Katholieke
Universiteit
Leuven

**Department of
Computer Science**

# DOCUMENT PROCESSING

## The complete architecture

Software Architecture (H09B5a and H07Z9a) – Part 2b

**Jeroen Reinenbergh (r0460600)**
**Jonas Schouterden (r0260385)**

Academic year 2014–2015

# Contents

# 1 Introduction

The goal of this project was to design a document processing system. In this document, we describe the final architecture, which was designed based on the requirements from the domain analysis and the priorities of these requirements given by our stakeholders. Section 2 lists the architectural decisions for all non-functional requirements and discusses the final architecture. Section 3 provides and discusses the main context and decomposition diagrams of our architecture (i.e. the context and primary diagram of the component-and-connector view) along with a discussion of the main architectural decisions involved. Section 4 provides and discusses the more fine-grained decompositions of some of the major components in the main decomposition. Section 5 provides and discusses the deployment of the components of the component-and-connector view on physical nodes. Finally, Section 6 illustrates how our architecture accomplishes the most important functionality and data flows using sequence diagrams. Afterwards, Appendix A lists and describes all components of the component-and-connector view and their interfaces and Appendix B lists and describes the data types used in these interfaces.

# 2 Overview

This section gives a high-level but complete overview of the system: it lists the design decisions for all non-functional requirements and provides a discussion concerning the strong and weak points of the architecture.

## 2.1 Architectural decisions

In this section, we give an overview of the architectural decisions made in our architecture in order to achieve the requirements given in the domain analysis.

**Av1a & Av2a: Notifying the appropriate operator within 1 minute**

**Av1b: Storing the status of an individual job**

**Av2b: Temporarily storing documents that should be delivered via the personal document store for at least 3 hours in case the PDSDB is unavailable and providing a clear message to users in order to fail gracefully**

**DocumentStorageCache** This component temporarily stores the id's of all generated documents that are to be delivered through the personal document store during downtime of the `PDSDB` component up to a maximum of 3 hours. When the latter component turns operational again, the `DocumentStorageManager` retrieves all documents and their corresponding meta data from the `DocumentDB` using the previously mentioned id's and subsequently stores them in the `PDSDB`. Note that the recipient therefore perceives a maximum total downtime of 3 hours plus the time needed for the `DocumentStorageManager` to transfer all documents and meta data that the cached id's refer to.
Note that the `DocumentDB` and the `PDSDB` store documents in exactly the same way, both including the document itself, its meta data and its id, although there is no need for the meta data in the former database. This storage tactic ensures that the `DocumentStorageManager` does not have to fetch or convert any information when transferring documents between both databases, making the transfer as efficient as possible. This does not introduce any overhead to the storage system, since the meta data is but a fraction of the document data. Finally, we would like to stress that the `DocumentStorageManager` implicitly pings the `PDSDB` when storing document data in it. The echo message then, in turn, consists of the write confirmation that is subsequently received. If one of those writes should fail, all subsequent writes are converted into document id writes to the aforementioned cache. Upon revival of the `PDSDB`, this database will send a single heartbeat to the `DocumentStorageManager`, causing this component to begin transferring the missing document data. Once all cached document IDs are processed, subsequent writes to the `PDSDB` will no longer be redirected through the `DocumentStorageCache`.

**Av1b: Storing the status of an individual job**

**Av3: Zoomit failure**

**P2: Document lookups**

**Sharding for DocumentDB for P2** In order to improve performance of the `DocumentDB`, we chose to partition this database into multiple shards. This approach was driven by the need for fast response time while keeping in mind the high storage cost for documents. More precisely, a `ShardingManager` is responsible for reading and writing all documents in one of the `DocumentDBShards`, making sure that every document is stored only once. This sharding technique provides roughly the same advantages in response time as active replication, but is significantly less costly when it comes to storage capacity. Note that there must also be a (sub)component monitoring all requests to the shards, in order to throttle excessive requests when necessary. Since this is a rather simple task and the (sub)component must be aware of the details concerning the sharding architecture to efficiently fulfil its purpose, this functionality is delegated to the `ShardingManager` itself. Finally, this `ShardingManager` is also responsible for implicitly pinging all `DocumentDBShards` upon reading and writing in them.

**Dedicated DocumentDB and JobDB** Since a large number of document lookups and downloads should not affect the performance of other functionality of the system, both link mappings and documents are stored in dedicated databases, each deployed on a different node. This decision ensures that documents can be looked up via the personal document store or a notification in a timely fashion, because it prohibits either of those two components to be a bottleneck in the document lookup process. Since, according to the previous decomposition, the `PDSDB` is deployed on a separate node too, this component's performance is already satisfactory and no changes need to be made to improve it.

**P3: Status overview for customer administrators**

**M1: New type of document: bank statements**

**M2: Multiple print & postal services**

**M3: Dynamic selection of the cheapest of print & postal services**

## 2.2 Discussion

Use this section to discuss your architecture in retrospect. For example, what are the strong points of your architecture? What are the weak points? Is there anything you would have done otherwise with your current experience? Are there any remarks about the architecture that you would give to your customers? Etc.

**Alternative for DocumentStorageCache** We could just as well do without the previously introduced `DocumentStorageCache` by letting the `DocumentStorageManager` actively look for documents in the `DocumentDB` that are not yet, but should be, present in the `PDSDB` upon revival of the `PDSDB`. A clear advantage of this approach is that the downtime of the `PDSDB` component is no longer restricted by the aforementioned 3-hour cache. An important disadvantage, however, lies in the fact that the `DocumentStorageManager` is burdened with a significant amount of extra work and will require more expensive hardware to cope with this. Since the support for a longer downtime of the `PDSDB` is out of scope, this approach was not chosen.

# 3 Client-server view (UML Component diagram)

The context diagram of the client-server view. Discuss which components communicate with external components and what these external components represent.

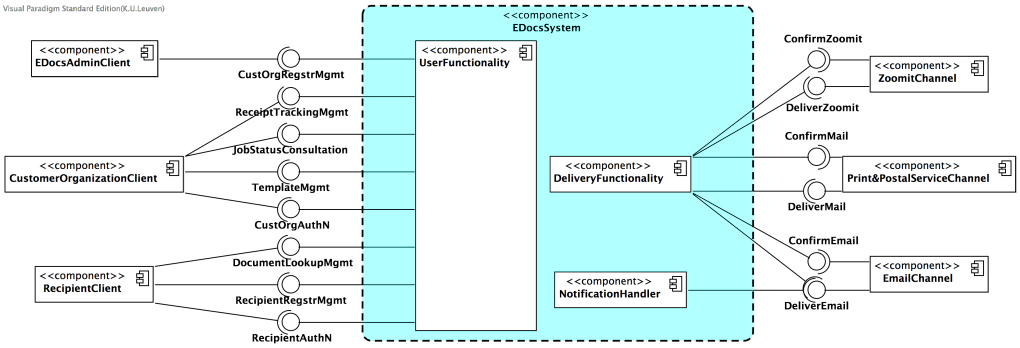


Figure 1: Context diagram for the client-server view.

The primary diagram and accompanying explanation.

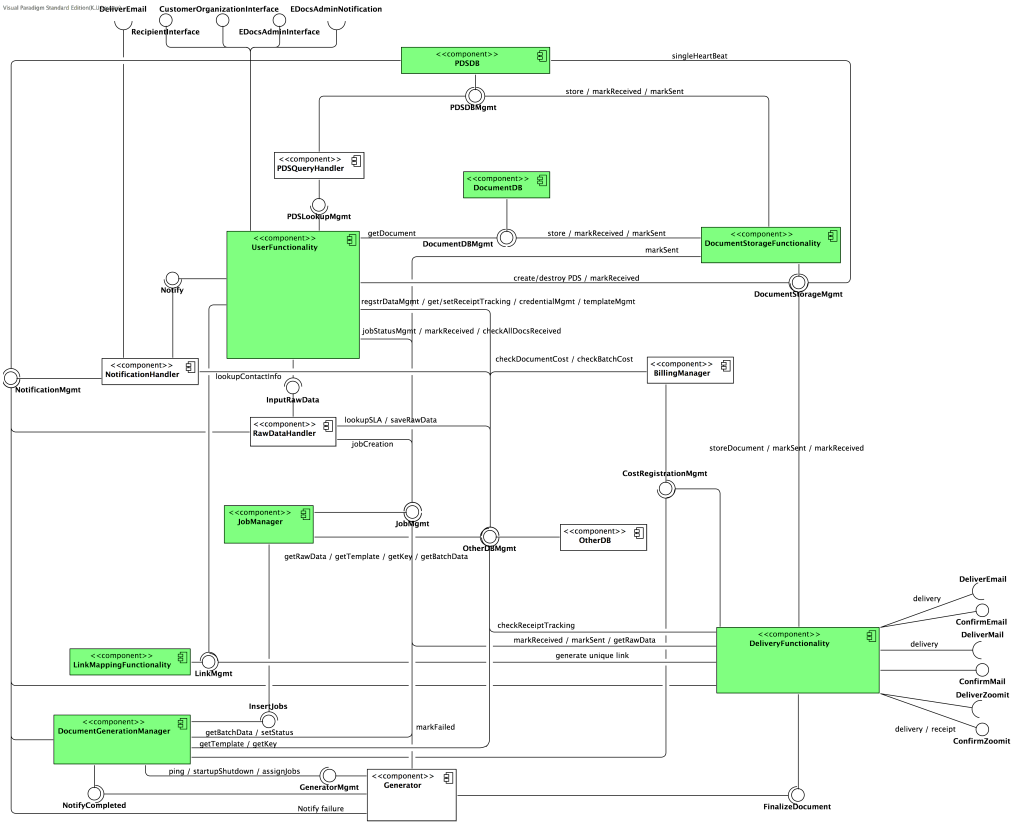


Figure 2: Primary diagram of the client-server view.

## 3.1 Main architectural decisions

Discuss your architectural decisions for the most important requirements in more detail using the components of the client-server view. Pay attention to the solutions that you employed and the alternatives that you considered. The explanation here must be self-contained and complete. Imagine you had to describe how the architecture supports the core functionality to someone that is looking at the client-server view only. Hide unnecessary details (these should be shown in the decomposition view).

**UserFacade** We introduce this component to be able to distinguish between registered recipients and unregistered recipients in the first steps of the document lookup process. The `UserFacade` also maps incoming links to documents that are located either in the `PDSDB` or in the `DocumentDB` by first fetching the correct mapping from the `LinkMappingDB` mentioned above. Another responsibility of the `UserFacade` is marking

documents as received, since this is the last internal component through which the document is passed before the requesting recipient actually receives it and failure of another component in the document lookup pipeline can no longer prevent this from happening.

**PDSFacade** This component handles all read requests that are intended for the personal document store. This extra level of indirection calculates and aggregates all intermediate query results as to relieve the `PDSDB` of this burden, which is also the reason why both components are best deployed on different nodes. Upon querying the personal document store, this component first collects the recipient's documents' meta data and subsequently performs queries on this collection. This approach introduces no significant overhead, because only those documents that are needed, will be fetched in their entirety. It is also the responsibility of the `PDSFacade` to check the recipient's ID against the recipient ID that can be retrieved from the document meta data and to only pass on those documents for which they are a match.

**DocumentStorageManager** The storage of documents is handled by the `DocumentStorageManager`, which receives generated documents from the intermediary `OtherFunctionality2` component and stores them in either the `DocumentDB` (for unregistered recipients) or in both the `DocumentDB` and the `PDSDB` (for registered recipients) according to the method that is invoked on it. The necessity for this component follows from the fact that synchronisation is needed between the two storage components mentioned above. Note that the PDSDocMgmt interface in the `PDSDB` component now requires an extra method to save documents.

**LinkManager** Motivatie voor LinkManager: Checks expiration date ALS DAT NODIG IS–¿ reason: links naar de pdsdb vervallen niet (zolang de gebruik geregistreerd is) LinkManager maps link to (document ID, place where the document is stored)-pairs –¿ REASON: the unique link has two possible sources: an e-mail to an unregistered recipient or an email to a registered recipient. For an unregistered recipient, the RecipientFacade must look with the documentid for the document in the documentDB. For a registered recipient, the RecipientFacade must look with the documentid for the document in the PDSDB. (Mogelijk een boolean ofzo) Does NOT do mapping removal after x years –¿ there has to be a notification when the link has expired

### 3.1.1 ReqX: requirement name

Describe the design choices related to *ReqX* together with the rationale of why these choices where made.

#### Alternatives considered

**Alternative(s) for choice 1** Explain what alternative(s) you considered for this design choice and why they where not selected.

# 4 Decomposition view (UML Component diagram)

Discuss the decompositions of the components of the client-server view which you have further decomposed.

