KU LEUVEN
DEPARTMENT OF COMPUTER SCIENCE

Document Processing Service (DocProc)

Part 2

Initial DocProc architecture

Academic Year 2020 – 2021

H09B5B: Software Architectuur
H07Z9B: Software Architecture

Contents

A. Client-server view (UML Component Diagram)

Figures

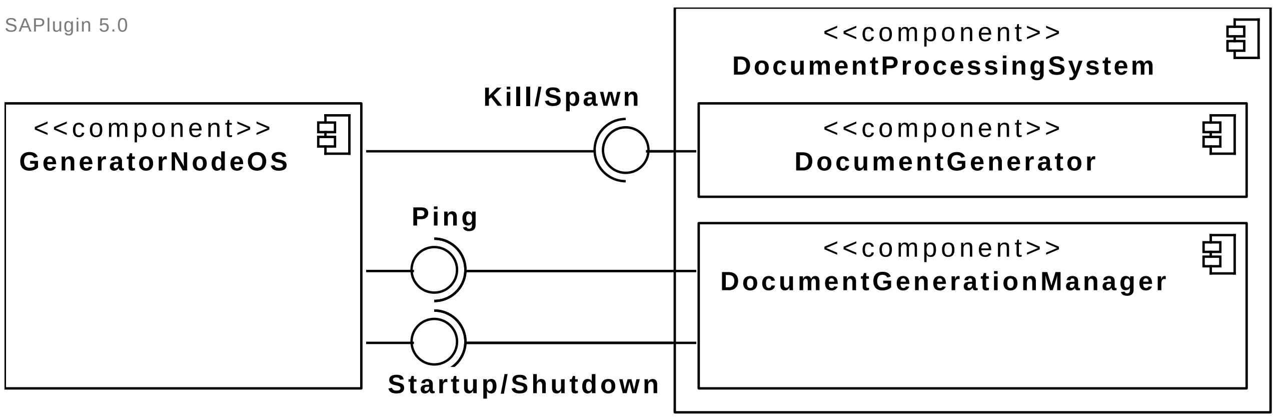
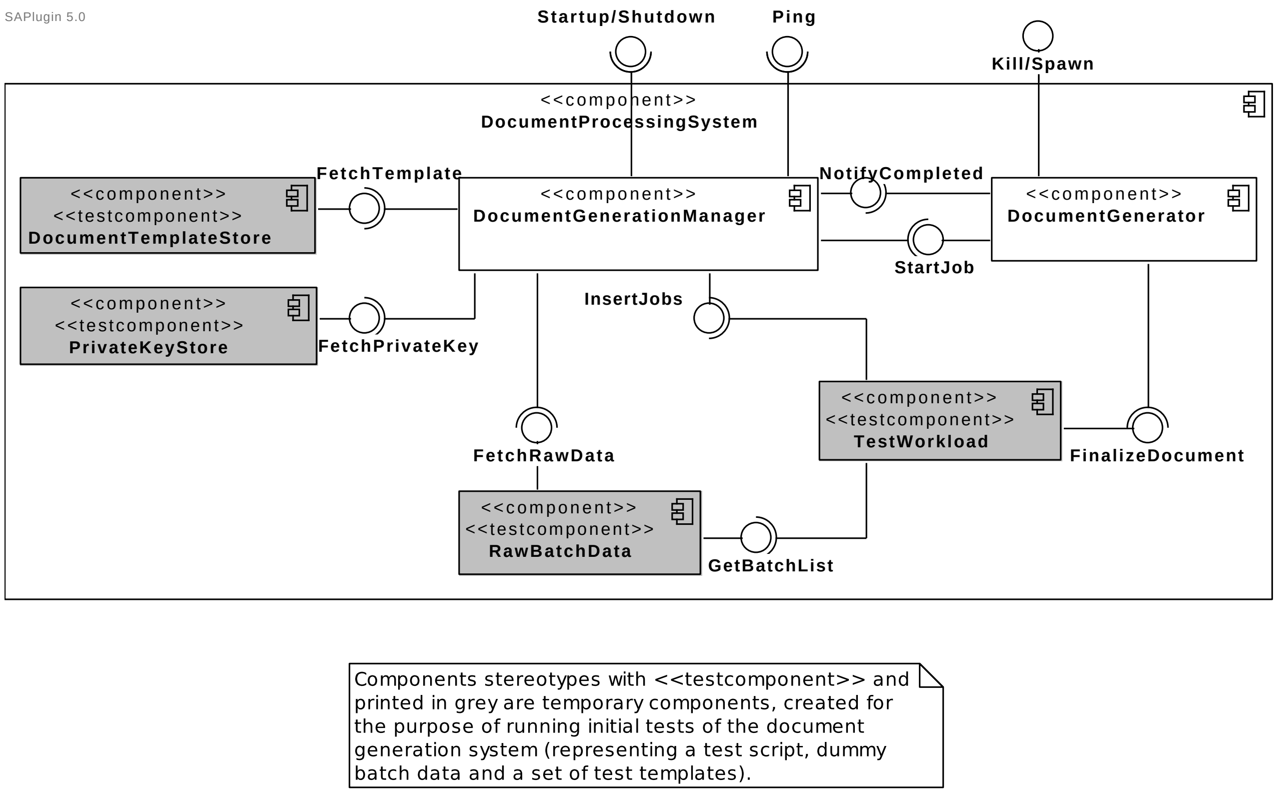


Figure A.1: Context diagram for the client-server view.



Components stereotypes with <<testcomponent>> and printed in grey are temporary components, created for the purpose of running initial tests of the document generation system (representing a test script, dummy batch data and a set of test templates).

Figure A.2: Initial version representing the document generation components.

