

Purlox App Architecture Documentation

1. Introduction

Purlox is a modular, extensible Flask-based service designed to:

1. **Ingest** BET (Brunauer–Emmett–Teller) analyzes Excel files from users.
2. **Parse & Validate** those workbooks for structure, required data, and physical unit consistency.
3. **Persist** extracted metadata, BET parameters, and raw isotherm data in a relational database.
4. **Serve** data via a user-friendly web interface and a machine-consumable JSON API for ELNs (e.g., eLabFTW).
5. **Monitor & Scale** using Docker orchestration, CI/CD pipelines, and best-practice logging and error handling.

This reference guide covers:

1. Detailed Project Layout and Module Responsibilities
 2. Database Schema, ORM Models, Indices, and Migration Strategy
 3. End-to-End Sequence Workflows with Conditional Flows
 4. Comprehensive API Specification and Versioning Strategy
 5. Deployment Architecture, Networking, and Environment Configuration
 6. Advanced Testing, CI/CD Pipelines, and Quality Gates
 7. Observability, Logging, and Performance Considerations
 8. Security, Authentication, and Future Enhancements
-

2. Project Structure

```
/ (root)
    ├── purlox_app/          # Main Python package
    │   ├── __init__.py       # `create_app()` factory, extension init
    │   ├── app.py            # Core application setup, blueprint registration
    │   ├── config.py         # Config classes, env var loader
    │   ├── models.py          # SQLAlchemy ORM definitions
    │   ├── db_manager.py     # DB session, transactional helpers
    │   ├── excel_processor.py # Workbook validation & parsing
    │   └── api/              # API blueprint
    │       ├── __init__.py   # Blueprint factory
    │       └── routes.py      # `/api/data` and `/api/health` endpoints
    ├── web/                 # Web UI blueprint
    │   ├── __init__.py       # Blueprint factory
    │   └── routes.py          # `/upload`, `/files`, `/file/<id>` routes
    └── utils/               # Shared utilities
        ├── logger.py         # Configures Python `logging` (handlers, formatters)
        ├── validators.py      # Schema & value checks for Excel content
        └── metrics.py         # Prometheus instrumentation
    ├── templates/           # Jinja2 templates
    │   ├── base.html
    │   ├── uploads.html
    │   ├── list_files.html
    │   ├── view_files.html
    │   └── api.html
    ├── static/              # CSS, JS (Chart.js), images
    ├── tests/               # pytest-based tests
    │   ├── conftest.py        # fixtures: `app`, `client`, `db_session`
    │   ├── test_upload.py
    │   ├── test_api.py
    │   ├── test_processor.py
    │   └── test_db_manager.py
    ├── docs/                # Architecture diagrams, ER diagrams, API spec
    ├── scripts/             # Helper scripts (db migrations, seed data)
    │   ├── migrate.py
    │   └── seed_sample_data.py
    ├── docker-compose.yml    # Multi-service orchestration
    ├── Dockerfile            # Flask app build
    ├── requirements.txt       # Dependencies with pinned versions
    ├── RUN.py                # CLI entrypoint: `flask run` wrapper
    └── .github/              # GitHub Actions workflows
        └── ci.yml            # Lint, test, build, deploy
```

2.1 Module Responsibilities

2.1.1 `create_app()` and `app.py`

- **Factory Pattern:** Accepts `config_name`, applies `config.py` class.
- **Extensions:** Initialize SQLAlchemy, Flask-Migrate, CORS, PrometheusMetrics.
- **Blueprints:** Register `/web` (UI), `/api` (data & health)
- **Error Handlers:** Custom JSON responses for `HTTPError`, `ValidationError`.
- **Middleware:** Request logging, timing, authentication stub.

2.1.2 Configuration (`config.py`)

- **BaseConfig:** Shared defaults:
 - `UPLOAD_FOLDER`, `MAX_CONTENT_LENGTH` (e.g., 10MB)
 - Allowed Excel MIME types and extensions
 - `LOG_LEVEL`, metrics path
- **DevelopmentConfig:** `DEBUG=True`,
`SQLALCHEMY_DATABASE_URI=sqlite:///data/dev.db`
- **ProductionConfig:** `DEBUG=False`, Postgres/MySQL URI, security headers
- **Environment Loading:** `.env` parsed via `python-dotenv`

2.1.3 ORM Models (`models.py`)

- **FileInfo:**
 - Columns: `id` (PK), `filename`, `upload_time`, `uploader`, `checksum`
 - Unique index on `(filename, checksum)` to prevent duplicate uploads
- **BETParameter:**

- Columns: `id` (PK), `file_id` (FK), `surface_area`, `pore_volume`, `c_constant`
- Check constraints to ensure `REAL >= 0`
- **DataPoint:**
 - Columns: `id`, `param_id` (FK), `pressure`, `adsorption`
 - Composite index on (`param_id`, `pressure`)

Indexes and constraints are defined via SQLAlchemy `Index` and `CheckConstraint` for data integrity.

