

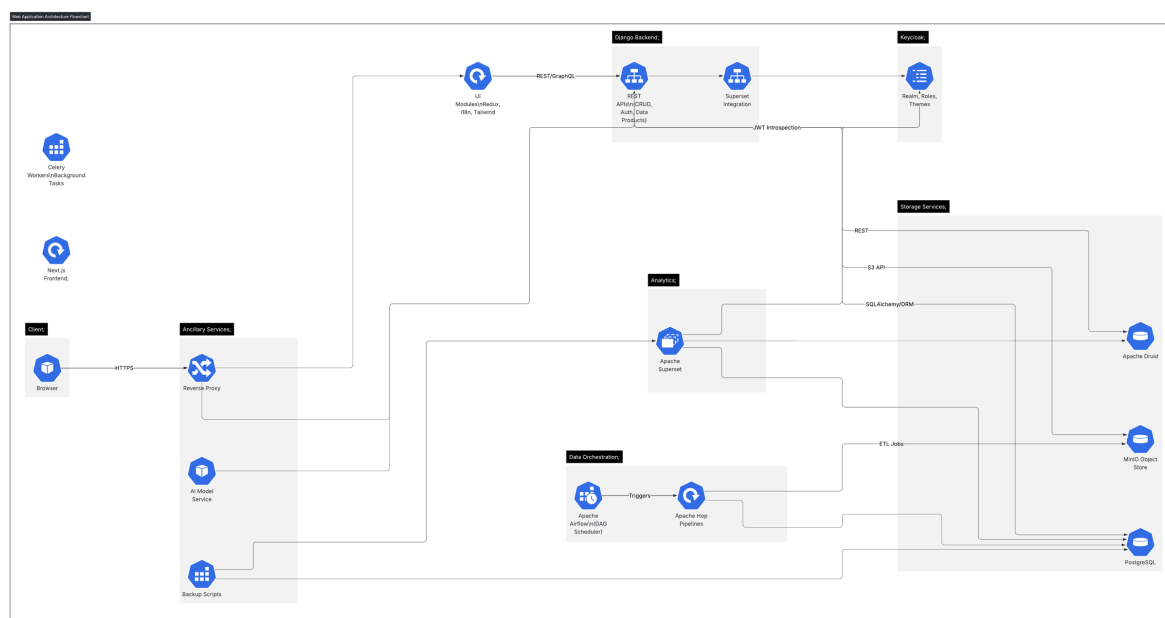
AU-DOHP Architecture and Developer Documentation

1 Overview

The AU-DOHP (Digital One Health Platform) repository hosts a collection of containerised services that together form a manual-deployment health data platform. It emphasises modularity and open-source technologies yet intentionally omits automated deployment pipelines, data lineage tracking, and centralised metadata management. User administration is centralised and therefore does not fully adhere to Data Mesh principles, which favour federated identity and role management.

2 High-Level Architecture

The overall ecosystem relies on Docker Compose for manual orchestration. Each service is self-contained but communicates through well-defined APIs and shared infrastructure (PostgreSQL, MinIO, Apache Druid). The following diagram depicts the major components, networking flows, and supporting services.



Architectural Notes

1. **No automated deployment:** the above components are launched manually via `docker-compose` files.
2. **No data lineage or metadata catalogue:** pipelines lack automatic provenance capture, and no central metadata registry exists.
3. **Centralised identity:** Keycloak controls authentication; domain teams cannot self-manage identities, diverging from Data Mesh ideals.

3 Service Modules

3.1 Frontend (frontend/)

- **Framework:** Next.js with TypeScript, React, Tailwind CSS.
- **State Management:** Redux Toolkit for global state; `i18next` for localisation.
- **E2E Testing:** Cypress configured under `cypress/`.
- **Structure:** feature modules in `src/modules/` (e.g., `data-product`, `pipeline`, `user`), each with UI components, hooks, and services.

- **Build & Quality:** scripts in `package.json`; linting via ESLint; formatting via Prettier.

3.2 Backend (backend/)

- **Framework:** Django + Django REST Framework, Celery for asynchronous tasks.
- **Main Apps:**
 - `api/`: Endpoint routing and generic serializers.
 - `users/ & accounts/`: User models, admin registration, Keycloak integration.
 - `roles/`: Role definitions and decorators.
 - `data_product/`: CRUD for data product entities and metadata fields.
 - `data_contract/`: Contract models, validations, and relations to pipelines.
 - `pipeline/`: Pipeline models, transformation rules, helper validators.
 - `process/`: Long-running business processes.
 - `backup/ & storage/`: MinIO integration and backup utilities.
 - `supersets/`: Communication layer for Superset dashboards.
 - `ai_model/`: Hooks to optional ML service.
- **Configuration:** `core/settings.py` obtains parameters (DB, MinIO, Keycloak endpoints) from environment variables.
- **Migrations:** executed manually via `python manage.py migrate`.

