DookuG module JAVA documentation

Table of Contents

Version: 1.1.0-SNAPSHOT

1. Overview

The project aims to enable general template-based document generation for its clients.

1.1. Components

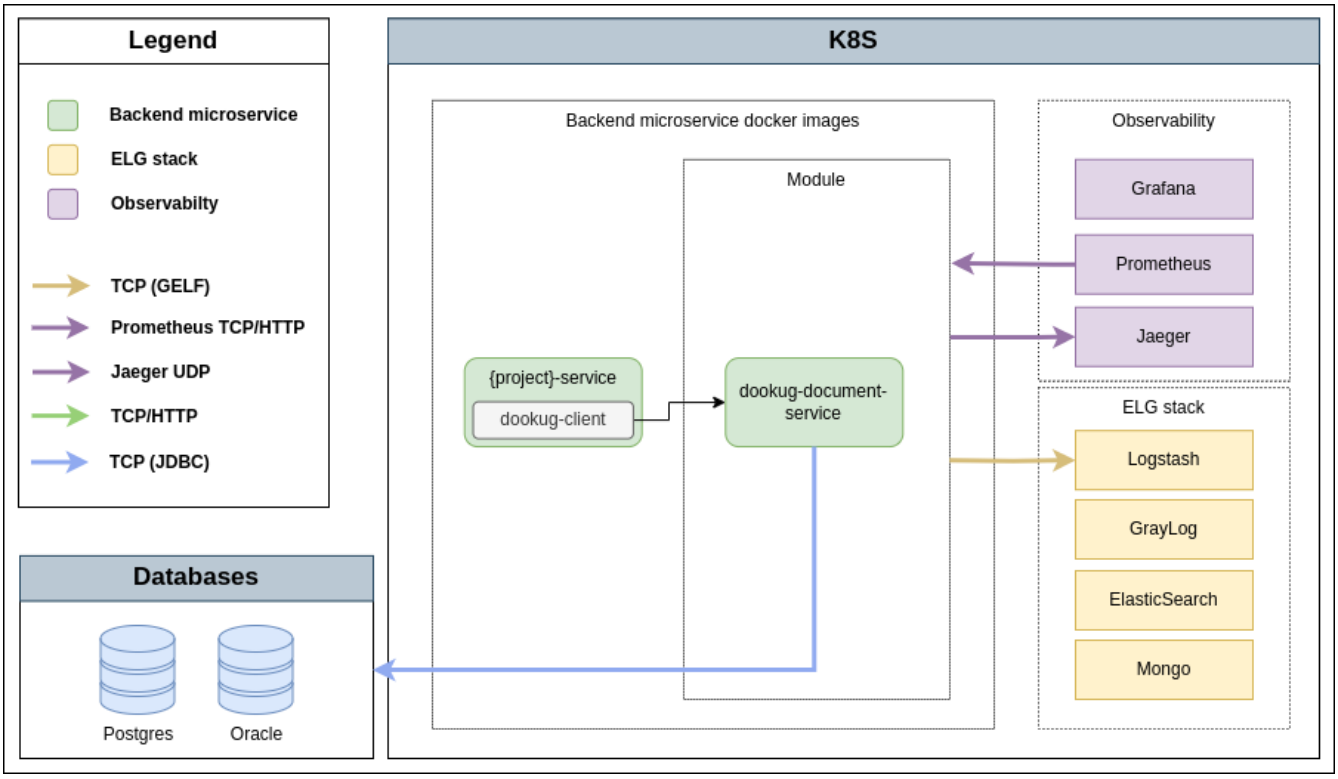

Figure 1. Architecture Diagram

Table 1. DookuG Backend 1.1.0-SNAPSHOT

Type	Module	Version
Internal	Oracle	21.3.0-xe
	PostgreSQL	dwh postgres image 0.10.0

2. Backend

2.1. Services

2.1.1. DOOKUG-DOCUMENT Service

2.1.1.1. Document Generation

The module provides multiple options for document generation. For each option, we need to specify the template and how the document should be processed by the engine, in what format we want to receive the file, and how we want to store the generated document. If the format of the generated document is not a STRING value but an engine that can process the document is not provided, or if STRING format is desired but PDF_BOX engine is specified, an INVALID_INPUT error will be returned. When generating documents based on the initial template, parameters must also be specified. There are several options for specifying it, one of which is to send it in key-value pairs. The other option is to expect a json structure that is to be sent in base64binary form in the request.

2.1.1.2. Document generation based on request body

```
POST /internal/dookug/document/generate/inline
```

During generation, we submit the initial template and the data associated with the generation in the request body: template and engine processing the document, response format, and document storage method.

Sample document generation request - JSON parameters in base64binary form (trimmed for readability):

```
Request method: POST
Request URI:    http://localhost:8082/internal/dookug/document/generate/inline
Headers:        Accept=application/json
                Content-Type=application/json
```

```
{
  "context": {
    "requestId": "431AZYLPPS6LJE01",
    "timestamp": "2023-02-22T09:12:23.533Z"
  },
  "generatorSetup": {
    "generatorEngine": "PDF_BOX",
    "templateEngine": "HANDLEBARS",
    "parametersData": "ewogICAgIklOU1RJVFVURV9OQU1FIjogIlNpw7Nmb2tpIGvDs3Jow...",
    "responseFormat": "PDF",
    "documentStorageMethod": "DATABASE"
  },
  "templates": [
    {
      "templateName": "main_template",
      "templateContent": "PCFET0NUWVBFIGh0bWw+CjxodG1sPgo8aGVhZD4KCjxzdHls...",
      "initial": true
    },
    {
      "templateName": "head_template",
      "templateContent": "PHRhYmxlIHN0eWxlPSJoZWlnaHQ6IDE2MXB4OyB3aWR0aDog...",
      "initial": false
    }
  ]
}
```

In the response, we receive the generated document and the filename in the HTTP header.

2.1.1.3. Document generation based on request body, with metadata response

```
POST /internal/dookug/document/generate/inline/metadata
```

There’s an option to receive metadata describing the document instead of the document itself. In this case, the request URI and HTTP header differ when submitting the request.

```
Request method: POST
Request URI:
http://localhost:8082/internal/dookug/document/generate/inline/metadata
Headers:        Accept=application/json
                Content-Type=application/json
```

2.1.1.4. Document generation based on multipart form

```
POST /internal/dookug/document/generate/inline/multipart
```

To generate, we need to submit the initial template and the data associated with the generation. The latter matches the data sent for document generation based on the request body.

