# **Enterprise Integration Application Requirements and Documentation**

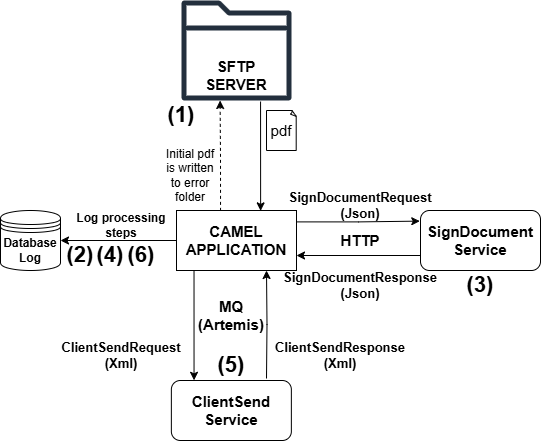
## **Overview**

The Enterprise Integration Application facilitates the secure processing of PDF documents by:

1. Reading PDF documents from an SFTP server.
2. Digitally signing the documents using the SignDocument service.
3. Distributing the signed documents to specified clients via the ClientSend service.
4. Logging all operations into a PostgreSQL database for auditing and tracking purposes.
5. Deploying the application in a scalable Kubernetes environment.
6. Writing failed PDF documents to a separate error folder for retry if any errors occur during processing.

### **Data Flow**

1. File is read from the SFTP server.
2. Metadata is extracted from the file name and logged.
3. Document is sent to the SignDocument service.
4. Signed document is received and logged.
5. Signed document is sent to the specified clients via the ClientSend service.
6. Final status is logged in PostgreSQL.
7. If an error occurs, the original PDF is written to a designated error folder for later retry.



## **Functional Requirements**

### **1. File Reception**

* **Protocol**: The application will poll an SFTP server for new PDF files.
* **File Naming Convention**: The file name will contain metadata in the following format: documentId\_ownerId\_documentType\_clientId.pdf
  + - documentId: Unique identifier for the document.
    - ownerId: Specifies the owner of the document.
    - documentType: Specifies the type of the document.
    - clientId: Specifies the client to whom the signed document should be sent.
* **File Processing**:
  + The application will parse the file name to extract metadata.
  + The PDF file will be read and prepared for signing.
  + If an error occurs during processing, the PDF will be moved to an error folder for manual or automated retry later.

### **2. Document Signing**

* **Business Logic**: The application will sign the document by making a request to the SignDocument service.
* **Message Protocol**: The application will use HTTP to communicate with the SignDocument service.
* **Request Payload**: The request will be in JSON format with the following structure:

{  
 "document": "<Base64-encoded PDF>",  
 "ownerId": "<Owner Information>"  
 "signtype": "<Signature Type, default: CAdES-C>",  
 "apikey": "<API Key>",  
 "documentType": "<Document Type>"  
}

* **Response Payload**: The response will be in JSON format with the following structure:

{  
 "signedDocument": "<Base64-encoded Signed PDF>",  
 "status": "<Status of the Signing Process: Success/Error>",  
 "message": "<Optional Message>",  
 "timestamp": "<Timestamp of the Response>"  
}

### **3. Client Notification**

* **Business Logic**: The application will send the signed document to the client by using the ClientSend service.
* **Message Protocol**: The application will use ArtemisMQ to communicate with the ClientSend service.
* **Request Payload**: The request will be in XML format with the following structure:

<ClientSendRequest>  
 <documentId>12345</documentId>  
 <ownerId>Owner ID</ownerId>  
 <document>Base64EncodedSignedPDF</document>  
 <clientId>Client123</clientId>  
 <timestamp>2025-01-01T12:00:00Z</timestamp>  
</ClientSendRequest>

* **Response Payload**: The response will be in XML format with the following structure:

<ClientSendResponse>  
 <status>Success/Error</status>  
 <message>Document sent successfully</message>  
<timestamp>2025-01-01T12:00:01Z</timestamp>

### **4. Logging**

* **Database**:
  + PostgreSQL will be used to store logs.
* **Log Details**:
  + documentId: Unique document identifier.
  + ownerId: Owner of the document.
  + timestamp: Time of the event.
  + status: Status of the operation (e.g., received, signed, sent to client).
  + errorDetails: If applicable, stores error messages or codes.

### **5. Deployment**

* **Kubernetes Deployment**: The application will be containerized using Docker.