Documentation

This is the developer-facing documentation for the FLUXPORT backend (FastAPI + SQLAlchemy + PostgreSQL). It focuses on architecture, data model, API surface, and service workflows.

Quick map

- Architecture: `architecture.md`
- API surface: `api.md`
- Data model: `data-model.md`
- Reporting engine: `reporting.md`
- Dev setup and operations: `dev-setup.md`

At a glance

- Runtime: FastAPI + Uvicorn
- ORM: SQLAlchemy 2.x
- Migrations: Alembic
- DB: PostgreSQL (configured via `DATABASE_URL`)
- Contracts: OpenAPI in `openapi.json`, Postman collection in `collection.json`

System context

flowchart LR UI[Flutter Web UI] -->|HTTP JSON| API[FastAPI App] API -->|SQLAlchemy| DB[(PostgreSQL)] API -->|Excel export| FILES[In-memory XLSX stream]

Folder layout (core)

- `app/main.py`: FastAPI app and router registration
- `app/api`: HTTP routers (CRUD, metadata, reports)
- `app/services`: domain services (PO, logistics, number range)
- `app/models`: SQLAlchemy models (business, lookup, and RBAC)
- `app/schemas`: Pydantic schemas
- `app/db`: DB session and base
- `app/core/reports`: report config + dynamic query engine
- `alembic/`: migrations

This backend is a layered FastAPI service with SQLAlchemy models, service classes for domain workflows, and a dynamic reporting engine.

High-level component layout

flowchart TB subgraph API R1[API Routers] R2[Lookup Factory Router] R3[Metadata + Access Queries] R4[Reports API] end subgraph Services S1[PurchaseOrderService] S2[LogisticsService] S3[NumberRangeService] S4[ReportQueryEngine] end subgraph Data M1[SQLAlchemy Models] M2[Alembic Migrations] DB[(PostgreSQL)] end R1 --> S1 R1 --> S2 R1 --> S3 R4 --> S4 S1 --> M1 --> DB S2 --> M1 --> DB S3 --> M1 --> DB S4 --> M1 --> DB M2 --> DB

Runtime entrypoint

- `app/main.py` creates the FastAPI app, applies CORS middleware, and includes routers.
- Core routers are registered directly (users, roles, metadata, access queries, number ranges).
- Domain APIs are grouped in `app/api/v1/endpoints/api.py` under `/api/v1`.

Configuration

- `app/core/config.py` loads `DATABASE_URL` from env with a default fallback.
- `app/db/session.py` creates the SQLAlchemy engine and `SessionLocal`.
- `check_env.py` prints the resolved database URL.

API layers

CRUD routers (direct)

Located in `app/api/routers` and follow a REST pattern: - `users`, `roles`, `permissions`, `role_permissions` - `user_roles`, `user_departments`, `user_countries`, `user_attributes` - `domains`, `object_types`, `masteraddr`, `forwarders` - `number_range` (sys number ranges) - `metadata` (table metadata for UI) - `user_profile`, `access_queries` (joined profile + RBAC queries)

Lookup factory (dynamic routers)

`app/api/v1/endpoints/lookup_factory.py` dynamically builds CRUD routers for lookup tables. Configured in `app/api/v1/endpoints/api.py` via `LOOKUP_CONFIG`.

Excel export)

Services

PurchaseOrderService

File: `app/services/purchase_order_service.py`

Responsibilities: - Validate vendor existence and activity - Recompute line totals and header totals server-side - Assign initial status and persist header/items - Retrieve PO with joined lookup relationships

LogisticsService

File: `app/services/logistics_service.py`

Responsibilities: - Create shipment header and items - Lock PO items (row-level) to prevent overship - Validate remaining quantity - Update PO item status when fully shipped

NumberRangeService

File: `app/services/number_range_get.py`

Responsibilities: - Create and update range configurations - Generate next number using row-level lock - List and delete ranges

ReportQueryEngine

File: `app/core/reports/query_engine.py`

Responsibilities: - Build dynamic queries based on config - Apply joins, projection, filters, and sorting - Provide consistent behavior for data and export

Cross-cutting patterns

- SQLAlchemy sessions are per-request via `Depends(get_db)`
- Errors are surfaced as HTTP errors at router level
- Soft delete exists for `users` via `?mode=soft` (sets `is_active=false`)
- Dynamic UIs consume `/metadata/{table}` and use type info for forms/tables

This page summarizes the main HTTP endpoints and their roles.