3.3 Authentication (keycloak/)

- **Realm Configuration:** `realm/realm.json` exports initial roles, clients, and groups.
- **Custom Event Listener:** `keycloak-custom-event-listener/` (Java, Maven) to intercept login events or propagate hooks.
- **Theme:** `theme/speedymesh/` provides login pages and email templates.
- **Limitations:** Domain teams depend on central admins for role or group changes, limiting Data Mesh self-service.

3.4 Data Orchestration

- **Apache Hop (hop/):**
 - Pipelines under `hop/pipelines/` with templates for FHIR, DHIS2, CSV ingestion.
 - Hop transforms connect to various sources, outputting to MinIO or PostgreSQL.
- **Apache Airflow (airflow/):**
 - DAGs under `airflow/dags/` schedule Hop jobs and other tasks.
 - `airflow.cfg` retains default lineage settings; no lineage backend is enabled.

3.5 Analytics (superset/)

- **Superset Integration:** Contains `Dockerfile`, `superset_config.py`, and bootstrap scripts.
- **Auth:** Keycloak OAuth is configured; dashboards imported via `superset/import/`.
- **Usage:** Visualises data from PostgreSQL and Apache Druid.

3.6 Storage Services

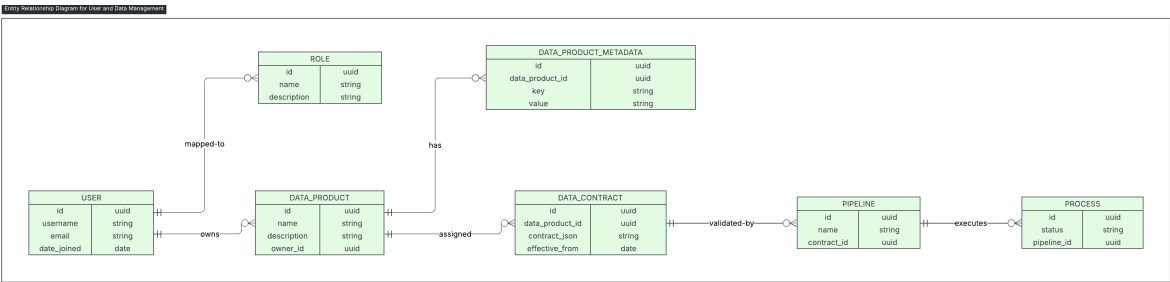
- **PostgreSQL:** main relational database for Django models and Superset metadata.
- **MinIO:** S3-compatible object store; `minio/setup.sh` provisions buckets.
- **Apache Druid:** analytics DB for fast aggregations, used by Superset.

3.7 Ancillary Modules

- **Reverse Proxy (nginx/):** central entry point terminating TLS and routing to services.
- **AI Model Service (ai-model/):** stand-alone microservice for experimental inference tasks.
- **Backup Scripts (backups/):** shell scripts documenting manual backup routines for PostgreSQL and Superset.

4 Database Schema and Relationships

The platform uses PostgreSQL for core data. The following Mermaid ER diagram outlines principal tables in the backend, highlighting their interdependencies. This is a conceptual view; actual implementation may include additional columns and junction tables.