Sample document generation multipart form request with key-value parameters:

```
Request method: POST
Request URI:
http://localhost:8082/internal/dookug/document/generate/inline/multipart
Headers:        Accept=application/octet-stream
                Content-Type=multipart/form-data

Content-Disposition: form-data; name="REQUEST"
Content-Type: application/json
{
    "context": {
      "requestId": "42ZZBR3LKJZ9IT0X",
      "timestamp": "2023-02-21T12:57:52.113Z"
    },
    "generatorSetup": {
      "generatorEngine": "PDF_BOX",
      "templateEngine": "HANDLEBARS",

"parameters":[{"key":"first","value":"első"},{"key":"second","value":"í123456789öüóőúű
áé-.,<>#&@{};*¤ß$"}],
      "responseFormat": "PDF",
      "documentStorageMethod": "DATABASE"
    }
}
--2b17c723-ec5b-4c99-8dcf-c6d8972c4564
Content-Disposition: form-data; name="TEMPLATE"
Content-Type: application/octet-stream

DookuG simple test with prameters first: [{{first}}], second: [{{second}}]
```

In the response, we receive the generated document and the filename in the HTTP header.

2.1.1.5. Document generation based on multipart form, with metadata response

```
POST /internal/dookug/document/generate/inline/multipart/metadata
```

There's also an option to receive metadata describing the document instead of the document itself. In this case, the request URI and HTTP header differ when submitting the request.

```
Request method: POST
Request URI:
http://localhost:8082/internal/dookug/document/generate/inline/multipart/metadata
Headers:        Accept=application/json
                Content-Type=multipart/form-data
```

2.1.1.6. Document generation based on stored template

```
POST /internal/dookug/document/generate/storedTemplate
```

To generate, we need to submit the name of the template from which the document is generated, along with the data associated with the generation. The latter matches the data sent for document generation based on the request body, supplemented with how the template is stored.

Sample document generation based on stored template request - JSON parameters in base64binary form (trimmed for readability):

```
Request method: POST
Request URI:    http://localhost:8082/internal/dookug/document/generate/storedTemplate
Headers:        Accept=application/json
                Content-Type=application/json
```

```
{
  "context": {
    "requestId": "431BIFEOUJ0ODU01",
    "timestamp": "2023-02-22T09:26:45.121Z"
  },
  "generatorSetup": {
    "templateStorageMethod": "DATABASE",
    "template": {
      "templateName": "DEV_TEMPLATE_HANDLEBARS",
      "templateLanguage": "HU",
      "validityDate": "2023-02-22T09:26:45.130074Z"
    },
    "generatorEngine": "PDF_BOX",
    "templateEngine": "HANDLEBARS",
```

```
    "parametersData": "ewogICJ0aXRsZSI6ICJwZWxkYSBjaW0iLAogICJjdXJyZW50WWV...",
    "responseFormat": "PDF",
    "documentStorageMethod": "DATABASE"
  }
}
```

In the response, we receive the generated document and the filename in the HTTP header.

2.1.1.7. Document generation based on stored template, with metadata response

```
POST /internal/dookug/document/generate/storedTemplate/metadata
```

There's an option to receive metadata describing the document instead of the document itself. In this case, the request URI and HTTP header differ when submitting the request.

The template key stored in the database consists of the templateName and templateLanguage values.

```
Request method: POST
Request URI:
http://localhost:8082/internal/dookug/document/generate/storedTemplate/metadata
Headers:        Accept=application/json
                Content-Type=application/json
```

Saving the document depends on the `documentStorageMethod` parameter. It can take two values: `NONE` and `DATABASE`. If `NONE` is specified, the document is not saved, and therefore cannot be queried later. In case of `DATABASE`, the generated document is saved in a database table, from where it can be retrieved later. Other data related to the document are also saved in the database:

- identifier of the initial template - if the template is not saved, this parameter is not filled
- filename of the generated file - generated from the unique identifier of the document, the name of the initial template, and the timestamp of the generation long value
- file format
- document status - DONE, FAILED, PENDING, SYNCING
- parameters related to the document
- document storage format - in case of the field DATABASE, this field is also filled with DATABASE value

---

During generation, the initial template is provided in any form of the response, the generated file is received, or metadata describing the document is received, as a DocumentMetadataResponse type object.

Sample DocumentMetadataResponse:

```
{
    "context": {
      "requestId": "42ZZBQ5K7W43FI6W",
      "timestamp": "2023-02-21T12:57:50.888Z"
    },
    "funcCode": "OK",
    "metadata": {
      "documentId": "42ZZBQ3ISCXWVO6V",
      "storageMethod": "DATABASE",
      "filename": "filename.pdf",
      "format": "PDF",
      "status" : "DONE"
    }
}
```

2.1.1.8. Query Document Metadata

```
POST /internal/dookug/document/storedTemplate/metadata/query
```

The purpose of querying document metadata is to retrieve document information that meets the specified filtering criteria.

The endpoint supports pagination, meaning data can be retrieved across multiple pages. In the request, you can specify which page of data and how many elements per page you want to retrieve. Accordingly, the response includes total count of elements and the number of pages they span. If not specified, the endpoint defaults to returning the first 15 elements.

The following filtering criteria can be used:

- templateId - identifier of the template used for document generation
- status - status of the document
- format - file format of the document
- storageMethod - storage method of the document
- storageId - unique identifier of the document storage
- filename - name of the document file

Sorting parameters can be:

- filename
- documentStorageMethod
- format
- status

For sorting, you can specify whether to sort in ascending or descending order for each parameter individually. In addition to the mentioned sorting options, there is a default sorting by document identifier.

Sample DocumentMetadataQueryRequest:

```
Request method: POST
Request URI:
http://localhost:8082/internal/dookug/document/storedTemplate/metadata/query
Headers:        Accept=application/json
                Content-Type=application/json; charset=UTF-8
```

```
{
    "context": {
      "requestId": "43183LDKQNC2R702",
      "timestamp": "2023-02-22T09:15:14.168Z"
    },
    "paginationParams": {
      "rows": 10,
      "page": 1
    },
    "queryParams": {
      "status": "DONE",
      "storageMethod": "DATABASE",
      "filename": "filename.pdf",
      "format": "PDF",
      "templateId" : "MAIN_TEMPLATE"
    }
}
```

If documents are found based on the submitted parameters, the response returns a list of up to 100 elements.

Sample DocumentMetadataQueryResponse:

```
{
    "context": {
      "requestId": "43183LDKQNC2R702",
      "timestamp": "2023-02-22T09:15:14.168Z"
    },
    "funcCode": "OK",
    "rowList": [
      {
        "documentId": "43183KXXW2KCI206",
        "storageMethod": "DATABASE",
        "filename": "filename.pdf",
        "format": "PDF",
        "status" : "DONE"
```

```
      }
    ]
}
```

2.1.1.9. Get Document

```
POST /internal/dookug/document/content/{documentId}
```

The purpose of this endpoint is to retrieve a previously generated and saved document based on the provided identifier.

Sample document retrieval request:

```
Request method: GET
Request URI:
http://localhost:8082/internal/dookug/document/content/43183KXXW2KCI206 ①
Headers:        Content-Type=application/octet-stream
```

① Identifier of the generated document

If no document is found for the submitted identifier, an ENTITY_NOT_FOUND error is returned.

In the response - for an existing document identifier - the generated document is returned, and the file name is included in the HTTP headers.

