Semester Project Information

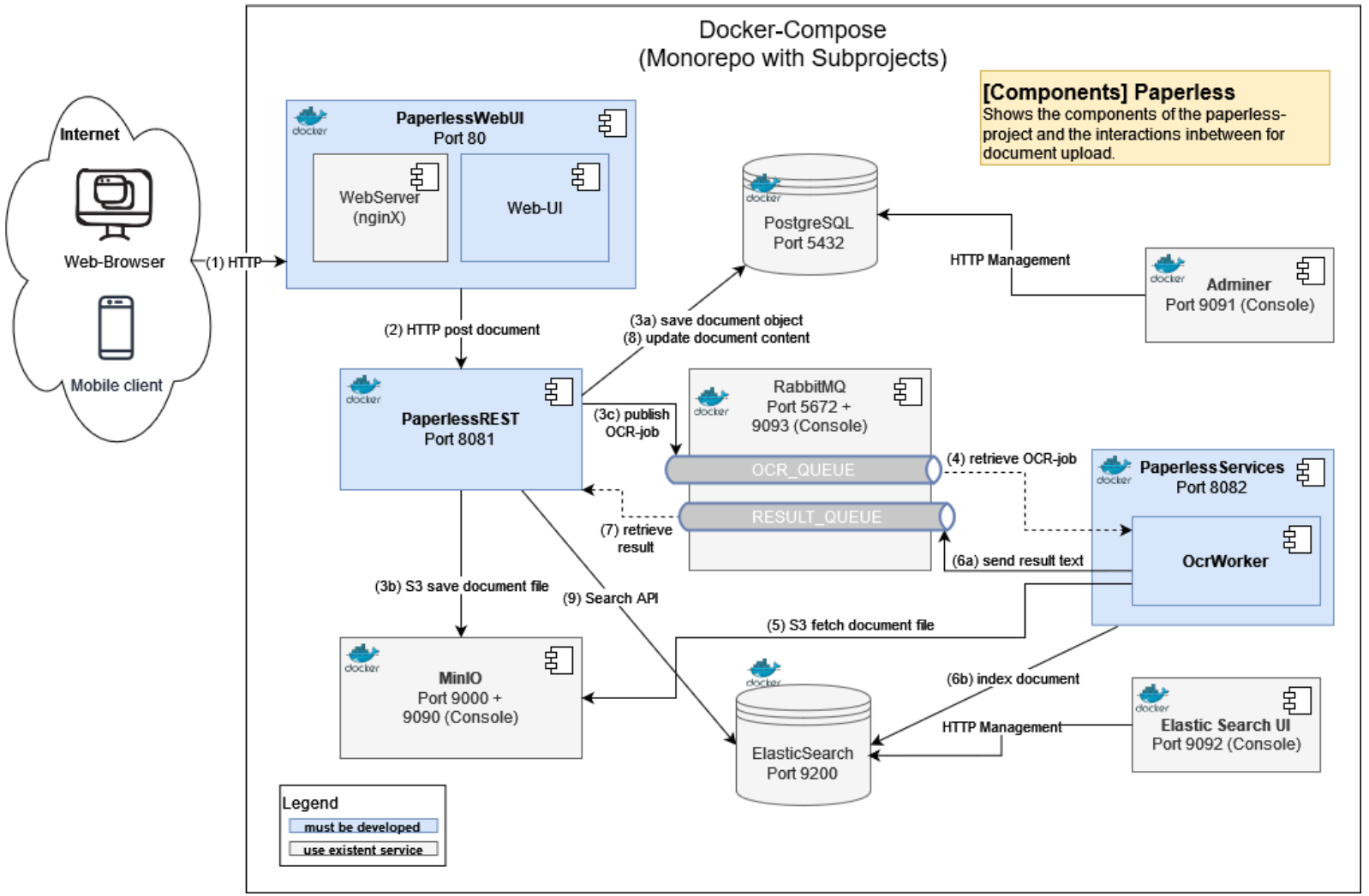
SWKOM | BIF5

Semester Project:

Document Management System

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

Semester Project Architecture



Use Cases

1. Upload document
    - Automatically performs OCR
    - Is indexed for full-text search in ElasticSearch
2. Search for a document
    - Full-text and fuzzy search in ElasticSearch
3. Manage documents
    - Update, delete, metadata

Further Usecases optional

Technology Stack

Java

- JDK LTS (>=17)
- Spring Boot (>=3)

