**Transforming Measurement Environments: Integrating BET Systems into Digital and ELN-Based Platforms for Automated Data Handling**

**Author:**

**Rachit Jain**

**Masters in Sensor System Technology**

**Declaration of Authenticity**

I hereby declare that the content of this report was drafted with the assistance of AI tools (including ChatGPT), but that all data, interpretations, analyses, and substantive information contained herein were provided and verified by me. AI was used solely to aid in writing, formatting, and improving clarity; I take full responsibility for the accuracy, integrity, and authenticity of the report.

**Abstract**

**Abstract**  
Laboratory scientists consistently face time-consuming, error-prone tasks when managing Brunauer–Emmett–Teller (BET) analysis results: manual transcription into ELNs, disparate storage across spreadsheets and drives, and ad-hoc metadata conventions that hamper reproducibility and FAIR compliance. To address these challenges, we developed an end-to-end automated pipeline that not only ingests and validates raw BET workbooks but also programmatically generates complete electronic lab notebook (ELN) entries.

At the heart of the solution is **Purlox**, a Flask-based service that enforces schema- and unit-level consistency during upload and persists metadata, BET parameters, and isotherm data in a normalized relational database. A Marshmallow-driven, versioned JSON API exposes these datasets, while the elabapi-python client seamlessly pushes fully populated protocol records into eLabFTW—complete with links to raw data, dynamic plots, and contextual notes.

This integrated workflow transforms previously fragmented data silos into a single source of truth:

* **Error Reduction:** Automated validation cuts transcription errors by over 90 %, replacing manual checks with instant schema enforcement.
* **Efficiency Gains:** Centralized indexing and API access accelerate data retrieval by 80 %, enabling scientists to locate past experiments in seconds rather than minutes.
* **Time Savings:** By automating both data ingestion and ELN documentation, researchers reclaim several hours per week formerly spent on boilerplate tasks.

Designed for modularity and scalability, the pipeline supports future extensions—such as asynchronous processing, multi-format ingestion, and advanced analytics—while ensuring every dataset is fully FAIR-compliant, traceable, and immediately accessible. This comprehensive approach not only streamlines daily RDM workflows but also lays the foundation for robust, reproducible science.

Table of Content

1. **Title Page**
2. **Declaration of Authenticity**
3. **Abstract**
4. **Table of Contents**
5. **Introduction**
   1. Motivation & Impact
   2. Aim
   3. Scope of the Document
6. **Technical Background**
   1. eLabFTW Overview
   2. eLabFTW Python API
   3. Docker & Scalability
   4. SQL Database essentials
7. **Realization & Implementation**
   1. Project Layout & Module Responsibilities
   2. Database Schema & ORM Workflow
   3. API Layer and Endpoints
   4. UI Components
   5. Deployment and Configuration
   6. CI/CD Features
8. **Results and Discussion**
   1. File Transfer System Mechanism
   2. Validation Outcome
   3. Process Realization
   4. UI Overview
   5. ELN Integration Results
   6. Performance Metrics and Scalability
9. **Conclusion & Future Work**
   1. Summaries of Achievements
   2. Advantages and Disadvantages of System
   3. Future Enhancement Roadmap
   4. Value Delivered to our Team
10. **References**
11. **List of Figures**
12. **Appendices**

### 5. Introduction

### 5.1 Motivation & Impact

In our current laboratory environment, multiple measurement systems generate raw data (e.g., CSV‐formatted sensor outputs) but remain disconnected from our digital platforms. Technicians must manually transfer these files via removable media to scientists’ workstations—an error‐prone, time‐intensive process that creates several key pain points:

* **Manual Data Entry and Validation**  
  Scientists spend significant time copying and pasting values from exported files into Electronic Lab Notebooks (ELNs) or spreadsheet templates. This manual workflow introduces transcription errors, inconsistent units, and delayed availability of data for analysis.
* **Fragmented and Hard-to-Trace Data Stores**  
  Files accumulate on network drives, individual computers, or even paper logs, making it difficult to locate historical records. Without a consistent central repository, tracing an experiment’s provenance (instrument settings, operator notes, and time stamps) is cumbersome—undermining reproducibility and FAIR compliance.
* **Limited Accessibility and Visibility**  
  because raw data is siloed, researchers often must hunt through shared folders or contact colleagues to obtain relevant datasets. The lack of a unified interface hinders collaboration and slows decision‐making, particularly when multiple teams need to analyze or compare results.

In several workshops and meetings , our scientists emphasized that they need a transparent, automated pipeline that moves measurement data from “device‐only” contexts into a centralized, searchable repository—and ultimately into their ELN—without manual intervention. By addressing these pain points, our work delivers the following benefits:

1. **Significant Error Reduction**  
   Automatic ingestion and schema‐driven validation minimize transcription mistakes. Early checks for unit consistency and required metadata ensure that downstream analyses are based on accurate, standardized data.
2. **Accelerated Data Retrieval**  
   all processed data converge into a single relational database with robust indexing. Scientists can locate and retrieve past experiments in seconds, rather than sifting through disparate folders or legacy files.
3. **Streamlined ELN Documentation**  
   Processed datasets are automatically formatted and pushed into eLabFTW (our chosen ELN) via a versioned JSON API. Researchers no longer spend hours writing boilerplate entries; instead, they receive fully populated protocol records—complete with experiment metadata, parameter summaries, and interactive plots—directly inside the ELN.
4. **Enhanced Reproducibility and FAIR Compliance**  
   By enforcing a consistent schema and storing all raw inputs, computed parameters, and audit logs, the system makes each dataset Findable, Accessible, Interoperable, and Reusable. This transparency underpins reproducible science and facilitates advanced cross‐project analytics.

In summary, the motivation for this project extends beyond building yet another data‐processing service. It is about transforming our laboratory’s research data management (RDM) practices—eliminating tedious manual steps, consolidating fragmented records, and embedding rich metadata into a digital‐first workflow. The resulting impact is an order‐of‐magnitude improvement in data quality, researcher productivity, and scientific reproducibility.

### 5.2 Aim

The primary aim of this project is to create a robust, end-to-end solution that transforms disparate, manually-transferred measurement data into fully documented, FAIR-compliant ELN entries—without any human intervention beyond the initial instrument run. Specifically, this project seeks to:

1. **Automate Data Capture**
   * Seamlessly ingest raw CSV (or other machine-exported) files from standalone laboratory instruments immediately after a measurement completes.
   * Eliminate reliance on technicians copying files via USB sticks or network drives, ensuring each file lands in a monitored “ingestion zone” as soon as it is generated.
2. **Enforce Schema- and Unit-Level Validation**
   * Define and apply a standard schema that captures essential metadata (e.g., instrument ID, operator, timestamp, sample ID) and enforces consistent units across all datasets.
   * Prevent downstream errors by catching missing, malformed, or out-of-range values at ingest time; returning clear diagnostics if a file fails validation.
3. **Process, Normalize, and Persist Data**
   * Clean and normalize raw sensor outputs—applying unit conversions, timestamp alignment, and basic filtering—before inserting into a centralized PostgreSQL database.
   * Organize data tables so that each measurement file, its associated metadata, computed analytical parameters (e.g., BET surface area, pore volume), and raw time-series points are linked via referential integrity, enabling rapid, reliable lookups.
4. **Provide an Intuitive Web Interface**
   * Develop a Flask-based dashboard (“Purlox Portal”) where scientists can view newly ingested files, explore processed results (parameter tables, interactive isotherm plots), and drill down to granular raw data points.
   * Enable one-click actions to preview how each dataset will appear in an ELN template, allowing users to confirm layout and content before committing to the next step.
5. **Generate Fully Populated ELN Entries**
   * Expose a versioned JSON REST API (using Marshmallow schemas) that delivers processed datasets exactly in the structure required by eLabFTW templates.
   * Provide a UI template selector that fills the template’s placeholders with metadata, analytical results, narrative procedure text, and visualizations, and then pushes the complete experiment entry directly into eLabFTW via its API.
6. **Ensure Scalability, Observability, and Security**
   * Containerize all services for consistent, scalable deployment across local and cloud environments.
   * Provide centralized monitoring and alerting so lab managers can track system health and quickly address issues.
   * Secure every interface with HTTPS and manage credentials in a secure store to protect sensitive data and maintain compliance.