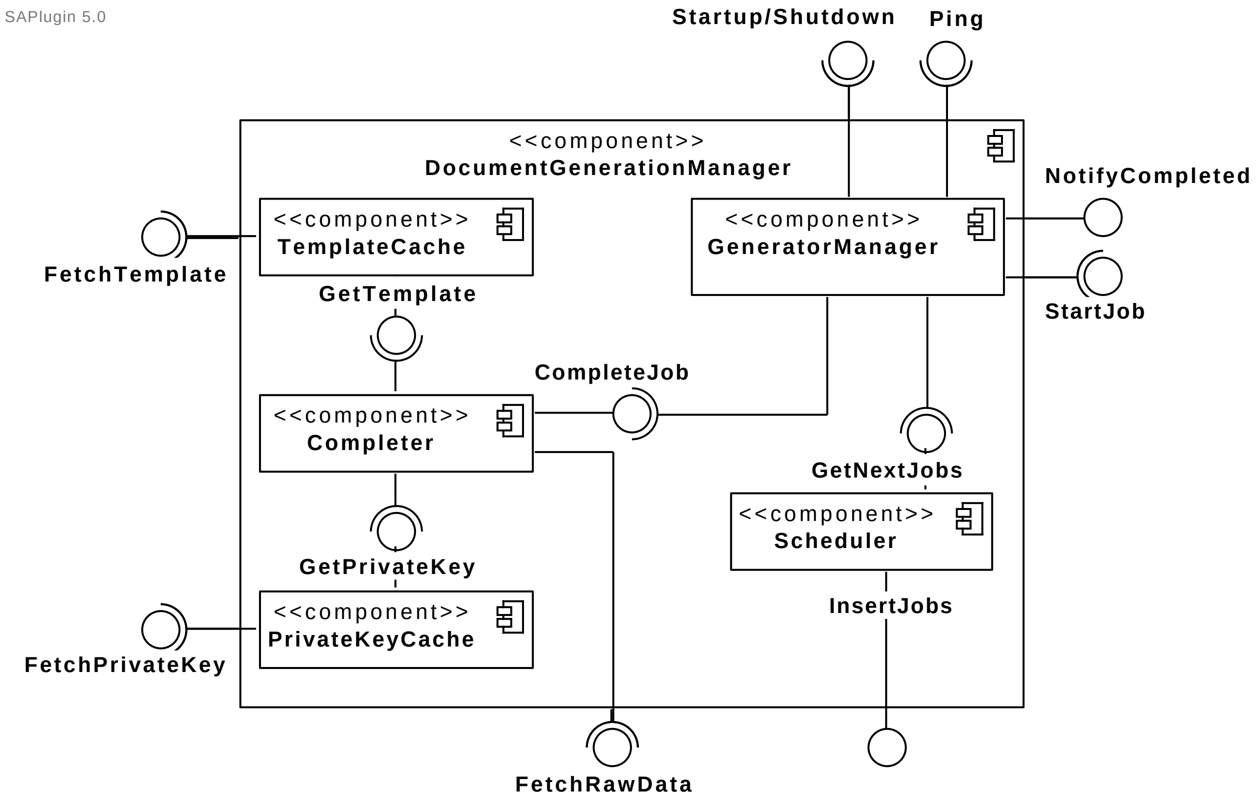


Figure A.3: Decomposition of the DocumentGenerationManager component

B. Module View (UML Component Diagram

Figures

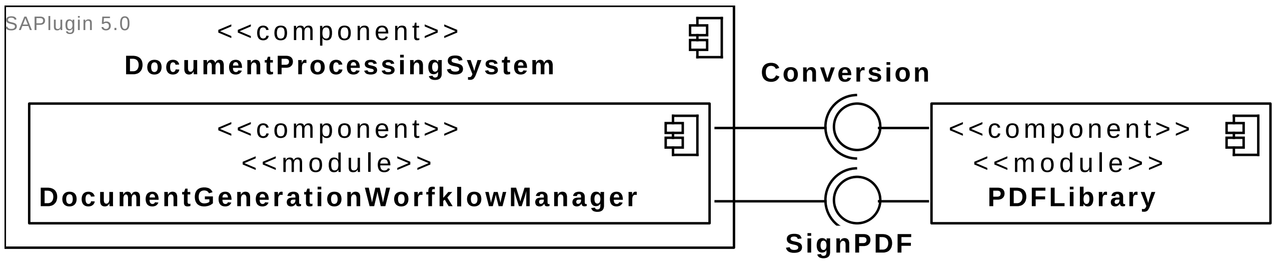


Figure B.1: Context diagram for the module view.