## 4.1 DeliveryFunctionality

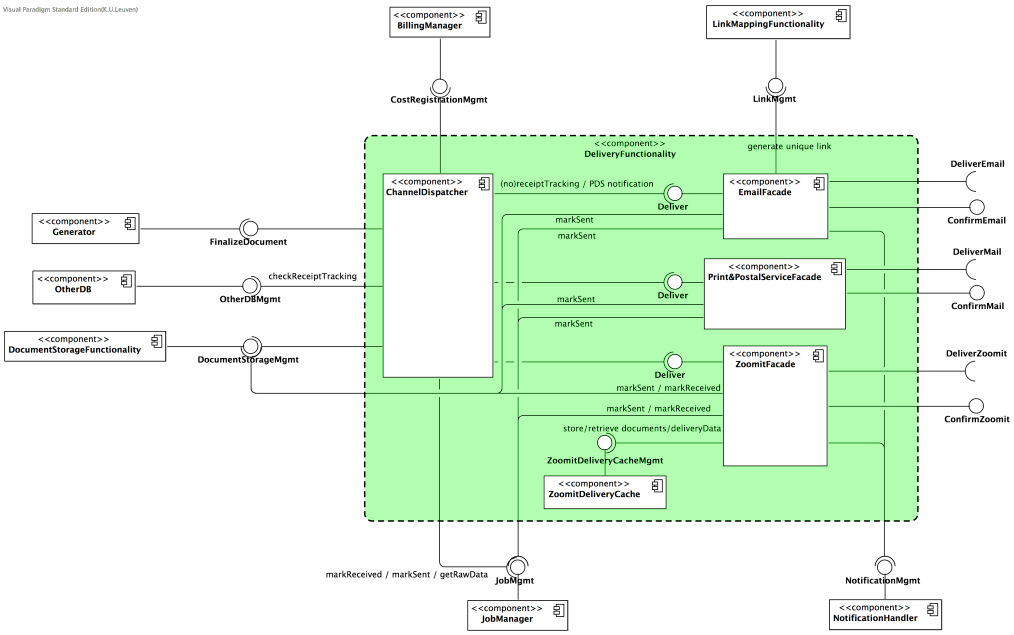


Figure 3: Decomposition of `DeliveryFunctionality`

Describe the decomposition of `ComponentX` and how this relates to the requirements.

## 4.2 DocumentDB

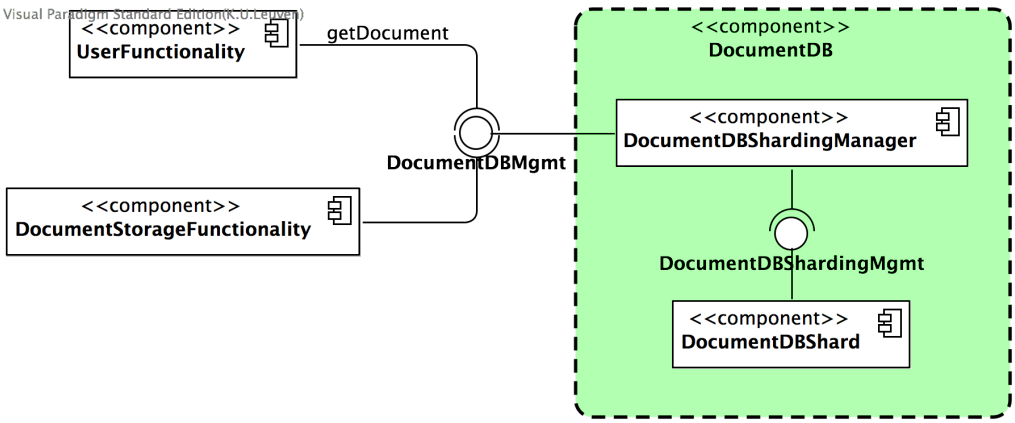


Figure 4: Decomposition of `DocumentDB`

## 4.3 DocumentGenerationManager

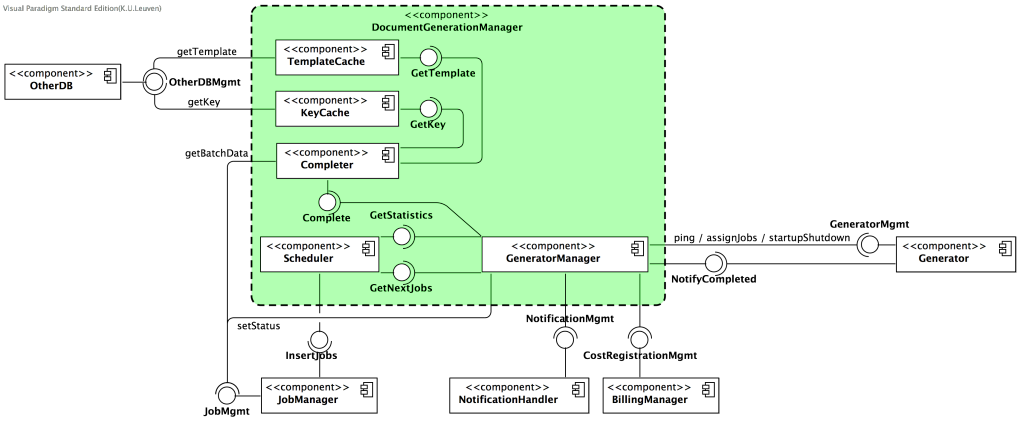


Figure 5: Decomposition of `DocumentGenerationManager`

## 4.4 DocumentStorageFunctionality

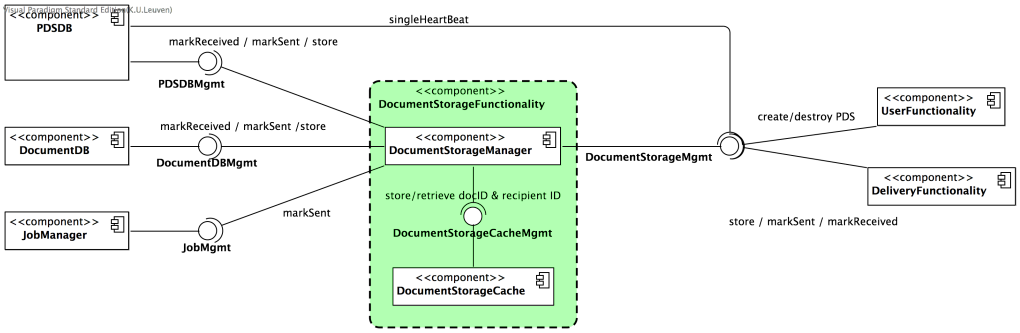


Figure 6: Decomposition of `DocumentStorageFunctionality`

## 4.5 JobManager

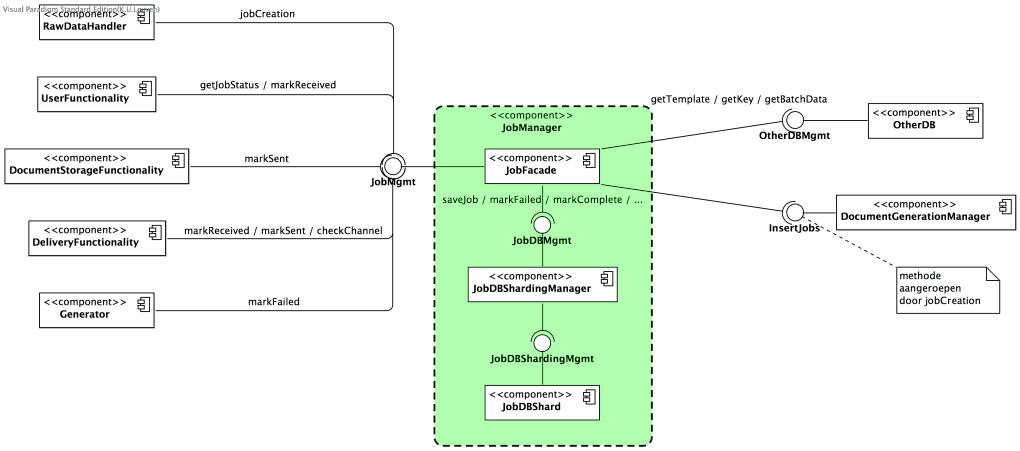


Figure 7: Decomposition of `JobManager`

## 4.6 LinkFunctionality

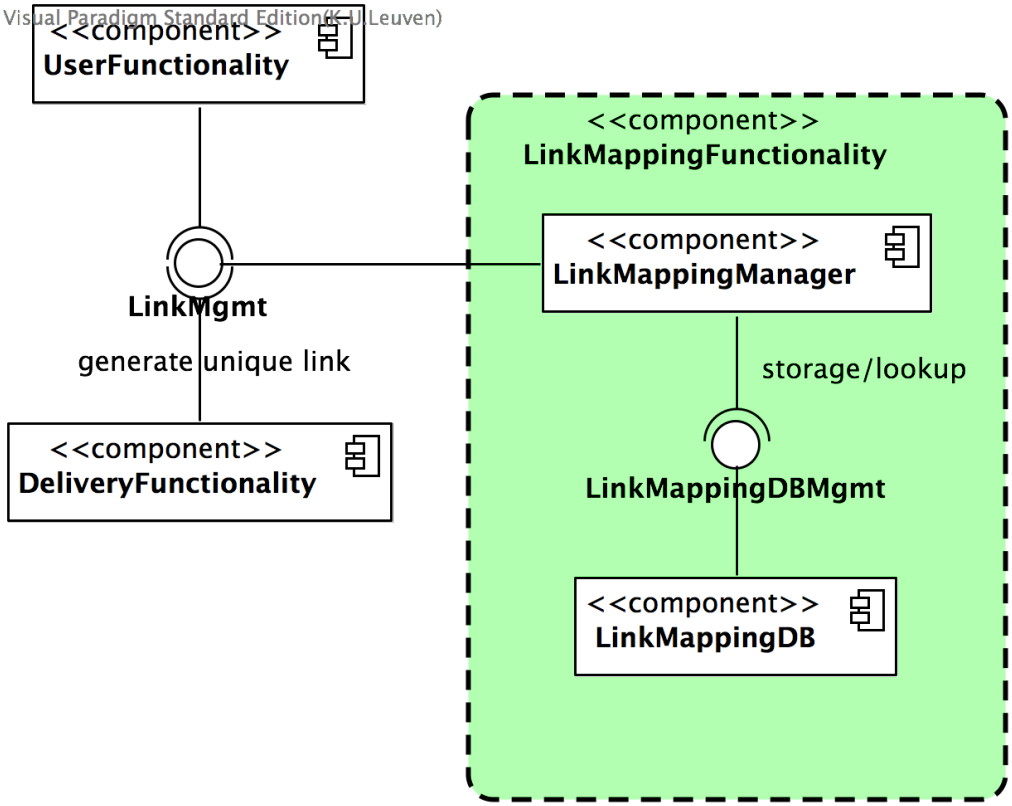


Figure 8: Decomposition of `LinkFunctionality`

## 4.7 PDSDB

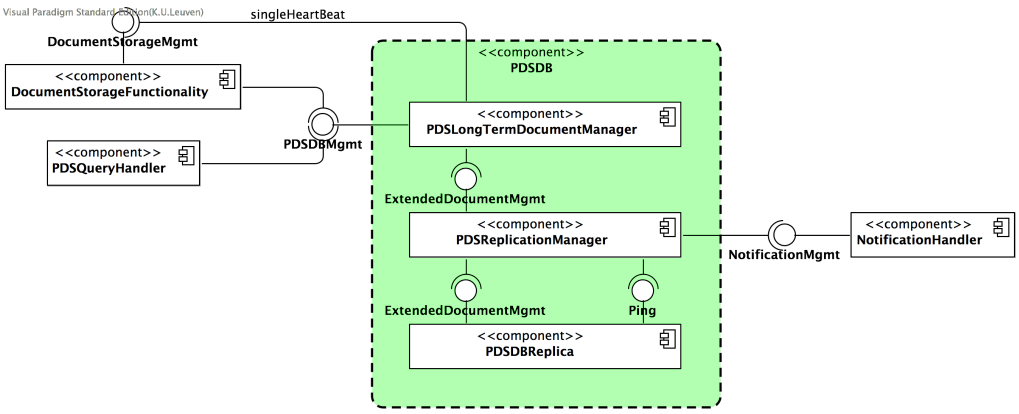


Figure 9: Decomposition of `PDSDB`

## 4.8 UserFunctionality

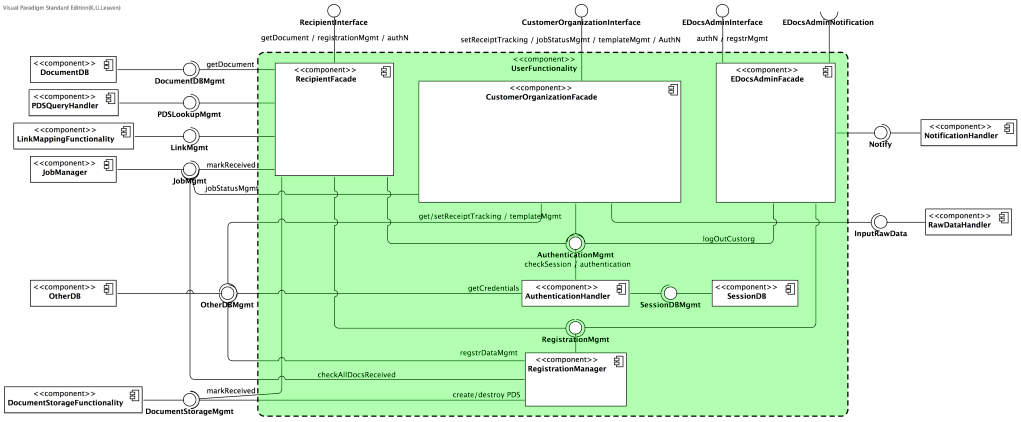


Figure 10: Decomposition of `UserFunctionality`

# 5 Deployment view (UML Deployment diagram)

Describe the context diagram for the deployment view. For example, which protocols are used for communication with external systems and why?