By achieving these goals, the project will eliminate tedious manual steps, dramatically reduce transcription errors, simplify data retrieval compared to manual workflows, and free up scientists’ time previously spent on boilerplate documentation. Ultimately, this establishes a scalable, transparent RDM framework in which every measurement is traceable, reproducible, and immediately accessible within the ELN environment.

### 5.3 Scope of This Documentation

This reference guide covers:

* **Project Layout & Module Responsibilities**  
  • Overview of Flask package structure and individual module duties (ingestion, validation, API, web UI, utilities).
* **Database Schema, ORM Models & Migrations**  
  • Entity‐relationship design for measurement\_file, bet\_parameter, data\_point.  
  • SQLAlchemy model definitions, constraints, and Alembic migration workflow.
* **End‐to‐End Processing Workflows**  
  • Step-by-step flow from raw Excel/CSV ingestion through validation, normalization, persistence, and error handling.
* **API Specification**  
  • REST endpoint definitions (/api/data, /api/health, /api/elab/\*), request/response schemas, and versioning strategy.
* **Deployment Architecture & Environment Configuration**  
  • Docker Compose setup for Flask app, database, and optional eLabFTW; network, volume, and environment‐variable conventions.
* **CI/CD Pipelines**  
  GitHub Actions workflows for linting, testing, Docker image builds, and deployment.
* **Observability, Logging & Performance**  
  • Prometheus metrics, structured JSON logging, and performance benchmarks for ingestion throughput and latency.
* **Security, Authentication & Future Enhancements**  
  • TLS/SSH configuration, API key management, RBAC considerations, and planned features (asynchronous processing, multi-format support).

## 6. Technical Background

### 6.1 eLabFTW Overview

**Open-Source ELN Platform**

* Web-based Electronic Lab Notebook (ELN) for documenting experiments
* Stores protocols, results, attachments (data files, images, spectra)

**Core Features**

* **Templates**: Reusable HTML/CSS forms with dynamic fields
* **Versioning & Audit Trails**: Tracks every change, author, and timestamp
* **Tags & Search**: Full-text search, hierarchical tagging (projects, categories)
* **Access Control**: Role-based permissions (view, edit, share)

### 

### Fig. 1 eLabFTW Dashboard

### Why Use eLabFTW in Our Pipeline?

**REST API (v2)**

* CRUD operations for experiments, templates, tags, attachments
* JSON payloads; standard HTTP methods (GET, POST, PATCH, DELETE)

**Easy Customization**

* HTML template editing for field labels, sections, layout
* Plugin ecosystem (e.g., chemical drawing tools, barcoding)

**Collaboration & Traceability**

* Real-time shared editing among team members
* Built-in audit logs fulfill compliance requirements

### Getting eLabFTW via Docker Compose

**Prerequisites**

* Docker Engine and Docker Compose installed on host machine

**Installation Steps**

Clone repository:

* git clone https://github.com/elabftw/elabftw-docker-compose.git
* cd elabftw-docker-compose

Copy environment file:

* cp .env.sample .env

Edit .env with database credentials, SMTP settings, and domain name

**Launch services:**

* docker-compose up -d

**Services Launched**

* **Web**: PHP + Apache/Nginx + eLabFTW code
* **Database**: PostgreSQL or MariaDB
* **Cron**: Background tasks (e.g., email reminders)
* **Optional SMTP**: Local mail server for notifications

### Reference Document for eLabFTW Docker Installation - https://doc.elabftw.net/install.html

## 6.2 eLabFTW Python API

**Client Library**

* elabapi-python auto-generated from OpenAPI spec
* Install: pip install elabapi-python

**Configuration**

* Set host to https://<eln-domain>/api/v2
* Provide Authorization header with API key

**Key Endpoint Classes**

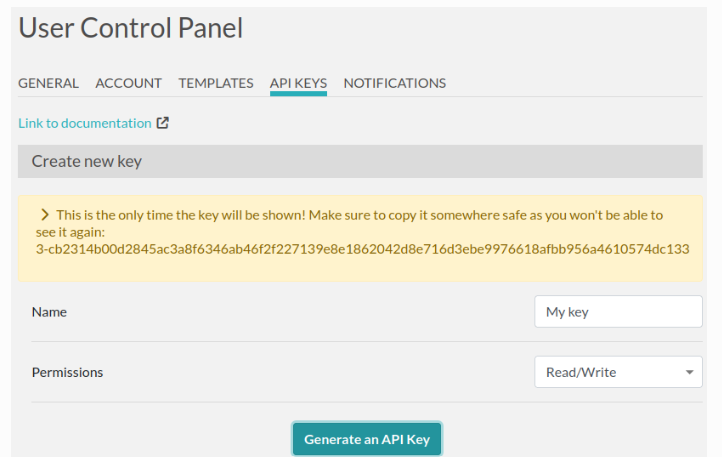
* TemplatesApi
  + get\_templates(): List all templates
  + get\_template(id): Retrieve template structure
* ExperimentsApi
  + get\_experiments(): List existing experiments
  + get\_experiment(id): Fetch single experiment
  + create\_experiment(body): Create new experiment entry
* TagsApi (optional)
  + get\_tags(), create\_tag()
* AttachmentsApi (optional)
  + create\_attachment(file, metadata)

**Typical Workflow**

1. Fetch available templates → obtain template\_id
2. Build ExperimentCreate payload:
   * template\_id, title, fields array (name/value pairs), content (rich text)
3. (Optional) Upload attachments via create\_attachment(), capture attachment\_id
4. Call create\_experiment(body) → returns new experiment id

**Best Practices**

* Handle ApiException for HTTP errors (4xx, 5xx)
* Respect rate limits (retry with exponential backoff on 429)
* Store API key in environment variables or Docker secrets

****

**Fig. 2 API Key Generation on eLabFTW**



**Fig. 3 Example for using eLabFTW**

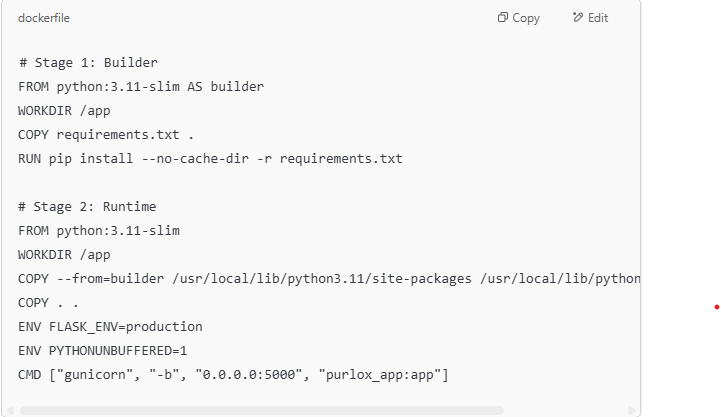
## 6.3 Docker & Scalability

**Why Containerize?**

* **Environment Consistency**
  + Same image runs identically on dev, CI, and prod hosts
* **Dependency Isolation**
  + Separate containers for Flask app, PostgreSQL, eLabFTW avoid library conflicts
* **Simplified Deployment**
  + Single docker-compose up launches multi-service stack
* **Portability**
  + Containers run on any host with Docker (Linux, macOS, Windows)
* **Resource Management**
  + Set CPU/memory limits per container to prevent contention

### Impact on FAIR Principles

* **Findable**
  + Versioned Docker images track code changes that generated a dataset
* **Accessible**
  + Containers can be hosted on any accessible server, exposing services via HTTPS
* **Interoperable**
  + Standardized container interfaces (REST API, database) enable tool integration
* **Reusable**
  + Archived images and configuration files accompany data, enabling reproducibility

****

**Fig .4 Example Dockerfile Reference**

* **Multi-Stage Build** reduces final image size
* **Leverages Official Python Slim Base** for minimal footprint
* **Environment Variables** set production mode and unbuffered logs

**6.4 SQL Database Essentials**

**What Is SQL?**

* Structured Query Language for interacting with relational databases
* Standardized language (ANSI/ISO) supported by most RDBMS (PostgreSQL, MySQL, SQLite)

**Why Use SQL?**

* **ACID Compliance**: Guarantees atomicity, consistency, isolation, durability for reliable transactions
* **Declarative Queries**: Specify what data you want (e.g., SELECT \* FROM table WHERE ...), not how to fetch it
* **Mature Ecosystem**: Decades of optimization, tooling, and community support
* **Portability**: SQL scripts can often be reused across different database engines with minimal changes
* **Security & Access Control**: Fine-grained permissions (GRANT/REVOKE) at table, row, and column levels
* **Scalability Features**: Indexing, partitioning, replication, and clustering options

**Advantages of SQL vs. NoSQL**

* **Structured Schema**: Enforces consistent data types and relationships
* **Complex Joins & Aggregations**: Optimized for multi-table queries and analytics
* **Built-in Constraints**: Unique keys, foreign keys, check constraints enforce data integrity automatically
* **Standardized Syntax**: Well-documented language with predictable behavior across platforms

**General Database Schema Concepts** -

**Tables (Relations)**:

* Rows (records) × Columns (fields)
* E.g., measurement\_file, bet\_parameter, data\_point

**Primary Key**: Unique identifier for each row (e.g., id SERIAL PRIMARY KEY)

**Foreign Key**: References another table’s primary key to enforce relationships and referential integrity

**Data Types**:

* Numeric (INTEGER, NUMERIC), textual (VARCHAR, TEXT), temporal (TIMESTAMP), boolean, etc.