2.1.4 Excel Parsing (`excel_processor.py`)

- **Validation:**
 1. Ensure the workbook contains required sheets: `Metadata`, `BET`, `Isotherm`.
 2. Column headers match expected patterns (using regex).
- **Parsing Pipeline:**
 1. **Metadata Extraction:** Uploader ID, file creation timestamp, instrument details.
 2. **BET Section:** Read specific cells (e.g., `B5:B7`) for C-constant, area, volume.
 3. **Isotherm Data:** Load tabular data into `pandas.DataFrame` for row-wise parsing.
 4. **Unit Conversion:** Convert units (e.g., cm³ STP to mmol/g) based on user settings.

Return Structure:

```
{
  'file_info': FileInfo(fields),
  'bet_parameters': BETParameter(fields),
  'data_points': [DataPoint(field_dict), ...]
}
```

- **Error Handling:** Raises `ValidationError` with detailed messages; caught in route for 400 responses.

2.1.5 Database Operations (`db_manager.py`)

- **Session Management:** Uses `scoped_session` for thread safety.
 - **Atomic Transactions:** `insert_file_payload()` wraps inserts in `try/except` with rollback.
 - **Query Helpers:**
 - `list_files(page, per_page)`: Returns paginated `FileInfo` objects.
 - `get_full_payload(file_id)`: Joins across models to nest BET and data points.
 - **Migration Scripts:** Leverages Flask-Migrate; `scripts/migrate.py` automates revision creation.
-

3. Data Model & Schema

3.1 ER Diagram

See `docs/er_diagram.svg` for a visual representation of table relationships.

3.2 SQL DDL

```
CREATE TABLE file_info (
    id SERIAL PRIMARY KEY,
    filename VARCHAR(256) NOT NULL,
    checksum CHAR(64) NOT NULL,
    upload_time TIMESTAMP WITH TIME ZONE DEFAULT now(),
    uploader VARCHAR(64),
    CONSTRAINT uq_file_checksum UNIQUE (filename, checksum)
);
```

```
CREATE TABLE bet_parameter (
    id SERIAL PRIMARY KEY,
    file_id INTEGER NOT NULL REFERENCES file_info(id) ON DELETE CASCADE,
    surface_area NUMERIC CHECK (surface_area >= 0),
    pore_volume NUMERIC CHECK (pore_volume >= 0),
```

```

c_constant NUMERIC DEFAULT 0,
created_at TIMESTAMP WITH TIME ZONE DEFAULT now()
);