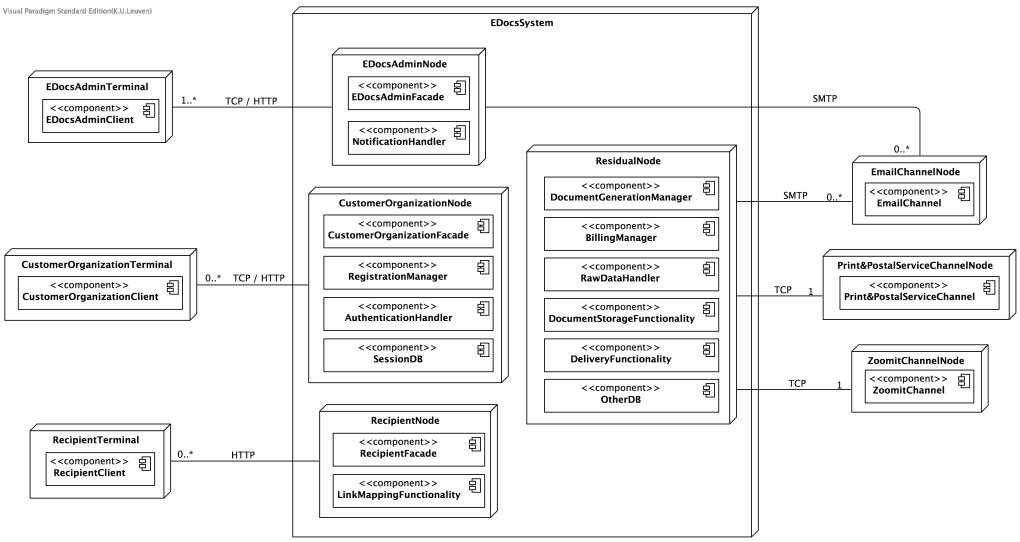


Figure 11: Context diagram for the deployment view.

The primary deployment diagram itself and accompanying explanation. Pay attention to the parts of the deployment diagram which are crucial for achieving certain non-functional requirements. Also discuss any alternative deployments that you considered.

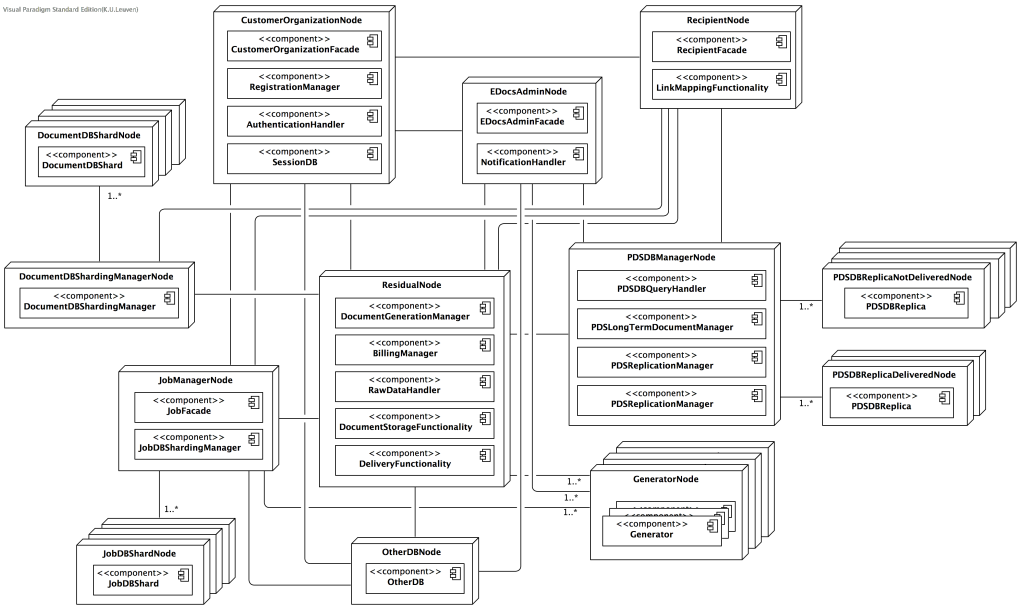


Figure 12: Primary diagram for the deployment view.

# 6 Scenarios

Illustrate how your architecture fulfills the most important data flows. As a rule of thumb, focus on the scenario of the domain description. Describe the scenario in terms of architectural components using UML Sequence diagrams and further explain the most important interactions in text. Illustrating the scenarios serves as a quick validation of the completeness of your architecture. If you notice at this point that for some reason, certain functionality or qualities are not addressed sufficiently in your architecture, it suffices to document this, together with a rationale of why this is the case according to you. You do not have to further refine you architecture at this point.

## 6.1 Scenario 1

Shortly describe the scenario shown in this subsection. Show the complete scenario using one or more sequence diagrams.

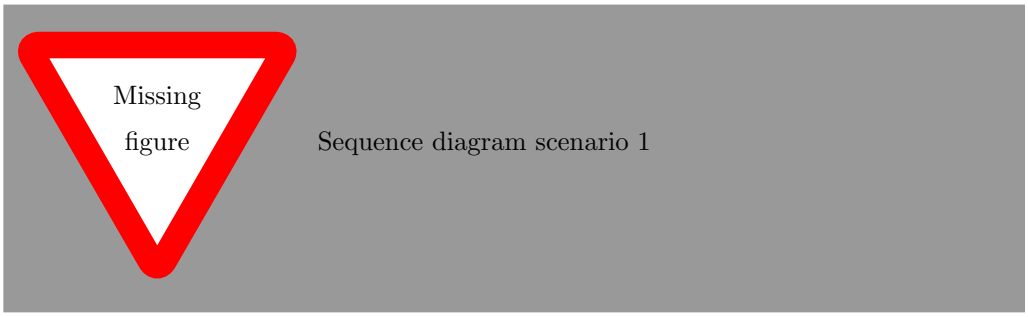


Figure 13: The system behavior for the first scenario.

## 6.2 A registered recipient logs in

Figure 14 depicts the sequence diagram of a Registered Recipient logging in, according to *UC1: Log in*. The Registered Recipient provides his details to the `RecipientFacade`. The `Authentication` compares these credentials to those stored in the `OtherDB`. if the given credentials match the stored credentials, a new session is opened in the `SessionDB` and the corresponding session identifier is returned to the registered recipient. Otherwise, an InvalidCredentialsException is thrown.

Note that the use case *UC1: Log in* also has the Customer Organization as a primary actor. The login procedure for customer organizations is almost identical to the procedure for registered recipient, which is why we do not provide a separate sequence diagram for this primary actor. Compared to figure 14, the Customer Organization sends it login request to the `CustomerOrganizationFacade` instead of the `RecipientFacade`. Also, another method call to `OtherDB` is used to ask for the credentials of a customer organization.

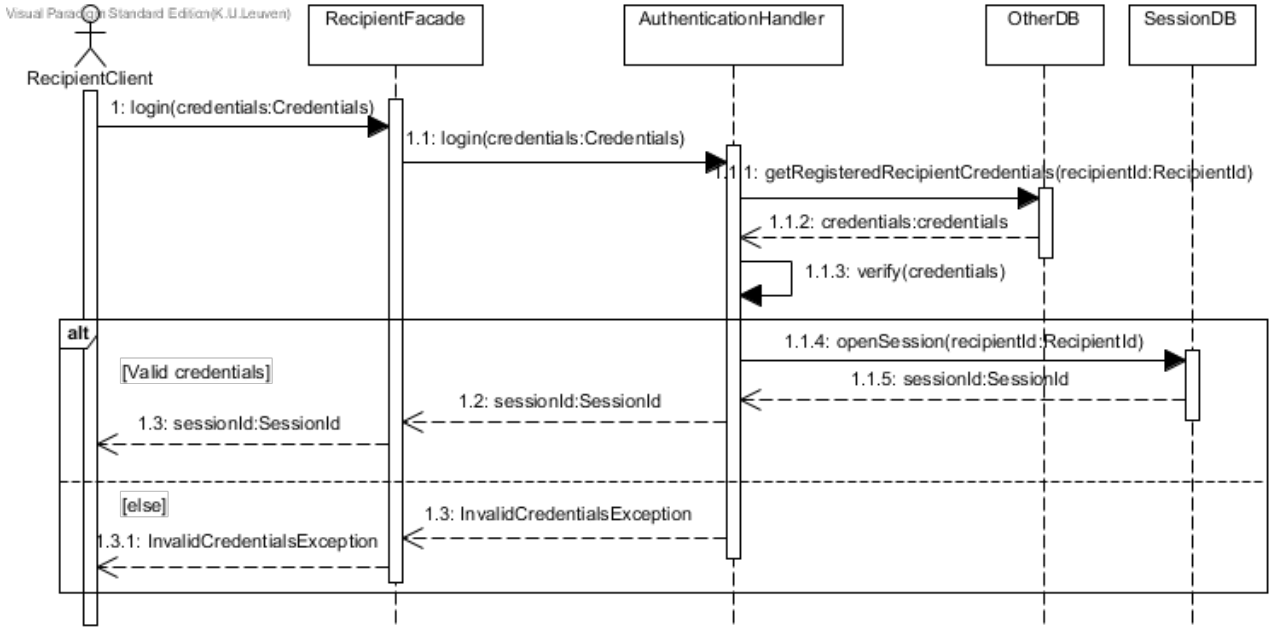


Figure 14: The login behaviour of a Registered Recipient.

## 6.3 A registered recipient logs out

Figure 15 shows the behaviour of a Registered Recipient logging out, according to *UC2: Log out*. The Registered Recipient provides his recipient id and session id to the `RecipienntFacade`. The `AuthenticationHandler` first verifies the session (figure 15). If the session is valid, it logs out the Registered Recipient.
Note that the logout procedure for Customer Organizations is almost the same. The only difference is that the Customer Organization sends his logout request to the `CustomerOrganizationFacade` instead of the `RecipientFacade`. Because of this reason, we do not give a separate sequence diagram for the Customer Organization logging out.

## 6.4 Updating a document template

Figure 16 shows the flow when a Customer Administrator updates the template used for generating documents of a given type. It corresponds to *UC20: Update document template*. It consists of two main parts. First, the customer administrator asks what the possible document types are that the customer organization is allowed to generate. The system verifies whether the session of the customer administrator is valid (as detailed in figure 17 and then returns a list of the allowed document types using the `OtherDB`. Together with the allowed document types, it returns the time when the currently stored templates corresponding to the document types were uploaded.
Secondly, the customer administrator provides the document type it wants to update and the new template to the `CustomerOrganizationFacade`. After verifying the validity of the session again (figure 17), the `CustomerOrganization` asks the `OtherDB` to check whether the document type provided by the customer organization is valid and allowed. If it is not allowed or invalid, an InvalidDocumentTypeException gets thrown. Otherwise, the `CustomerOrganizationFacade` stores the template in the `OtherDB` together with the time when it has received the template, the document type and the customer organization id.