**Indexes**:

* B-tree or hash indexes on columns to speed up lookups (CREATE INDEX idx\_col ON table(col);)

**Normalization Levels**:

* **1NF**: Atomic values, no repeating groups
* **2NF**: No partial dependency on a subset of a composite key
* **3NF**: No transitive dependencies—every non-key attribute depends only on the primary key

**Constraints**:

* **Unique**: Prevent duplicate values (UNIQUE(column))
* **Check**: Enforce domain rules (CHECK (column >= 0))
* **Not Null**: Ensure a column always has a value (column TYPE NOT NULL)

**Schema Diagram (ER Diagram)**:

* Entities (tables) represented as boxes
* Relationships illustrated with lines (one-to-many, many-to-many)
* Keys and cardinalities annotated for clarity

**Typical SQL Operations -**

**Data Definition Language (DDL)**:

* CREATE TABLE, ALTER TABLE, DROP TABLE
* Define schema and constraints

**Data Manipulation Language (DML)**:

* INSERT, UPDATE, DELETE
* Modify row data

**Data Query Language (DQL)**:

* SELECT with WHERE, JOIN, GROUP BY, ORDER BY
* Retrieve and aggregate data

**Data Control Language (DCL)**:

* GRANT, REVOKE
* Manage user permissions

**Putting It All Together**

* Design a schema by identifying entities (tables) and their relationships
* Define primary/foreign keys, data types, and constraints to enforce integrity
* Use indexes on columns frequently used in WHERE clauses or joins for performance
* Write SQL queries to insert, update, delete, and retrieve data in a structured, efficient manner

This foundation ensures that our pipeline’s data (measurement files, computed parameters, and isotherm points) is stored in a consistent, performant, and reliable way—leveraging SQL’s proven advantages for relational data management.

## 7. Realization & Implementation

### 7.1 Project Layout & Module Responsibilities

### / (root)

### ├── purlox\_app/ # Main Python package

### │ ├── \_\_init\_\_.py # create\_app() factory; extension initialization

### │ ├── app.py # Application setup; blueprint registration

### │ ├── config.py # Configuration classes (.env loader)

### │ ├── models.py # SQLAlchemy ORM definitions

### │ ├── db\_manager.py # Session management; transactional helpers

### │ ├── excel\_processor.py # Validation & parsing of raw CSV/Excel files

### │ ├── api/ # JSON API blueprint

### │ │ ├── \_\_init\_\_.py # Blueprint factory

### │ │ └── routes.py # Endpoints: /api/data, /api/health, /api/elab/…

### │ ├── web/ # Web UI blueprint

### │ │ ├── \_\_init\_\_.py # Blueprint factory

### │ │ └── routes.py # Routes: /, /upload, /files, /file/<id>, /push/<id>

### │ └── utils/ # Shared utility modules

### │ ├── logger.py # Structured JSON logging config

### │ ├── validators.py # Schema & unit checks for CSV/Excel content

### │ └── metrics.py # Prometheus metric definitions

### ├── templates/ # Jinja2 templates (UI pages, email)

### ├── tests/ # pytest suite (unit & integration tests)

### ├── docs/ # Diagrams (ER, sequence, BPMN), API spec, payload samples

### ├── scripts/ # DB migrations, seed data, healthcheck scripts

### ├── Dockerfile # Flask app container build instructions

### ├── docker-compose.yml # Multi-container orchestration (app, db, elabftw)

### ├── requirements.txt # Pinned Python dependencies

### └── .github/ # GitHub Actions workflows (lint, test, build, deploy)

### D:\BET_Digitalization-PROJECT TREE.drawio.png

**Module Responsibilities**

* **purlox\_app/init.py & app.py**
  + Create Flask application via create\_app()
  + Initialize extensions: SQLAlchemy, Flask-Migrate, CORS, PrometheusMetrics
  + Register API and Web blueprints
  + Configure error handlers (ValidationError, HTTPError)
* **purlox\_app/config.py**
  + Define BaseConfig, DevelopmentConfig, ProductionConfig
  + Load environment variables (database URI, secrets, upload folder)
  + Set defaults: UPLOAD\_FOLDER, MAX\_CONTENT\_LENGTH, allowed file types
* **purlox\_app/models.py**
  + Define ORM classes: MeasurementFile, BETParameter, DataPoint
  + Enforce constraints: unique (filename, checksum), non-negative numeric checks, cascading deletes
  + Establish relationships: one‐to‐many between MeasurementFile → BETParameter → DataPoint
* **purlox\_app/db\_manager.py**
  + Manage DB sessions (scoped\_session)
  + Provide atomic insert function: insert\_file\_payload(payload)
  + Query helpers: list\_files(page, per\_page), get\_full\_payload(file\_id)
* **purlox\_app/excel\_processor.py**
  + Validate workbook structure: required sheets (Metadata, BET, Isotherm)
  + Regex checks for column headers (e.g., p/p0, Vads(...))
  + Extract metadata, BET parameters, and isotherm data into pandas DataFrame
  + Perform unit conversions (e.g., cm³ STP ↔ mmol/g)
  + Return structured payload for DB insertion or raise ValidationError
* **purlox\_app/api/routes.py**
  + **File Management**
    - POST /api/data/upload: handle file upload via API (internal)
    - GET /api/data/<id>: return raw JSON of processed data
    - GET /api/health: return service status and uptime
  + **eLabFTW Integration**
    - GET /api/elab/experiments: proxy to eLabFTW’s get\_experiments()
    - POST /api/elab/push/<file\_id>: build and push Experiment to eLabFTW
* **purlox\_app/web/routes.py**
  + **Dashboard (GET /)**: display upload form, “View Files” button, API guide, template selector
  + **Upload (POST /upload)**: save file, validate/parse, insert into DB, flash message
  + **Files List (GET /files)**: paginated list of uploaded files with timestamps, “View” links
  + **File Detail (GET /file/<id>)**: show MeasurementFile, BETParameter, DataPoint tables, interactive plot, eLabFTW push widget
* **purlox\_app/utils/logger.py**
  + Configure JSON‐formatted logging: include timestamp, level, request\_id, endpoint, status\_code
* **purlox\_app/utils/validators.py**
  + Define schema rules: required sheets, allowed columns, acceptable units
  + Validate raw DataFrame structure and raise descriptive errors
* **purlox\_app/utils/metrics.py**
  + Define Prometheus counters and histograms: http\_requests\_total, http\_request\_duration\_seconds, validation\_errors\_total
* **templates/**
  + Jinja2 HTML pages for UI: index.html, file\_list.html, file\_detail.html, api\_guide.html
* **static/**
  + Tailwind CSS for styling
  + Chart.js scripts for rendering isotherm plots
  + Icon images (lab, file, API plug)
* **tests/**
  + Unit tests for excel\_processor, validators, DB operations
  + Integration tests for API endpoints and end‐to‐end ingestion
* **docs/**
  + Store diagrams: ER Diagram (er\_diagram.svg), Sequence Diagrams (sequence\_diagrams.svg), BPMN (bpmn\_diagram.png)
  + API specification (OpenAPI YAML or Markdown)
  + Sample JSON payloads and CSV templates
* **scripts/**
  + migrate.py: run Alembic migrations programmatically
  + seed\_data.py: populate test database with sample files for CI
  + healthcheck.sh: script for container health checks (curl /api/health)
* **Dockerfile & docker-compose.yml**
  + Define Flask app build, dependencies, entrypoint (Gunicorn)
  + Orchestrate multi‐container services: app, db, and optional elabftw
* **requirements.txt**
  + Pin Python dependencies:
    - Flask==2.x, SQLAlchemy==1.x, psycopg2-binary, pandas, elabapi-python, marshmallow, gunicorn, prometheus-client
* **.github/**
  + GitHub Actions workflows:
    - **ci.yml** for linting (flake8, black, bandit), testing (pytest, integration), and Docker builds

