINT_TYPE = 'PRIMARY KEY'
execute ('ALTER TABLE mwf_form_attachments DROP CONSTRAINT ' + @var1);
alter table mwf_form_attachments
  add constraint form_attachments_pk primary key (attachment_id, record_id);
```

Or:

```
declare @sql nvarchar(200)
declare @constName nvarchar(100)
set @constName = (
  select top 1 TC.CONSTRAINT_NAME
  from INFORMATION_SCHEMA.TABLE_CONSTRAINTS TC
  inner join INFORMATION_SCHEMA.CONSTRAINT_COLUMN_USAGE CC on
    TC.CONSTRAINT_NAME = CC.CONSTRAINT_NAME
  where TC.CONSTRAINT_TYPE = 'UNIQUE'
  and COLUMN_NAME='email'
  and TC.TABLE_NAME='mwf_user')
print @constName
set @sql = 'ALTER TABLE mwf_user DROP CONSTRAINT '+ @constName
Exec sp_executesql @sql;
```

2.7.4. Postgres

During the schema migration, PL/SQL code must be executed. The database user must be able to execute code as shown below:

```
DO $$DECLARE r record;
BEGIN
  FOR r IN
    SELECT
      tc.constraint_name, tc.table_name
    FROM
      information_schema.table_constraints AS tc
      JOIN information_schema.key_column_usage AS kcu
        ON tc.constraint_name = kcu.constraint_name
      JOIN information_schema.constraint_column_usage AS ccu
        ON ccu.constraint_name = tc.constraint_name
    WHERE constraint_type = 'UNIQUE'
      AND tc.table_name='mwf_user'
      AND (ccu.column_name='user_name' OR ccu.column_name='email')
  LOOP
    EXECUTE 'ALTER TABLE ' || quote_ident(r.table_name) ||
      ' DROP CONSTRAINT ' || quote_ident(r.constraint_name) ||
      ';';
  END LOOP;
END$$;
```

3. Reporting web app

This chapter provides you with comprehensive information about the system and browser requirements, as well as a detailed guide to the installation and configuration of the Formcentric Analytics Reporting web app.

3.1. System requirements

- Formcentric Analytics Backend 4.5.2

3.1.1. Java

Formcentric Analytics can be used with the following Java versions. Execution must take place inside a servlet container, version 5.0 or later (e.g. Apache Tomcat 10). Alternatively, you can execute the Spring Boot JAR (recommended).

Java	Status
OpenJDK 21	supported
OpenJDK 17	supported

3.2. Browser requirements

- Google Chrome (latest version)
- Mozilla Firefox (latest version or ESR)
- Microsoft Edge (latest version)

3.3. Installation

The Reporting web app is supplied in two variants. You can run Reporting with a WAR in a servlet container or as a standalone Spring Boot JAR.

We recommend the use of the Spring Boot JAR. This offers various advantages compared with a WAR deployment. These include the following:

- Both the application and logging are easier to configure (see below)
- Simple updates to new versions, since only the JAR needs to be swapped out
- The JAR is simpler to run in a Docker container
- You do not need to select a compatible servlet container – a suitable Tomcat servlet container is included with the JAR.

You can verify that the installation has been successful by accessing the Reporting web app and logging in. After an initial installation, the username is *admin* and the password is *admin*.

3.3.1. Spring Boot

You only need to read this section if you want to use the Spring Boot JAR, as recommended.

Logging
The Spring Boot variant for Reporting uses Logback. For configuration, we recommended creating a file with the name *logback-spring.xml* alongside the executable JAR and referencing this file in the *application.properties* (see below).

You can find the standard configuration used in the JAR at */BOOT-INF/classes/logback-spring.xml*. We recommend that you use this as the starting-point for your configuration.

A general description of the XML configuration for Logback can be found at the following address:https://logback.qos.ch/manual/configuration.html.

Configuration
To configure Reporting, we recommend creating an *application.properties* in the same directory as the JAR. Alternatively, you can set the parameters with environment variables. For further Backend configuration details, please see Section 2.4, “Configuration”.

An example of the content in *application.properties* is shown below:

```
### Backend Connection Settings
analytics.backend.url=http://analytics-backend:8080/

#########################################################################
## You can configure logging the 'Spring Boot way' (shown below) but
## we recommend to go with the logback-spring.xml configuration.
#########################################################################

logging.config=file:logback-spring.xml

### logging the spring boot way:
#logging.level.root=INFO
#logging.level.com.formcentric=DEBUG
```

Here are several examples of configuring the Backend with environment variables, which is a popular option for containers:

```
MANAGEMENT_SERVER_PORT=9090
MANAGEMENT_ENDPOINTS_WEB_EXPOSURE_INCLUDE=prometheus,health
ANALYTICS_ACTUATOR_SECURITY_ENABLED=false
```

3.3.2. Servlet container

You only need to read this section if you want to run the Backend in a servlet container.

Logging
You configure logging in the file *WEB-INF/classes/logback-spring.xml*.

A general description of the XML configuration for Logback can be found at the following address:https://logback.qos.ch/manual/configuration.html.

Configuration
To configure Reporting, you can either unpack the WAR file and complete your configuration in *WEB-INF/classes/application-reporting.properties* or you can set the parameters with environment variables. For further Backend configuration details, please see Section 2.4, “Configuration”.

3.4. Configuration

This section explains all of the configuration parameters used with Formcentric Analytics.

If you run the application within FirstSpirit, you can complete your configuration in Server Manager.

analytics.backend.url
URL of the Formcentric Analytics backend already deployed.

analytics.security.oauth2.clientSecret
Here, you must enter the same password that you used before during the Backend configuration.

3.4.1. Request ID

Formcentric Analytics can accept a request ID from the http request headers and output this to the log. This enables the tracking of a request across multiple log entries. To do this, Formcentric Analytics sets a variable *REQ-ID* in the mapped diagnostic context (MDC) of the logging framework used (see Section 3.3.1, “Spring Boot” and Section 3.3.2, “Servlet container”).

You use the following parameters to configure this functionality.

analytics.http.requestId.enabled: You use this parameter to activate (*true*) or deactivate (*false*) this function. This functionality is activated by default.

analytics.http.requestId.headerName: You use this parameter to configure the name of the http header from which Formcentric Analytics accepts the request ID. The default value here is *x-request-id*.

analytics.http.requestId.internalRequestIdPrefix: If the request does not contain a request ID, Formcentric Analytics generates a request ID automatically. You use this parameter to generate the prefix for this automatically generated ID, which lets you distinguish IDs in the log more easily.

analytics.http.requestId.externalRequestIdPrefix: Formcentric Analytics can set a prefix in front of an external request ID (the ID from the request header) so as to make it clear in the log that this ID was not generated by Formcentric Analytics.

3.4.2. SSO/OpenID Connect integration (optional)

To use OIDC in your Reporting web app, several settings must also be configured in the Reporting web app alongside the configuration of the Backend (see Section 2.4.5, “SSO/OpenID Connect integration (optional)”).

analytics.oidc.clientId: Enter the identifier of the application that is being used for the user login process.

analytics.oidc.authorityUrl: Enter the URI of the authorisation server that you have included in your JWTs under *iss*. For this setting, your authorisation server must supply a provider configuration endpoint (see https://openid.net/specs/openid-connect-discovery-1_0.html#ProviderConfig). This endpoint is also known as an authorisation server metadata endpoint (see https://datatracker.ietf.org/doc/html/rfc8414#section-3). If this is not the case, please contact our support team. We can help you complete your configuration.

analytics.oidc.scopes: Enter the scopes that will be queried by the OIDC server (use spaces to separate). If this setting is not configured, the default values *openid email profile* are used. Note that the *openid* scope must always be specified.

analytics.oidc.showLogout: Use this setting to specify whether the logout button is displayed in Reporting. The default value is *true*. If the user clicks the button, the user is also logged out of the OIDC server. If this action is not required, you can deactivate the button.

analytics.oidc.logoutView: This setting is for redirection handling after logout from Reporting (using the logout button – see above). You can have the user either redirected to an internal logout page (*ANALYTICS*) or taken directly to the login page for the OIDC server (*OIDC*). The Analytics logout page displays a single button. If clicked, the user is taken to the OIDC server login page. This page lets the user see that they have been logged out. If this parameter is not set, the default value *OIDC* is used. This means that the user is taken directly to the OIDC server login page after logging out.

3.4.3. Encrypting configuration parameters

In the default configuration, sensitive data in configuration files is stored in plaintext. In the event of a security breach, attackers would gain access to valid login credentials. For this reason, you are given the option of storing passwords in an encrypted format. In this case, passwords are decrypted only when the application starts, using the stored encryption password. The password to be used for encryption must be stored in an environment variable before the application starts. The default environment variable used for encryption is *MWF_ENCRYPTION_PASSWORD*.

To encrypt or decrypt values, a command line tool is available, which can be accessed with the command

```
mvn dependency:copy -Dartifact=com.formcentric:encryption-cli:2.0:jar
      -DoutputDirectory=.
```

. The tool either encrypts or decrypts a value when executed.

```
java -jar encryption-cli-1.0.jar -p 'encryptionPassword'
      -e 'toEncrypt'
```

If run *without* parameters, the tool prompts the user for the password and the values to be encrypted/decrypted. Please note that the parameters must be entered in single quotation marks. The tool can be run with the following parameters:

Parameter	**Description**
-p or --encryptionPassword	Specifies the password to be used for encryption or decryption.
-d or --decrypt	Decrypts the value given after this parameter.
-e or --encrypt	Encrypts the value given after this parameter.
-? or --help	Shows the help file for the tool.