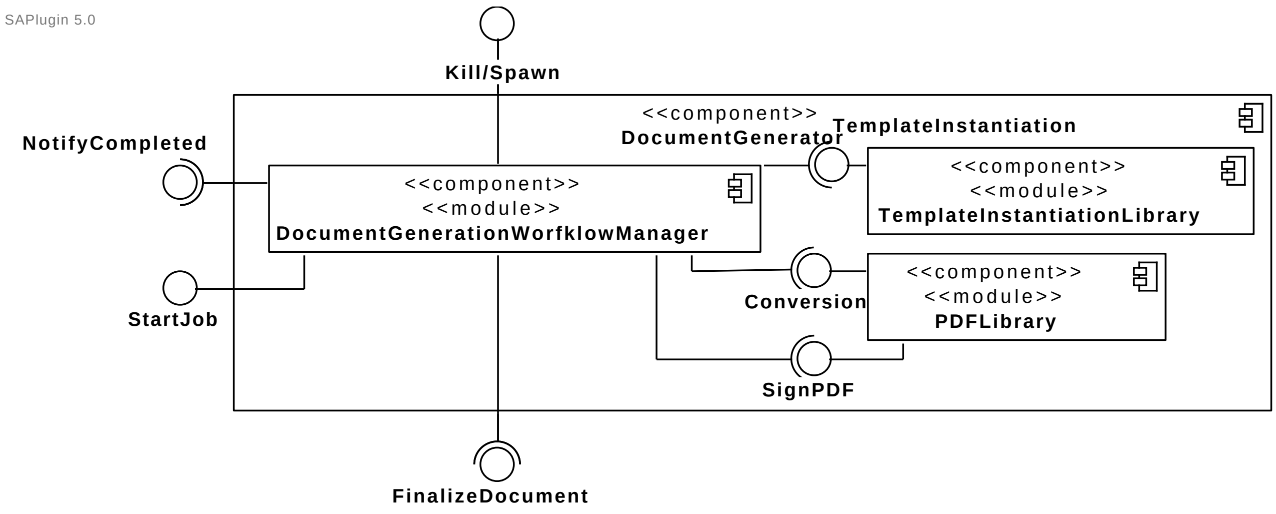


Figure B.2: Decomposition of the DocumentGenerator component

C. Deployment view (UML Deployment Diagram)

Figures

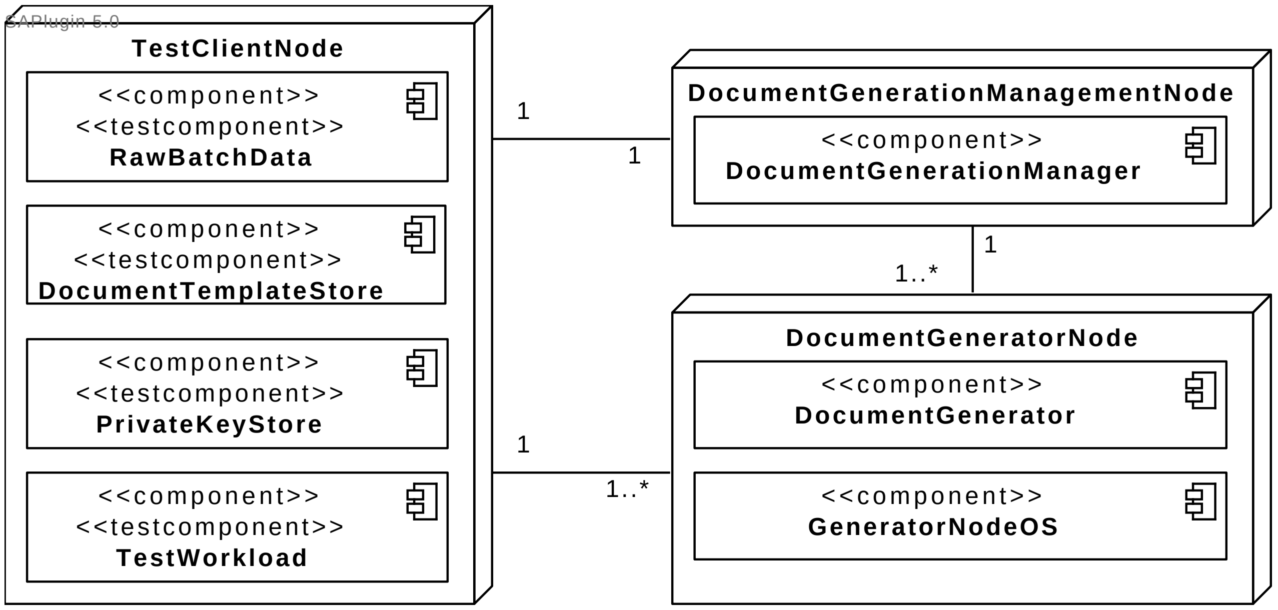
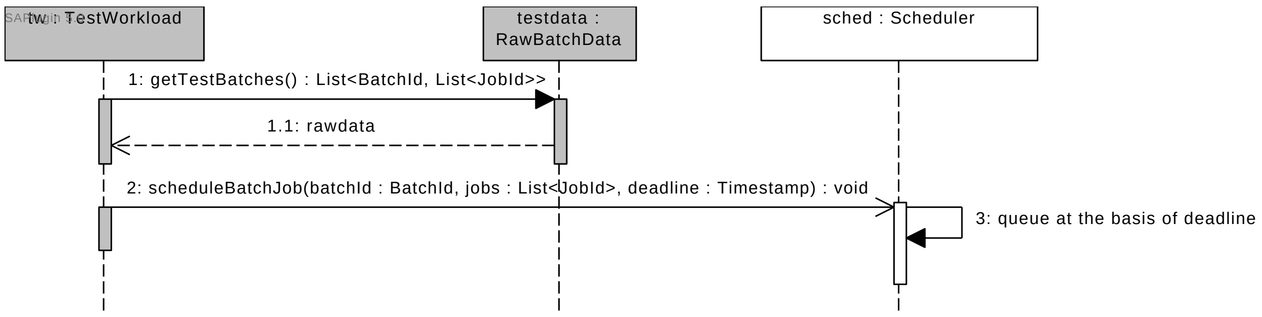


Figure C.1: Deployment view of the document generation system

D. Process View (UML Sequence Diagram)

Figures



tw : TestWorkload
testdata : RawBatchData
sched : Scheduler
1: getTestBatches() : List<BatchId, List<JobId>>
1.1: rawdata
2: scheduleBatchJob(batchId : BatchId, jobs : List<JobId>, deadline : Timestamp) : void
3: queue at the basis of deadline