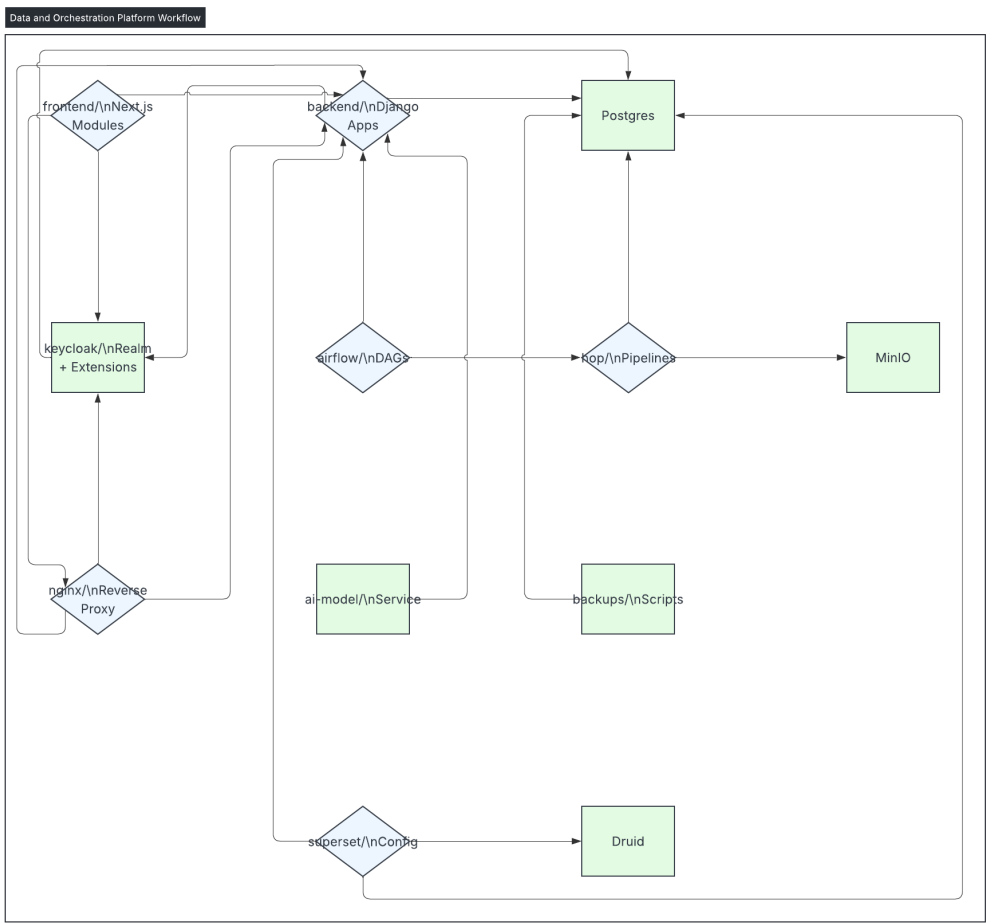


Notes on Storage

- **MinIO** buckets are not represented in the ER diagram but serve as object repositories for raw or processed files associated with **DATA_PRODUCTS**.
- **Druid** hosts denormalised event tables for analytics; its schema varies per ingestion pipeline and is not centrally tracked.
- **No metadata catalogue**: the relationships above are managed within PostgreSQL; there is no overarching metadata system to describe pipelines or datasets.

5 Internal Library and Module Structure

The repository behaves as a multi-module “library” in the sense that each directory provides reusable components. The diagram below highlights top-level modules and their primary dependencies.



Each module is developed and maintained independently but shares common environment variables and docker-compose networks. None are deployed automatically; administrators manually build images, run migrations, and start services.

6 Operational Considerations

1. Manual Deployment Only

- All services rely on hand-crafted docker-compose files (docker-compose.yml, .dev.yml, .prod.yml).
- Operators manually execute commands such as `docker-compose up --build` and `docker-compose down`.

2. Lack of Data Lineage and Metadata Management

- No automated provenance tracking exists for pipelines or datasets.
- Metadata is limited to application tables in PostgreSQL; there is no cross-system catalogue like Amundsen or DataHub.

3. Centralised User Management

- Keycloak hosts all realms, clients, and roles.
- Domain teams cannot self-provision users or roles, conflicting with Data Mesh requirements for decentralised ownership.

4. No Automated Testing Pipeline

- While unit and integration tests exist (Django tests, Cypress scripts), there is no CI server automatically executing them. Running tests remains a manual developer responsibility.

7 Developer Guide

7.1 Prerequisites

- **Docker & Docker Compose** for orchestrating services.
- **Python 3.10+**, `virtualenv`, and `pip` for backend development.
- **Node.js** and `npm`/`yarn` for the frontend.
- **Java & Maven** if developing Keycloak extensions.
- **Optional:** `psql`, `mc` (MinIO CLI), `airflow` CLI, `hop-run.sh`.

7.2 Local Setup Workflow

1. Clone Repo & Configure Environment

- Copy sample `.env` files or craft new ones specifying credentials for PostgreSQL, MinIO, and Keycloak.

2. Start Core Stack

```
docker-compose -f docker-compose.dev.yml up --build
```

3. Backend Development

```
cd backend pip install -r requirements.txt python manage.py migrate python manage.py runserver 0.0.0.0:8000
```

4. Frontend Development

```
cd frontend npm install npm run dev
```

5. Keycloak Configuration

- Access Keycloak admin console, import `keycloak/realms/realms.json` if using an external instance.

6. Superset

- Initialise metadata DB, import dashboards using `superset/import/` scripts.

7. Data Pipelines

- Develop Hop pipelines in `hop/`, schedule them via Airflow DAGs in `airflow/dags/`.

7.3 Coding Standards

- **Backend:** PEP8, flake8 (if run manually), and Django best practices.
- **Frontend:** ESLint, Prettier; commit messages follow conventional style.
- **Tests:**
 - Django app tests reside in respective tests.py or tests/ directories.
 - Cypress scripts for e2e coverage of core UI flows.

8 Known Limitations and Future Work

AREA	CURRENT STATE	PLANNED ENHANCEMENTS
Data Lineage	None	Integrate lineage in Airflow/Hop, adopt catalog (e.g., DataHub).
Metadata Management	No central catalog	Introduce a metadata service to track datasets/pipelines.
Deployment	Manual Docker Compose	Establish CI/CD pipelines (e.g., GitHub Actions, GitLab CI).
User Management	Centralised Keycloak realm	Move toward federated identity and fine-grained domain roles.
Monitoring	Basic logging only	Add Prometheus/Grafana and alerting.
Governance	No audit trails beyond logs	Implement comprehensive auditing and access reviews.