2.2. OpenAPI Documentation

Download OpenAPI Files:

- dookug-rest-service
  - yaml
  - json

3. DookuG Module Client

The client aims to support management of the DookuG module, which encompasses various server-side methods.

3.1. Technology

- Java 17
- JEE 10 (There is a client for supporting JEE 8)
- Eclipse Microprofile 4.1 and 5.0

3.2. Usage

Using the client requires only an entry in the pom.xml file, depending on the desired JEE version:

```
<dependency>
   <groupId>hu.icellmobilsoft.dookug</groupId>
   <artifactId>dookug-client-jee10</artifactId>
   <version>${version.dookug.client}</version>
</dependency>
```

or

```
<dependency>
   <groupId>hu.icellmobilsoft.dookug</groupId>
   <artifactId>dookug-client-jee8</artifactId>
   <version>${version.dookug.client}</version>
</dependency>
```

followed by an @Inject,

```
@Inject
private DookugClient dookugClient;
```

which defines client calls to all endpoints and allows the use of its offered methods. The API calls are made using the MicroProfile Rest Client, so configurations can be set using the familiar microprofile-config. (see below)

WARNING	It is the responsibility of the client application to mount resources under the `/home/icellmobilsoft/resources` mount point within the container that it uses, which are referenced in the templates!

3.3. Operation

The client currently has multiple methods to generate documents, retrieve metadata of the created documents, and fetch the generated document itself.

During document generation, some methods return the generated document:

```
public GeneratedDocumentDto postDocumentGenerateEntityBody(Collection<TemplateType>
templates, Collection<ParameterType> parameters) throws BaseException;

public GeneratedDocumentDto postDocumentGenerateEntityBody(Collection<TemplateType>
templates, ParametersDataType parametersData) throws BaseException;

public GeneratedDocumentDto postDocumentGenerateEntityBody(Collection<TemplateType>
```

```
templates) throws BaseException;

public GeneratedDocumentDto postDocumentGenerateMultipart(InputStream template,
Collection<ParameterType> parameters) throws BaseException;

public GeneratedDocumentDto postDocumentGenerateMultipart(InputStream template,
ParametersDataType parametersData) throws BaseException;

public GeneratedDocumentDto postDocumentGenerateMultipart(InputStream template) throws
BaseException;

public GeneratedDocumentDto postDatabaseStoredTemplateDocumentGenerate(String
templateName, OffsetDateTime templateValidity, Collection<ParameterType> parameters)
throws BaseException;

public GeneratedDocumentDto postDatabaseStoredTemplateDocumentGenerate(String
templateName, OffsetDateTime templateValidity, ParametersDataType parametersData)
throws BaseException;
```

It is important to note that `ParametersDataType` type parameters can be created manually, but a `ParametersDataBuilder` helper class is provided to configure the desired settings using a fluent API. This was required by the SAXON template engine, as additional parameters are necessary for generating XSLT templates, without which documents cannot be generated. The desired configuration can be easily extracted from the builder for the SAXON engine by calling the `getSaxonParameters(byte[], byte[], boolean)` method, where the FOP configuration content, the XML dataset, and the compression status of the XML can be specified, or through overloaded methods, just the content of the XML file.

```
GeneratedDocumentDto resp = client.postDatabaseStoredTemplateDocumentGenerate(
        TemplateLanguageType.HU, ①
        ParametersDataBuilder.getSaxonParameters(

FileUtil.readFileFromResource(DocumentServiceTestConstant.XSLT_TEMPLATE_PARAMS).getByt
es(StandardCharsets.UTF_8))); ②
```

① The language of the template

② The content of the XML dataset (uncompressed in this case)

These methods return a `GeneratedDocumentDTO` upon a successful call, which contains the generated filename, the generated object as a stream, and the HTTP status code.

Parameters must include the list of templates and the parameters if the templates contain variables that need to be replaced with values, such as in the case of the HANDLEBARS template engine. The parameters can be key-value pairs or a JSON object that essentially contains these key-value pairs.

The methods differ in the way the template for document generation is provided. In the `postDocumentGenerateEntityBody()` methods, the structure used as a template is sent in the request body. In the `postDocumentGenerateMultipart()` methods, the template is provided as a form

parameter, InputStream. In the `postDatabaseStoredTemplateDocumentGenerate()` methods, only the template name needs to be provided, as the endpoint handles it according to the specified storage method.

Additionally, methods are available for generating documents where only the metadata of the created document is returned instead of the document itself:

```
public DocumentMetadataResponse
postDocumentGenerateEntityBodyMetadata(Collection<TemplateType> templates,
Collection<ParameterType> parameters) throws BaseException;

public DocumentMetadataResponse
postDocumentGenerateEntityBodyMetadata(Collection<TemplateType> templates,
ParametersDataType parametersData) throws BaseException;

public DocumentMetadataResponse
postDocumentGenerateEntityBodyMetadata(Collection<TemplateType> templates) throws
BaseException;

public DocumentMetadataResponse postDocumentGenerateMultipartMetadata(InputStream
template, Collection<ParameterType> parameters) throws BaseException;

public DocumentMetadataResponse postDocumentGenerateMultipartMetadata(InputStream
template, ParametersDataType parametersData) throws BaseException;

public DocumentMetadataResponse postDocumentGenerateMultipartMetadata(InputStream
template) throws BaseException;

public DocumentMetadataResponse
postDatabaseStoredTemplateDocumentGenerateMetadata(String templateName, OffsetDateTime
templateValidity, Collection<ParameterType> parameters) throws BaseException;

public DocumentMetadataResponse
postDatabaseStoredTemplateDocumentGenerateMetadata(String templateName, OffsetDateTime
templateValidity, ParametersDataType parametersData) throws BaseException;

public DocumentMetadataResponse postStoredTemplateDocumentGenerateMetadata(String
templateName, OffsetDateTime templateValidity, TemplateStorageMethodType
templateStorageMethodType, Collection<ParameterType> parameters, ParametersDataType
parametersData) throws BaseException;
```

These methods return a `DocumentMetadataResponse` upon a successful call, which includes the unique identifier of the document, filename, storage method, format, and status.

Similarly, the client calls are distinguished in the same way by how the template is provided.

To view already generated files, there is an option to query the metadata of the documents:

```
public DocumentMetadataQueryResponse
```

```
postDocumentMetadataQuery(DocumentMetadataQueryParamsType queryParams) throws
BaseException;

public DocumentMetadataQueryResponse
postDocumentMetadataQuery(DocumentMetadataQueryRequest queryRequest) throws
BaseException;
```

The query returns a `DocumentMetadataQueryResponse` upon a successful call, which contains the metadata of documents matching the specified parameters, as well as pagination data: total number of items, total number of pages, page number, and the number of items returned in the query.

Filtering conditions can include the document name, format, storage method, template identifier from which the document was generated, the identifier of the document's storage, and the status of the document.