Figure D.1: scheduling document generation jobs

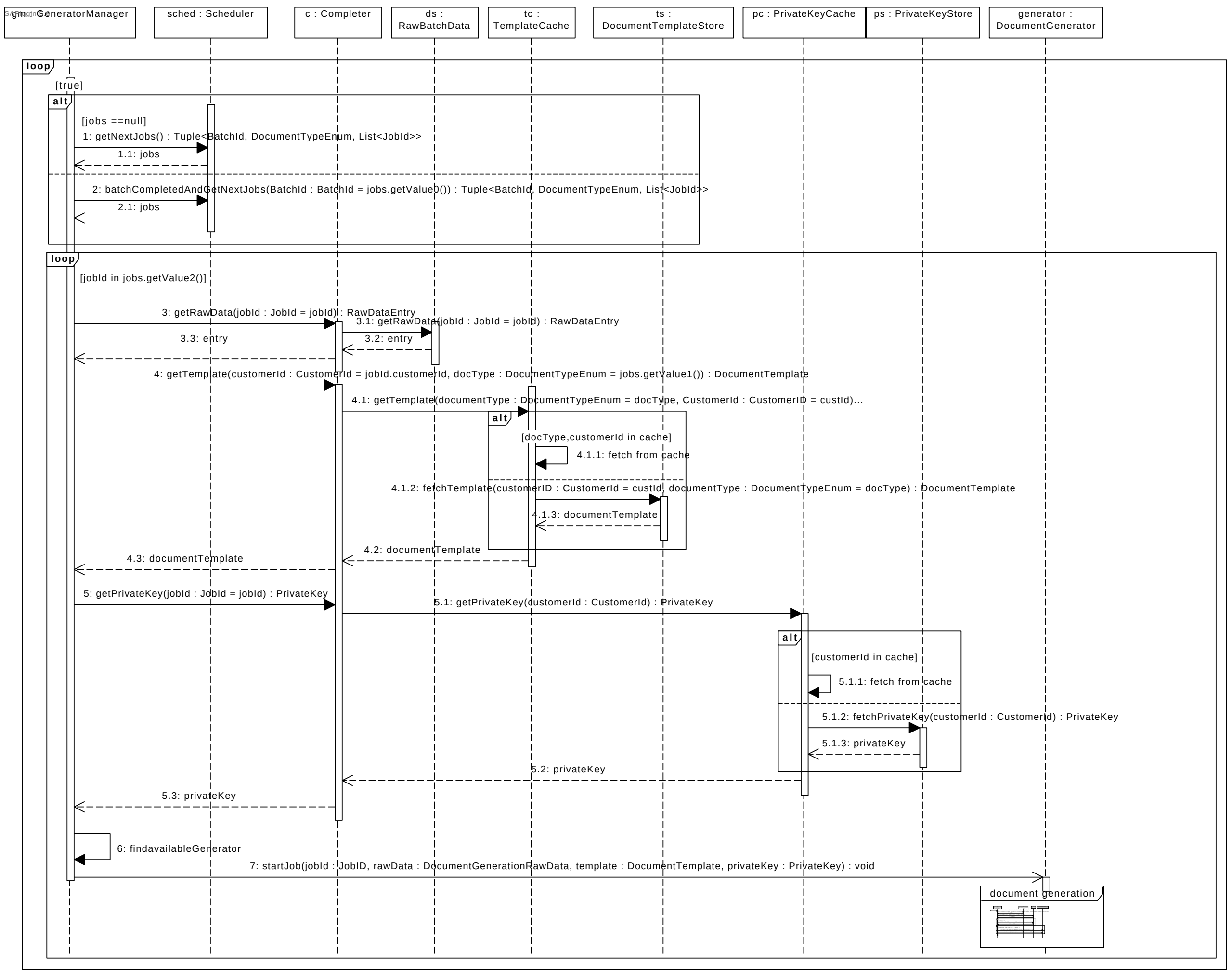

gm : GeneratorManager
sched : Scheduler
c : Completer
ds : RawBatchData
tc : TemplateCache
ts : DocumentTemplateStore
pc : PrivateKeyCache
ps : PrivateKeyStore
generator : DocumentGenerator
loop
[true]
alt
[jobs ==null]
1: getNextJobs() : Tuple<BatchId, DocumentTypeEnum, List<JobId>>
1.1: jobs
2: batchCompletedAndGetNextJobs(BatchId : BatchId = jobs.getValue0()) : Tuple<BatchId, DocumentTypeEnum, List<JobId>>
2.1: jobs
loop
[jobId in jobs.getValue2()]
3: getRawData(jobId : JobId = jobId) : RawDataEntry
3.1: getRawData(jobId : JobId = jobId) : RawDataEntry
3.2: entry
3.3: entry
4: getTemplate(customerId : CustomerId = jobId.customerId, docType : DocumentTypeEnum = jobs.getValue1()) : DocumentTemplate
4.1: getTemplate(documentType : DocumentTypeEnum = docType, CustomerId : CustomerID = custId)...
alt
[docType,customerId in cache]
4.1.1: fetch from cache
4.1.2: fetchTemplate(customerID : CustomerId = custId, documentType : DocumentTypeEnum = docType) : DocumentTemplate
4.1.3: documentTemplate
4.2: documentTemplate
4.3: documentTemplate
5: getPrivateKey(jobId : JobId = jobId) : PrivateKey
5.1: getPrivateKey(customerId : CustomerId) : PrivateKey
alt
[customerId in cache]
5.1.1: fetch from cache
5.1.2: fetchPrivateKey(customerId : CustomerId) : PrivateKey
5.1.3: privateKey
5.2: privateKey
5.3: privateKey
6: findavailableGenerator
7: startJob(jobId : JobID, rawData : DocumentGenerationRawData, template : DocumentTemplate, privateKey : PrivateKey) : void
document generation

Figure D.2: executing document generation jobs

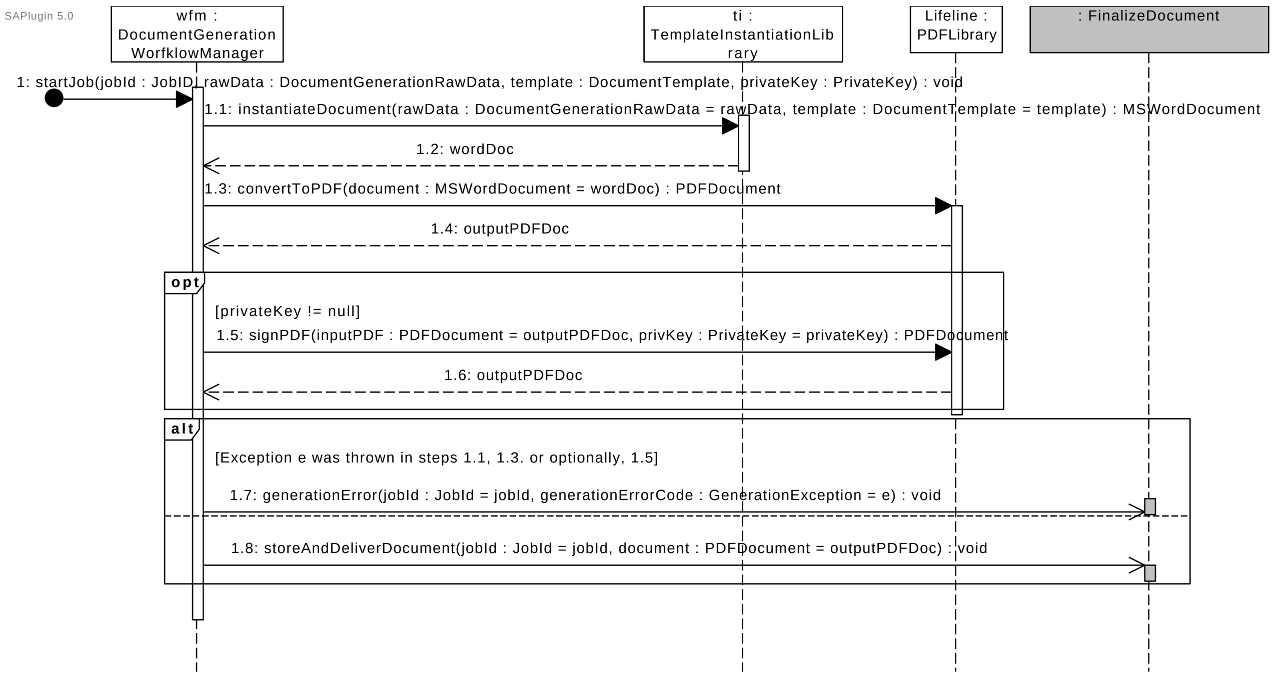

SAPlugin 5.0
wfm : DocumentGeneration WorfklowManager
ti : TemplateInstantiationLibrary
Lifeline : PDFLibrary
: FinalizeDocument
1: startJob(jobId : JobID, rawData : DocumentGenerationRawData, template : DocumentTemplate, privateKey : PrivateKey) : void
1.1: instantiateDocument(rawData : DocumentGenerationRawData = rawData, template : DocumentTemplate = template) : MSWordDocument
1.2: wordDoc
1.3: convertToPDF(document : MSWordDocument = wordDoc) : PDFDocument
1.4: outputPDFDoc
opt
[privateKey != null]
1.5: signPDF(inputPDF : PDFDocument = outputPDFDoc, privKey : PrivateKey = privateKey) : PDFDocument
1.6: outputPDFDoc
alt
[Exception e was thrown in steps 1.1, 1.3. or optionally, 1.5]
1.7: generationError(jobId : JobId = jobId, generationErrorCode : GenerationException = e) : void
1.8: storeAndDeliverDocument(jobId : JobId = jobId, document : PDFDocument = outputPDFDoc) : void