```

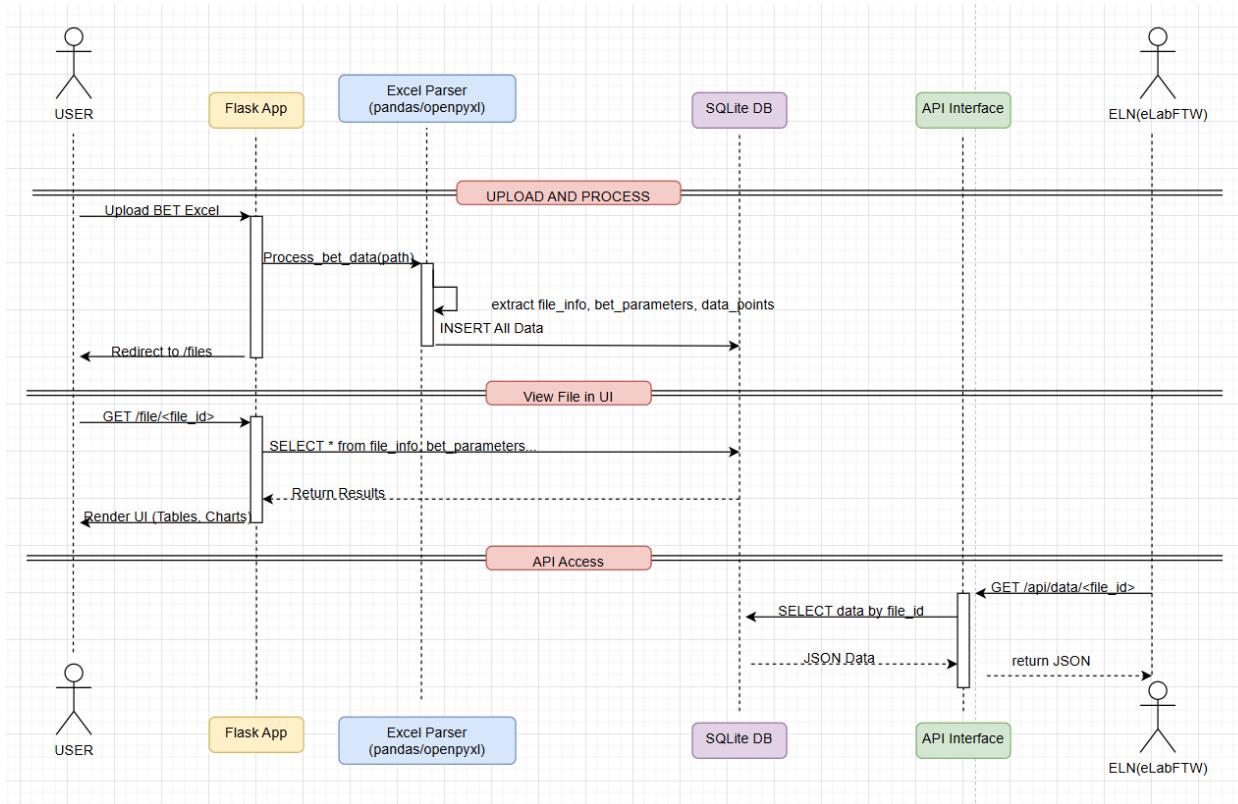
```

CREATE TABLE data_point (
    id SERIAL PRIMARY KEY,
    param_id INTEGER NOT NULL REFERENCES bet_parameter(id) ON DELETE CASCADE,
    pressure NUMERIC NOT NULL,
    adsorption NUMERIC NOT NULL,
    CONSTRAINT idx_pressure UNIQUE (param_id, pressure)
);

```

4. Core Processing Workflows

Sequence diagrams illustrate normal and error scenarios.



4.1 Upload & Processing

| Step | Component | Action |
|------|----------------|--|
| p | | |
| 1 | User (Browser) | Submits <code>POST /upload</code> with multipart form containing Excel file. |

| | | |
|---|-----------------|---|
| 2 | Web Blueprint | Validates MIME type, extension, and file size via <code>validators.py</code> . |
| 3 | excel_processor | Opens workbook, validates sheets, extracts and converts data; raises <code>ValidationError</code> on schema mismatch. |
| 4 | db_manager | Within transaction: <ul style="list-style-type: none"> Insert <code>FileInfo</code> (compute SHA-256). Insert <code>BETParameter</code> linked to file. Bulk insert <code>DataPoint</code> rows. |
| 5 | Flask App | Commits transaction and emits <code>upload_success</code> metric; OR rollback on error and return 400 with details. |
| 6 | Response | Redirects to <code>/files</code> on success; flashes messages via session. |

4.2 View File in UI

| Step | Component | Action |
|------|----------------|---|
| 1 | User (Browser) | Clicks link to <code>GET /file/<id></code> . |
| 2 | Web Blueprint | Calls <code>db_manager.get_full_payload(id)</code> ; returns nested dict. |
| 3 | Template | Populates <code>view_files.html</code> with: <ul style="list-style-type: none"> Definition list for BET params <code><canvas></code> element for isotherm plot (Chart.js) Download link for raw JSON (via <code>GET /api/data/<id></code>) |
| 4 | Browser | Renders charts and tables; JS fetches metrics endpoint for dashboard overhead (optional). |

4.3 API Data Access

| Step | Component | Action |
|------|--------------|---|
| 1 | Client (ELN) | Sends <code>GET /api/data/<id>?version=1.0</code> . |