It is also possible to query a previously generated document:

```
public GeneratedDocumentDto getDocumentContent(String documentId) throws
BaseException;
```

The response returns the generated document as a stream, the filename, and the HTTP status code upon a successful call.

The unique identifier of the document must be provided as a parameter.

The client also offers additional configuration options that can influence the generation process:

```
setGeneratorEngineType()
```

```
setTemplateEngineType()
```

```
setResponseFormat()
```

```
setDocumentStorageMethodType()
```

```
setDigitalSigningType()
```

The `setGeneratorEngineType()` allows setting the engine used for output generation, which currently can be:

- `PDF_BOX` — uses [Apache PdfBox](https://pdfbox.apache.org/)

- `SAXON` — uses [Saxon HE](https://github.com/Saxonica/Saxon-HE/)
- `NONE`

The `setTemplateEngineType()` allows setting the template 'type', which currently can be:

- `HANDLEBARS` — uses [Handlebars](https://handlebarsjs.com/)
- `NONE`

The `setResponseFormat()` allows setting the response format:

- `PDF`
- `STRING`

The `setDocumentStorageMethodType()` allows setting the document storage method:

- `NONE`
- `DATABASE`

The `setDigitalSigningType(digitalSigningType)` allows controlling whether the generated PDF document should have a digital signature. The `digitalSigning` expects three additional parameters:

- `signatureName` - the name of the signature (optional)
- `signatureReason` - the reason for signing (optional)
- `keyAlias` - the identifier of the key in the keystore, used to identify the signing key (optional, but recommended as it defaults to searching for the `test` key)

If no settings are specified, the default values are `PDF_BOX` + `HANDLEBARS` + `PDF` + `NONE` without a digital signature.

The `postDocumentGenerateEntityBody()` methods call the following REST endpoint in the module:

```
POST /internal/dookug/document/generate/inline
```

The client sends:

- `ContextType`
- the received `TemplateType` list
- the received `ParameterType` list
- the `GeneratorSetup` object, which can be controlled by the client's `set` methods.

If the request is correct, the generated object is returned.

NOTE	For Multipart and StoredTemplate clients, the process is the same, differing only in the REST API calls.

The `postDocumentMetadataQuery()` methods call the following REST endpoint in the module:

```
POST /internal/dookug/document/storedTemplate/metadata/query
```

The client sends:

- `ContextType`
- the received filtering conditions
- pagination parameters
- sorting settings

If the request is correct, the metadata of the documents matching the specified parameters is returned.

Example of client usage:

```
 @Inject
 private DookugClient dookugClient;
 ...
 //template object
 TemplateType template = new
TemplateType().withTemplateName("main").withTemplateContent("DookuG client simple test
with parameters first: [{{first}}], second:
[{{second}}]".getBytes(StandardCharsets.UTF_8));

 //parameters
 ParameterType parameter1 = new ParameterType().withKey("first").withValue("első");
 ParameterType parameter2 = new
ParameterType().withKey("second").withValue("í189öüóőúűáé-.,<>#&@{};*¤ß$");
 ...
 client.setResponseFormatType(ResponseFormatType.STRING);
 client.setGeneratorEngineType(GeneratorEngineType.NONE);
 GeneratedDocumentDto response =
dookugClient.postDocumentGenerateEntityBody(List.of(template),
List.of(parameter1,parameter2));
```

Or similarly, generating a document but with a PDF document format, multipart input, and returning the metadata:

```
 @Inject
 private DookugClient dookugMultipartClient;
 ...
 //template as byte array
 byte[] template = "DookuG client simple test with parameters first: [{{first}}],
second: [{{second}}]".getBytes(StandardCharsets.UTF_8);

 //parameters
 ParameterType parameter1 = new ParameterType().withKey("first").withValue("első");
 ParameterType parameter2 = new
```

```
ParameterType().withKey("second").withValue("í189öüóőúűáé-.,<>#&@{};*¤ß$");
 ...
 client.setResponseFormatType(ResponseFormatType.PDF); //ez a default
 client.setGeneratorEngineType(GeneratorEngineType.PDF_BOX); //ez a default
 client.setTemplateEngineType(GeneratorEngineType.HANDLEBARS); //ez a default
 DocumentMetadataResponse response =
dookugMultipartClient.postDocumentGenerateMultipartMetadata(new
ByteArrayInputStream(template), List.of(parameter1, parameter2));
```

3.3.1. Using Saxon-HE Engine in the Client

To use the Saxon-HE engine in the client, an XSLT template is required for generating a PDF file from an XML. In this case, only PDF can be the output format. You also need to provide a `fop-config.xml` file in the request, which helps regulate the use of fonts, for example.

```
<?xml version="1.0" encoding="UTF-8"?>
<fop version="1.0">
    <renderers>
        <renderer mime="application/pdf">
            <fonts>
                <!-- TTF fonts -->
                <font kerning="yes" embed-
url="/home/icellmobilsoft/fonts/Roboto/Roboto-Regular.ttf">
                    <font-triplet name="Roboto" style="normal" weight="normal" />
                </font>
                <font kerning="yes" embed-
url="/home/icellmobilsoft/fonts/Roboto/Roboto-Bold.ttf">
                    <font-triplet name="Roboto" style="normal" weight="bold" />
                </font>
            </fonts>
        </renderer>
    </renderers>
</fop>
```

Handlebars can also be used with SAXON, where you can substitute the usual `{{VARIABLE}}` variables with desired text parts, as well as create nested templates (this is mainly used for that).

The major change from other engines is that you need to specify the `XML` as a data source in the `generatorSetup` in the SAXON case, in addition to other fields:

- XML: as data source
- XSLT: as template
- fopConfig: transformer configuration

3.4. Error Handling

The client can only return a BaseException, but if a RestClientResponseException is received during

the API call, it will return the wrapped BaseException contained within it!

4. Configuration

4.1. Backend

The module's configuration is managed via MicroProfile Config, which allows specifying necessary values in multiple ways.

MicroProfile Config can retrieve a given key from all available sources and uses the highest priority value.

The base configuration is provided via `project-defaults.yml`, which can be extended and may vary per environment. It's not necessary to specify every value; only those that differ from the default settings.

Possible modes in order of priority:

- System variables
- Environment variables

4.1.1. DOOKUG-DOCUMENT service

The meanings of the emojis used in the table:

 - meaning that it is a startup parameter.