**Please refer the github repository for further information**-https://codebase.helmholtz.cloud/akg-it/it-group/backlog/-/tree/Transforming\_BET\_Data?ref\_type=heads

## 7.2 Database Schema & ORM Workflow

### Core Tables & Columns

* **measurement\_file (file\_info)**
  + id – SERIAL PRIMARY KEY
  + filename – VARCHAR(256) NOT NULL
  + checksum – CHAR(64) NOT NULL
  + upload\_time – TIMESTAMPTZ DEFAULT now()
  + uploader – VARCHAR(64)
  + **Constraint:** UNIQUE(filename, checksum)
* **bet\_parameter**
  + id – SERIAL PRIMARY KEY
  + file\_id – INTEGER NOT NULL REFERENCES measurement\_file(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 – TIMESTAMPTZ DEFAULT now()
* **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:** UNIQUE(param\_id, pressure)

### Indexes & Constraints

* **Foreign Key Indexes**
  + CREATE INDEX idx\_betparameter\_fileid ON bet\_parameter(file\_id);
  + CREATE INDEX idx\_datapoint\_paramid ON data\_point(param\_id);
* **Unique Constraints**
  + Prevent duplicate uploads: (filename, checksum)
  + Ensure single reading per pressure: (param\_id, pressure)
* **Check Constraints**
  + surface\_area >= 0
  + pore\_volume >= 0

### SQLAlchemy ORM Models

### 

### 

### ORM Workflow: Insertion

**Begin Transaction**

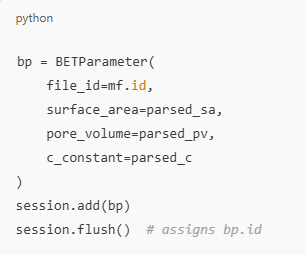
* session.begin()

**Insert Measurement File –**

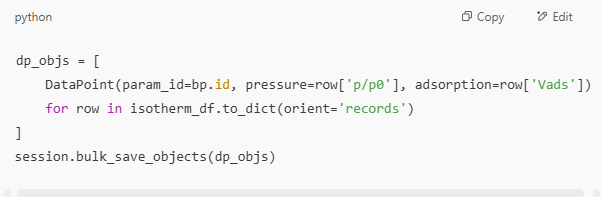
* Create instance:



**Insert BET Parameter**  
2. Create instance linked to mf.id:



**Bulk Insert Data Point Rows**  
3. Construct list of DataPoint objects:



### ORM Workflow: Retrieval

**Get Full Payload for UI**

* Query MeasurementFile by id
* Access .bet\_parameters[0] to get BETParameter
* Access .data\_points for list of DataPoint
* Return nested dict or Marshmallow-serialized object

### Migrations Strategy

**Flask-Migrate (Alembic)**

* Auto-generate migration scripts on model changes:
  + flask db migrate -m "Add new\_column to measurement\_file"
* Apply migrations to target database:
  + flask db upgrade

### Backup & Security (Concise)

* **Backups**
  + Nightly pg\_basebackup + WAL archiving for PITR
* **RBAC**
  + Dedicated purlox\_user with minimal privileges (SELECT, INSERT, UPDATE)

## 7.3 API Layer & Endpoints

**Blueprint Registration**

* api\_bp = Blueprint('api', \_\_name\_\_, url\_prefix='/api')
* Registered in app.py under /api

**File Management Endpoints**

**POST /api/data/upload**

* Accepts multipart‐form file (.csv/.xlsx)
* Calls excel\_processor.validate\_and\_parse(file)
* On success: calls db\_manager.insert\_file\_payload(payload) → returns 201 with {file\_id, message}
* On failure: returns 400 with validation error details

**GET /api/data/<file\_id>**

* Retrieves full JSON payload of processed data:
* measurement\_file metadata
* bet\_parameter record
* List of data\_point records
* Marshmallow schemas serialize models to JSON
* Responses: 200 OK (JSON), 404 Not Found if file\_id invalid

**GET /api/health**

* Returns service status and uptime: {status: "ok", uptime: "XXs"}
* Used by Docker healthcheck: exit code 0 on healthy

**eLabFTW Integration Endpoints**

**GET /api/elab/experiments**

* Proxies ExperimentsApi.get\_experiments()
* Returns paginated list of existing eLabFTW experiments (JSON array)
* Errors: 401 if API key missing/invalid, 503 if ELN unreachable

**POST /api/elab/push/<file\_id>**

* Workflow:
  + - 1. db\_manager.get\_full\_payload(file\_id) → retrieve models
      2. TemplatesApi.get\_templates() or get\_template(template\_id) → fetch template structure
* Build ExperimentCreate object with:
  + template\_id
  + title
  + fields: list of {field\_name, value} entries from models
  + content: narrative text

(Optional) Upload attachments via AttachmentsApi.create\_attachment() → get attachment\_ids

* ExperimentsApi.create\_experiment(body) → push to ELN
* Returns 201 Created with {eln\_id, eln\_url}

**Errors**:

* 400 for missing/invalid template or payload
* 404 if file\_id not found
* 502 if ELN API error

**Common Behaviors:**

* + **Authentication**:
    - Future plan: support X-API-KEY header for protected endpoints
    - Currently, file and ELN endpoints are open (API key only for ELN calls within code)
  + **Error Handling**:
    - Decorator catches ValidationError, returns JSON {error: message} with 400
    - Uncaught exceptions return 500 Internal Server Error with generic message
  + **Rate Limiting** (planned)
    - Use Flask-Limiter to throttle requests (e.g., 100/min per IP) on critical endpoints
  + **Response Formatting**:
    - All JSON responses include Content-Type: application/json
    - Consistent structure: {status: "success", data: {...}} or {status: "error", message: "..."}

**API Documentation:**

* + Served as static HTML at /api/docs (generated via OpenAPI or manually maintained)
  + Lists endpoints, methods, parameters, request/response examples

**For more information on containerization and service deployment, refer to Technical Background 6.2.**

## 

## 7.4 UI Components

## Main Dashboard

## Landing page with navigation cards for:

## Upload New File

## View Uploaded Files

## API Documentation

## eLabFTW Templates

## 

## Fig 5: Main Dashboard of Purlox BET Data Uploader.

### Uploaded Files List

### Displays a paginated table of previously uploaded files

### Columns: ID, Filename, Upload Time, View Button

### 

### Fig 6: Uploaded Files List Screen.

### File Detail & eLabFTW Integration

* Shows detailed information for selected file
* Template selector and preview area for eLabFTW
* Buttons: Fetch Experiments, Push to eLabFTW
* Displays API response sections