Figure D.3: document generation

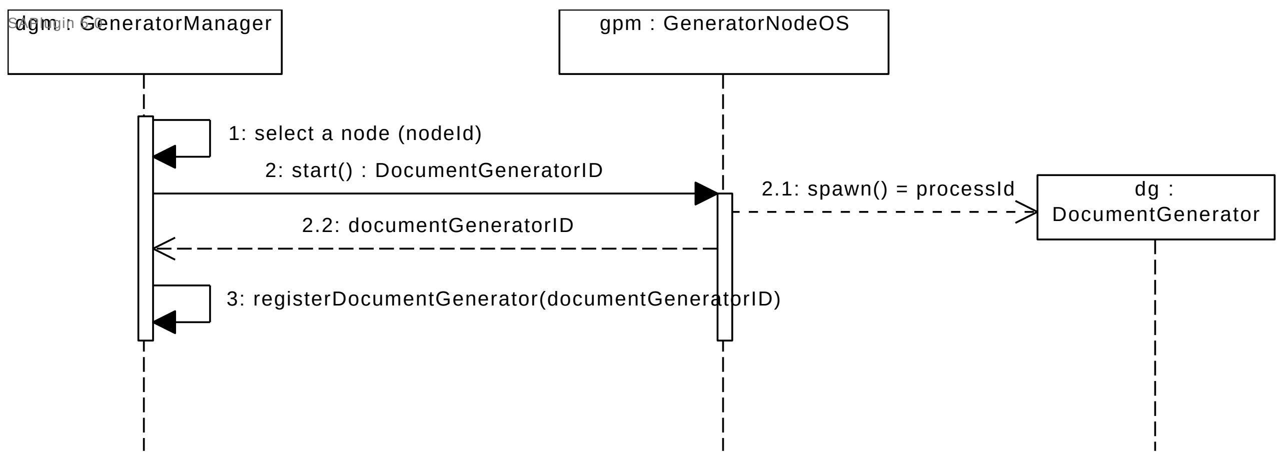


Figure D.4: start DocumentGenerator

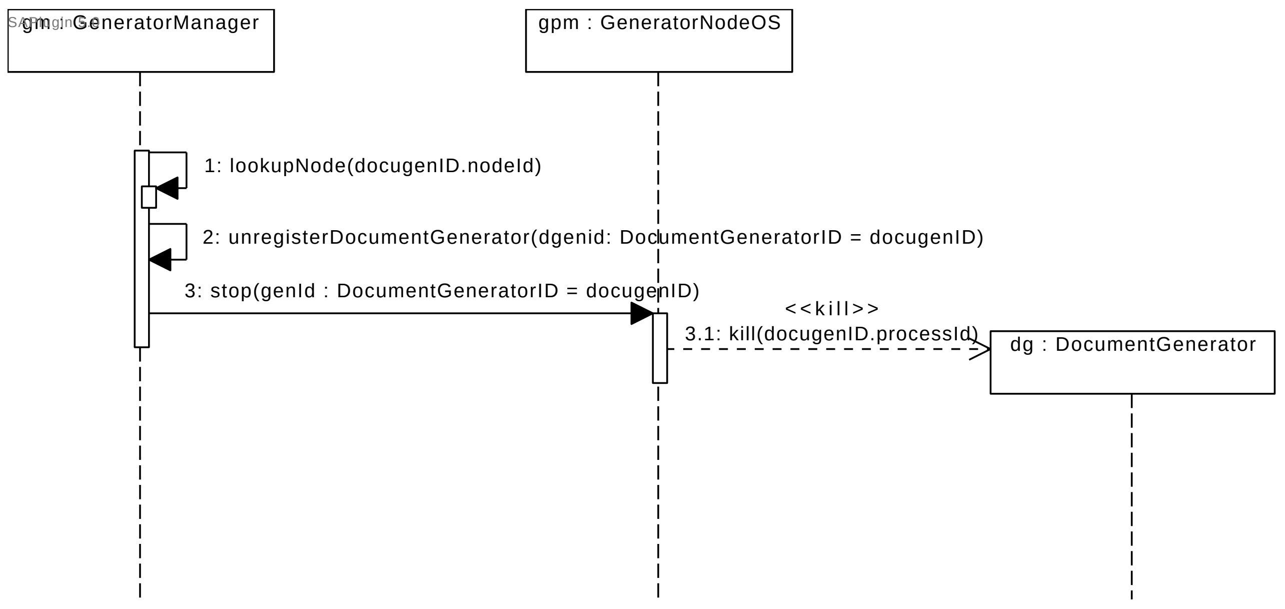

gm : GeneratorManager
gpm : GeneratorNodeOS
1: lookupNode(docugenID.nodeId)
2: unregisterDocumentGenerator(dgenid: DocumentGeneratorID = docugenID)
3: stop(genId : DocumentGeneratorID = docugenID)
<<kill>>
3.1: kill(docugenID.processId)
dg : DocumentGenerator

Figure D.5: kill DocumentGenerator

E. Element catalog

E.1 Components

E.1.1 Completer

Responsibility: Before assigning a an individual document processing job (**JobId**) to a DocumentGenerator instance, the GenerationManager asks the Completer for the raw data and the required meta-data (type of document, the document template and optionally the key for signing) to perform these jobs.
Super-components: DocumentProcessingSystem ▷ DocumentGenerationManager
Sub-components: None
Provided interfaces: CompleteJob
Required interfaces: FetchRawData, GetPrivateKey, GetTemplate
Deployed on: DocumentGenerationManagementNode
Visible on diagrams: figs. A.3 and D.2

E.1.2 DocumentGenerationManager

Responsibility: The DocumentGeneratorManager schedules and executes the document processing jobs (in batch). It plans and decides which jobs should be processed in the near future based on the deadlines of all jobs that should still be processed and and triggers the execution of individual document generation tasks (individual documents).
The DocumentGeneratorManager starts up new Generator instances if needed.
Super-components: DocumentProcessingSystem
Sub-components: Scheduler, GeneratorManager, Completer, TemplateCache, PrivateKey-Cache
Provided interfaces: InsertJobs, NotifyCompleted
Required interfaces: FetchPrivateKey, FetchRawData, FetchTemplate, Ping, StartJob, Startup/Shutdown
Deployed on: DocumentGenerationManagementNode
Visible on diagrams: figs. A.1, A.2, A.3 and C.1

E.1.3 DocumentGenerator

Responsibility: The DocumentGenerator generates a PDF document by completing a given template with document information specified in the raw data entry. If required, it signs the document with the specified private key.
The DocumentGenerator instances that have received the signal to shut down will complete their assigned group of document generation jobs and report back completion to the DocumentGenerationManager before actually shutting down.
Super-components: DocumentProcessingSystem
Sub-components: None
Provided interfaces: Kill/Spawn, StartJob
Required interfaces: FinalizeDocument, NotifyCompleted
Deployed on: DocumentGeneratorNode
Visible on diagrams: figs. A.1, A.2, B.2, C.1 and D.2

E.1.4 DocumentProcessingSystem

Responsibility: This component represents the entire DocumentProcessingSystem (level 1 of the decomposition) and groups all components involved in the document processing system.
Super-components: None
Sub-components: DocumentGenerator, DocumentGenerationManager, TestWorkload, RawBatchData, DocumentTemplateStore, PrivateKeyStore
Provided interfaces: Kill/Spawn

Required interfaces: ≺ `Ping`, ≺ `Startup/Shutdown`
Deployed on: DocumentGeneratorNode, TestClientNode, DocumentGenerationManagementNode
Visible on diagrams: figs. A.1, A.2 and B.1

E.1.5 DocumentTemplateStore

Responsibility: This is test template folder that for our test customers holds templates for both invoices and payslips
Super-components: ⧉ `DocumentProcessingSystem`
Sub-components: None
Provided interfaces: ⊸ `FetchTemplate`
Required interfaces: None
Deployed on: TestClientNode
Visible on diagrams: figs. A.2, C.1 and D.2