 - meaning that this parameter can be overridden during runtime

dookug keys

Key	Source	Description	Default value	Since	Features
dookug.client.document	hu.icellmobilsoft.dookug.api.dto.constants.ConfigKeys.Client	**Client configuration** In the application using the DookuG client, you need to set: - the server REST URL (e.g., dookug.client.document/mp-rest/url: http://localhost:8082) - optionally the REST connectTimeout (e.g., dookug.client.document/mp-rest/connectTimeout: 5000) - optionally the REST readTimeout (e.g., dookug.client.document/mp-rest/readTimeout: 60000)	-	0.1.0	
dookug.service.cache.keystore.enabled	hu.icellmobilsoft.dookug.api.dto.constants.ConfigKeys.Cache.Keystore	Does the module use keystore caching for document signing?	true	1.1.0	
dookug.service.cache.keystore.enablestatistic	hu.icellmobilsoft.dookug.api.dto.constants.ConfigKeys.Cache.Keystore	Metrics related to the Template cache should be generated. By default, they are not generated.	false	1.1.0	
dookug.service.cache.keystore.ttl	hu.icellmobilsoft.dookug.api.dto.constants.ConfigKeys.Cache.Keystore	How long until the system invalidates the cache content. By default, 1440 minutes.	1440	1.1.0	

Key	Source	Description	Default value	Since	Features
dookug.service.cache.template.enabled	hu.icellmobilsoft.dookug.api.dto.constants.ConfigKeys.Cache.Template	Does the module use template caching?	false	0.6.0	
dookug.service.cache.template.enablestatistic	hu.icellmobilsoft.dookug.api.dto.constants.ConfigKeys.Cache.Template	Metrics related to the Template cache should be generated. By default, they are not generated.	false	0.5.0	
dookug.service.cache.template.ttl	hu.icellmobilsoft.dookug.api.dto.constants.ConfigKeys.Cache.Template	How long until the system invalidates the cache content. By default, 60 minutes.	60	0.5.0	
dookug.service.engine.handlebars.escapingstrategy	hu.icellmobilsoft.dookug.api.dto.constants.ConfigKeys.Handlebars.EscapingStrategy	**Handlebars configuration** Configuration storing the `com.github.jknack.handlebars.EscapingStrategy` key.	In the Handlebars engine, HTML will be the default strategy if no value is configured with this key.	0.1.0	
dookug.service.engine.handlebars.helper.javascript.directory	hu.icellmobilsoft.dookug.api.dto.constants.ConfigKeys.Handlebars.Helper	**Handlebars configuration** Stores the path to the directory containing JavaScript files with Handlebars handlers in the Docker container.	/home/icellmobilsoft/handlebars/helper/js	0.1.0	

Key	Source	Description	Default value	Since	Features
dookug.service.engine.saxon.fopconfig	hu.icellmobilsoft.dookug.api.dto.constants.ConfigKeys.Saxon	**Saxon configuration** Path to the Fop config file within the container	/home/icellmobilsoft/fop-config/fop-config.xml	0.1.0	
dookug.service.engine.saxon.xslt.language.default	hu.icellmobilsoft.dookug.api.dto.constants.ConfigKeys.Saxon	**Saxon configuration** Default language	HU	0.1.0	
dookug.service.engine.saxon.xslt.language.variable	hu.icellmobilsoft.dookug.api.dto.constants.ConfigKeys.Saxon	**Saxon configuration** The name of the language variable in the XSLT template	lang	0.1.0	
dookug.service.interface.parametersdata.gzipped	hu.icellmobilsoft.dookug.api.dto.constants.ConfigKeys.Interface	**Interface configuration** Logical config key. The module expects the incoming "parametersData" field in the request to be compressed using gzip.	false	0.1.0	

4.1.2. Pdf Signature configuration

Unresolved directive in config/backend.adoc - include::../generated/dookug-pdf-signature-config.adoc[leveloffset=+1]

4.2. Client

The operation of the client can be influenced using MicroProfile Config.

```
dookug:
    client:
        document/mp-rest/url: http://localhost:8082
        document/mp-rest/connectTimeout: 5000 #millisec
        document/mp-rest/readTimeout: 60000 #millisec
```

It is advisable to specify the following parameters for the client operation:

Parameter Key	Mandatory	Description
dookug.client.document/mp-rest/url	yes	Base URL of the DookuG module
dookug.client.document/mp-rest/connectTimeout		Connection Timeout (defaults to 5 seconds if unspecified)
dookug.client.document/mp-rest/readTimeout		Read Timeout (defaults to 1 minute if unspecified)
dookug.service.cache.template.ttl	no	Template cache expiration time in minutes, default is 60 minutes
dookug.service.cache.template.enablestatistic	no	Whether to generate metrics for caching, defaults to false

Additional parameters can also be specified, as outlined in the [Microprofile RestClient documentation](https://download.eclipse.org/microprofile/microprofile-rest-client-2.0/microprofile-rest-client-spec-2.0.html#mpconfig:~:text=Client%20CDI%20Support-,Support%20for%20MicroProfile%20Config,-Configuration%20Keys).

5. Installation, Deployment

The Dookug module must be accessible in the environments of the project(s) that intend to utilize the service. Each instance of the service, along with its infrastructure requirements, should be installed and configured for every environment (development/test/production).

5.1. DOOKUG-DOCUMENT Service

5.1.1. Service configuration

Interface

```
DOOKUG_SERVICE_INTERFACE_PARAMETERSDATA_GZIPPED=true
```

PDF signing

```
DOOKUG_SERVICE_ENGINE_PDF_DIGITALSIGN_SIGNATURE_DEFAULT_NAME="Original document"
DOOKUG_SERVICE_ENGINE_PDF_DIGITALSIGN_SIGNATURE_DEFAULT_REASON="Certified by Dookug"
DOOKUG_SERVICE_ENGINE_PDF_DIGITALSIGN_SIGNATURE_DEFAULT_KEYSTORE=/home/icellmobilsoft/
keys/keystore.p12
DOOKUG_SERVICE_ENGINE_PDF_DIGITALSIGN_SIGNATURE_DEFAULT_KEYSTOREPASS=123456
DOOKUG_SERVICE_ENGINE_PDF_DIGITALSIGN_SIGNATURE_DEFAULT_KEYSTORETYPE=PKCS12
```

Saxon

```
DOOKUG_SERVICE_ENGINE_SAXON_FOPCONFIG=/home/icellmobilsoft/fop-config/fop-config.xml
```

```
DOOKUG_SERVICE_ENGINE_SAXON_XSLT_LANGUAGE_VARIABLE=lang
DOOKUG_SERVICE_ENGINE_SAXON_XSLT_LANGUAGE_DEFAULT=HU
```

Handlebars

```
DOOKUG_SERVICE_ENGINE_HANDLEBARS_HELPER_JAVASCRIPT_DIRECTORY=/home/icellmobilsoft/hand
lebars/helper/js
DOOKUG_SERVICE_ENGINE_HANDLEBARS_ESCAPINGSTRATEGY=HTML_ENTITY
```

5.1.2. Recommended K8S Configuration

Parameter	Value	Description
JAVA_OPTS	-Xms2000m -Xmx2000m -XX:MetaspaceSize=96M -XX:MaxMetaspaceSize=256m -XX:+PrintCommandLineFlags -XX:+UseG1GC	Maximize module heap requirement to 2G memory with UseG1GC algorithm.