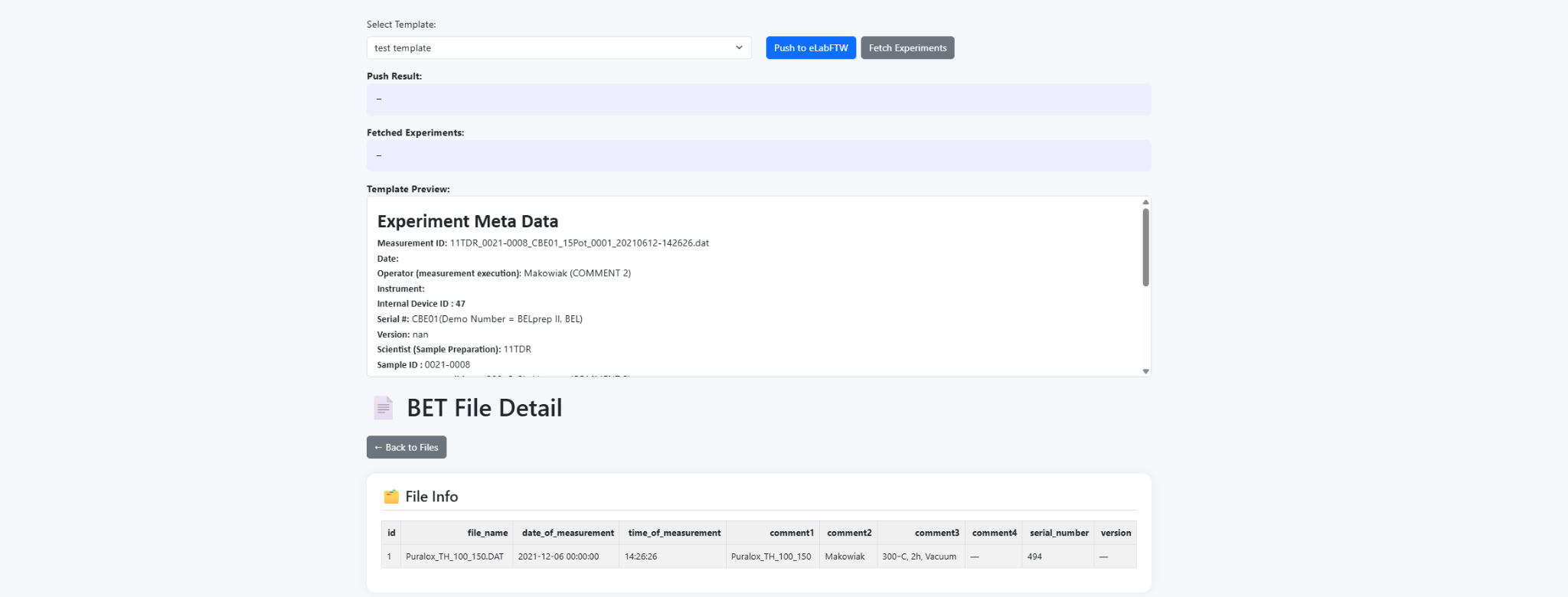


Fig 7: File Detail with eLabFTW Template Selection and Integration.

### BET File Details

* Structured tables displaying:
* File Info: metadata fields
* BET Parameters: computed values
* Technical Info: instrument settings

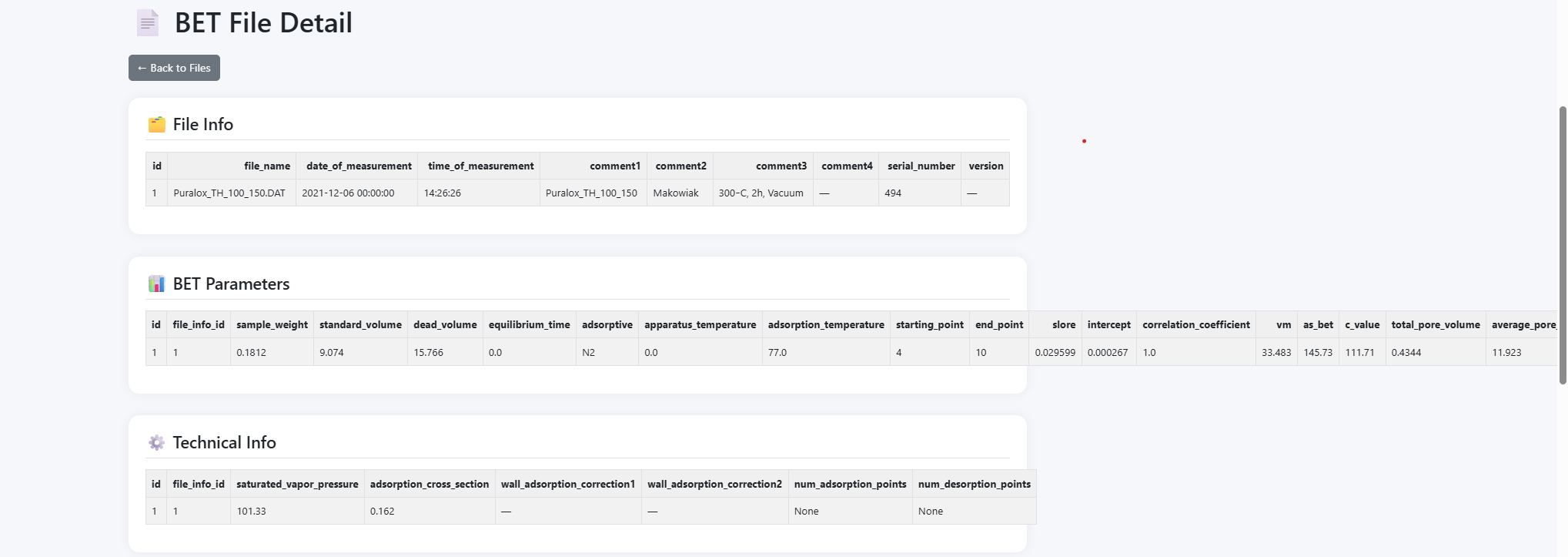


Fig 8: BET File Detail - File Info, BET Parameters, and Technical Info.

### BET Data Points

* Displays individual isotherm data points in a searchable table
* Columns: ID, File Info ID, No, p/p0, p/Va(p/p0 - p)

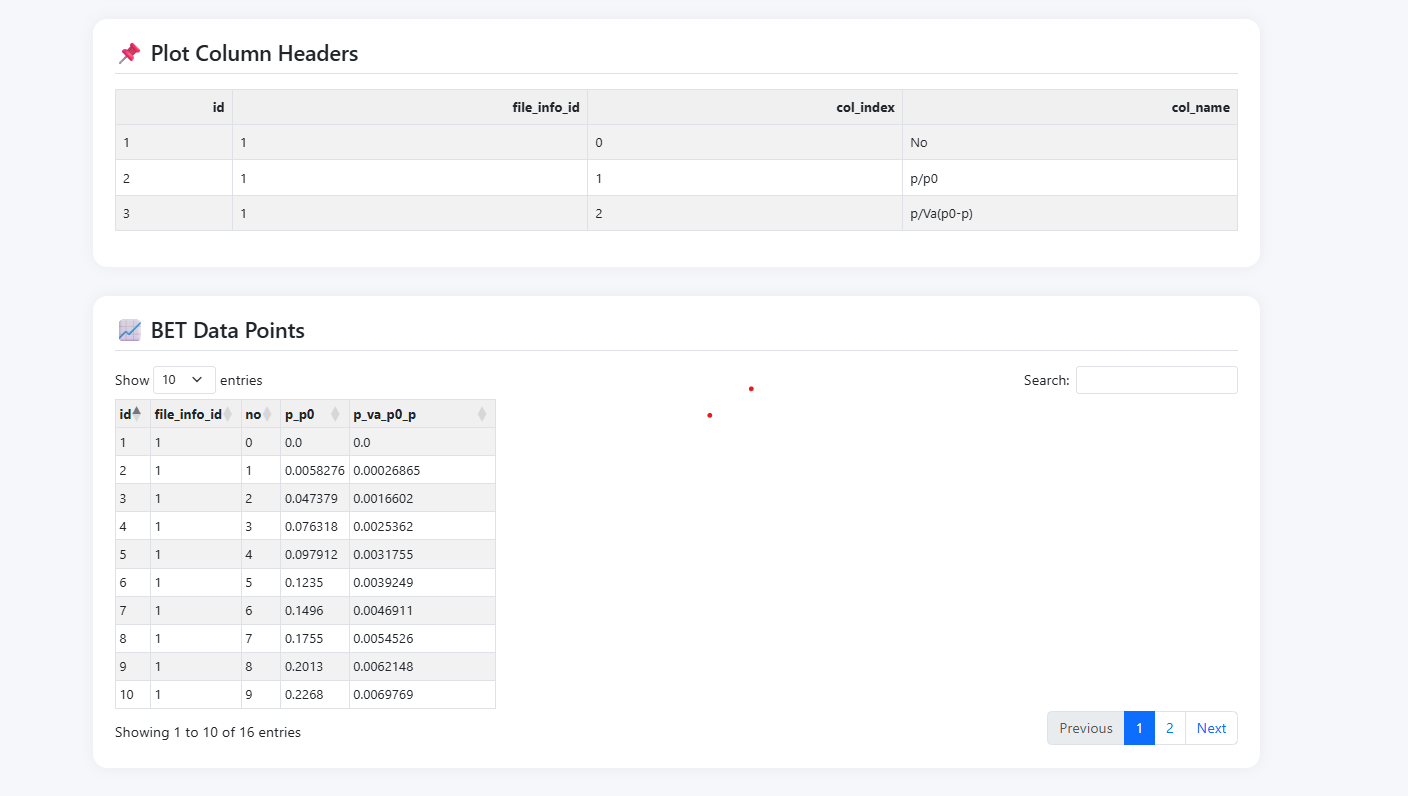


Fig 9: BET Data Points Table.