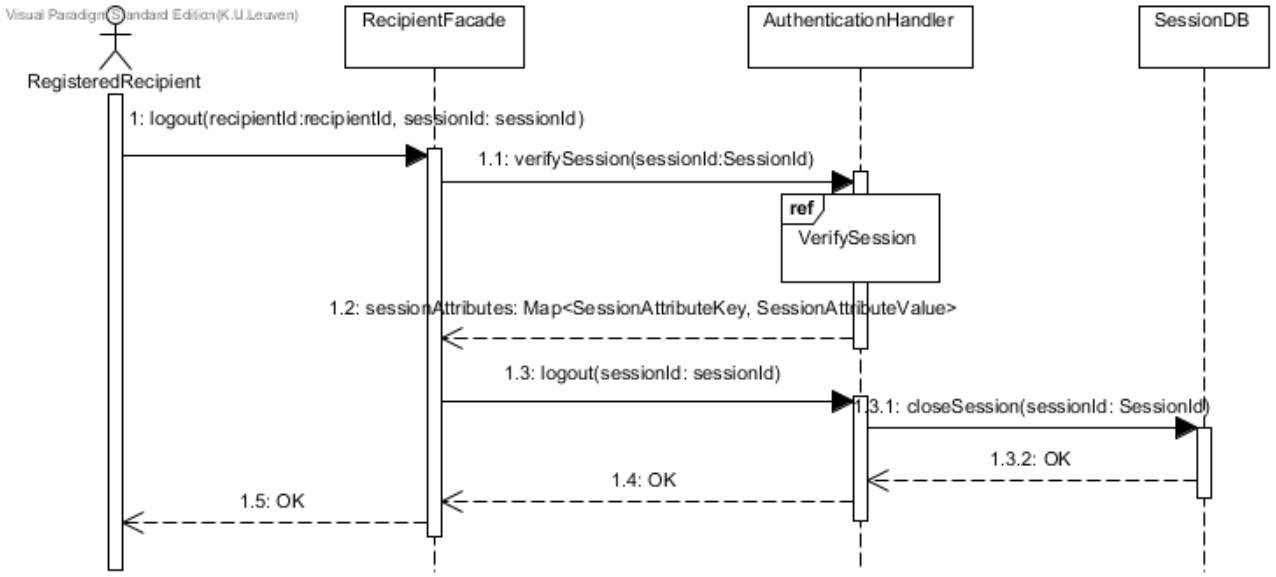


Figure 15: The system behavior for the first scenario.

## 6.5 Verifying a session

Figure 17 how it is verified whether a session is valid0 The `AuthenticationHandler` uses the `SessionDB` to verify whether each incoming session identifier belongs to an existing session. If it belongs to an existing session, the corresponding session attributes are returned to the caller, otherwise an NoSuchSessionException is thrown.

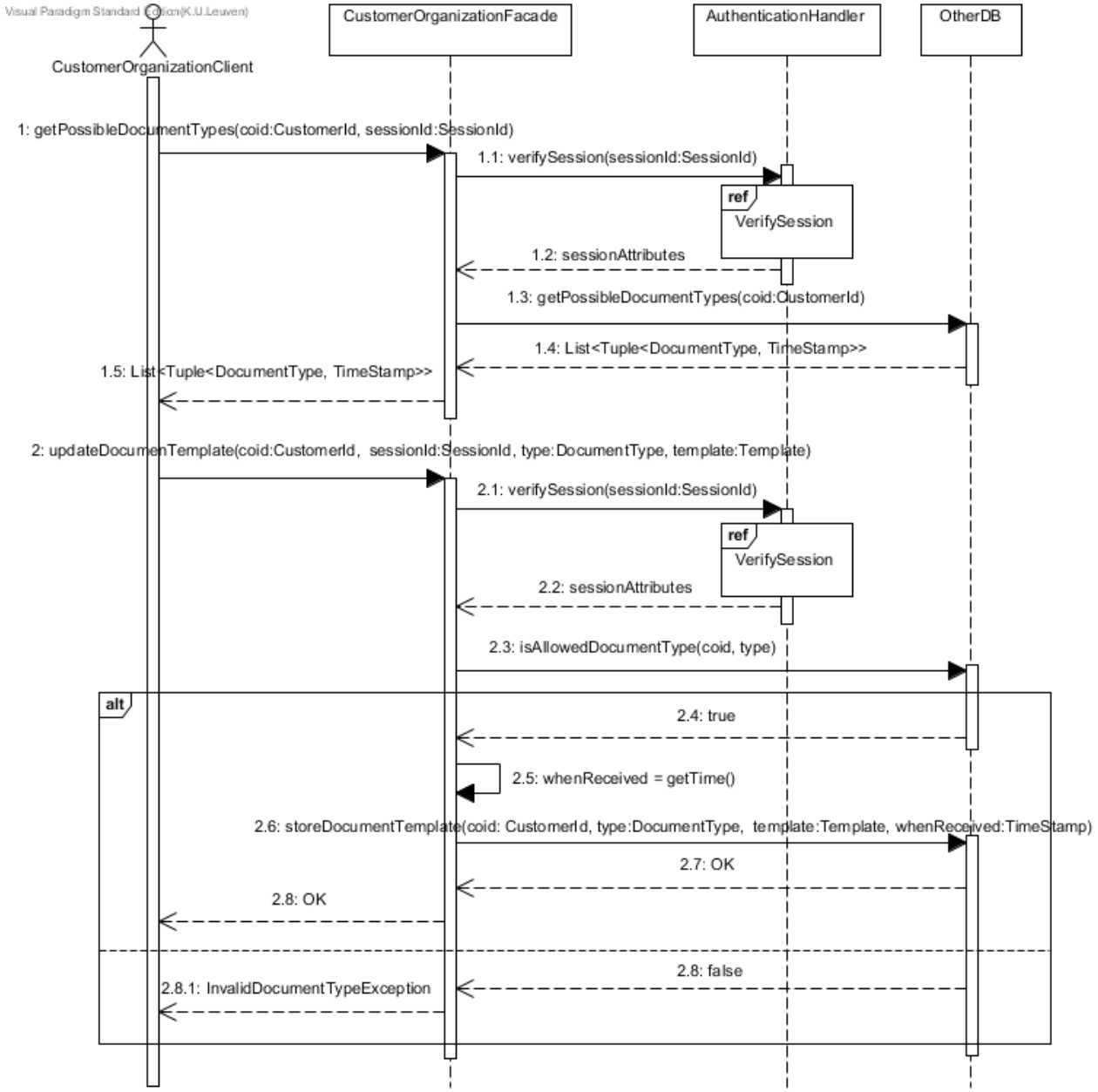


Figure 16: The system behavior for updating a document template.

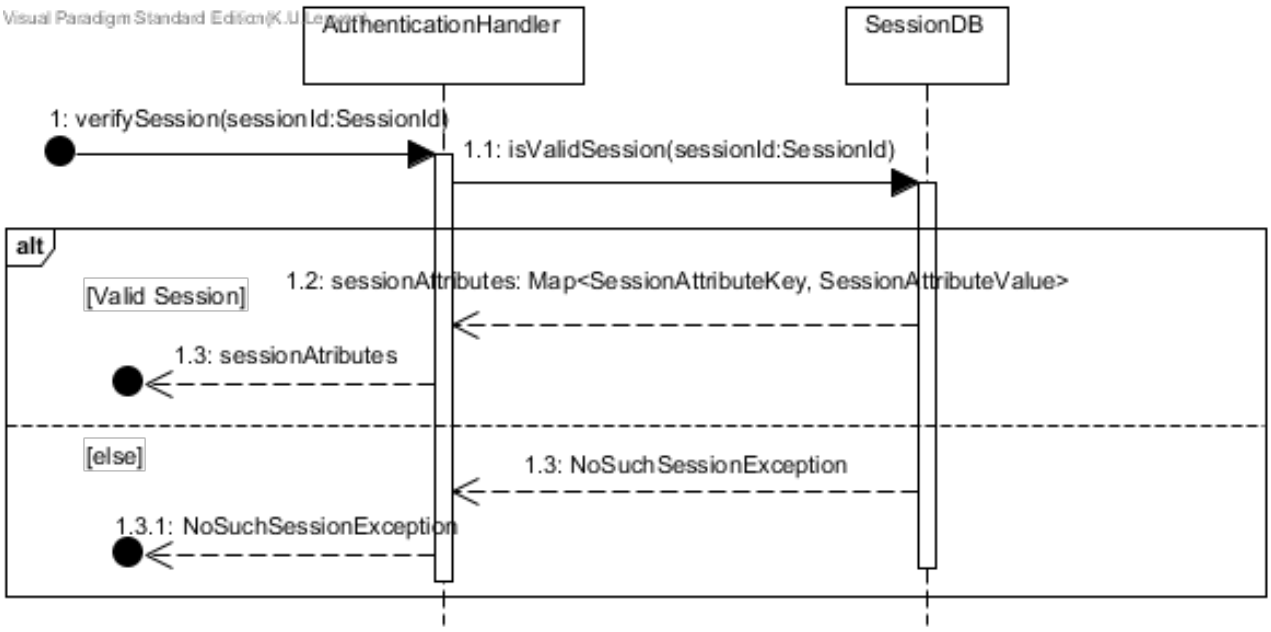


Figure 17: verifySession: sequence diagram depicting the verification whether a session is valid.

# A Element catalog

In this section, we list all the components and the interfaces they provide. Per component, we describe its responsibilities, declare its super-component (if any) and list its sub-components (if any).

List all components and describe their responsibilities and provided interfaces. Per interface, list all methods using a Java-like syntax and describe their effect and exceptions if any. List all elements and interfaces alphabetically for ease of navigation.

## A.1 AuthenticationHandler

- **Description:** The `AuthenticationHandler` is responsible for authenticating Registered recipients and Customer organizations. The architecture does not specify the means of authentication (e.g. the type of credentials). The credentials are stored in the `UserDB`.
- **Super-component:** `UserFunctionality`
- **Sub-components:** None

### Provided interfaces

- AuthN
  - `RecipientId getRecipientId(SessionId sessionId) throws NoSuchSessionException`
    - Effect: The `AuthenticationHandler` fetches and returns the Registered Recipient's identifier corresponding to the `sessionId` from the `SessionDB`.
    - Exceptions:
      - NoSuchSessionException: Thrown if no session exists with the given identifiers, or if the session belongs to a customer organization.
  - `CustomerId getCustomerId(SessionId sessionId) throws NoSuchSessionException`
    - Effect: The `AuthenticationHandler` fetches and returns the Customer Organization's identifier corresponding to the `sessionId` from the `SessionDB`.
    - Exceptions:
      - NoSuchSessionException: Thrown if no session exists with the given identifier, or if the session belongs to a registered recipient.

- `Boolean logout(SessionId sessionId)`
  - Effect: The `AuthenticationHandler` will remove the session with the given id from the `SessionDB`. If no such session exists, nothing is changed and no exception is thrown.
  - Exceptions: None
- `SessionId login(Credentials credentials) throws InvalidCredentialsException`
  - Effect: The `AuthenticationHandler` verifies the `credentials` using the `UserDB`. If they are correct, the `AuthenticationHandler` creates a new session using the `SessionDB`, stores the id of the user (i.e. the Registered Recipient id or Customer Organization id) as an attribute in this session and returns the id of the new session. The id of the user is present in the given credentials.
  - Exceptions:
    - InvalidCredentialsException: Thrown if the given credentials are invalid.

- CheckSession
  - `Map<SessionAttributeKey, SessionAttributeValue> verifySession(SessionId sesionId) throws NoSuchsSessionException`
    - Effect: The `AuthenticationHandler` verifies whether a session with the given id exists in the `SessionDB` and if so, returns all its associated attributes.
    - Exceptions:
      - NoSuchSessionException: Thrown if no session exists with the given identifiers.