Base paths

- Root CRUD routers: `/`
- Versioned domain APIs: `/api/v1`
- Number ranges: `/api/v1/sys-number-ranges`
- Reports (visibility): `/api/v1/reports/visibility`

Core CRUD routers (root)

These follow a consistent REST pattern: list, get, create, update, delete.

- `users` (includes `GET /users/by-email`, delete supports `?mode=soft|hard`)
- `roles`, `permissions`, `role-permissions`
- `user-roles`, `user-departments`, `user-countries`, `user-attributes`
- `object-types`, `domains`, `masteraddr`, `forwarders`
- `metadata` (schema-like table metadata for UI)

RBAC and access queries

- `GET /user-profile?email=...` or `?username=...`
- Joined profile response: user + roles + permissions + departments + countries + attributes
- `GET /access-queries/by-permission?permission_id=...`
- Returns permission, roles containing it, and users in those roles
- `GET /access-queries/by-role?role_id=...` or `?role_name=...`
- Returns role, permissions, and users holding the role

Number range management

Base path: `/api/v1/sys-number-ranges`

Supported operations: - `GET /api/v1/sys-number-ranges` (list) - `POST /api/v1/sys-number-ranges` (create) - `PATCH /api/v1/sys-number-ranges/{range_id}` (update) - `DELETE /api/v1/sys-number-ranges/{range_id}` (delete)

- `/uom_lookup`
- `/po_status_lookup`, `/po_type_lookup`, `/purchase_org_lookup`, `/po_item_status_lookup`
- `/shipment_status_lookup`, `/transport_mode_lookup`, `/milestone_type_lookup`, `/container_type_lookup`
- `/cost_component_lookup`
- `/document_type_lookup`, `/text_type_lookup`

These endpoints are created dynamically and share the same CRUD semantics.

Domain endpoints

Purchase Orders

Base: `/api/v1/purchase-orders`

- `POST /` creates a PO header + items
- `GET /{po_id}` returns a PO with items and joined lookups
- `GET /` paginated list, supports optional `vendor_id`

Shipments

Base: `/api/v1/shipments`

- `POST /` creates shipment with validation against PO items
- `GET /{shipment_id}` retrieves shipment with items

Reports (Visibility)

Base: `/api/v1/reports/visibility`

- `GET /metadata` returns UI metadata for column definitions and filters
- `GET /data` returns report data with pagination
- `GET /export` streams an Excel file based on selected columns

API contracts

- `openapi.json` is committed at repo root
- `collection.json` provides a Postman collection

FLUXPORT Backend Docs

This section summarizes the SQLAlchemy models and their logical groupings. Table names are shown in brackets.

RBAC and users

- User [users]: id, username, email, clearance, is_active, created_at
- Role [roles]: id, name
- Permission [permissions]: id, action_key, object_type
- UserRole [user_roles]: id, user_id, role_id
- RolePermission [role_permissions]: id, role_id, permission_id, role_name
- UserDepartment [user_departments]: id, user_id, department
- UserCountry [user_countries]: id, user_id, country_code
- UserAttribute [user_attributes]: id, user_id, key, value

erDiagram users ||--o{ user_roles : has roles ||--o{ user_roles : assigns roles ||--o{ role_permissions : grants permissions ||--o{ role_permissions : contains users ||--o{ user_departments : member_of users ||--o{ user_countries : assigned_to users ||--o{ user_attributes : has

Master data

- MasterAddr [masteraddr]: name, addr_type, country, city, street, phone, email, validity
- PartnerRole [partner_role_lookup]: role_code, role_name
- PartnerMaster [partner_master]: partner_identifier, role_id, legal_name, payment terms, currency, addr_id
- CompanyMaster [company_master]: company_code, branch_code, legal_name, addr_id
- Forwarder [forwarder]: forwarder_id, branch_id, validity
- Domain [domains]: domain_name, technical_key, display_label, is_active
- ObjectType [object_types]: object_type, object_description
- SystemQualifier [system_qualifier]: category, code, label, description

Product and pricing

- ProductMaster [product_master]: sku_identifier, type_id, uom_id, hs_code, origin, weight, volume
- ProductTypeLookup [product_type_lookup]
- UomLookup [uom_lookup]
- PricingType [pricing_type_lookup]