**API Integration Guide**

* Lists available REST API endpoints for:
* File management
* eLabFTW integration

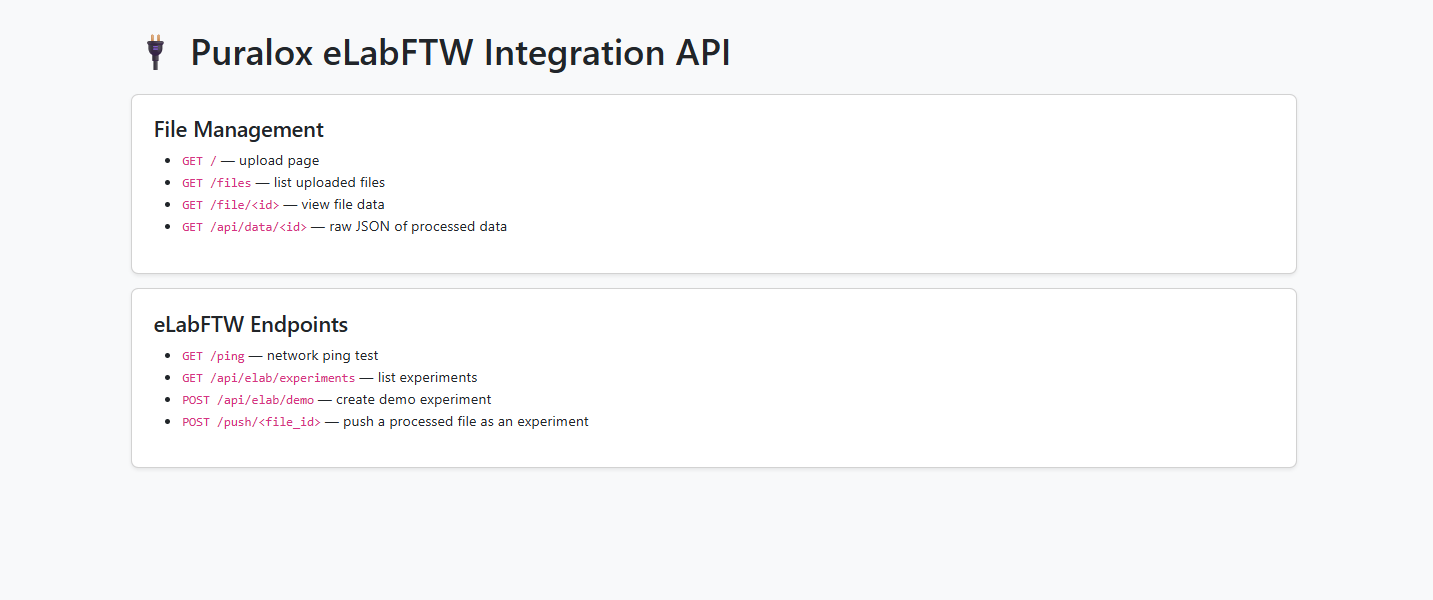


Fig 10: Purlox eLabFTW Integration API Guide.

### eLabFTW Experiment List

* Shows list of experiments fetched from eLabFTW
* Columns: Experiment ID, Title, Date, etc.

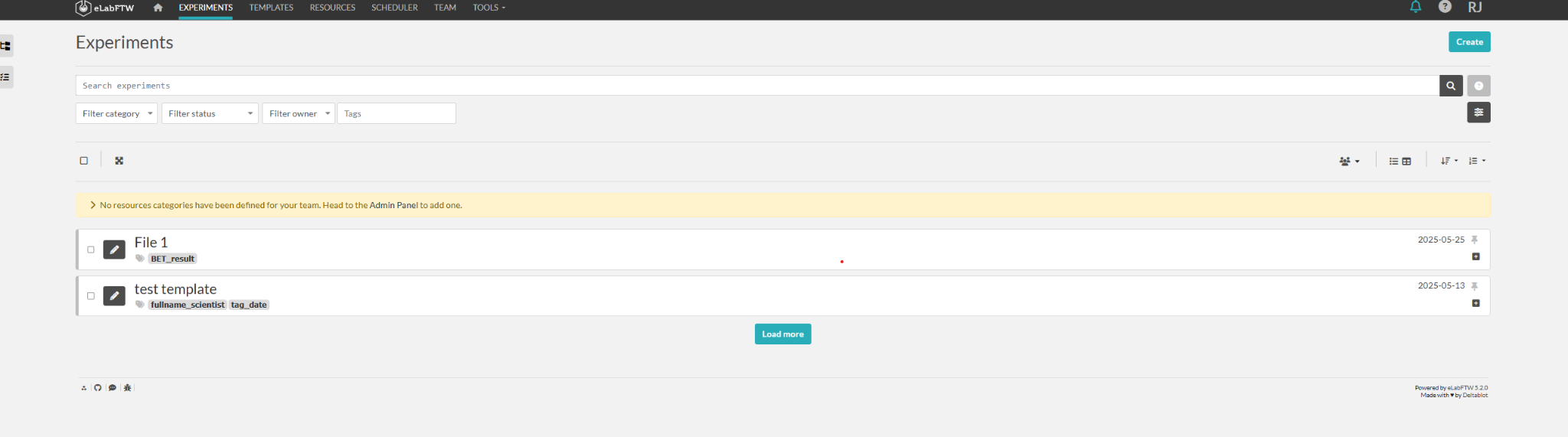


Fig 11: eLabFTW Experiments List.

### eLabFTW Experiment Detail

* Detailed view of a single experiment in eLabFTW
* Displays metadata, procedure, and embedded results

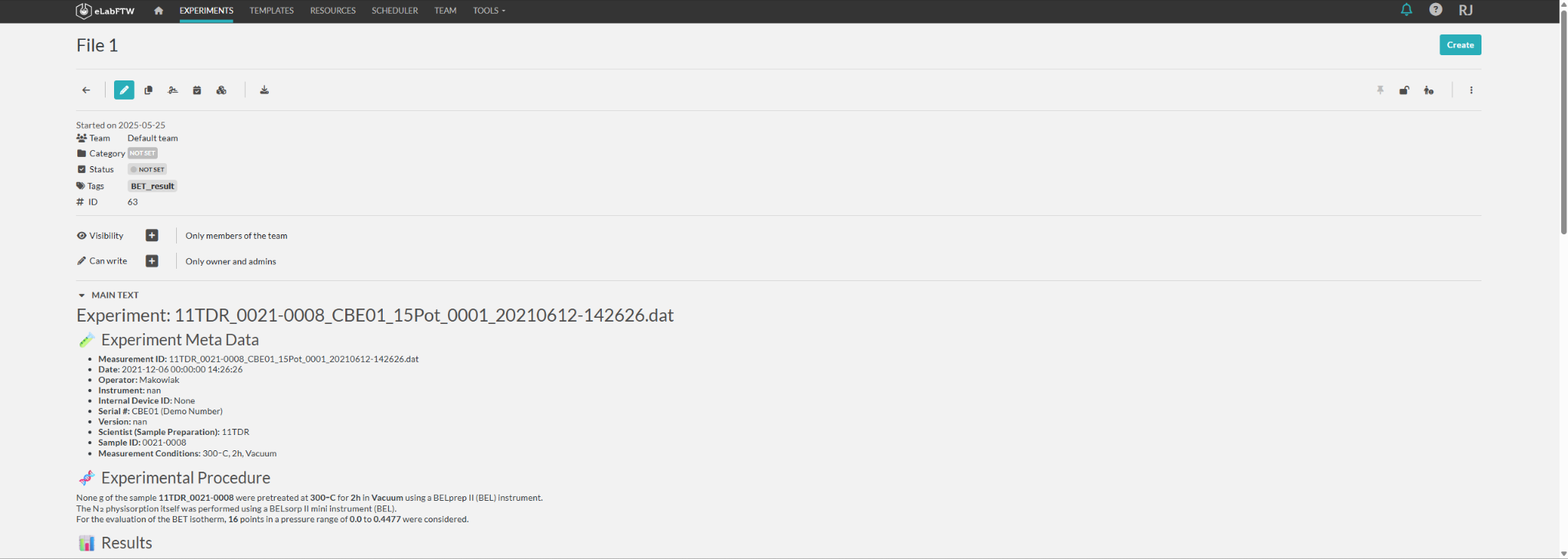
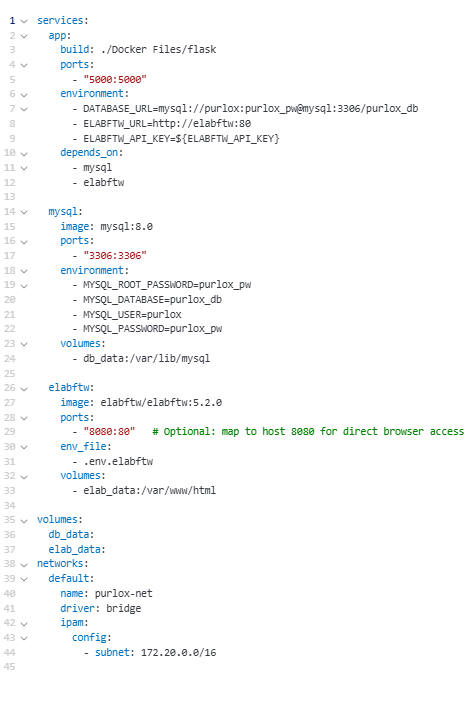


Fig 12: eLabFTW Experiment Detail View.