-
- 2 API Blueprint Parses `version` param; routes to `routes.py:get_data_v1()`; fetches payload via `db_manager.get_full_payload()`.
- 3 Serializer Applies Marshmallow schemas for JSON output, omitting internal fields.
- 4 Response Returns `200 OK` with JSON; sets `Cache-Control: max-age=60` for read-heavy clients.
- 5 Error Path Invalid ID → `404 Not Found`; DB error → `500 Internal Server Error`; malformed version → `400 Bad Request`.
-

5. API Reference

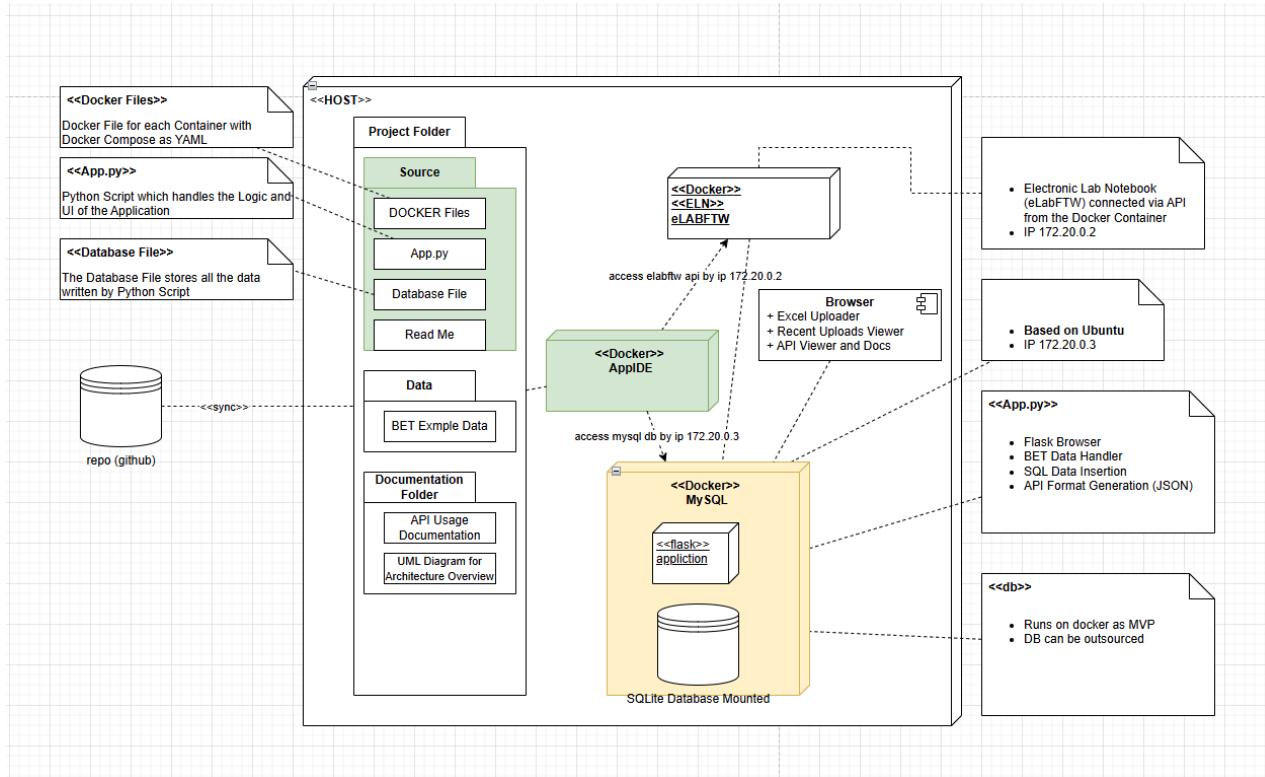
| Endpoint | Method | Query Params | Auth Required | Response Code | Notes |
|--|--------|---|---------------|------------------|--|
| <code>/api/data/<file_id></code> | GET | <code>version</code> , <code>format</code> | No* | 200, 400, 404 | JSON or CSV output via <code>?format=csv</code> |
| <code>/api/health</code> | GET | — | No | 200 | Returns <code>{status: "ok", uptime: "..."}</code> |
| <code>/api/list-file</code> s* | GET | <code>page</code> , <code>per_page</code> | No* | 200 | Paginated list of recent uploads |

* Authentication via API key header `X-API-KEY` can be enabled in future.

