

An aerial photograph of a cityscape at sunset or sunrise. The sky is a mix of blue and orange. In the foreground, a large, modern building complex with white facades and many windows is visible. A rainbow is visible on the left side of the image, arching over the buildings. In the background, other city buildings and a tall smokestack are visible against the horizon.

# Semester Project Information

SWEN3 | BIF5



An aerial photograph of a city at sunset. The sky is a mix of orange, yellow, and blue. A rainbow is visible in the lower-left corner. In the foreground, there are several modern, multi-story buildings with white and grey facades. In the background, a city skyline is visible, including a tall chimney emitting smoke.

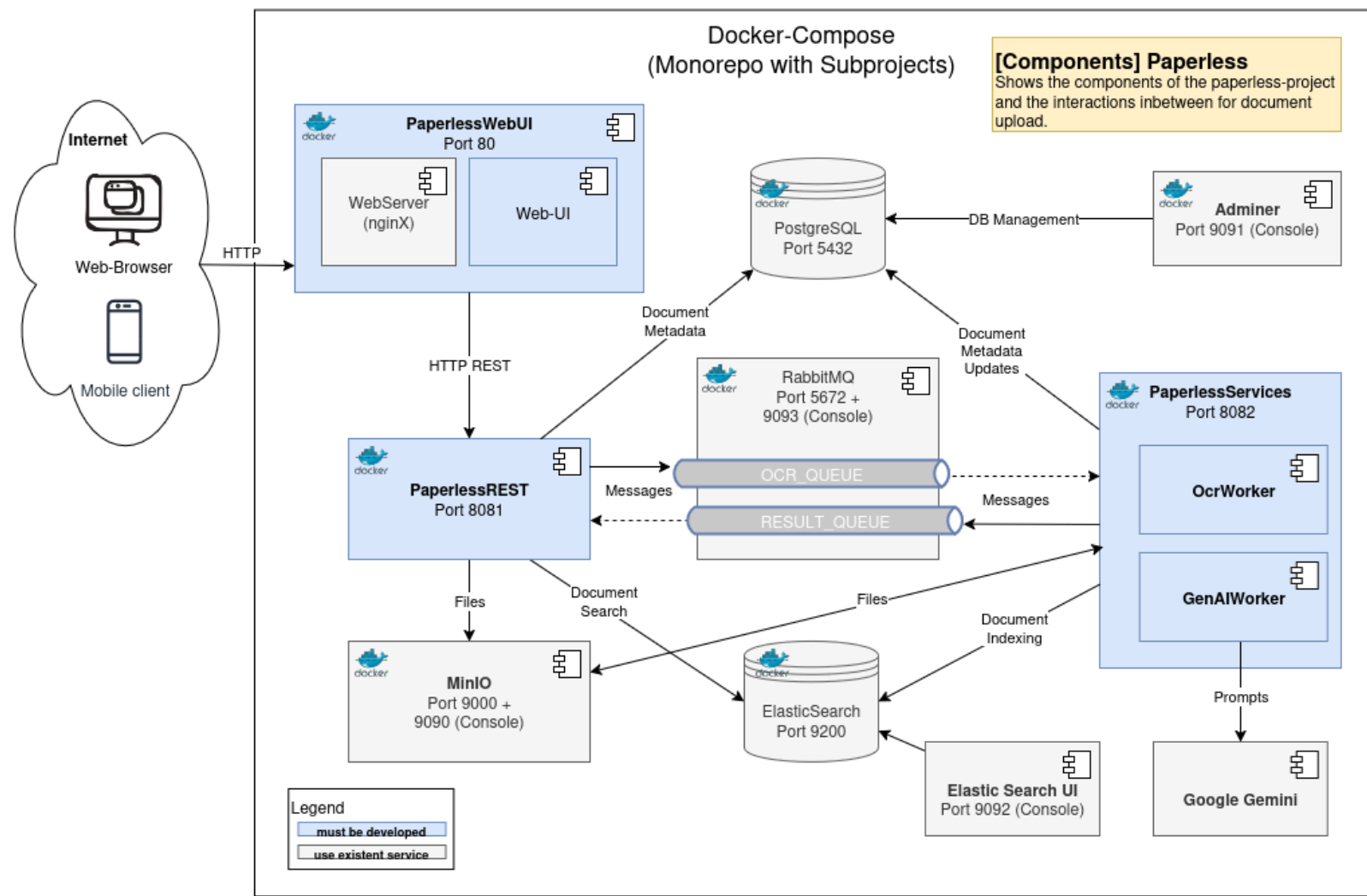
# Semester Project:

# Document Management System

*a Document management system for archiving documents in a FileStore,  
with automatic OCR (queue for OC-recognition),  
automatic summary generation (using Gen-AI),  
tagging and full text search (ElasticSearch).*

COR = Optical Character Recognition (Texterkennung)

# Semester Project Architecture



# Use Cases

## 1. Upload document

- Automatically performs OCR
- Is indexed for full-text search in Elasticsearch
- a summary is automatically generated

ElasticSearch = Such-Engine, die vor allem für Volltextsuche entwickelt wurde, arbeitet mit JSON Dokumenten; indexed: in einer speziellen Suchstruktur gespeichert

## 2. Search for a document

- Full-text and fuzzy search in Elasticsearch

fuzzy search = Tippfehler ignorieren

## 3. Manage documents

- Update, delete, metadata

## 4. Your additional individually defined usecase (Sprint 6)

- Your own choice
- Must include additional entities!

