FH University of Applied Sciences TECHNIKUM WIEN

Semester Project

BIF5-SWKOM

Semester Project: Paperless

Overview

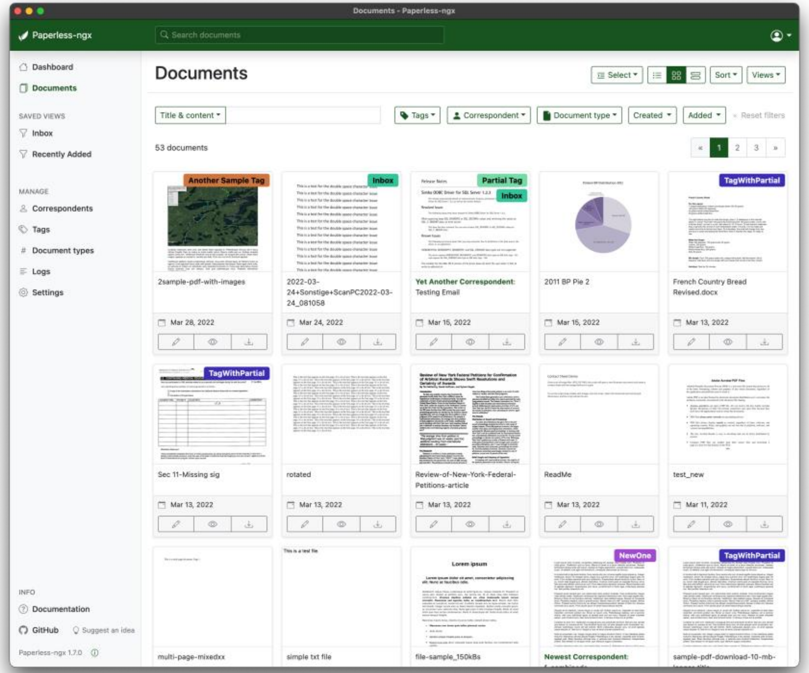
Paperless

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

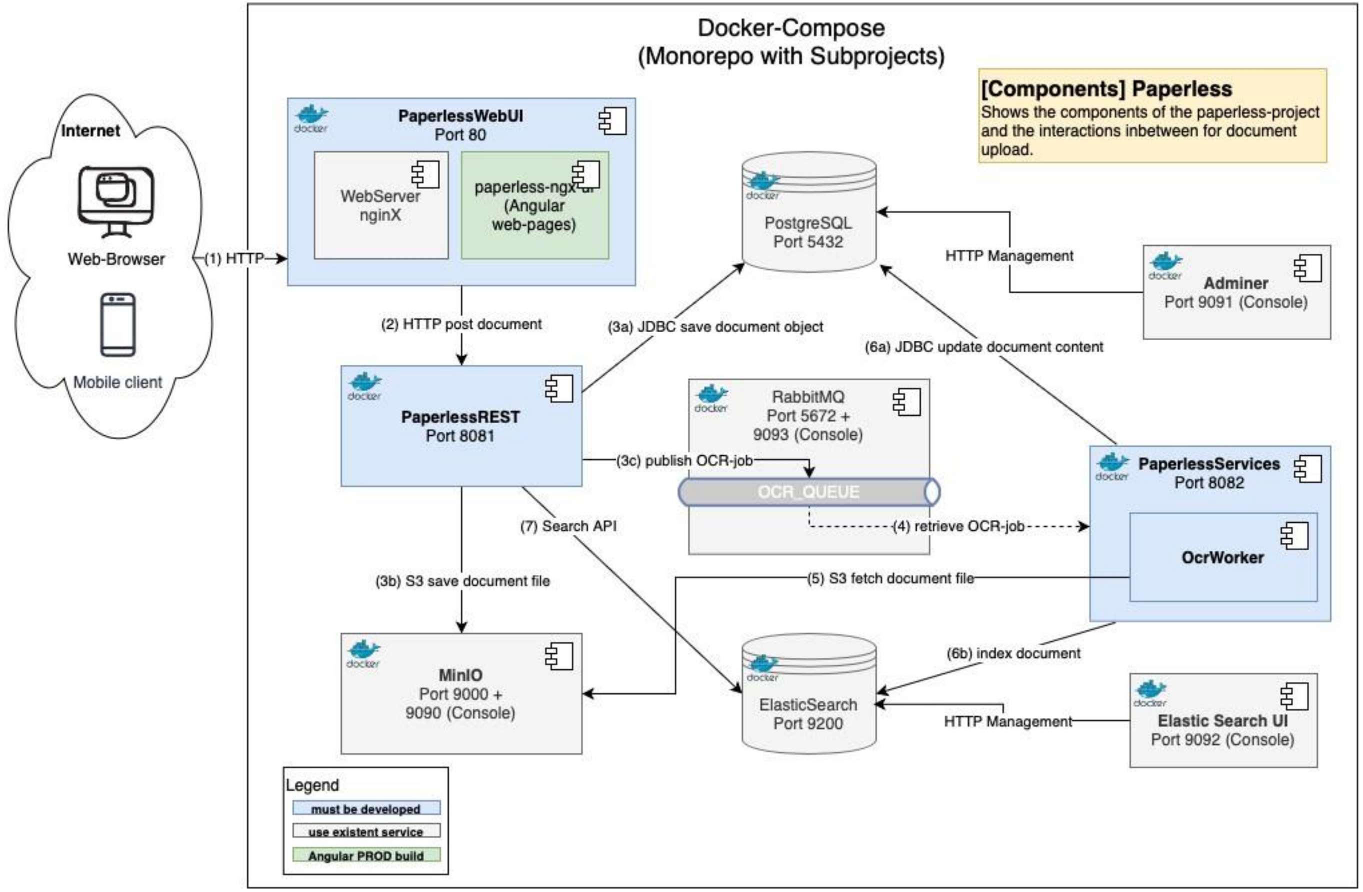
Paperless Frontend

- https://docs.paperless-ngx.com/

- **Paperless-ngx** is a *community-supported* open-source document management system that transforms your physical documents into a searchable online archive so you can keep, well, *less paper*.



Paperless Architecture



Some assumptions

- Paperless-ngx used as Frontend
- Investigate the API and Paperless-ngx specificas with a proxy
- Extend REST-Server to provide paperless-ngx needs

Use cases

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

- Further Usecases optional

Semester Project: Paperless

Organization

FH University of Applied Sciences TECHNIKUM WIEN

Sprint 1	Sprint 2	Sprint 3	Sprint 4	Sprint 5	Sprint 6	Sprint 7
• REST API • OpenAPI • CI/CD	• UI • Object Mapping • Validation	• Data Access Layer • Repositories • O/R Mapper	• Queueing • DI & Logging →Code Review	• Service Agent (OCR) • Error Handling	• Elastic Search • Use Cases	• Integration Tests • Finalization →Code Review

Sprint 1: Project setup, REST, CI/CD

1. Java Project with Maven setup
2. Remote Repository setup,
   all team members are able to commit/push
3. REST Server generated based on OpenAPI-specs
4. Requests to endpoints return a hardcoded result
5. The CI/CD pipeline is setup so that on every push on main/master: the project is built, tests are ran, and the artifact with the artifact (jar or assembly) is uploaded to the repository
6. Initial docker-compose.yml, used to run the REST-server inside a container