Requests/limits settings

```
spec:
  containers:
    resources:
      limits:
        cpu: "3"
        memory: 3G
      requests:
        cpu: "3"
        memory: 3G
```

5.1.3. Observability

5.1.3.1. Health - startup/liveness/readiness

The service supports the use of K8S probes.

Started: http://localhost:9990/health/started

Live: http://localhost:9990/health/live

Ready: http://localhost:9990/health/ready

6. Additional informations

6.1. Template Cache

The application caches the TEMPLATE data stored in the database using GUAVA. The caches have a defined lifespan, and the time resets with each new request.

Configuration Parameters:

You can specify the duration for which templates are kept in the cache using the DOOKUG_SERVICE_CACHE_TEMPLATE_TTL parameter, in minutes. The default is 60 minutes.

You can specify whether metrics generation is needed using the DOOKUG_SERVICE_CACHE_TEMPLATE_ENABLESTATISTIC parameter. The default is false, meaning no metrics are generated.

Metrics similar to the following are generated:

```
# TYPE application_cache_hit_count gauge
application_cache_hit_count{name="template"} 0.0
# TYPE application_cache_miss_count gauge
application_cache_miss_count{name="template"} 1.0
# TYPE application_cache_size gauge
application_cache_size{name="template"} 1.0
```

6.2. Helpers - Helper Functions for Use in Template Files

6.2.1. Built-in Helpers

Handlebars provides built-in helpers, whose documentation can be found here.

6.2.2. Custom Helpers

It is possible to use general helpers provided by the module. In the examples, the ... parts render if the expression evaluates to true.

6.2.2.1. Usage

When using helpers, specify the parameters for the given helper after the helper keyword. Helpers can be used during inline generation and database-stored template-based generation. The parameters can be hardcoded values or come in JSON format during the document generation call.

Helpers can be combined with each other.

Token	Description	Usage	Available from Version
equals	Compares the values of two elements; renders if the evaluation is true.	`{{#if (equals yourField 'white')}} ... {{/if}}`	0.1.0
before	Renders if the provided parameter is an earlier date than the given value. The latter can also be a parameter.	`{{#if (before yourDate '2023-08-13T05:40:55Z')}} ... {{/if}} {{#if (before yourDate checkDate)}} ... {{/if}}`	0.1.0
between	Renders if the provided parameter falls between the two given date values. The latter can also be parameters.	`{{#if (between yourDate '2023-08-13T05:40:55Z' '2023-08-15T05:40:55Z')}} ... {{/if}} {{#if (between yourDate startDate endDate)}} ... {{/if}}`	0.1.0
dateMinusMinutes	Subtracts a given number of minutes from the specified date.	`{{dateMinusMinutes '2023-08-13T05:40:55Z' 60}}  // evaluates to '2023-08-13T04:40:55Z' {{dateMinusMinutes yourDate 60}} // The following two examples evaluate to the same date {{dateMinusMinutes yourDate minutesToSubtract}}`	0.1.0
datePlusMinutes	The counterpart to dateMinusMinutes, it adds minutes to the given date.	`{{datePlusMinutes '2023-08-13T05:40:55Z' 60}} - evaluates to '2023-08-13T06:40:55Z' {{datePlusMinutes yourDate 60}} {{datePlusMinutes yourDate minutesToAdd}}`	0.1.0
declare	Creates a new variable and immediately renders it. The created variables are global, regardless of where they were created in the templates.	`{{declare 'myColor' 'white'}} // immediately renders the word 'white' ... {{#if myColor}} {{myColor}} {{/if}} // renders 'white'`	0.1.0

declareVoid	Works similarly to the declare helper, but does not immediately render the given value. declare can be replaced by declareVoid followed by immediate invocation.	{{declareVoid 'myColor' 'white'}}
		...
		{{#if myColor}} {{myColor}} {{/if}}
		// renders 'white'
		Replacing declare with declareVoid:
		{{declareVoid 'myColor' 'white'}}
		{{myColor}} // renders 'white' immediately after creation
			0.1.0

formatDate	Transforms the given parameter, accepted in Java 8 date format, according to the specified pattern. The pattern only accepts date formats; otherwise, it throws an error.	yourDate = '2024-01-09';
		
		{{formatDate '2023-08-13' 'yyyy.MM.dd'}} // '2023.08.13'
		{{formatDate '2023-08-13' 'MM.dd.yyyy'}} // '08.13.2023'
		{{formatDate yourDate 'yyyy/MM/dd'}} // '2024/01/09'
			0.5.0

formatDateTime	Transforms the given parameter, accepted in Java 8 date and time format, according to the specified pattern. The helper handles full ISO date format input and can generate appropriate output using the pattern. The helper can also handle time zones; if a Java-accepted Zone ID is provided after the output pattern, it converts the input time to the time zone's output.	yourDate = '2024-01-09T15:30:04Z'
		yourDateTimeFormat = 'MM.dd-HH:mm'
		yourDateFormat = 'yyyy/MM.dd'
		yourTimeFormat = 'hh:mm:ss a'
		
		{{formatDateTime '2023-08-13T05:40:55Z' 'yyyy-MM-dd HH:mm:ss'}} // '2023-08-13 06:40:55'
		{{formatDateTime yourDate 'yyyy-MM-dd HH:mm:ss'}}
		// '2024-01-09 15:30:04'
		{{formatDateTime yourDate yourDateTimeFormat}}
		// '01.09-15:30'
		{{formatDateTime yourDate yourDateFormat}}
		// '2024/01.09'
		{{formatDateTime yourDate yourTimeFormat}}
		// '03:30:04 PM'
		{{formatDateTime '2023-08-13T05:40:55Z' 'yyyy-MM-dd HH:mm:ss' 'CET'}} // '2023-08-13 08:40:55'
			0.5.0

formatTime	Transforms the given parameter, accepted in Java 8 time format, according to the specified pattern. The pattern only accepts time formats; otherwise, it throws an error.	`yourTime = '15:30:55Z';`
		
		`{{formatTime '15:30:55Z' 'HH:mm:ss'}}  // '15:30:55'`
		`{{formatTime yourTime 'h:mm A'}}`
		`// '3:30 PM'`
			0.5.0

formatNumber	Helper used for formatting numbers, following Java number formatting conventions.	`number = 1234.567;`
		`percentage = 0.4567`
		
		`{{formatNumber number '#'}}`
		`// "1235", rounded integer`
		`{{formatNumber number '0.00'}}`
		`// "1234.57", number rounded to 2 decimals`
		`{{formatNumber number '000000.00'}}`
		`// "001234.57", number padded with leading zeros`
		`{{formatNumber number '#,###.##'}}`
		`// "1,234.57", comma-separated, grouped by thousands`
		`{{formatNumber number '$#,##0.00'}}`
		`// "$1,234.57", currency expression`
		`{{formatNumber percentage '0.00%'}}`
		`// "45.67%", percentage expression of a value between 0 and 1`
		`{{formatNumber number '0.###E0'}}`
		`// "1.235E8", number in scientific notation`
			0.1.0

and	Logical AND operator for N values. Renders if the logical AND operator evaluates to true.	{{#if (and falseValue trueValue notExistingValue)}} YES {{else}} NO {{/if}} // renders "NO"
		{{#if (and trueValue trueValue trueValue)}} YES {{/if}}
		// renders "YES" since all three values are true
		myValue = 'black'
		{{#if (and (equals myValue 'black')}} YES {{else}} NO {{/if}}
		// renders "YES", both the variable and the string evaluation are true
		{{#if (and (equals myValue 'white')}} YES {{else}} NO {{/if}}
		// renders "NO", both the variable and the string evaluation are false
		