- PurchaseOrderHeader [po_header]: po_number, type_id, status_id, purchase_org_id, vendor_id, total_amount, currency
- PurchaseOrderItem [po_item]: po_header_id, item_number, product_id, status_id, quantity, unit_price, line_total
- POScheduleLine [po_schedule_line]: po_item_id, shipment_header_id, quantity, delivery_date
- PurchaseOrderStatusLookup [po_status_lookup]
- PurchaseOrderTypeLookup [po_type_lookup]
- PurchaseOrgLookup [purchase_org_lookup]
- PurchaseOrderItemStatusLookup [po_item_status_lookup]

erDiagram po_header ||--o{ po_item : contains po_item ||--o{ po_schedule_line : plans partner_master ||--o{ po_header : vendor po_type_lookup ||--o{ po_header : type po_status_lookup ||--o{ po_header : status po_item_status_lookup ||--o{ po_item : status

Logistics

- ShipmentHeader [shipment_header]: shipment_number, status_id, mode_id, carrier_id, ETA/ETD
- ShipmentItem [shipment_item]: shipment_header_id, po_schedule_line_id, shipped_qty
- ShipmentContainer [shipment_container]: container_type_id, container_number, seal_number
- ShipmentMilestone [shipment_milestone]: milestone_id, event_datetime, location
- ShipmentStatusLookup [shipment_status_lookup]
- TransportModeLookup [transport_mode_lookup]
- MilestoneTypeLookup [milestone_type_lookup]
- ContainerTypeLookup [container_type_lookup]

erDiagram shipment_header ||--o{ shipment_item : packs shipment_header ||--o{ shipment_container : contains shipment_header ||--o{ shipment_milestone : tracks po_schedule_line ||--o{ shipment_item : fulfills shipment_status_lookup ||--o{ shipment_header : status transport_mode_lookup ||--o{ shipment_header : mode container_type_lookup ||--o{ shipment_container : type

Finance and landed cost

- CostComponentLookup [cost_component_lookup]: component_code, component_name, is_tax

- DocumentTypeLookup [document_type_lookup]
- DocumentAttachment [document_attachment]: type_id, file_path, shipment_id, po_header_id, partner_id
- TextTypeLookup [text_type_lookup]
- TextMaster [text_master]: type_id, content, PO/shipment/partner/product foreign keys

System numbering

- SysNumberRange [sys_number_ranges]: doc_category, doc_type_id, prefix, current_value, padding, include_year, is_active

Number range generation uses row-level locking to ensure unique sequence increments.

The reporting system is a metadata-driven, SQLAlchemy-based query engine that powers a dynamic UI grid.

Components

- `app/core/reports/visibility_config.py`
- Defines `VISIBILITY_REPORT_CONFIG` with:
  - report id
  - base model
  - field definitions (path, label, group, filter_type)
  - join paths and icon formatting rules
- `app/core/reports/query_engine.py`
- Builds queries dynamically based on selected columns, filters, and sort
- Applies joins only when required
- `app/api/v1/endpoints/reports.py`
- Exposes metadata, data, and export endpoints

Request flow

sequenceDiagram participant UI as Flutter UI participant API as FastAPI participant QE as ReportQueryEngine participant DB as Postgres UI->>API: GET /api/v1/reports/visibility/metadata API-->>UI: metadata (fields, filters, defaults) UI->>API: GET /api/v1/reports/visibility/data?select=... API->>QE: build_query(select, filters, sort) QE->>DB: SQL with joins/projection DB-->>QE: rows QE-->>API: rows API-->>UI: data payload

Filters and sorting

- Filters are applied per field based on `filter_type`:
- `search` -> `ilike`
- `select` -> `in`
- `date_range` and `numeric_range` -> caller should map to filter keys
- Sorting uses `sort` with optional leading `-` for DESC

Export

`GET /api/v1/reports/visibility/export`: - Uses the same query engine for consistency - Generates an XLSX stream in-memory and returns it as a download

Notes for extension

- Add new columns by editing `VISIBILITY_REPORT_CONFIG["fields"]`

FLUXPORT Backend Docs

Prerequisites

- Python 3.11+
- PostgreSQL (local or remote)

Install dependencies

```
python -m venv venv
.\venv\Scripts\activate
pip install -r requirements.txt
```

Configure database

`DATABASE_URL` is read from the environment in `app/core/config.py`. Set it to a Postgres DSN, for example:

```
set DATABASE_URL=postgresql://user:pass@host:5432/dbname
```

You can validate the resolved URL with:

```
python check_env.py
```

Run the API

```
python -m uvicorn app.main:app --port 8000 --reload
```

Health check:

```
curl http://127.0.0.1:8000/health
```

Migrations

Alembic config lives in `alembic.ini`.

Common commands:

```
alembic upgrade head
alembic revision --autogenerate -m "your message"
```

API contracts

- `openapi.json` contains the exported OpenAPI schema
- `collection.json` contains a Postman collection