## A.2 BillingManager

- **Description:** The `BillingManager` is responsible for all billing tasks. This includes billing the Customer Organization for the generation and delivery of non-recurring document processing jobs.
- **Super-component:** None
- **Sub-components:** None

**Provided interfaces**

- CostRegistrationMgmt
  - `void addDocumentGenerationCost(CustomerId cuId, JobId jobId, DocumentPriority priority)`
    - Effect: The `BillingManager` adds the cost for generation documents corresponding to the job identified by `jobId` to the bill of the customer organization identified by `cuId`. The document had a priority `priority`. This method is called by the `GenerationManager`.
    - Exceptions: None
  - `void addDocumentDeliveryCostbyEmail(CustomerId cuId, JobId jobId)`
    - Effect: The `BillingManager` adds the cost for delivering the document by e-mail to the bill of the customer organization identified by `cuId`. The document corresponds to the job identified by `JobId`. This method is called by the `ChannelDispatcher`.
    - Exceptions: None
  - `void addDocumentDeliveryCostbyPrintAndPostalService(CustomerId cuId, JobId jobId)`
    - Effect: The `BillingManager` adds the cost for delivering the document by e-mail to the bill of the customer organization identified by `cuId`. The document corresponds to the job identified by `JobId`. This method is called by the `ChannelDispatcher`.
    - Exceptions: None
  - `void addDocumentDeliveryCostbyZoomit(CustomerId cuId, JobId jobId)`
    - Effect: The `BillingManager` adds the cost for delivering the document by Zoomit to the bill of the customer organization identified by `cuId`. The document corresponds to the job identified by `JobId`. This method is called by the `ChannelDispatcher`.
    - Exceptions: None

## A.3 ChannelDispatcher

- **Description:** The `ChannelDispatcher` is responsible for choosing the correct delivery channel for a generated document. It also forwards the document to the `DocumentStorageManager`, which will store the document.
- **Super-component:** `Deliveryfunctionality`
- **Sub-components:** None

**Provided interfaces**

- FinalizeDocument
  Note that the methods in this interface are made idempotent. The methods of this interface are called by `Generator` instances.
  - `void storeAndDeliverDocument(JobId jobid, Document doc)`
    - Effect: The `ChannelDispatcher` will store the given document `document` and deliver it. This method is made idempotent. To filer duplicate method calls, it has the `JobId` of the document as an argument. This idempotence is to account for the case when case when a `Generator` fails after forwarding the document and before reporting completion to the `DocumentGenerationManager`. In this case, it can be that the `DocumentGenerationManager` restarts jobs for which a document has already been stored or delivered.
    - Exceptions: None
  - `void generationError(JobId jobid, Error error)`
    - Effect: Describe the effect of the operation
    - Exceptions: None

## A.4 CustomerOrganizationClient

- **Description:** The `CustomerOrganizationClient` is external to the eDocs system and represents a client device of a Customer Organization (i.e. Customer Administrator and Customer Information System) that communicates with the eDocs System.
- **Super-component:** None
- **Sub-components:** None

**Provided interfaces**

- InterfaceA
  - `returntType1 operation1(ParamType param) throws SomeException`
    - Effect: Describe the effect of the operation
    - Exceptions:
      - SomeException: Describe when the exception is thrown.
  - `void operation2(ParamType2 param)`
    - Effect: Describe the effect of the operation
    - Exceptions: None

## A.5 CustomerOrganizationFacade

- **Description:** The `CustomerOrganizationFacade` provides the main interface of the system to the Customer Organization (i.e. Customer Administrator and Customer Information System).
- **Super-component:** `UserFunctionality`
- **Sub-components:** None

**Provided interfaces**

- AuthN
  - `Boolean logout(SessionId sessionId)`
    - Effect: The `AuthenticationHandler` will remove the session with the given id from the `SessionDB`. If no such session exists, nothing is changed and no exception is thrown.
    - Exceptions: None
  - `SessionId login(Credentials credentials) throws InvalidCredentialsException`
    - Effect: The `AuthenticationHandler` verifies the `credentials` using the `UserDB`. If they are correct, the `AuthenticationHandler` creates a new session using the `SessionDB`, stores the id of the user (i.e. the Registered Recipient id or Customer Organization id) as an attribute in this session and returns the id of the new session. The id of the user is present in the given credentials.
    - Exceptions:
      - InvalidCredentialsException: Thrown if the given credentials are invalid.
- TemplateMgmt
  - `List<Tuple<DocumentType,TimeStamp>> getPossibleDocumentTypes(SessionId sessionId, CustomerId cuId) throws NotAuthenticatedException`
    - Effect: The `CustomerOrganizationClient` will return a list of the document types that the customer organization is allowed to generate. It also indicates the current template for each document type by returning for each document type the time when the last template was uploaded.
    - Exceptions:
      - NotAuthenticatedException: Thrown if the given session identifier is invalid.
  - `Boolean updateDocumentTemplate(SessionId sessionId, CustomerId cuId, DocumentType documentTyp Template template) throws NotAuthenticatedException, InvalidDocumentTypeException`
    - Effect: The `CustomerOrganizationClient` will update the current template for the given `documentType` to the given `template` for the customer organization identified by `cuId`. Returns true when it succeeds.
    - Exceptions:
      - NotAuthenticatedException: Thrown if the given session identifier is invalid.
      - InvalidDocumentTypeException: Thrown if the given document type is invalid or not allowed for the given customer organization.

## A.6 Completer

- **Description:** The `Completer` is responsible for fetching the raw data an applicable meta-data for a group of `JobIds` when a `Generator` instance requires a new group of jobs.
- **Super-component:** `DocumentGenerationManager`
- **Sub-components:** None

**Provided interfaces**

- Complete
  - `CompletePartialBatchData getComplete(BatchId batchId, List<JobId> jobIds )`
    - Effect: The `Completer` fetches data needed by a `Generator` for generation of the documents corrseponding to the`JobIds` belonging to the same batch, which is identified by `BatchId`.
    - Exceptions: None

## A.7 DocumentDB

- **Description:** The `DocumentDB` is responsible for actually storing all the documents. It stores documents regardless of the fact of a document is also stored in the `PDSDB` It receives read and write requests from the `DocumentStoragManager`.
- **Super-component:** None
- **Sub-components:** `DocumentDBShardingManager` and `DocumentDBShard`

**Provided interfaces**

- InterfaceA
  - `returntType1 operation1(ParamType param) throws SomeException`
    - Effect: Describe the effect of the operation
    - Exceptions:
      - SomeException: Describe when the exception is thrown.
  - `void operation2(ParamType2 param)`
    - Effect: Describe the effect of the operation
    - Exceptions: None
- InterfaceB
  - `returntType2 operation3()`
    - Effect: Describe the effect of the operation
    - Exceptions: None

## A.8 DocumentDBShard

- **Description:** A `DocumentDBShard` is responsible for storing a partition of all the documents.
- **Super-component:** `DocumentDB`
- **Sub-components:** None

**Provided interfaces**

- InterfaceA
  - `returntType1 operation1(ParamType param) throws SomeException`
    - Effect: Describe the effect of the operation
    - Exceptions:
      - SomeException: Describe when the exception is thrown.
  - `void operation2(ParamType2 param)`
    - Effect: Describe the effect of the operation
    - Exceptions: None
- InterfaceB
  - `returntType2 operation3()`
    - Effect: Describe the effect of the operation
    - Exceptions: None

## A.9 DocumentDBShardingManager

- **Description:** The `DocumentDBShardingManager` manages the storage of the documents over multiple `DocumentDBShards`.
- **Super-component:** The `DocumentDB`
- **Sub-components:** None

**Provided interfaces**

- InterfaceA
  - `returntType1 operation1(ParamType param) throws SomeException`
    - Effect: Describe the effect of the operation
    - Exceptions:
      - SomeException: Describe when the exception is thrown.
  - `void operation2(ParamType2 param)`
    - Effect: Describe the effect of the operation
    - Exceptions: None
- InterfaceB
  - `returntType2 operation3()`
    - Effect: Describe the effect of the operation
    - Exceptions: None

## A.10 DocumentGenerationManager

- **Description:** The `DocumentGenerationManager` monitors the availability of the `Generator` components using the Ping interface. The `DocumentGenerationManager` keeps track of the jobs assigned to and being processed by the `Generators`. To minimize the overhead of the job coordination, the `DocumentGenerationManager` assigns jobs to the `Generators` in groups of more than one job that are part of the same batch. If a `Generator` fails to complete its jobs, the `DocumentGenerationManager` can restart these failed jobs.
  It prioritizes jobs based on thei deadlines ansd schedules them according to *P1*.
- **Super-component:** None
- **Sub-components:** `Completer`, `GenerationManager`, `KeyCache`, `Scheduler`, `TemplateCache`

**Provided interfaces**

- InsertJobs
  - `returntType1 operation1(ParamType param) throws SomeException`
    - Effect: Describe the effect of the operation
    - Exceptions:
      - SomeException: Describe when the exception is thrown.
- NotifyCompleted
  - `void notifyCompletedAndGiveMeMore(GeneratorId id)`
    - Effect: The `DocumentGenerationManager` gets notified that the document processing jobs assigned to the `Generator` identified by an `id` are completed.
    - Exceptions: None
  - `void notifyCompletedAndIAmShuttingDown(GeneratorId id)`
    - Effect: The `DocumentGenerationManager` gets notified that the document processing jobs assigned to the `Generator` identified by an `id` are completed.
    - Exceptions: None

## A.11 DocumentStorageCache

- **Description:** The `DocumentStorageCache` is responsible for storing the `DocumentIds` and `UserIds` when the `PDSDB` fails. According to Av2, the system should temporarily store at least 3 hours of documents to be delivered via the personal document store. When the `PDSDB` fails, the documents that are supposed to also be saved in the `PDSDB` are saved in the `DocumentDB`, just as usual. But in this case the `DocumentStorageManager` also stores the `DocumentIds` and `UserIds` of those documents in the `DocumentStorageCache` for at least 3 hours. This way, the `DocumentStorageManager` can transfer these documents from the `DocumentDB` to the `PDSDB` using this information if the `PDSDB` comes back online within 3 hours. The requirements do not specify what happens after 3 hours, so in this architecture, the behaviour after those 3 hours is undefined.
- **Super-component:** `DocumentStorageFunctionality`
- **Sub-components:** None

**Provided interfaces**

- InterfaceA
  - `returntType1 operation1(ParamType param) throws SomeException`
    - Effect: Describe the effect of the operation
    - Exceptions:
      - SomeException: Describe when the exception is thrown.
  - `void operation2(ParamType2 param)`
    - Effect: Describe the effect of the operation
    - Exceptions: None
- InterfaceB
  - `returntType2 operation3()`
    - Effect: Describe the effect of the operation
    - Exceptions: None

## A.12 DocumentStorageFunctionality

- **Description:** The `DocumentStorageManager` is responsible for storing the generated documents in the correct database. If the document belongs to a registered recipient, it sends a write request to both the `DocumentDB` and the `PDSDB`. Otherwise, it only sends a write request to the `DocumentDB`, which stores all the documents.
  It is also responsible for copying documents from the `DocumentDB` to the `PDSDB` when an Unregistered Recipient registers to the eDocs system.
  Another responsibility of the `DeliveryFunctionality` is storing at least 3 hours of documents when the `PDSDB` fails.
- **Super-component:** None
- **Sub-components:** `DocumentStorageCache` and `DocumentStorageManager`

**Provided interfaces**

- InterfaceA
  - `returntType1 operation1(ParamType param) throws SomeException`
    - Effect: Describe the effect of the operation
    - Exceptions:
      - SomeException: Describe when the exception is thrown.
  - `void operation2(ParamType2 param)`
    - Effect: Describe the effect of the operation

        * Exceptions: None

- InterfaceB
  - `returntType2 operation3()`
    * Effect: Describe the effect of the operation
    * Exceptions: None

## A.13 DocumentStorageManager

- **Description:** The `DocumentStorageManager` is responsible for storing the generated documents in the correct database. If the document belongs to a registered recipient, it sends a write request to both the `DocumentDB` and the `PDSDB`. Otherwise, it only sends a write request to the `DocumentDB`, which stores all the documents.
  It is also responsible for copying documents from the `DocumentDB` to the `PDSDB` when an Unregistered Recipient registers to the eDocs system.
  Another responsibility of the `DocumentStorageManager` is storing at least 3 hours of documents when the `PDSDB` fails.
- **Super-component:** `DocumentStoragefunctionlity`
- **Sub-components:** None

**Provided interfaces**

- InterfaceA
  - `returntType1 operation1(ParamType param) throws SomeException`
    * Effect: Describe the effect of the operation
    * Exceptions:
      · SomeException: Describe when the exception is thrown.
- InterfaceB
  - `returntType2 operation3()`
    * Effect: Describe the effect of the operation
    * Exceptions: None

## A.14 DeliveryFunctionality

- **Description:** The `DeliveryFunctionality` is responsible for delivering the documents generated by eDocs to the recipients.
- **Super-component:** `ChannelDispatcher`, `EmailFacade`, `Print&PostalServiceFacade`, `ZoomitFacade` and `ZoomitDeliveryCache`
- **Sub-components:** the direct sub-components, if any.