		Combined usage:
		{{#if (and (equals 'black' 'white') (equals 'white' 'white'))}} YES {{else}} NO {{/if}} // renders "NO", the first is false, the second is true
			0.1.0

or	Logical OR operator for N values. Renders if the logical OR operator evaluates to true.	{{#if (or falseValue trueValue notExistingValue)}} YES {{else}} NO {{/if}} // renders "YES"
		{{#if (or falseValue falseValue falseValue)}} YES {{/if}}
		// renders nothing
		myValue = 'white'
		{{#if (or myValue1 myValue2)}} YES {{else}} NO {{/if}}
		// renders "YES" if myValue1 OR myValue2 is defined (not null)
		
		Combined usage:
		{{#if (or (equals 'black' 'white') (equals 'white' 'white'))}} YES {{else}} NO {{/if}} // renders "YES", false OR true evaluates to true
			0.1.0

not	Logical NOT operator. Negates the given parameter; renders if the evaluation is true, otherwise does not render.	`{{#if (not falseValue)}} YES {{else}} NO {{/if}} // YES {{#if (not existingValue)}} YES {{else}} NO {{/if}} // NO {{#if (not (equals 'black' 'white'))}} YES {{else}} NO {{/if}} // YES`	0.1.0
in	Checks if the first parameter matches any of the subsequent elements .	`myValue = 'white' {{#if (in myValue 'black' 'gray')}} YES {{else}} NO {{/if}} // NO {{#if (in myValue 'black' 'white' 'gray')}} YES {{else}} NO {{/if}} // YES`	0.1.0
math	Helper for basic mathematical operations. The first parameter is the operator, and the other two parameters are the operands. The list of usable operators: “+”, “-”, “*”, “/”, “%” If an invalid operator is used, the evaluation result is: “-1”. Otherwise, the result of the mathematical operation on the operands corresponding to the operator.	`num1 = 5 num2 = 8 num3 = 100 num4 = 20  {{math '+' num1 num2}} // 13 {{math '-' num3 53}} // 47 {{math '*' num2 num4}} // 160 {{math '/' num3 num4}} // 5 {{math '%' num4 num2}} // 40 {{math 'A' num1 num2}} // -1`	0.1.0

6.2.2.2. Additional Helper Functions

The following 3rd party helper functions can be used in the project:

- StringHelpers

6.3. Using Custom Fonts

This documentation is intended for the Apache PDFBox engine and focuses on using custom fonts.

NOTE	A limitation of the engine is that it can only embed TTF fonts. Additionally, if you want to use weight, styles, and variants settings, it is worth noting that PDFBox cannot always emulate these properly unless you load different variations of the font.

In the docker-compose file, we add the `fonts` directory to our image as `/home/icellmobilsoft/fonts`,

so we can access the fonts in this directory locally from the templates.

We have two options for using custom fonts.

1. Load the font directly in the CSS
2. Load the font programmatically, then reference it in the CSS using `font-family`

In the first option, you can specify the font's location in the CSS within the `@font-face` at-rule using the `src: url()` descriptor.

In this case, internet access will be required during generation, which may not always be possible as the company's policy may prohibit the module from accessing external URLs.

This can be mitigated either by making the fonts accessible through an internal URL or by preloading some frequently used fonts in the module and referencing their local availability using a `file://` URL. If the module will contain preloaded fonts, it is advisable to mention this in the documentation or provide the information through an endpoint.

TEMPLATE:

```
<style>
    @font-face {
        font-family: 'Cairo';
        font-style: normal;
        src: url(file:/home/icellmobilsoft/fonts/Cairo/Cairo-Regular.ttf); ①
    }

    @font-face {
        font-family: 'IndieFlower';
        font-style: normal;
        src: url(file:fonts/IndieFlower/IndieFlower-Regular.ttf); ②
    }
</style>
```

① Loading Cairo-Regular.ttf from the `/home/icellmobilsoft/fonts` directory.

② Here we also load the font from the above directory. The server considers `/home/icellmobilsoft` as the root, so the relative path works.

TIP

Using Google Fonts

```
<style>
    @import
url('https://fonts.googleapis.com/css?family=Quicksand&amp;display=swap');
①

    .quicksand { ②
      font-family: 'Quicksand', Arial;
      font-weight: 700;
      font-style: normal;
```

```
        font-size: 38px;
        line-height: 1.15;
        letter-spacing: -.02em;
        color: rgba(0, 0, 0, 0.8);
        -webkit-font-smoothing: antialiased;
    }
</style>
...
<p class="quicksand">Google Quicksand Font úőüöóéáyí</p> ③
```

① Importing the Quicksand font from Google Fonts

② Setting the CSS class to use the font

③ Displaying text with the specified font

If we want to load fonts programmatically from the file system, we can use the `builder.useFonts` method during rendering, which we can reference in the CSS.

JAVA:

```
builder.useFont(new File("fonts/NotoSansThai/NotoSansThai-Regular.ttf"),
"notosansthai-regular"); ①
```

① Loading NotoSansThai-Regular.ttf

TEMPLATE:

```
<style>
    @font-face {
        font-family: 'notosansthai-regular'; ①
        font-style: normal;
        -fs-font-subset: complete-font; ②
    }
</style>
```

① Using the previously loaded font

② This is just an example; this setting is not really necessary as it embeds the entire font, whereas by default only the subset is embedded, which is the correct operation.

The downside is that all fonts are loaded during rendering, even those not used in the document template.

NOTE The PDFBox Fonts Wiki is available here

6.4. Digitally Signing PDFs

This documentation is intended for the Apache PDFBox engine and describes the process of digitally signing PDFs using a self-signed certificate, which is added to the image as

`/home/icellmobilsoft/keys/keystore.p12` using the docker-compose file.

The keystore is located in the `/etc/docker-compose/keys` folder and was created on November 7, 2023, with the following command:

```
keytool -genkeypair -storepass 123456 -storetype pkcs12 -alias test -validity 10958 -v
-keyalg RSA -keystore keystore.p12
```

WARNING	This means it will expire on November 7, 2053.