Sprint 2: Integrate UI

1. A webserver service paperlessNGX webpages integrated
2. paperlessNGX communicates with REST server
3. UI: the dashboard shows up in the browser (with fake data)
4. Extend docker-compose.yml to run the UI in an additional container

Sprint 3: Data Access Layer (DAL), ORM, 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 4: Queues integration (RabbitMQ)

1. Extend docker-compose.yml to run RabbitMQ in a container
2. Extend docker-compose.yml to run MinIO in a container
3. Integrate Queues into REST Server
4. 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)
   - Store the PDF-file on MinIO
5. Failure/exception-handling (with layer-specific exceptions) implemented
6. Logging in remarkable/critical positions integrated
7. Prepare for the mid-term Code-Review

Sprint 5: Services Integration - OCR

1. Create an additional application for running the OCR service
2. Tesseract for Ghostscript 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 from MinIO
   - Perform the OCR-recognition
   - Store the text-result in PostgreSQL
   - Show functionallity with unit-tests
4. Extend docker-compose.yml to run the OCR-service in a container

Sprint 6: Services Integration - Elasticsearch

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 7: Use Cases, Integration-Test, Finalization

1. Implement another use-cases in your project, at least „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

Technology Stack

Java – Groups

- JDK LTS 17
- Spring Boot
- RabbitMQ
- PostgreSQL
- GitHub + GitHub Actions

C# - Groups

- .NET 7.0
- ASP.Net
- RabbitMQ
- PostgreSQL
- Linux or Windows
- Azure DevOps or GitHub + GitHub Actions
- (Azure WebApps)