# Technology Stack

## Java

- JDK LTS ( $\geq 21$ )
- Spring Boot ( $\geq 3$ )

## C#

- .Net  $\geq 8.0$
- ASP.Net

## Both

- Docker (Desktop)
- RabbitMQ tauscht Nachrichten zwischen Anwendungen aus, indem es Warteschlangen (Queues) verwaltet
- PostgreSQL
- ElasticSearch
- GenAI (Google Gemini or similar)  
generative AI



# Semester Project Sprints

An aerial photograph of a cityscape during sunset or sunrise. The sky is a mix of blue and orange. In the foreground, there are several modern, multi-story buildings with white facades and many windows. A rainbow is visible on the left side of the image, arching over a street. A bridge or walkway connects two of the buildings. In the background, a city skyline is visible, including a tall chimney emitting smoke. The overall atmosphere is calm and urban.

# Sprints

Sprint 1	Sprint 2	Sprint 3	Sprint 4	Sprint 5	Sprint 6	Sprint 7
Project-Setup	(Web-)UI	Queuing	Workers,	GenAI	ELK	Integr.-Test
REST API			MinIO, OCR		Use Cases	Batch-Processing
DAL		<b>Mid-Term Code Review</b>				<b>Final Code Review</b>

# Sprint 1: Project-Setup, REST API, DAL (with Mapping)

1. Java/C# Project Setup
2. Remote Repository setup,  
all team members are able to commit/push
3. REST Server created  
Endpoints defined by the team (code-first)
4. ORM is integrated to persist the entities on the PostgreSQL database, use the repository pattern
5. Show correct function with unit-tests, mock out the “production” database
6. Initial docker-compose.yml, used to run the REST-server & database inside containers



# Sprint 1: Assignment

- Date of submission: `dd.mm.yyyy`  
(see Moodle Course: „Submission – Sprint 1)
- Submission contents:
  - ZIP-Archive with the full project
- MUST-HAVE Check Criteria:
  - No build error
  - Servers (REST & PostgreSQL) start successfully
  - REST Endpoints functioning, data is persisted in DB



# Sprint 2: (Web-)UI

1. Webserver service (e.g. nginx or else) integrated
2. Dashboard and detail-pages are served by the webserver
3. The Webpage communication with the REST server
4. Extend docker-compose.yml to run the UI in an additional container

# Sprint 2: Assignment

- Date of submission: `dd.mm.yyyy`  
(see Moodle Course: „Submission – Sprint 2)
- Submission contents:
  - ZIP-Archive with the full project
- MUST-HAVE Check Criteria:
  - No build error (docker compose build)
  - docker compose up successfully starts containers
  - GET <http://localhost/> returns the functioning paperless-frontend





# Sprint 3: Queues integration (RabbitMQ)

1. Extend docker-compose.yml to run RabbitMQ in a container
2. Integrate Queues into REST Server
3. on document upload the REST-Server should also
  - send a message to the RabbitMQ
  - will be processed by an "empty" OCR-worker
4. Failure/exception-handling (with layer-specific exceptions) implemented
5. Logging in remarkable/critical positions integrated
6. Prepare for the mid-term Code-Review

# Sprint 3: Assignment

- Date of submission: `dd.mm.yyyy`  
(see Moodle Course: „Submission – Sprint 3)
- Submission contents:
  - ZIP-Archive with the full project
- MUST-HAVE Check Criteria:
  - No build error (docker compose build)
  - docker compose up starts all containers successfully
  - POST <http://localhost/>... some PDF-Document  
will lead to a log-entry at the worker-service (for to be processed with OCR)



# Sprint 4: Worker Services (OCR, MinIO)

1. Create an additional application for running the OCR service
2. Tesseract for Ghostscript (or the like) integrated and working, show function with unit-tests.
3. Extend REST Server to store PDF document in MinIO
4. Implement the OCR-worker service to
  - retrieve messages from the queue (sent by REST-Server on document-upload),
  - fetch the original PDF-document from MinIO
  - perform the OCR-recognition
  - show functionality with unit-tests
5. Extend docker-compose.yml to run the MinIO and OCR-service in a container