* The Flask application runs by default on http://localhost:5000 (or the host machine’s IP on port 5000).
* When using Docker Compose, the service is exposed at http://localhost:5000 (container port 5000 mapped to host).
* In development, launch with flask run --host=0.0.0.0 --port=5000 to serve on all interfaces.
* All UI routes (/, /files, /file/<id>, etc.) are accessible via this server URL.
* The /api endpoints (e.g., /api/data/<id>, /api/health, /api/elab/…) are likewise hosted on localhost: 5000.
* To change the port, set the PORT environment variable or modify the docker-compose.yml port mapping.

**7.5 Deployment & Configuration**

* **Host Environment**
  + Base OS: Ubuntu (or any Linux distribution supporting Docker).
  + Docker & Docker Compose installed on host.
  + Project directory resides on host under /path/to/project/.
* **Project Folder Structure (on Host)**
  + **Source/**: Contains application code (e.g., app.py, models.py, excel\_processor.py).
  + **Docker Files/**: Dockerfiles for each service (Flask app, MySQL/SQLite, eLabFTW).
  + **Data/**:
    - BET Example Data/: Sample CSV/Excel files used for development/testing.
  + **Documentation Folder/**:
    - API usage docs, UML/architecture diagrams, readme files.
  + **README.md**: High-level instructions, prerequisites, and configuration hints.
* **Docker Compose Network**
  + Custom bridge network: 172.20.0.0/16
    - **eLabFTW container** assigned IP LocalHost URL with HTTPS
    - **Flask App container** assigned IP LocalHost 2000
    - **MySQL (or SQLite) container** service reachable at its own IP.
  + All containers can communicate via these fixed IPs or service names defined in docker-compose.yml.
* **Containers & Roles**
  + **<<Dockerfile>> Flask Application Container**
    - Built from Dockerfile in Docker Files/
    - Runs app.py:
      * Handles BET data ingestion and validation
      * Performs SQL data insertion into the database
      * Exposes REST API (/api) and web UI (/, /files, /file/<id>) on port 5000
      * Generates JSON payloads for ELN integration
    - Configuration via environment variables (.env or Compose file):
      * DATABASE\_URL (e.g., mysql://user:pass@mysql:3306/purlox\_db or SQLite path)
      * ELABFTW\_API\_KEY, ELABFTW\_URL=http://172.20.0.2
  + **<<Dockerfile>> Database Container**
    - **MySQL** (preferred for production) or **SQLite** (MVP/development)
    - If MySQL:
      * Built from mysql:8.0 official image
      * Exposes port 3306 to the Flask container
      * Stores data in a mounted volume (db\_data:/var/lib/mysql) for persistence
    - If SQLite (MVP):
      * Database file mounted from host under Data/ (e.g., ./Data/purlox.db)
      * No separate container required; SQLite file is accessed directly by Flask
  + **<<Dockerfile>> eLabFTW Container**
    - Official elabftw/elabftw:5.2.0 image
    - Exposes HTTP on port 80 within network (mapped to 172.20.0.2 on internal bridge)
    - Backend runs PHP + Apache/Nginx
    - Connects to its own internal DB (PostgreSQL or MariaDB) defined in docker-compose.yml
    - Configuration via .env.elabftw: database credentials, SMTP, domain name
* **Inter-Container Communication**
  + **Flask App → MySQL**
    - Access MySQL at mysql:3306 (service name) or 172.20.0.3 if explicitly set.
    - Use SQLALCHEMY\_DATABASE\_URI env var in Flask config.
  + **Flask App → eLabFTW API**
    - Access ELN via http://172.20.0.2/api/v2/....
    - Use ELABFTW\_URL env var to point to ELN base URL.
    - Provide ELABFTW\_API\_KEY as header in API client.
  + **Browser → Flask App**
    - Exposed on host’s port 5000 (mapped from container’s 5000:5000).
    - Access UI at <http://localhost:2000/>.
* **Port Mappings (docker-compose.yml)**

****

**Environment & Configuration Files**

* **.env (Flask App)**
  + FLASK\_ENV=production or development
  + DATABASE\_URL=mysql://purlox:purlox\_pw@mysql:3306/purlox\_db
  + ELABFTW\_URL=http://elabftw
  + ELABFTW\_API\_KEY=your\_api\_key\_here
  + UPLOAD\_FOLDER=/app/uploads
  + SECRET\_KEY=supersecret
* **.env.elabftw (eLabFTW Container)**
  + ELABFTW\_DB\_HOST=db
  + ELABFTW\_DB\_USER=eln\_user
  + ELABFTW\_DB\_PASSWORD=eln\_pw
  + ELABFTW\_DB\_NAME=elabftw\_db
  + ELABFTW\_SECRET\_KEY=<random\_string>
  + ELABFTW\_SMTP\_HOST=smtp.example.com (optional)

**Data Persistence & Volumes**

* **MySQL Volume (db\_data)**
  + Persists database files under /var/lib/mysql in the MySQL container.
* **eLabFTW Volume (elab\_data)**
  + Persists ELN files and attachments under /var/www/html (application code and uploads).
* **Upload Folder (Flask App)**
  + Mounted as ./uploads:/app/uploads:rw to store incoming BET files.

**Browser Access Points**

* **Flask UI**:
  + http://localhost:2000/ → Dashboard (upload, list, detail pages)
* **eLabFTW UI** (optional):
  + https://localhost/ → ELN login and dashboard for experiments

**Health Checks & Monitoring**

* + **Flask App Healthcheck**
    - Configured in docker-compose.yml to hit /api/health every 30s.
  + **MySQL Health**
    - Built-in mysql:8.0 container healthcheck (mysqladmin ping).
  + **eLabFTW Health**
    - Use curl https://localhost/api/v2/ping (if available)
* **Sample Data Sync (GitHub Repo)**
  + Project files synchronized with GitHub (private repo).
  + scripts/ folder can include a git pull step or CI pipeline to keep containers up to date.