E.1.6 GeneratorManager

Responsibility: The GenerationManager iteratively fetches document processing jobs as scheduled. It triggers the execution by assigning the individual document generation tasks to `DocumentGenerator` instances.
It keeps track of DocumentGenerators, spawns and kills `DocumentGenerator` instances as needed.
The GenerationManager monitors the availability of the Generator using ping/echo (every 4 seconds).
Super-components: ⧉ `DocumentProcessingSystem` ▷ ⧉ `DocumentGenerationManager`
Sub-components: None
Provided interfaces: ⊸ `NotifyCompleted`
Required interfaces: ≺ `CompleteJob`, ≺ `GetNextJobs`, ≺ `Ping`, ≺ `StartJob`, ≺ `Startup/Shutdown`
Deployed on: DocumentGenerationManagementNode
Visible on diagrams: figs. A.3, D.2, D.4 and D.5

E.1.7 GeneratorNodeOS

Responsibility: This component is responsible for managing the `DocumentGenerator` processes running on a single `DocumentGenerator` node. It instantiates and kills processes of the `DocumentGenerator` and responds to ping messages.
Super-components: None
Sub-components: None
Provided interfaces: ⊸ `Ping`, ⊸ `Startup/Shutdown`
Required interfaces: ≺ `Kill/Spawn`
Deployed on: DocumentGeneratorNode
Visible on diagrams: figs. A.1, C.1, D.4 and D.5

E.1.8 PrivateKeyCache

Responsibility: This is a local cache that keeps a copy of the private keys used recently.
Super-components: ⧉ `DocumentProcessingSystem` ▷ ⧉ `DocumentGenerationManager`
Sub-components: None
Provided interfaces: ⊸ `GetPrivateKey`
Required interfaces: ≺ `FetchPrivateKey`
Deployed on: DocumentGenerationManagementNode
Visible on diagrams: figs. A.3 and D.2

E.1.9 PrivateKeyStore

Responsibility: The `PrivateKeyStore` keeps track of the private keys belonging to different customer organisations, for the purpose of adding digital signatures to generated documents.
Super-components: ⧉ `DocumentProcessingSystem`
Sub-components: None
Provided interfaces: ⊸ `FetchPrivateKey`
Required interfaces: None

Deployed on: TestClientNode
Visible on diagrams: figs. A.2, C.1 and D.2

E.1.10 RawBatchData

Responsibility: This is a dummy set of raw data for initial testing purposes.
Super-components: DocumentProcessingSystem
Sub-components: None
Provided interfaces: FetchRawData, GetBatchList
Required interfaces: None
Deployed on: TestClientNode
Visible on diagrams: figs. A.2, C.1, D.1 and D.2

E.1.11 Scheduler

Responsibility: The scheduler plans the generation of incoming batches taking into account (i) their inherent priority (customer), (ii) the deadline of the job, and (iii) the deadlines of all other ongoing jobs
Super-components: DocumentProcessingSystem ▷ DocumentGenerationManager
Sub-components: None
Provided interfaces: GetNextJobs, InsertJobs
Required interfaces: None
Deployed on: DocumentGenerationManagementNode
Visible on diagrams: figs. A.3, D.1 and D.2

E.1.12 TemplateCache

Responsibility: The TemplateCache stores already fetched templates to minimize I/O
Super-components: DocumentProcessingSystem ▷ DocumentGenerationManager
Sub-components: None
Provided interfaces: GetTemplate
Required interfaces: FetchTemplate
Deployed on: DocumentGenerationManagementNode
Visible on diagrams: figs. A.3 and D.2

E.1.13 TestWorkload

Responsibility: This is a temporary test script that triggers the batch generation of a set of dummy documents, as specified in the RawBatchData file.
Super-components: DocumentProcessingSystem
Sub-components: None
Provided interfaces: FinalizeDocument
Required interfaces: GetBatchList, InsertJobs
Deployed on: TestClientNode
Visible on diagrams: figs. A.2, C.1 and D.1

E.2 Modules

E.2.1 DocumentGenerationWorfklowManager

Responsibility: This component steers the entire process of generating documents and reports the results (storage and delivery of finished documents, or reporting about exceptions and failures)
Super-components: None
Sub-components: None
Provided interfaces: Kill/Spawn, StartJob
Required interfaces: Conversion, FinalizeDocument, NotifyCompleted, SignPDF, TemplateInstantiation
Deployed on: DocumentGeneratorNode

Visible on diagrams: figs. B.1, B.2 and D.3

E.2.2 PDFLibrary

Responsibility: This is Apache PDFBox. See https://pdfbox.apache.org/
Super-components: None
Sub-components: None
Provided interfaces: -○ Conversion, -○ SignPDF
Required interfaces: None
Deployed on: DocumentGeneratorNode
Visible on diagrams: figs. B.1, B.2 and D.3

E.2.3 TemplateInstantiationLibrary

Responsibility: This library provides the functions to create a document. This is done with a template (Microsoft Word document) and involves filling in the generic parameters with information provided in the raw data uploaded by the customer.
Super-components: None
Sub-components: None
Provided interfaces: -○ TemplateInstantiation
Required interfaces: None
Deployed on: DocumentGeneratorNode
Visible on diagrams: figs. B.2 and D.3

E.3 Interfaces

E.3.1 CompleteJob

Provided by: Completer
Required by: GeneratorManager
Operations:

- **PrivateKey** getPrivateKey(**JobId** jobId)
  - Effect: The private key to be used in a specific document processing job is obtained.
  - Returns: Returns the requested **PrivateKey** object.
  - Sequence Diagrams: fig. D.2
- **RawDataEntry** getRawData(**JobId** jobId)
  - Effect: The raw data that is to be used to generate a specific document is fetched through this operation.
  - Returns: Returns the requested **RawDataEntry** object.
  - Sequence Diagrams: fig. D.2
- **DocumentTemplate** getTemplate(**CustomerId** customerId, ***DocumentTypeEnum*** docType)
  - Effect: The template to be used in a specific document processing job is to be used.
  - Returns: Returns the requested **DocumentTemplate**.
  - Sequence Diagrams: fig. D.2

Diagrams: fig. A.3

E.3.2 Conversion

Provided by: None
Required by: None
Operations:

- **PDFDocument** convertToPDF(**MSWordDocument** document) throws *PDFConversionException*
  - Effect: This operation is used to convert any Microsoft Word document into a PDF. If the conversion fails, a PDFConversionException is thrown.
  - Returns: Returns the **PDFDocument** object that is the result of the conversion.
  - Sequence Diagrams: fig. D.3

Diagrams: figs. B.1 and B.2

E.3.3 FetchPrivateKey

Provided by: PrivateKeyStore
Required by: DocumentGenerationManager, PrivateKeyCache
Operations:

- **PrivateKey** fetchPrivateKey(**CustomerId** customerId)
  - Effect: The private key for a specific specific customer is fetched from persistent storage (e.g. read from file).
  - Returns: Returns the requested **PrivateKey** object.
  - Sequence Diagrams: fig. D.2

Diagrams: figs. A.2 and A.3

E.3.4 FetchRawData

Provided by: RawBatchData
Required by: Completer, DocumentGenerationManager
Operations:

- **RawDataEntry** getRawData(**JobId** jobId)
  - Effect: This method is used to read the raw data for a specific job from the test file
  - Returns: Returns the requested raw data entry.
  - Sequence Diagrams: fig. D.2

Diagrams: figs. A.2 and A.3

E.3.5 FetchTemplate

Provided by: DocumentTemplateStore
Required by: DocumentGenerationManager, TemplateCache
Operations:

- **DocumentTemplate** fetchTemplate(**CustomerId** customerID, ***DocumentTypeEnum*** documentType)
  - Effect: The template for a specific document type and a specific customer is fetched from persistent storage (e.g. read from file)
  - Returns: Returns a **DocumentTemplate** object.
  - Sequence Diagrams: fig. D.2

Diagrams: figs. A.2 and A.3

E.3.6 FinalizeDocument

Provided by: TestWorkload
Required by: DocumentGenerator
Operations:

- void generationError(**JobId** jobId, ***GenerationException*** generationErrorCode)
  - Effect: This operation is used to signal that a specific document generation process has failed.
  - Sequence Diagrams: fig. D.3
- void storeAndDeliverDocument(**JobId** jobId, **PDFDocument** document)
  - Effect: This operation is used to store the generated PDF document in the document, archive and trigger the delivery phase.
  - Sequence Diagrams: fig. D.3

Diagrams: figs. A.2 and B.2

E.3.7 GetBatchList

Provided by: RawBatchData
Required by: TestWorkload
Operations:

- List<**BatchId**, List<**JobId**>> getTestBatches()
  - Effect: This temporary operation is used to read the test batch from persistent storage (e.g. from a CSV file).
  - Returns: Returns a list of batches (ids), together with the corresponding individual job identifiers for each batch.

– Sequence Diagrams: fig. D.1

Diagrams: fig. A.2

E.3.8 GetNextJobs

Provided by: Scheduler
Required by: GeneratorManager
Operations:

- Tuple<**BatchId**, ***DocumentTypeEnum***, List<**JobId**>> batchCompletedAndGetNextJobs(**BatchId** BatchId)
  - Effect: This method is called to signal completion of a batch job that was fetched earlier and to fetch the next batch job in line from the queue
  - Returns: Returns a tuple that combines the identifiers of the next jobs to be executed (List<**JobId**)), the type of document that is processed (***DocumentTypeEnum***) and the identifier of the batch to which these jobs belong (**BatchId**).
  - Sequence Diagrams: fig. D.2
- Tuple<**BatchId**, ***DocumentTypeEnum***, List<**JobId**>> getNextJobs()
  - Effect: This method is called to fetch the next batch job in line from the queue, consisting of a **BatchId**, the DocumentType (Enum) to be instantiated, and the list of document processing jobs, consisting of a unique ID and the raw data.
  - Returns: Returns a tuple that combines the identifiers of the next jobs to be executed (List<**JobId**)), the type of document that is processed (***DocumentTypeEnum***) and the identifier of the batch to which these jobs belong (**BatchId**).
  - Sequence Diagrams: fig. D.2

Diagrams: fig. A.3

E.3.9 GetPrivateKey

Provided by: PrivateKeyCache
Required by: Completer
Operations:

- **PrivateKey** getPrivateKey(**CustomerId** customerId)
  - Effect: The private key to be used to sign a generated document of a specific customer is obtained through this operation. If the private key is not in the cache (cache miss), it will be fetched from persistent storage and kept in cache.
  - Returns: Returns the requested **PrivateKey** object.
  - Sequence Diagrams: fig. D.2

Diagrams: fig. A.3

E.3.10 GetTemplate

Provided by: TemplateCache
Required by: Completer
Operations:

- **DocumentTemplate** getTemplate(***DocumentTypeEnum*** documentType, **CustomerId** CustomerId)
  - Effect: The template to be used to generate an document of a specific type for a specific customer is obtained through this operation. If the template is not in the cache (cache miss), it will be fetched from persistent storage and kept in cache.
  - Returns: Returns the requested **DocumentTemplate** object.
  - Sequence Diagrams: fig. D.2

Diagrams: fig. A.3

E.3.11 InsertJobs

Provided by: DocumentGenerationManager, Scheduler
Required by: TestWorkload
Description : This interface groups the operations used to schedule document generation processes at the level of an entire batch.
Operations:

- void scheduleBatchJob (**BatchId** batchId, List<**JobId**> jobs, **Timestamp** deadline)
  - Effect: This operation is used to schedule a batch job in the document generation subsystem. Scheduling will be based on the deadline and the available resources.
  - Sequence Diagrams: fig. D.1

Diagrams: figs. A.2 and A.3

E.3.12 Kill/Spawn

Provided by: DocumentGenerator, DocumentProcessingSystem
Required by: GeneratorNodeOS
Operations:

- void kill(int processId)
  - Effect: Sends a termination signal to a DocumentGenerator process to end its activities. This will effectively end the process after finishing document generation processes currently active.
  - Sequence Diagrams: None
- int spawn()
  - Effect: Spawns a new DocumentGeneration process and runs the startup logic of a document generation job: provisioning resources (such as memory/cpu/local storage), loading libraries, etc.
  - Returns: Returns the OS-level identifier of the process on the node (process ID/pid).
  - Sequence Diagrams: None

Diagrams: figs. A.1, A.2 and B.2

E.3.13 NotifyCompleted

Provided by: DocumentGenerationManager, GeneratorManager
Required by: DocumentGenerator
Description : This interface is used to notify the DocumentGenerationManager of completion of a job that was assigned previously. The DocumentGenerationManager can keep track of throughput of these jobs to assess the overall performance.
Operations:

- void notifyJobCompletion(**JobId** jobId)
  - Effect: Signal to notify that a document processing job was completed and that the DocumentGenerator is free again.
  - Sequence Diagrams: None

Diagrams: figs. A.2, A.3 and B.2

E.3.14 Ping

Provided by: GeneratorNodeOS
Required by: DocumentGenerationManager, DocumentProcessingSystem, GeneratorManager
Description : ping - send ICMP ECHO_REQUEST to network hosts.
Operations:

- boolean ping()
  - Effect: ping - send ICMP ECHO_REQUEST to network hosts. Is used to verify that the node is still operational and responsive.
  - Returns: Returns an echo if the system is operational and responsive.
  - Sequence Diagrams: None

Diagrams: figs. A.1, A.2 and A.3

E.3.15 SignPDF

Provided by: None
Required by: None
Operations:

- **PDFDocument** signPDF(**PDFDocument** inputPDF, **PrivateKey** privKey) throws *PDFSigningException*
  - Effect: This operation is used to digitally sign a PDF with the provided privateKey. Using the corresponding public key, the authenticity of the document can be verified. If the signing process fails, a PDFSigningException is thrown.