# Sprint 4: Assignment

- Date of submission: `dd.mm.yyyy`  
(see Moodle Course: „Submission – Sprint 4)
- Submission contents:
  - ZIP-Archive with the full project
- MUST-HAVE Check Criteria:
  - No build error (docker compose build)
  - docker compose up successfully starts all containers
  - POST <http://localhost/>... some PDF-Document
    - will lead to store the PDF on MinIO
    - will lead to a log-output on the PaperlessService.OcrWorker (stating OCR result)



# Sprint 5: Generative AI-Integration

1. Extend docker-compose.yml to include configuration for GenAI service
2. Add a new GenAI-worker service
3. On document upload, after OCR is complete:
  - Send the extracted text to a GenAI API (Google Gemini)
  - Receive a summary as a response
4. Extend REST Server to store the summary in the database
5. Logging in critical positions is integrated
6. Exceptions and API failures are handled properly

# Sprint 5: Assignment

- Date of submission: `dd.mm.yyyy`  
(see Moodle Course: „Submission – Sprint 5)
- Submission contents:
  - ZIP-Archive with the full project
- MUST-HAVE Check Criteria:
  - No build error
  - docker compose up starts all required containers including PaperlessServices
  - POST <http://localhost:8080/> ... some PDF-Document
    - will lead to the summary generated with GenAI
    - will lead to summary stored in the database





# Sprint 6: Elasticsearch Integration, additional Use-Case

1. Elasticsearch integrated in worker-service and working
  - Show function with unit-tests.
2. Implement the indexing-worker to
  - Store the text-content (the former OCR-result) in Elasticsearch
  - Show functionality with unit-tests
3. Implement the use-case „search for documents“ in your project.
4. Implement your additional use-case in your project (this must contain additional entities)

# Sprint 6: Assignment

- Date of submission: `dd.mm.yyyy`  
(see Moodle Course: „Submission – Sprint 6)
- Submission contents:
  - ZIP-Archive with the full project
- MUST-HAVE Check Criteria:
  - No build error
  - docker compose up starts all required containers
  - HelloWorld.pdf will be uploaded via Paperless Frontend <http://localhost/>
  - Search function executed in <http://localhost/> for term „Hello“ will result in showing up the HelloWorld.pdf document



# Sprint 7: Integration-Test, Batch-Processing, Finalization

1. Show the functionality of use case „document upload“ with an integration-test
2. Create an additional application for running the scheduled service that reads daily XML files from an input folder to process access logs from external systems via a batch process.
  - Consider how such a process is best integrated into the existing architecture.
  - Define an appropriate XML format for the access statistics.
  - Extend the PostgreSQL database so that the daily access count is stored per document.
  - Implement the batch service to be scheduled (e.g. daily at 01:00 AM) to retrieve and process the XML files. Ensure the input folder and filename patterns are configurable.
  - Provide a sample XML file to demonstrate the functionality.
  - Ensure that processed XML files are appropriately archived or removed to prevent redundant processing.
3. Project finalization
4. Prepare for the final code-review

# Sprint 7: Assignment

- Date of submission: `dd.mm.yyyy`  
(see Moodle Course: „Submission – Sprint 7)
- Submission contents:
  - ZIP-Archive with the full project
- MUST-HAVE Check Criteria:
  - No build error
  - docker compose up starts all required containers
  - Provided integration-test (write an HOWTO in README.md) will be executed and should run successfully to the end.
  - The batch process must successfully read the sample XML file, process the data, and persist it in the database as demonstrated by relevant database queries.





# Semester Project Grading

An aerial photograph of a cityscape during sunset or sunrise. The sky is a mix of blue and orange. In the foreground, a large, modern white building complex with many windows is prominent. A rainbow is visible on the left side of the image, arching over a parking lot. In the background, other city buildings and a tall smokestack are visible against the hazy horizon.

# Grading

- 35% Continuous Sprint Submissions (code quality + completeness)
  - Incremental Points per Sprint
    - E.g. Sprint 3 = Sprint 1 Today, Sprint 2 Today, Sprint 3 Today
    - Continued option to improve
  - Hand-ins are graded and teams are asked for presentations
- 65% Code Reviews (code quality + knowledge + completeness)
  - 30% Mid-Review
  - 70% End-Review
- All parts have to be positive

# Code Review

**Code review** is systematic examination ... of computer source code. It is intended to find and fix mistakes overlooked in the initial development phase, improving both the overall quality of software and the developers' skills.

# Code Review Criterias

- Unit Tests, Code Coverage (> 70%)
- REST Service
- Queuing
- Business Layer & Logic
- Data Access Layer
- Entities & Entity Mapping
- Validation
- Exception Handling
- Logging
- Dependency Injection
- Service Agents
- Implementation of Use Cases