6.4.1. Configuration

The signature can be requested in the API calls within the GeneratorSetup request. Here you can specify the `signatureName`, `signatureReason`, and the private key alias (`keyAlias`) in case the keystore file contains multiple private keys.

```
...
<ns2:generatorSetup>
    <ns2:generatorEngine>PDF_BOX</ns2:generatorEngine>
    <ns2:templateEngine>NONE</ns2:templateEngine>
    <ns2:responseFormat>PDF</ns2:responseFormat>
    <ns2:digitalSignatureProfile>sampleProfile</ns2:digitalSignatureProfile> ①
        ...
```

① The name of the signature profile only in case you need PDF signing

The signature can be customized with the following configuration keys:

```
dookug:
  service:
    engine:
      pdf:
        digitalsign:
          sampleProfile: ①
            name: Example Ltd. ②
            reason: Certified ②
            keystore: /home/icellmobilsoft/keys/keystore.p12 ③
            keystorePass: 123456 ④
            keystoreType: PKCS12 ④
            keyAlias: key_test ⑤
```

① the name of the signatureProfile

② `name` and `reason` default values are used if the request does not contain these values.

③ `keystore` is mandatory and specifies the location of the signing keys.

④ `keystorePass` and `keystoreType` specify the password and type of the keystore file, respectively. These are also mandatory.

⑤ the identifier of the private key within the keystore

6.4.2. Visible signature (using the EU DSS ESIGN library)

In case you want to add visible (clickable) signature to the PDF document you need to add some extra configuration.

```
dookug:
  service:
    engine:
      pdf:
        digitalsign:
          sampleProfile:
            keystore: /home/icellmobilsoft/keys/keystore.p12 ③
            keystorePass: 123456
            keystoreType: PKCS12
            keyAlias: key_test
            dss: ①
              imageFile: /home/icellmobilsoft/pdfsign/sample/signature.png ②
              showOnPage: 1 ③
              position:
                left: 18 ④
                top: 2 ⑤
              width: 2 ⑥
              useTimestamp: false ⑦
```

① the key for DSS library

② the image you want to add to the document

③ the image will added to this page: `-1`: last page, `1`:first page, `0`:wont add, `n`:n-th page

④ X position of the image in cm (from top left corner)

⑤ Y position of the image in cm (from top left corner)

⑥ the width of the image in cm

⑦ you can add the signature timestamp to the image

The available configuration keys can be found here.

6.5. Keystore Cache for digital signature

The application caches the keystore data configured for signature profiles using GUAVA. The caches have a defined lifespan, and the time resets with each new request.

Configuration Parameters:

You can specify the duration for which templates are kept in the cache using the DOOKUG_SERVICE_CACHE_KEYSTORE_TTL parameter, in minutes. The default is 1440 minutes (1 day).

You can specify whether metrics generation is needed using the DOOKUG_SERVICE_CACHE_KEYSTORE_ENABLESTATISTIC parameter. The default is false, meaning no metrics are generated.

Metrics similar to the following are generated:

```
# TYPE application_cache_hit_count gauge
application_cache_hit_count{name="keystore"} 0.0
# TYPE application_cache_miss_count gauge
application_cache_miss_count{name="keystore"} 1.0
# TYPE application_cache_size gauge
application_cache_size{name="keystore"} 1.0
```

7. Release notes

7.1. v0.4.0

The project has been released on GitHub opensource.

- maven pom change:
  - `<groupId>hu.icellmobilsoft.dookug.common</groupId>` → `<groupId>hu.icellmobilsoft.dookug</groupId>`
  - `<groupId>hu.icellmobilsoft.dookug.document</groupId>` → `<groupId>hu.icellmobilsoft.dookug</groupId>`
  - `<groupId>hu.icellmobilsoft.dookug.client</groupId>` → `<groupId>hu.icellmobilsoft.dookug</groupId>`

7.2. v0.5.0

7.2.1. Changes / updates

- Caching of templates in memory https://github.com/i-Cell-Mobilsoft-Open-Source/DookuG-backend/issues/4
- Observability: DB health check added
- TemplateLanguage type has been changed from enum to string (`EN` → `EN` or `en_GB`).
- The jandex.idx file has been removed from dookug-client-jee10. It caused Unsatisfied Dependency Exception during the deployment.

7.2.2. Migration

Replace `TemplateLanguageType.*` occurrences with the string value of the language.

7.3. v0.6.0

7.3.1. Changes / updates

- Documentation fixes (formatting, missing 0.5.0 release notes include)
- Add com.github.jknack.handlebars.helper.StringHelpers build in helper to helpers
- Rename formatDate to formatDateTime
- Create formatDate which handles date format
- Create formatTime which handles time format
- Add IT test for owned helpers
- GET `/system/evict` REST endpoint added
- Template caching has been fixed
- Template caching can be configured with the environment variable named `DOOKUG_SERVICE_CACHE_TEMPLATE_ENABLED` (`true` by default)

7.3.2. Migration

Changes are backwards compatible doesnt need any migration.

7.4. v1.0.0

7.4.1. Changes / updates

- Jakarta EE10 upgrade
- Test fix
- gitHub Workflows - java 17 upgrade
- Bugfix: The `not` helper function has been fixed, it evaluated the `false` parameter incorrectly.
- Handlebars version bump: 4.3.1 → 4.4.0
- Handlebars caching: Handlebars template engine now uses its in-built caching mechanism for compiled templates
- coff:ee upgrade - migration doc: 2.5.0 → 2.6.0
- coff:ee upgrade - migration doc: 2.6.0 → 2.7.0

NOTE	The client is not affected by the coffee upgrade, it remains 2.6.0 coffee . The reason for this is that you should upgrade to 2.7.0 on the project that uses coffee.

IMPORTANT	Because of the coffee upgrade, the use of interfaces had to be removed from the clients, because it already uses the coffee 2.7.0 BaseException, and would cause a break in the client.

- For `dookug-model`, `org.hibernate` groupId has been replaced by `org.hibernate.orm`.

- Roaster upgrade migration: 2.1.0 → 2.2.0
- In tests, the coffee.model.base.java.time.timezone.id property in the BaseIT class is set to UTC.
- New documentation structure

7.4.2. Migration

- Replace the dookug-client dependency in your project depending on whether you are using JEE 8 or JEE 10.

```
<dependency>
    <groupId>hu.icellmobilsoft.dookug.client</groupId>
    <artifact>dookug-client-jee10</artifact> ①
    <version>1.0.0</version>
</dependency>

<dependency>
    <groupId>hu.icellmobilsoft.dookug.client</groupId>
    <artifact>dookug-client-jee8</artifact> ②
    <version>1.0.0</version>
</dependency>
```

① in case you have EE10 application

② in case you have EE8 application

7.5. v1.1.0

7.5.1. Changes / updates

- Documentation has been translated to english.
- Using the EU DSS Esig library to sign PDF documents (optional)
- PDF signing: `signature` key removed from configuration.
- GitHub workflows for docker build and release build
- The coffee-module-etcd dependency has been removed.
- The signatureName and signatureReason has been removed from the request - These are set up in the application configuration under profile name keys
- Possible extSessionId dupication (in client calls) fixed

7.5.2. Migration

Changes are backwards compatible doesnt need any migration.