**Provided interfaces**

- InterfaceA
  - `returntType1 operation1(ParamType param) throws SomeException`
    * Effect: Describe the effect of the operation
    * Exceptions:
      · SomeException: Describe when the exception is thrown.
- InterfaceB
  - `returntType2 operation3()`
    * Effect: Describe the effect of the operation
    * Exceptions: None

## A.15 EDocsAdminClient

- **Description:** The `EDocsAdminClient` is external to the eDocs system and represents a client device of an administrator of eDocs that communicates with the eDocs System.
- **Super-component:** None
- **Sub-components:** None

**Provided interfaces**

- InterfaceA
  - `returntType1 operation1(ParamType param) throws SomeException`
    - Effect: Describe the effect of the operation
    - Exceptions:
      - SomeException: Describe when the exception is thrown.
  - `void operation2(ParamType2 param)`
    - Effect: Describe the effect of the operation
    - Exceptions: None
- InterfaceB
  - `returntType2 operation3()`
    - Effect: Describe the effect of the operation
    - Exceptions: None

## A.16 EmailChannel

- **Description:** The `EmailChannel` is responsible for the delivery emails. It is external to the eDocs system and represents a mail server of an e-mail provider.
- **Super-component:** None
- **Sub-components:** None.

**Provided interfaces**

- InterfaceA
  - `returntType1 operation1(ParamType param) throws SomeException`
    - Effect: Describe the effect of the operation
    - Exceptions:
      - SomeException: Describe when the exception is thrown.
  - `void operation2(ParamType2 param)`
    - Effect: Describe the effect of the operation
    - Exceptions: None
- InterfaceB
  - `returntType2 operation3()`
    - Effect: Describe the effect of the operation
    - Exceptions: None

## A.17 EmailFacade

- **Description:** The `EmailFacade` is responsible for creating and sending emails used in the delivery of documents. It will send documents to Unregistered recipients by e-mail when receipt tracking is turned off. When receipt tracking is turned on for an Unregistered Recipient, it will send an e-mail containing a short description of the received document and a unique link, which can be followed to get document. For Registered Recipients, it will send an e-mail containing a short description of the document and a link to the document. It also marks jobs as sent using the `JobManager`.
- **Super-component:** `DeliveryFunctionality`
- **Sub-components:** None

**Provided interfaces**

- InterfaceA
  - `returntType1 operation1(ParamType param) throws SomeException`
    - Effect: Describe the effect of the operation
    - Exceptions:
      - SomeException: Describe when the exception is thrown.
- InterfaceB
  - `returntType2 operation3()`
    - Effect: Describe the effect of the operation
    - Exceptions: None

## A.18 Generator

- **Description:** A `Generator` generates the documents and forwards them to `DeliveryFunctionality` to store and deliver them. Its availability is monitored by the `DocumenGenerationManager` with the Ping interface. A `Generator` is also responsible of notifying the `NotificationHandler` that it does not have all of the data required to fill in the template.
- **Super-component:** None
- **Sub-components:** None

**Provided interfaces**

- AssignJobs
  - `void assignJobs(CompletePartialBatchData batchData)`
    - Effect: Describe the effect of the operation
    - Exceptions:
      - SomeException: Describe when the exception is thrown.
- Startup/ShutDown
  - `void startUp(GeneratorId generatorId)`
    - Effect: Starts up the `Generator` instance an gives it the given `GeneratorId`.
    - Exceptions: None
  - `void shutDown()`
    - Effect: The `Generator` completes its assigned group of document generation jobs and report back completion to the `DocumentGenerationManager`, after which is shuts down.
    - Exceptions: None
- Ping
  - `Echo ping()`
    - Effect: The `Generator` will respond to the ping request by sending an echo response. This is used by the `GeneratorManager` to check whether the `Generator` is available.
    - Exceptions: None

## A.19 GeneratorManager

- **Description:** The `GenerationManager` is responsible for monitoring the `Generator` instances. It starts up or shuts down these instances based on the number of required instances indicated by the `Scheduler`.
- **Super-component:** `DocumentGenerationManager`
- **Sub-components:** None

**Provided interfaces**

- NotifyCompleted
  - `void notifyCompletedAndGiveMeMore(GeneratorId id)`
    - Effect: The `DocumentGenerationManager` gets notified that the document processing jobs assigned to the `Generator` identified by an `id` are completed.
    - Exceptions: None

## A.20 JobDBShard

- **Description:** The `JobDBShard` is responsible for storing partition of all the jobs.
- **Super-component:** `JobManager`
- **Sub-components:** None

**Provided interfaces**

- InterfaceA
  - `returntType1 operation1(ParamType param) throws SomeException`
    - Effect: Describe the effect of the operation
    - Exceptions:
      - SomeException: Describe when the exception is thrown.
  - `void operation2(ParamType2 param)`
    - Effect: Describe the effect of the operation
    - Exceptions: None
- InterfaceB
  - `returntType2 operation3()`
    - Effect: Describe the effect of the operation
    - Exceptions: None

## A.21 JobDBShardingManager

- **Description:** The `JobDBShardingManager` manages the storage of the documents over multiple `JobDBShards`.
- **Super-component:** `JobManager`
- **Sub-components:** None

**Provided interfaces**

- InterfaceA
  - `returntType1 operation1(ParamType param) throws SomeException`
    - Effect: Describe the effect of the operation
    - Exceptions:
      - SomeException: Describe when the exception is thrown.
  - `void operation2(ParamType2 param)`
    - Effect: Describe the effect of the operation
    - Exceptions: None
- InterfaceB
  - `returntType2 operation3()`
    - Effect: Describe the effect of the operation
    - Exceptions: None

## A.22 JobFacade

- **Description:** The `JobFacade` is responsible creating jobs and storing them using the `JobDBShadingManager` over different `JobDBShards`. It is responsible for retrieving data connected to a specific job. It can retrieve the raw data or customer organization info for specific jobs. The `JobFacade` is also used for marking jobs as sent and received.
- **Super-component:** `JobManager`
- **Sub-components:** None

**Provided interfaces**

- SetStatus
  - `void setJobStatusAsTemporarilyFailed(List<JobId> statusesOfJobs)`
    - Effect: The `JobManager` marks the job as "temporarily failed" for each of the jobs identified by the given `JobIds`. Used by the `DocumentGenerationManager` for jobs that where assigned to a failed `Generator` instance.
    - Exceptions: None
- GetBatchData
  - `Tuple<JobId, RawData> getRawData(List<JobId> jobIds)`
    - Effect: Describe the effect of the operation
    - Exceptions: None
  - `BatchMetaData getMetaData(BatchId batchId)`
    - Effect: Describe the effect of the operation
    - Exceptions: None

## A.23 JobManager

- **Description:** The `JobManager` is responsible creating jobs and storing them. It is responsible for retrieving data connected to a specific job. It can retrieve the raw data or customer organization info for specific jobs. The `JobFacade` is also used for marking jobs as sent and received.
- **Super-component:** None
- **Sub-components:** `JobFacade`, `JobDBShardingManager` and `JobDBShard`

**Provided interfaces**

- SetStatus
  - `void setJobStatusAsTemporarilyFailed(List<JobId> statusesOfJobs)`
    - Effect: The `JobManager` marks the job as "temporarily failed" for each of the jobs identified by the given `JobIds`. Used by the `DocumentGenerationManager` for jobs that where assigned to a failed `Generator` instance.
    - Exceptions: None
- GetBatchData
  - `Tuple<JobId, RawData> getRawData(List<JobId> jobIds)`
    - Effect: Describe the effect of the operation
    - Exceptions: None
  - `BatchMetaData getMetaData(BatchId batchId)`
    - Effect: Describe the effect of the operation
    - Exceptions: None

## A.24 KeyCache

- **Description:** The `KeyCache` caches the keys which are most recently used for document generation. The `Completer` has to fetch a key every time a `Generator` instance requests new jobs, while the key will be the same for all jobs belonging to the same batch. The `KeyCache` avoids that the key storage system becomes a bottleneck for document generations. The keys are cached based on the `CustomerId` of a Customer Organization.
- **Super-component:** `DocumentGenerationManager`.
- **Sub-components:** None

**Provided interfaces**

- GetKey
  - `Key getKey(CustomerId customerId) throws NoSuchKeyException`
    - Effect: The `KeyCache` looks into its cache for the `Key` belonging to the customer organisation with id `customerId`. If the `Key` is in its cache, it returns it. If the `Key` is not in its cache, it asks `OtherDB` for the `Key` and stores it in its cache, after which it returns that `Key`.
    - Exceptions:
      - NoSuchKeyException: Thrown if there is no key for the given `customerId`.

## A.25 LinkMappingDB

- **Description:** The `LinkMappingDB` is responsible for actually storing a mapping between unique links and `DocumentIds`. With this information, it also stores information about where a document can be found, i.e. only in the `DocumentDB` or both in the `PDSDB` and the `DocumentDB`.
- **Super-component:** `LinkMappingFunctionality`
- **Sub-components:** None

**Provided interfaces**

- InterfaceA
  - `returntType1 operation1(ParamType param) throws SomeException`
    - Effect: Describe the effect of the operation
    - Exceptions:
      - SomeException: Describe when the exception is thrown.

  - `void operation2(ParamType2 param)`
    - Effect: Describe the effect of the operation
    - Exceptions: None

- InterfaceB
  - `returntType2 operation3()`
    - Effect: Describe the effect of the operation
    - Exceptions: None

## A.26 LinkMappingManager

- **Description:** The `LinkMappingManager` is responsible for creating unique links which points to a document. It also sends read and write requests to the `LinkMappingDB` to get and store the mappings between the unique links and the documents.
- **Super-component:** `LinkMappingFunctionality`
- **Sub-components:** None

**Provided interfaces**

- InterfaceA
  - `returntType1 operation1(ParamType param) throws SomeException`
    - Effect: Describe the effect of the operation
    - Exceptions:
      - SomeException: Describe when the exception is thrown.
- InterfaceB
  - `returntType2 operation3()`
    - Effect: Describe the effect of the operation
    - Exceptions: None

## A.27 LinkMappingFunctionality

- **Description:** The `LinkMappingFunctionality` is responsible for creating unique links which point to documents. It is also responsible for storing these mappings and ultimately mapping a link to a document.
- **Super-component:** None
- **Sub-components:** `LinkMappingManager` and `LinkMappingFunctionality`

**Provided interfaces**

- InterfaceA
  - `returntType1 operation1(ParamType param) throws SomeException`
    - Effect: Describe the effect of the operation
    - Exceptions:
      - SomeException: Describe when the exception is thrown.
  - `void operation2(ParamType2 param)`
    - Effect: Describe the effect of the operation
    - Exceptions: None
- InterfaceB
  - `returntType2 operation3()`
    - Effect: Describe the effect of the operation
    - Exceptions: None

## A.28 PDSDB

- **Description:** The `PDSDB` component is responsible for storing the database of documents in the personal document stores. That database is separated from all other persistent data so that its failure *"does not affect the availability of other types of persistent data"*, as required by *Av2*.
- **Super-component:** None
- **Sub-components:** `PDSDBReplica`, `PDSLongTermDocumentManager`, `PDSReplicationManager`

**Provided interfaces**

- DocumentMgmt
  - `Tuple<Document, MetaData> getDocument(DocumentId id)`
    - Effect: The `PDSDB` will fetch and return the document corresponding to `DocumentId` id.
    - Exceptions: None
  - `List<Document> getAllDocumentMetaDataOf(RecipientId recipientId)`
    - Effect: The `PDSDB` will fetch and return all the meta-data of the documents belonging to the Registered Recipient identified by `recipientId`.
    - Exceptions: None
  - `void storeDocument(DocumentId id, Document doc, DocumentMetaData md)`
    - Effect: The `PDSDB` will store the given document`doc` together with the provided meta-data `md`.
    - Exceptions: None
  - `void storeDocuments(List<Tuple<DocumentId, Document, DocumentMetaData>> documentList)`
    - Effect: Describe the effect of the operation
    - Exceptions: None
  - `List<Tuple<DocumentId, DocumentMetaData>> getAllDocumentMetaData(RecipientId recipientId) throws PDSUnavailableException`
    - Effect: The `PDSDB` fetches and returns the meta-data of all the documents of the Registered Recipient identified by `recipientId`.
    - Exceptions:
      - PDSUnavailableException: Thrown if the personal document store is unavailable.

