Team:

- Grancsa Robert, 343C1
- Toader Ana-Maria, 341C4

Contributions to the project were similar, as can be seen in the commits on github.

Auth service

A lightweight authentication layer that sits in front of your application and delegates sign-in/out, token refresh and user-info retrieval to **Amazon Cognito** using the **OpenID Connect** flow.
It is written in **TypeScript**, runs on **Express**, and ships with a production-ready Docker image plus a GitHub Actions workflow that automatically builds and publishes that image.

Key features:

- Provide a simple, self-contained service that:
  - Redirects users to Cognito for login and receives the authorization *code* callback (`/auth/login`, `/auth/callback`)
  - Handles logout through Cognito's `/logout` endpoint (`/auth/logout`)
  - Exposes management endpoints for refreshing tokens and retrieving user-info (`/management/refresh`, `/management/userinfo`)
- Store tokens and user data in an **Express session** (in-memory by default) so the rest of your stack can stay stateless.
- Log all significant events with **Winston** for easier troubleshooting.

CI/CD:

A GitHub Actions workflow:

1. Builds the Docker image on every push to `main`.
2. Publishes it to GitHub Container Registry (`ghcr.io`).
3. Updates the image tag in the downstream Kubernetes manifest (see `update-manifest` job).

Business logic

A lightweight micro-service that ingests PDF files, extracts and chunks their contents (text + tables), then stores the result in a downstream document store for fast semantic search.

It exposes a small REST/JSON API (and a simple HTML playground) guarded by an external auth service.

Key Features:

- PDF ingestion & OCR – powered by unstructured-io and PyMuPDF
- Sentence segmentation – spaCy `ro_core_news_lg` model
- Chunking for embeddings – Hugging Face `sentence-transformers` tokenizer
- Endpoints:
  - `POST /upload` – upload & index a PDF
  - `DELETE /delete` – remove a previously indexed file
  - `GET /search` – query your indexed content
  - `GET /get-documents` – list all files for the authenticated user
- HTML demo UI at `/` (drag-and-drop a PDF, run queries, etc.)
- Container-ready – single-stage Dockerfile, multi-arch image pushed to GHCR
- CI/CD – GitHub Actions workflow builds the image on every `main` push and bumps the tag in a separate k8s-infra repo
- Logs – structured files under ./logs/, plus console output.
- Uploads & temp files – saved to uploads/ and purged once the PDF has been processed.

CI/CD:

The workflow in `.github/workflows/build-and-push.yml` :

1. Builds a multi-arch image with Docker Buildx.
2. Pushes the image to ghcr.io.
3. Opens the k8s-infrastructure repo, patches the image tag in the deployment manifest, and pushes to main.

DB service

Provides a low-level OpenSearch Python client with wrapper methods for the OpenSearch REST API so that you can interact with your cluster more naturally in Python.

Key features:

- upload documents in bulk and automatically embed text with an ingest pipeline
- perform semantic (k-NN) search against your own per-user indices
- list or delete stored documents and indices
- API reference:

Method & path	Body (JSON)	Description
`POST /db-service/upload`	`{ "id": "<index>", "content": [...] }`	Bulk-upload documents (creates index if absent)
`GET /db-service/search`	`{ "id": "<index>", "query": "…" }`	Semantic search (k results, default 3)
`GET /db-service/get-documents`	`{ "id": "<index>" }`	List distinct file names stored in an index
`DELETE /db-service/delete`	`{ "id": "<index>", "filename": "file.pdf" }`	Delete all docs from a given file

All endpoints expect `Content-Type: application/json` .

CI/CD:

- **build-and-push.yml** uses Docker Buildx to build a multi-architecture image and push it to **GHCR**.
- A follow-up job updates the `db-service` image tag inside an external Kubernetes manifests repository.

K8s infrastructure

The repository contains the declarative Kubernetes manifests that deploy the **Document Manager** platform:

- **Auth Service** – Node.js API for authentication and Cognito integration
- **Business-Logic Service** – Python / Flask API that mediates between the UI, Auth and DB layers
- **DB Service** – Python / Flask wrapper around OpenSearch for vector-enabled storage and search
- **OpenSearch + Dashboards** – Backend datastore (stateful set) with an optional UI
- **Ingress** – Single public entry-point that routes traffic to the above services
- **Argo CD Application** – GitOps controller that keeps the cluster in sync with `main`
- **Kustomize** – Composition layer that stitches the individual manifests together

Access points:

URL path	Backend service
`/auth`	**auth-service** on port 3000
`/api`	**business-logic-service** on port 5000
`/dashboard`	**opensearch-dashboards** on port 5601

These routes are configured in `ingress.yaml` .