C#

- .Net >= 7.0
- ASP.Net

Both

- Docker (Desktop)
- RabbitMQ
- PostgreSQL
- ElasticSearch

Semester Project Sprints

Sprints

Sprint 1	Sprint 2	Sprint 3	Sprint 4	Sprint 5	Sprint 6	Sprint 7
Prj-Setup	(Web-)UI	DAL	Queuing,	OCR-Service	ElasticSearch	Integr.-Test
REST API		Mapping	DI,Logging	Error-Hdl.	Use Cases	Finalizazion
			Code Review			**Code Review**

Sprint 1: Project-Setup, REST API

1. Java/C# Project Setup
2. Remote Repository setup,
   all team members are able to commit/push
3. REST Server created
   Endopints defined by the team
4. Requests to endpoints return a hardcoded result
5. Initial docker-compose.yml, used to run the REST-server inside a container

Sprint 1: Assignment

- Date of submission: `dd.mm.yyyy`
  (see Moodle Course: „Submission – Sprint 1:REST Service API)
- Submission contents:
  - ZIP-Archive with the full project
- MUST-HAVE Check Criteria:
  - No build error
  - REST-Server starts
  - GET http://localhost:8081/ returns HTTP 200 OK, with some hardcoded data

Sprint 2: (Web-)UI

1. Webserver service (e.g. nginx or else) integrated
2. Some simple Dashboard is served by the webserver
3. The Webpage communication with the REST server
   at least some "Hello World" message
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:UI Integration)
- Submission contents:
  - ZIP-Archive with the full project
- MUST-HAVE Check Criteria:
  - No build error (docker compose build)
  - docker compose up starts RestServer- and WebServer-Container
  - GET http://localhost/ returns the paperless-frontend showing hard-coded fake data

Sprint 3: Data Access Layer (DAL), Mapping, PostgreSQL

1. Entities classes are created
2. DTOs are mapped to the entities using a mapping framework
   show correct function with Unit-Tests
3. Validation rules are defined on the entities
   show correct function with Unit-Test
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. Extend docker-compose.yml to run PostgreSQL as container

Sprint 3: Assignment

- Date of submission: `dd.mm.yyyy`
  (see Moodle Course: „Submission – Sprint 3:DAL (Persistence))
- Submission contents:
  - ZIP-Archive with the full project
- MUST-HAVE Check Criteria:
  - No build error (docker compose build)
  - docker compose up starts RestServer-, WebServer- and PostgreSQL-Container
  - POST http://localhost/... some PDF-Document
    will lead to a line INSERTED in the document-table in the database

Sprint 4: 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 (this will be processed by the OCR-worker in the next sprint)
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 4: Assignment

- Date of submission: `dd.mm.yyyy`
  (see Moodle Course: „Submission – Sprint 4:Message Broker)
- Submission contents:
  - ZIP-Archive with the full project
- MUST-HAVE Check Criteria:
  - No build error (docker compose build)
  - docker compose up starts RestServer-, WebServer-, PostgreSQL-, and RabbitMQ-Container
  - POST http://localhost/... some PDF-Document
    - will lead to an entry on the RabbitMQ-Server (checked on RabbitMQ-Console/MgMT-Interface http://localhost:9093/)
    - will lead to a log-output on the PaperlessService.OcrWorker

Sprint 5: OCR Services Integration

1. Create an additional application for running the OCR service
2. Tesseract for Ghostscript (or the like) integraded and working, show function with unit-tests.
3. Implement the OCR-worker service to
   - retrieve messages from the queue (sent by REST-Server on document-upload),
   - fetch the original PDF-document
   - Perform the OCR-recognition
   - Send the text-result back to the REST-Server via queue
   - Show functionallity with unit-tests
4. Extend docker-compose.yml to run the OCR-service in a container

Sprint 5: Assignment

- Date of submission: `dd.mm.yyyy`
  (see Moodle Course: „Submission – Sprint 5:Worker Services)
- 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 PDF-file OCR-processed in PaperlessServices
    - will lead to text-content stored

Sprint 6: ElasticSearch Integration

1. Elasticsearch integraded 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 functionallity with unit-tests

Sprint 6: Assignment

- Date of submission: `dd.mm.yyyy`
  (see Moodle Course: „Submission – Sprint 6:Elasticsearch)
- 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: Use-Cases, Integration-Test, Finalization

1. Implement another use-cases in your project, like „search for documents“
2. Show the functionallity of use case „document upload“ with an integration-test
3. Show the functionallity of your additional use-cases with an integration-test
4. Project finalization
5. Prepare for the final code-review

Sprint 7: Assignment

- Date of submission: `dd.mm.yyyy`
  (see Moodle Course: „Submission – Sprint 7:Finalization)
- 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.

Semester Project Grading

Grading

- 35% Continous 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