## A.29 PDSDBReplica

- **Description:** The `PDSDBReplica` is responsible for actually storing the documents.
- **Super-component:** `PDSDB`
- **Sub-components:** None

**Provided interfaces**

- ExtendedDocumentMgmt
  - `List<Document> getAllDocumentsOf(TimeStamp whenFailed)`
    - Effect: Describe the effect of the operation
    - Exceptions:
      - SomeException: Describe when the exception is thrown.
  - `List<Document> getDocumentsSince(TimeStamp whenFailed)`
    - Effect: Describe the effect of the operation
    - Exceptions:
      - SomeException: Describe when the exception is thrown.
  - `void storeDocuments(List<Tuple<DocumentId, Document, DocumentMetaData>> documentList)`

    - Effect: The PDSDBReplica will store the documents and their meta-data.
    - Exceptions: None
  - `void storeDocument(DocumentId documentId, Document document, DocumentMetaData md)`
    - Effect: The PDSDBReplica stores the given document with its DocumentId and meta-data.
    - Exceptions: None
  - `Tuple<Document, MetaData> getDocument(DocumentId id)`
    - Effect: The PDSDB will fetch and return the document corresponding to DocumentId id.
    - Exceptions: None
  - `List<Tuple<DocumentId, DocumentMetaData>> getAllDocumentMetaData(RecipientId recipientId) throws PDSUnavailableException`
    - Effect: The PDSDBReplica fetches and returns the meta-data of all the documents of the Registered Recipient identified by recipientId.
    - Exceptions: None

- Ping
  - `Echo ping()`
    - Effect: The PDSDBReplica will respond to the ping request by sending an echo response. This is used by the PDSReplicationManager to check whether the PDSDBReplica is available.
    - Exceptions: None

## A.30 PDSFacade

- **Description:** Responsibilities of the component.
- **Super-component:** The direct super-component, if any.
- **Sub-components:** the direct sub-components, if any.

**Provided interfaces**

- PDSDBDocMgmt
  - `List<Tuple<DocumentId, DocumentMetaData>> getAllDocumentMetaData(RecipientId recipientId) throws PDSUnavailableException`
    - Effect: The PDSFacade fetches and returns the meta-data of all the documents of the Registered Recipient identified by recipientId.
    - Exceptions:
      - PDSUnavailableException: Thrown if the personal document store is unavailable.
  - `void operation2(ParamType2 param)`
    - Effect: Describe the effect of the operation
    - Exceptions: None

## A.31 PDSLongTermDocumentManager

- **Description:** The PDSLongTermDocumentManager is responsible for managing the different storage clusters. Each cluster consists of a PDSReplicationManager and one or more PDSDBReplica instances. In the architecture, two clusters are defined. The PDSLongTermDocumentManager reads to and write from clusters, and periodically transfers documents from the one cluster to the other.
- **Super-component:** PDSDB
- **Sub-components:** None

**Provided interfaces**

- DocumentMgmt
  - `Tuple<Document, MetaData> getDocument(DocumentId id)`
    - Effect: The `PDSDB` will fetch and return the document corresponding to `DocumentId` id.
    - Exceptions: None
  - `List<Document> getAllDocumentMetaDataOf(RecipientId recipientId)`
    - Effect: The `PDSLongTermDocumentManager` will fetch and return all the meta-data of the documents belonging to the Registered Recipient identified by `recipientId`.
    - Exceptions: None
  - `void storeDocument(DocumentId id, Document doc, DocumentMetaData md)`
    - Effect: The `PDSDB` will store the given document`doc` together with the provided meta-data `md`.
    - Exceptions: None
  - `void storeDocuments(List<Tuple<DocumentId, Document, DocumentMetaData>> documentList)`
    - Effect: The `PDSLongTermDocumentManager` will store the documents and their meta-data.
    - Exceptions: None
  - `List<Tuple<DocumentId, DocumentMetaData>> getAllDocumentMetaData(RecipientId recipientId) throws PDSUnavailableException`
    - Effect: The `PDSLongTermDocumentManager` fetches and returns the meta-data of all the documents of the Registered Recipient identified by `recipientId`.
    - Exceptions:
      - PDSUnavailableException: Thrown if the personal document store is unavailable.

## A.32 PDSReplicationManager

- **Description:** The `PDSReplicationManager` is responsible for managing the `PDSDBReplicas`. The `PDSReplicationManager` passes read requests to one `PDSDBReplica` and writes to all `PDSDBReplicas`. It monitors their availabilty using the ping/echo.
- **Super-component:** `PDSDB`
- **Sub-components:** None

**Provided interfaces**

- ExtendedDocumentMgmt
  - `Tuple<Document, MetaData> getDocument(DocumentId documentId)`
    - Effect: The `PDSReplicationManager` will fetch and return the document corresponding to `DocumentId` id.
    - Exceptions: None
  - `List<Tuple<DocumentId, DocumentMetaData>> getAllDocumentMetaData(RecipientId recipientId) throws PDSUnavailableException`
    - Effect: The `PDSReplicationManager` fetches and returns the meta-data of all the documents of the Registered Recipient identified by `recipientId`.
    - Exceptions:
      - PDSUnavailableException: Thrown if the personal document store is unavailable.
  - `void storeDocument(DocumentId id, Document doc, DocumentMetaData md)`
    - Effect: The `PDSReplicationManager` will store the given document `doc` together with the provided meta-data `md`.
    - Exceptions: None
  - `void storeDocuments(List<Tuple<DocumentId, Document, DocumentMetaData>> documentList)`
    - Effect: The `PDSReplicationManager` will store the given list of documents and their meta-data.
    - Exceptions: None

## A.33 Print&PostalServiceChannel

- **Description:** The `Print&PostalServiceChannel` is responsible for the printing the document and sending it by mail. It is external to the eDocs system and represents the servers of a print &postal service.
- **Super-component:** None
- **Sub-components:** None

**Provided interfaces**

- InterfaceA
  - `returntType1 operation1(ParamType param) throws SomeException`
    - Effect: Describe the effect of the operation
    - Exceptions:
      - SomeException: Describe when the exception is thrown.
  - `void operation2(ParamType2 param)`
    - Effect: Describe the effect of the operation
    - Exceptions: None
- InterfaceB
  - `returntType2 operation3()`
    - Effect: Describe the effect of the operation
    - Exceptions: None

## A.34 Print&PostalServiceFacade

- **Description:** The `Print&PostalServiceFacade` is responsible for delivering a document to the `Print&PostalChanne` so it can be printed and sent by mail. It also marks jobs as sent using the `JobManager`.
- **Super-component:** `DeliveryFunctionality`
- **Sub-components:** None

**Provided interfaces**

- InterfaceA
  - `returntType1 operation1(ParamType param) throws SomeException`
    - Effect: Describe the effect of the operation
    - Exceptions:
      - SomeException: Describe when the exception is thrown.
  - `void operation2(ParamType2 param)`
    - Effect: Describe the effect of the operation
    - Exceptions: None
- InterfaceB
  - `returntType2 operation3()`
    - Effect: Describe the effect of the operation
    - Exceptions: None

## A.35 NotificationHandler

- **Description:** The `NotificationHandler` is responsible for sending notifications to the appropriate parties, e.g. the eDocs operators and the customer administrators.
- **Super-component:** None
- **Sub-components:** None

**Provided interfaces**

- NotifyOperator
  - `void notifyOperatorOfPDSDBReplicaFailure(PDSDBReplicaId replicaId, TimeStamp dateTime)`
    - Effect: The `NotificationHandler` will send the given `PDSDBReplicaId` of the failed `PDSDBReplica` with the given time of failure `dateTime` to the eDocs operators. This method is called by a `PDSReplicationManager`.
    - Exceptions: None
  - `void notifyOperatorOfDocumentGenerationFailure(NotificationMessage msg, TimeStamp whenFailed)`
    - Effect: The `NotificationHandler` will send a textual message `msg` to the eDocs operators, which contains further information about the specific failure. This method is called by the `DocumentGenerationManager`. More specifically, it is called by the `GeneratorManager`.
    - Exceptions: None

## A.36 OtherDB

- **Description:** The `OtherDB` is responsible for storing all information that is not required to be stored separately by non-functional requirements. For example, it stores the raw data and data about customer organizations and registered recipients. It also stores the templates for documents and the keys of customer organizations to sign the documents during generation.
- **Super-component:** None
- **Sub-components:** None

**Provided interfaces**

- GetKey
  - `Key getKey(CustomerId customerId)`
    - Effect: The `OtherDB` returns the key belonging to the Customer Organization identified by `customerId`.
    - Exceptions:
      - NoSuchKeyException: Thrown if there is no key for the given `customerId`.
- TemplateMgmt
  - `Template getTemplate(CustomerId customerId, DocumentType documentType, TimeStamp whenReceived)`
    - Effect: The `OtherDB` returns the `Template` belonging to the customer organisation with id `customerId` corresponding to a document of type `documentType` and received at time `whenReceived`.
    - Exceptions:
      - NoSuchTemplateException: Thrown if there is no template for the given arguments.
  - `Boolean storeDocumentTemplate(CustomerId cuId, DocumentType documentType, Template template, TimeStamp whenReceived) throws InvalidDocumentTypeException`
    - Effect: The `OtherDB` stores the given template with the given time stamp for the customer organization identified by the given `CustomerId`.
    - Exceptions:

· InvalidDocumentTypeException: Thrown if the given document type is invalid or not allowed for the given customer organization.

- UserDataMgmt
  - `Credentials getRegisteredRecipientCredentials(RecipientId recipientId) throws NoSuchRecipientException`
    - Effect: The `OtherDB` returns the credentials belonging to the Registered Recipient identified by `recipientId`.
    - Exceptions:
      · NoSuchRecipientException: Thrown if no Registered Recipient with the given credentials exists.
  - `Credentials getCustomerOrganizationCredentials(CustomerId customerId) throws NoSuchCustomerOrganizationException`
    - Effect: The `OtherDB` returns the credentials belonging to the Customer Organization identified by `customerId`.
    - Exceptions:
      · NoSuchCustomerOrganizationException: Thrown if no Customer Organization with the given credentials exists.
  - `List<Tuple<DocumentType, TimeStamp>> getPossibleDocumentTypes(CustomerId customerId)`
    - Effect: The `OtherDB` returns the a list of document types that the Customer Organization identified by `customerId` can generate. For each document type, it also returns the time when the last `Template` for that document type was uploaded.
    - Exceptions: None

## A.37 RecipientClient

- **Description:** The `RecipientClient` is external to the eDocs system and represents a client device of an unregistered or registered recipient of eDocs that communicates with the eDocs System.
- **Super-component:** None
- **Sub-components:** None

**Provided interfaces**

- InterfaceA
  - `returntType1 operation1(ParamType param) throws SomeException`
    - Effect: Describe the effect of the operation
    - Exceptions:
      · SomeException: Describe when the exception is thrown.
  - `void operation2(ParamType2 param)`
    - Effect: Describe the effect of the operation
    - Exceptions: None
- InterfaceB
  - `returntType2 operation3()`
    - Effect: Describe the effect of the operation
    - Exceptions: None

## A.38 RawDataHandler

- **Description:** The `RawDataHandler` is responsible for verifying the raw data and its entries. It forwards the validated raw data to the `JobManager` to create jobs.
- **Super-component:** None
- **Sub-components:** None

**Provided interfaces**

- InterfaceA
  - `void validateRawData(List<RawData> rawData) throws InvalidRawDataException`
    - Effect: The `RawDataHandler` verifies the received raw data. If it is correct, nothing happens.
    - Exceptions:
      - InvalidRawDataException: Thrown if the raw data is invalidated for some reason. This exception can contain a message about why the raw data was invalidated.
- InterfaceB
  - `returntType2 operation3()`
    - Effect: Describe the effect of the operation
    - Exceptions: None

## A.39 RecipientFacade

- **Description:** The `RecipientFacade` is responsible for the interaction of Registered and Unregistered Recipients with the eDocs system. It provides methods for authentication, for consulting the personal document store, for downloading documents, ...
- **Super-component:** `Userfunctionality`
- **Sub-components:** None

**Provided interfaces**

- AuthN
  - `SessionId login(Credentials credentials ) throws InvalidCredentialsException`
    - Effect: The `RecipientFacade` forwards the given `credentials` to the `AuthenticationHandler`, which verifies them and returns a new session identifier if correct. This session identifier can be used in future requests to the `RecipientFacade`.
    - Exceptions:
      - InvalidCredentialsException: Thrown if the `AuthenticationHandler` indicated that the given credentials where incorrect.
  - `Boolean logout(SessionId sessionId)`
    - Effect: The `RecipientFacade` removes the session corresponding to the `sessionId` using the `AuthenticationHandler`. As a result, this session cannot be used anymore to access the system without logging in again. If no session corresponds to the `sessionId`, it does not exist, nothing is changed but no exception is thrown.
    - Exceptions: None
- DocumentMgmt
  - `PDSOverview getPDSOverview(SessionId, RecipientId recipientId) throws NotAuthenticatedException, PDSUnavailableException`
    - Effect: The `RecipientFacade` first verifies the given session identifier `sessionId` using the `AuthenticationHandler`. The `RecipientFacade` then requests all the document meta-data of the documents of the recipient identified by `recipientId` from the `PDSFacade`. It generates a document overview , which is returned to the caller.
    - Exceptions:
      - NotAuthenticatedException: Thrown if the given session identifier is invalid.
      - PDSUnavailableException: Thrown if the personal document store is unavailable.