- Returns: Returns the signed **PDFDocument** object.
- Sequence Diagrams: fig. D.3

Diagrams: figs. B.1 and B.2

E.3.16 StartJob

Provided by: DocumentGenerator
Required by: DocumentGenerationManager, GeneratorManager
Description : This interface is used to start a document processing job.
Operations:

- void startJob(**JobId** jobId, **DocumentGenerationRawData** rawData, **DocumentTemplate** template, **PrivateKey** privateKey)
  - Effect: This operation is called to trigger the immediate start of a document generation job. If no private key is provided, the resulting **PDFDocument** will not be signed.
  - Sequence Diagrams: fig. D.2

Diagrams: figs. A.2, A.3 and B.2

E.3.17 Startup/Shutdown

Provided by: GeneratorNodeOS
Required by: DocumentGenerationManager, DocumentProcessingSystem, GeneratorManager
Description : This interface groups together operations that can be used to start and stop individual DocumentGenerators
Operations:

- **DocumentGeneratorID** start()
  - Effect: Starts a new DocumentGenerator.
  - Returns: Returns a **DocumentGeneratorID**, which refers to the node and the started process.
  - Sequence Diagrams: fig. D.4
- void stop(**DocumentGeneratorID** genId)
  - Effect: Stops a specific DocumentGenerator, referred to by it **DocumentGeneratorID**
  - Sequence Diagrams: fig. D.5

Diagrams: figs. A.1, A.2 and A.3

E.3.18 TemplateInstantiation

Provided by: None
Required by: None
Operations:

- **MSWordDocument** instantiateDocument(**DocumentGenerationRawData** rawData, **DocumentTemplate** template) throws *DocumentInstantiationException*
  - Effect: This operation triggers the instantiation of a template with specific data obtained from a raw data entry and generates a **MSWordDocument**. When the raw data does not provide the necessary parameters or when data can not be read, a DocumentInstantiationException is thrown.
  - Returns: Returns the generated **MSWordDocument** object.
  - Sequence Diagrams: fig. D.3

Diagrams: fig. B.2

E.4 Nodes

E.4.1 DocumentGenerationManagementNode

Responsibility: This represents a node on which the coordination and scheduling of document generation jobs is performed.
Visible on diagrams: fig. C.1

E.4.2 DocumentGeneratorNode

Responsibility: This represents a node on which document generation jobs are executed.

Visible on diagrams: fig. C.1

E.4.3 TestClientNode

Responsibility: This represents the node on which the test is initiated and coordinated.
Visible on diagrams: fig. C.1

E.5 Exceptions

- *DocumentInstantiationException* This exception is thrown when a problem has occurred during the instantiation of a document (i.e. filling in a template with raw data)
- *PDFConversionException* This exception is thrown when a problem has occurred during the conversion of documents towards the PDF format.
- *PDFSigningException* This exception is thrown when a problem has occurred while digitally signing a document.

E.6 Data types

- **Address**:
  The postal address of a recipient.
- **BatchId**:
  Attributes: **CustomerId** customerID, int batchId
  This represents the unique identifier of a Document generation batch in the system, it is a composite of the customerId and a unique batchId for that customer.
- **ByteArray**:
  Represents a byte array.
- **CustomerId**:
  Attributes: int id
  This class represents a unique identifier to a customer organisation as it is registered in the system.
- **DeliveryInformation**:
  Represents the delivery information for a single document, as provided in the raw data. Note: this data type has to be completed in a later stage.
- ***Document***:
  Attributes: **ByteArray** byteArray
  Subtypes: **△MSWordDocument**, **△PDFDocument**
  Base class to represent a generated document.
- **DocumentGenerationRawData**:
  Attributes: ***DocumentTypeEnum*** docType, Map<String, String> parameters
  This is the part of the raw data used for generating a single document, comprising of (i) the document type, (ii) the parameters to be filled in to generate a document of said type (encoded as String).
- **DocumentGeneratorID**:
  Attributes: int nodeId, int processId
  This is the identifier of a DocumentGenerator process in the system. It refer to the node and the process number on that node on which the DocumentGenerator is running.
- **DocumentId**:
  An identifier used by the document provider to uniquely identify a specific document.
- **DocumentTemplate**:
  Attributes: **ByteArray** byteArray
  Represents a Microsoft Word template file. This is a regular Word file with predefined generic parameters that have to be filled in during document generation.
- ***DocumentTypeEnum***:
  Subtypes: **△PayslipDocumentType**, **△InvoiceDocumentType**
  Enum structure to represent the different types of documents that can be generated by the DocProc system.
- ***GenerationException***:

Base class for any type of exception that may occur during the generation of documents.

- **InvalidInvoiceException**:
  Thrown when the provided invoice is not correctly formatted or does not contain all required information.
- **InvalidSupplierException**:
  Thrown when the identified supplier cannot send invoices via Zoomit to the customer.
- **Invoice**:
  Attributes: **PDFInvoice** pdf, string customerEndpointID, string registrationName, string supplierEndpointID, string dueDate, string paymentID
  An XML object containing the PDF of the invoice along with the necessary meta data.
- **InvoiceDocumentType** (△***DocumentTypeEnum***):
  Specific document type referring to invoices.
- **JobId**:
  Attributes: **BatchId** batchId, int jobId
  Unique identifier referring to a document processing job involving a single document. It refers to the identifier of the entire batch to which this individual job belongs.
- **MSWordDocument** (△***Document***):
  Represents a generated Microsoft Word document (.docx)
- **NoSuchSupplierException**:
  Thrown when the company with the given VAT identification number is not registered with Zoomit.
- **PayslipDocumentType** (△***DocumentTypeEnum***):
  Specific document type referring to payslips.
- **PDFDocument** (△***Document***):
  Represents a generated PDF document (.pdf) May or may not be digitally signed
- **PDFInvoice**:
  A PDF file representing an invoice.
- **PrintParameters**:
  Attributes: boolean single, string paperType, boolean colour
  Specifies how a document should be printed.
- **PrivateKey**:
  Attributes: **ByteArray** privateKey
  A private key used for digitally signing PDF documents, encoded as a byte array.
- **RawDataBatch**:
  Attributes: **Timestamp** deadline
  This represents a batch of raw data, as uploaded by the customer organisation. It is a collection of RawDataEntry objects.
- **RawDataEntry**:
  This data type represents the raw data used to generate and deliver a single document
- **Timestamp**:
  Timestamp representing a deadline in the system.

E.7 Unresolved issues

SA Plugin v5.0.12 (VP OpenAPI v16.1)

- [Diag.depl.07] : 1 (*Project does not contain a deployment context diagram.*)