5.1 Versioning Strategy

- URI versioning: `/api/v1/data/<file_id>` for breaking changes.
 - Response schema evolution tracked via `version` query param and content negotiation.
-

6. Configuration & Deployment



6.1 Docker Compose

```

version: '3.8'
services:
  app:
    build:
      context: .
      dockerfile: Dockerfile
    env_file: .env.production
    volumes:
      - ./Data:/app/Data:ro
      - db_data:/app/data
    ports:
      - '5000:5000'
    healthcheck:
      test: ["CMD", "curl", "--fail", "http://localhost:5000/api/health"]
      interval: 30s
      timeout: 10s
      retries: 3
    networks:
      purlox-net:
        ipv4_address: 172.20.0.10

```

```

db:
  image: postgres:14
  env_file: .env.production
  volumes:
    - pg_data:/var/lib/postgresql/data
networks:
  purlox-net:
    ipv4_address: 172.20.0.11

elabftw:
  image: elabftw/elabftw
  ports:
    - '172.20.0.2:80:80'
networks:
  purlox-net:
    ipv4_address: 172.20.0.12

volumes:
  db_data:
  pg_data:

networks:
  purlox-net:
    driver: bridge
  ipam:
    config:
      - subnet: 172.20.0.0/16

```

- **Healthchecks:** Ensures services restart on failure.
- **Resource Limits:** `deploy.resources.limits` can be added for CPU/memory caps.

6.2 Environment Management

- `.env.development & .env.production` for secrets and URLs.
 - Use Docker secrets or Vault for sensitive credentials in production.
-

7. Testing & CI/CD

7.1 Unit & Integration Tests (pytest)

- **Fixtures:**

- `db_session`: Isolated transaction per test, rolled back.
- `excel_file_path`: Parameterized sample files.

- **Coverage Targets:**

- $\geq 90\%$ for parsers and DB logic.
- $\geq 80\%$ for API and UI routes.

7.2 GitHub Actions Workflow (`.github/ci.yml`)

```
name: CI Pipeline
on: [push, pull_request]
jobs:
  lint:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - name: Set up Python
        uses: actions/setup-python@v3
        with: python-version: '3.11'
      - run: pip install flake8 black bandit
      - run: flake8 purlox_app/ tests/
      - run: black --check .
      - run: bandit -r purlox_app/
  test:
    runs-on: ubuntu-latest
    services:
      postgres:
        image: postgres:14
        env:
          POSTGRES_USER: test
          POSTGRES_PASSWORD: test
          POSTGRES_DB: test_db
        ports: ['5432:5432']
    steps:
      - uses: actions/checkout@v3
      - name: Install dependencies
        run: |
          pip install -r requirements.txt psycopg2-binary
      - name: Run migrations
        run: python scripts/migrate.py <--apply-->
      - name: Run tests
```

```
run: pytest --maxfail=1 --disable-warnings -q
```

7.3 Release & Deployment

- **Semantic Versioning:** Git tags `v<MAJOR>.<MINOR>.<PATCH>`.
 - **Container Registry:** Push Docker image on tag to Docker Hub or ECR.
 - **Deployment:** Pull image and `docker-compose up -d` on target host or orchestrate via Kubernetes (future).
-

8. Observability & Best Practices

- **Logging:** Structured JSON logs; forwarded to ELK or Splunk.
 - **Metrics:** Prometheus endpoint at `/metrics`; track request rates, error rates, latency histograms.
 - **Tracing:** OpenTelemetry integration for distributed traces.
 - **Security:**
 - Enforce TLS (redirect HTTP to HTTPS).
 - Input sanitization in uploads and query params.
 - Rate limiting via Flask-Limiter.
 - **Scaling:**
 - Horizontal: Multiple app instances behind Nginx or AWS ALB.
 - Vertical: Increase CPU/memory, switch to Postgres cluster.
 - **Data Migration:**
 - Use Flask-Migrate `alembic` scripts; backups via cron jobs and volume snapshots.
-

9. Future Enhancements

- **Background Processing:** Offload parsing to Celery + RabbitMQ/Redis.
- **User Management:** Integrate OAuth2 for user authentication/authorization.
- **Interactive Dashboards:** Add Plotly Dash or Grafana embedding.
- **Multi-format Support:** Handle CSV/JSON lab data imports.
- **Cloud Deployment:** Helm charts for Kubernetes; managed DB services.