## A.40 RegistrationManager

- **Description:** The `RegistrationManager` is responsible for the registration or unregistered recipients and customer organizations. For the registration of unregistered recipients, the `RegistrationManager` gets called by the `RecipientFacade`, as recipients can register themselves. Customer organizations get registered by an eDocs operator, so the `EDocsAdminClient` calls those methods.
- **Super-component:** `UserFunctionality`
- **Sub-components:** None

**Provided interfaces**

- InterfaceA
  - `returntType1 operation1(ParamType param) throws SomeException`
    - Effect: Describe the effect of the operation
    - Exceptions:
      - SomeException: Describe when the exception is thrown.
- InterfaceB
  - `returntType2 operation3()`
    - Effect: Describe the effect of the operation
    - Exceptions: None

## A.41 Scheduler

- **Description:** The `Scheduler` receives the new jobs initiated by a Customer Organization and adds them to a queue of all jobs that have not been processed yet. To lower the size of this queue, the Scheduler is only given the information it needs, i.e., the id of the batch, its deadline and the ids of the individual jobs. The raw data of each job and the meta-data of the batch is stored in `OtherDB` and fetched by the `Completer` when needed.
  The `Scheduler` also indicates to the `GenerationManager` the number of required `Generator` instances through its GetStatistics interface.
- **Super-component:** `DocumentGenerationManager`
- **Sub-components:** None

**Provided interfaces**

- GetNextJobs
  - `Tuple<BatchId, List<JobId>> getNextJobs()`
    - Effect: The `Scheduler` returns the `JobIds` of the group of jobs that belong to the batch identified by `BatchId` that should be generated next. This method is called by the `GeneratorManager` when a `Generator` instance requires a new group of jobs.
    - Exceptions: None
    - `Tuple<BatchId, List<JobId>> jobsCompletedAndGiveMeMore(List<JobId>)`
      - Effect: The `Scheduler` gets notified that the document processing jobs belonging to the list of `JobIds` are completed. It returns the a list of `JobIds` belonging to a batch identified by `BatchId`. The returned list of `JobIds` identify document processing jobs which are not yet started.
      - Exceptions: None
- InsertJobs
  - `void insertJobs(BatchId batchId, TimeStamp deadline, List<JobId> jobIds )`

        * Effect: The `Scheduler` adds the jobs identified by their `JobId` to its queue of all jobs that have not been processed yet. To lower the size of this queue, the Scheduler is only given the information it needs, i.e., the id of the batch, its deadline and the ids of the individual jobs. This method provides new jobs synchronously to the `Scheduler`, which it schedules synchronously. This means that when the method call returns, the given jobs are scheduled.
        * Exceptions: None

- GetStatistics
    - `int getNumberOfFutureJobs()`
        * Effect: The `Scheduler` returns the amount of documents that should be generated in the near future. The `GeneratorManager` queries this method at regular intervals and adjusts the number of `Generator` instances accordingly.
        * Exceptions: None

## A.42 SessionDB

- **Description:** The `SessionDB` stores the session identifiers for currently active sessions.
- **Super-component:** `UserFunctionality`
- **Sub-components:** None.

**Provided interfaces**

- SessionMgmt
    - `RecipientId getRecipientId(SessionId sessionId) throws NoSuchSessionException`
        * Effect: The `SessionDB` fetches and returns the Registered Recipient's identifier corresponding to the `sessionId` from the `sessionDB`.
        * Exceptions:
            · NoSuchSessionException: Thrown if no session exists with the given identifiers, or if the session belongs to a customer organization.
    - `CustomerId getCustomerId(SessionId sessionId) throws NoSuchSessionException`
        * Effect: The `SessionDB` fetches and returns the Customer Organization's identifier corresponding to the `sessionId` from the `sessionDB`.
        * Exceptions:
            · NoSuchSessionException: Thrown if no session exists with the given identifier, or if the session belongs to a registered recipient.
    - `SessionId openSession(RecipientId recipientId)`
        * Effect: The `SessionDB` generates a new session identifier for the given `recipientId` and stores this as an active session.
        * Exceptions: None
    - `void closeSession(SessionId sessionId) throws NoSuchSessionException`
        * Effect: The `SessionDB` closes the active session associated with the given `sessionId`.
        * Exceptions:
            · NoSuchSessionException: Thrown if no session exists with the given identifier.
    - `Map<SessionAttributeKey, SessionAttributeValue> isvalidSession(SessionId sesionId) throws NoSuchsSessionException`
        * Effect: The `SessionDB` verifies whether a session with the given id exists in the `SessionDB` and if so, returns all its associated attributes.
        * Exceptions:
            · NoSuchSessionException: Thrown if no session exists with the given identifiers.

## A.43 TemplateCache

- **Description:** The `TemplateCache` caches the templates which are most recently used for document generation. The `Completer` has to fetch a templateevery time a `Generator` instance requests new jobs, while the template will be the same for all jobs belonging to the same batch. The `TemplateCache` avoids that the template storage system becomes a bottleneck for document generations. The templates are cached based on the `CustomerId` of a Customer Organization, the type of the document and the date and time at which the batch was provided by the Customer Organization (in order to account for template updates).
- **Super-component:** `DocumentGenerationManager`
- **Sub-components:** None

**Provided interfaces**

- GetTemplate
  - `Template getTemplate(CustomerId customerId, DocumentType documentType, TimeStamp whenReceived`
    - Effect: The `TemplateCache` looks into its cache for the `Template` belonging to the customer organisation with id `customerId` corresponding to a document of type `documentType` and received at time `whenReceived`. If the `Template` is in its cache, it returns it. If the `Template` is not in its cache, it asks `OtherDB` for the `Template` and stores it in its cache, after which it returns that `Template`.
    - Exceptions:
      - NoSuchTemplateException: Thrown if there is no template for the given arguments.

## A.44 UserFunctionality

- **Description:** The `UserFunctionality` is responsible for the interaction of registered recipients, unregistered recipients, customer organizations and eDocs operators with the eDocs system. It provides methods to register, to login and to logout, to consult the personal document store and download documents, to consult the status of document processing jobs, ...
- **Super-component:** None
- **Sub-components:** `RecipientFacade`, `CustomerOrganizationClient`, `EDocsadminfacade`,`RegistrationManager` `AuthenticationHandler` and `SessionDB`

**Provided interfaces**

- InterfaceA
  - `returntType1 operation1(ParamType param) throws SomeException`
    - Effect: Describe the effect of the operation
    - Exceptions:
      - SomeException: Describe when the exception is thrown.
  - `void operation2(ParamType2 param)`
    - Effect: Describe the effect of the operation
    - Exceptions: None
- InterfaceB
  - `returntType2 operation3()`
    - Effect: Describe the effect of the operation
    - Exceptions: None

### A.45 ZoomitChannel

- **Description:** The `ZoomitChannel` is responsible for delivering documents via Zoomit. It is external to the system and represents the servers of Zoomit to which a document can be sent.
- **Super-component:** None
- **Sub-components:** None

**Provided interfaces**

- InterfaceA
  - `returntType1 operation1(ParamType param) throws SomeException`
    - Effect: Describe the effect of the operation
    - Exceptions:
      - SomeException: Describe when the exception is thrown.
  - `void operation2(ParamType2 param)`
    - Effect: Describe the effect of the operation
    - Exceptions: None
- InterfaceB
  - `returntType2 operation3()`
    - Effect: Describe the effect of the operation
    - Exceptions: None

### A.46 ZoomitFacade

- **Description:** The `ZoomitFacade` is responsible for sending documents to Unregistered Recipients through Zoomit. It is also responsible for receiving messages from Zoomit when a document has been received by Zoomit or when a Zoomit user as received his or her document. The `ZoomitFacade` can use the `JobManager` to mark jobs as sent or received.
- **Super-component:** `DeliveryFunctionality`
- **Sub-components:** None

**Provided interfaces**

- InterfaceA
  - `returntType1 operation1(ParamType param) throws SomeException`
    - Effect: Describe the effect of the operation
    - Exceptions:
      - SomeException: Describe when the exception is thrown.

## B Defined data types

List and describe all data types defined in your interface specifications. List them alphabetically for ease of navigation.

- `BatchId`: A piece of data uniquely identifying a batch of document processing jobs in the system. This architecture does not specify the exact format of this identifier, but possibilites are a long integer, a string, a URL etc.
- `BatchMetaData`: A data structure listing the metadata belonging to a batch of jobs. This includes the `CustomerID` of a Customer Organization, the `DocumentType` of the documents to be generated, the `TimeStamp` of when the batch was received, ...

- `CompletePartialBatchData`: A complex data structure listing all data a `Generator` needs to complete document generation jobs that are part of the same batch. It contains an array of `Tuple<JobId, RawData>`. The `JobIds` identify jobs that are all part of the same batch. The `RawData` belongs to these document processing jobs. Also listed in the `BatchMetaData` are the values of the `BatchMetaData`, `Key` and `Template` data types belonging to the batch.

  `CompletePartialBatchData` also contains a `BatchMetaData` entry, a `Key` and a `Template`. *Important to note:* a value of CompletePartialBatchData contains all information necessary to generate **some** jobs of belonging to same batch. It does not have to contain the information of all jobs belonging to same batch.

- `Credentials`: The authentication credentials of a Registered Recipient or Customer Organization. The credentials always contain an identifier of the recipient or customer organization and a proof of his or her identity. The architecture does not specify the specific credentials used, but a possibility is using a username and password.

- `CustomerId`: A piece of data uniquely identifying a Customer Organization in the system. This architecture does not specify the exact format of this identifier, but possibilites are a long integer, a string, a URL etc.

- `Document`: A data file corresponding to a document. The architecture specifies the format of this data type as a PDF-file.

- `DocumentId`: A piece of data uniquely identifying a document in the system.

- `DocumentMetaData`: The meta-data stored with a document in the `DocumentDB`. This meta-data differs from the `PDSDBMetaData`, in that it does not contain a `UserId`, but it does contain an `Email` address.

- `DocumentPriority`: A data type representing the priority of document generation jobs. They have values representing the Critical, Diamond, Gold and Silver priorities. The exact format of this data type is not specified by the architecture.

- `DocumentType`: A piece of data describing the type of a document. This architecture does not specify the exact format of this data type, but possibilities are a long integer, a string, a URL etc.

- `Echo`: The response to a ping message. This data element does not contain any meaningful data.

- `Error`: Description of data type.

- `GeneratordId`: A piece of data uniquely identifying a `Generator` in the system. This architecture does not specify the exact format of this identifier, but possibilites are a long integer, a string, a URL etc.

- `JobBatch`: Description of data type.

- `JobId`: A piece of data uniquely identifying a document processing job in the system.

- `Key`: A data structure containing the key of the Customer Organization which is used to sign its documents during the generation process. This architecture does not specify the exact format of this data type, but possibilites are a long integer, a string, a URL etc.

- `NotificationMessage`: A textual message which can be used to include extra information about the event of the notification.

- `PDSOverview` The overview of a personal document store that can be shown to a Registered Recipient. Through this overview, the Registered Recipient can consult documents. The architecture does not specify the exact format of such an overview, but a likely possibility is an HTML page.

- `PDSDBReplicaId`: A piece of data uniquely identifying a `PDSDBReplica` in the system. This architecture does not specify the exact format of this identifier, but possibilites are a long integer, a string, a URL etc.

- `PDSDBMetaData`: The meta-data stored with a document in the `PDSDB`. This meta-data differs from the `DocumentMetaData`, in that it does not contain a `Email` address, but it does contain a `UserID`.

- `RawData`: A data structure listing the raw data used in a document processing job.

- `RecipientId`: A piece of data uniquely identifying a Registered Recipient in the system. This architecture does not specify the exact format of this identifier, but possibilites are a long integer, a string, a URL etc.
- `SessionId`: A piece of data uniquely identifying a session of a registered recipient of customer organization in the eDocs system. This contains at least the user identifier (i.e. the `CustomerId` for a customer organization or the `RecipientId` for a registered recipient) and the time the session was initiated.
- `SessionAttributeKey`: The key of an attribute attached to a session. This architecture does not specify the exact format of this key. a possible value is a long integer or a flat string.
- `SessionAttributeValue`: The value of an attribute attached to a session. This value van be of any primitive type.
- `TimeStamp`: The representation of a time (i.e. date and time of day) in the system.
- `Template`: A document used as a template for the generation of documents.
- `TemplateId`: A data structure uniquely identifying a template in the system. It lists three values. It contains `CustomerId` which identifies the Customer Organization who the template belongs to. It also contains a `DocumentType`, specifying for which kind of document it is a template for. The last piece of information it contains is a `TimeStamp` specifying when the system received the template.