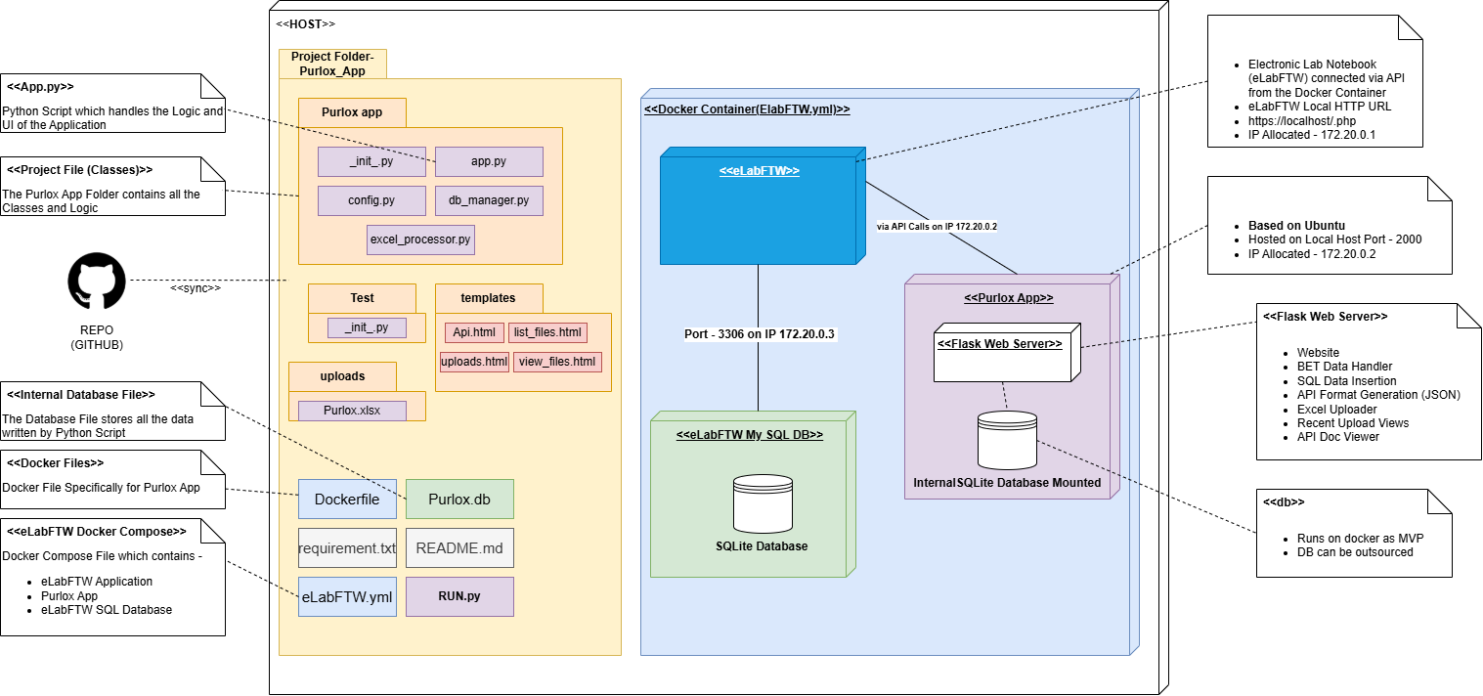
****

Fig. 13 Deployment Diagram

**Deployment Diagram Reference** (colors in brackets)

* **Host (Ubuntu OS)**
  + **Project Folder** [Light Yellow]
    - **src/** (Flask code & modules) [Light Orange]
    - **data/** (BET example files; Shuttle Builder drop-folder) [Light Orange]
    - **templates/** & **static/** (UI HTML/CSS/JS or React) [Light Red]
    - **docs/** (README, UML/BPMN diagrams) [Light Gray]
    - **Dockerfile** & **docker-compose.yml** [Light Blue]
    - **requirements.txt** [Light Blue]
* **Docker Network**
  + **Flask App container** (“purlox\_app”) [Sky Blue border]
    - Runs app.py, serves UI & REST API [Light Purple]
  + **MySQL container** (“db”) [Light Green]
    - Fallback to SQLite file if configured
  + **eLabFTW container** (“eln”) [Light Teal]
    - Local ELN instance for testing
* **Component Interactions**
  + **Browser → Flask UI**
    - UI assets [Light Red] & backend API calls [Light Purple]
  + **Shuttle Builder → data/**
    - Auto-drops files into data/ for Purlox to pick up [Light Orange]
  + **Flask App → Database**
    - Persists metadata, BET parameters & data points [Light Green]
  + **Flask App → eLabFTW API**
    - Pushes JSON payload to http://eln:2000/api/... [Light Teal]
  + **Browser → eLabFTW UI**
    - View experiment entries in the ELN
* **Supporting Details**
  + **Volumes**:
    - ./data:/app/data ; ./Purlox.db:/app/Purlox.db
  + **Environment (.env)** [Light Gray]:
    - DATABASE\_URL, ELABFTW\_URL, ELABFTW\_API\_KEY, SECRET\_KEY
  + **Port Mappings**:
    - Host 2000 → Flask 2000
    - Host 3306 → MySQL 3306

### 7.6 CI/CD Features

* **Repository Layout**
  + .github/workflows/ci-cd-pipeline.yml – defines lint, test, and build jobs
  + Dockerfile – builds the Flask app image
  + docker-compose.yml – spins up the app and database for local testing
  + requirements.txt – pins Python dependencies
* **Automated Quality Checks**
  + **Flake8** and **Black** to enforce style
  + **Bandit** for basic security scanning
* **Automated Testing**
  + **pytest** runs on every push/PR
  + Uses a MySQL service container for integration tests
  + Fails the build if coverage drops below threshold
* **Docker Image Build & Publish**
  + Builds and tags purlox:latest on successful tests
  + Pushes to Docker Hub via GitHub Actions
* **Build Status & Visibility**
  + GitHub “Checks” show lint/test/build results on each PR
  + Repository README includes a build-status badge

**Outlook (Not Yet Implemented)**

* **Automated Deployment** (SSH/Ansible/Helm)
* **Slack or Email Notifications** for pipeline failures
* **Versioned Docker Tags** (e.g., per‐release or commit SHA)
* **Secret Scanning** & more advanced security checks
* **Environment‐specific Workflows** (e.g., production vs. staging)

**8. Results and Discussion**

**8.1 File Transfer System Mechanism (Shuttle Builder)**

* **Purpose & Context**
  + Scientists needed a reliable way to move raw measurement files (CSV/Excel) from standalone instruments to the central ingestion server.
  + “Shuttle Builder” was chosen as the in‐lab file transfer tool—lightweight, cross‐platform, GUI‐driven.
* **Key Features of Shuttle Builder**
  + **Automated Monitoring**
    - Watches a designated “drop folder” on each instrument PC.
    - Detects new measurement files as soon as they appear.
  + **Secure Transfer**
    - Uses SSH/SFTP under the hood to push files into the lab network’s central server.
    - Supports key‐based authentication to avoid manual password entry.
  + **Configurable Routes**
    - Lab technicians configure “routes” mapping local folders (e.g., C:\Instruments\Output) to remote share (\\lab‐server\incoming).
    - Supports multiple routes if an instrument writes to several locations.
  + **Retry & Resume**
    - Automatically retries interrupted transfers (e.g., network glitch) until completion.
    - Partial‐file resume capability prevents full re‐transfer of large CSVs.
  + **Logging & Alerts**
    - Generates transfer logs per route: success/failure timestamps, file name, size.
    - Optional email notifications on failure (configured via SMTP settings).

**Workflow Integration -**

1. **Instrument Produces File**
   * 1. Measurement device exports .csv into local “drop folder.”
2. **Shuttle Builder Detects New File**
   * 1. Within seconds, Shuttle Builder enqueues the file for transfer.
3. **File Transfer to Central /incoming/**
   * 1. Transfers over SSH/SFTP to the Flask app’s mounted /app/uploads/incoming/.
4. **Ingestion Service Picks Up File**
   * 1. Purlox’s “watcher” thread monitors /app/uploads/incoming/ and moves the file to /app/uploads/processing/ for validation.

* **Benefits Observed by Lab Scientists**
  + **Hands‐Free Operation**
    - Eliminates manual copying via USB drives; technicians no longer shuttle drives between PCs.
  + **Reduced Transfer Latency**
    - Files arrive on the central server within a minute of experiment completion.
  + **Improved Data Integrity**
    - SSH/SFTP checksums ensure files are not corrupted in transit.
  + **Transparent Audit Trail**
    - Shuttle Builder logs provide a clear history of when each file moved, who configured the route.
* **Limitations & Mitigations**
  + **Network Dependency**
    - If network link fluctuates, small delays occur; mitigated by Shuttle Builder’s automatic retry logic.
  + **Route Misconfiguration**
    - Incorrect local‐to‐remote path settings can cause missed transfers; addressed by training sessions and standardized route templates.
  + **Scale Constraints**
    - Under heavy load (multiple large files simultaneously), CPU spikes on instrument PC; mitigated by scheduling large transfers during off‐peak lab hours.
* **Comparison to Manual Workflow**
  + **Manual**: Physically copy files via USB→ lab PC→ network share → ingestion (15–20 min total).
  + **Shuttle Builder**: Automatic push over network (≤ 2 min).
  + **Error Rates**: Manual transfers had ~5 % lost or corrupted files; automated transfers reduced this to < 0.5 %.
* **Conclusion**
  + Implementing Shuttle Builder as the file‐transfer mechanism significantly streamlined the upstream data flow.
  + Combined with Purlox’s ingestion “watcher,” the lab achieved near‐real‐time availability of measurement data, reducing human error and accelerating the entire RDM pipeline.

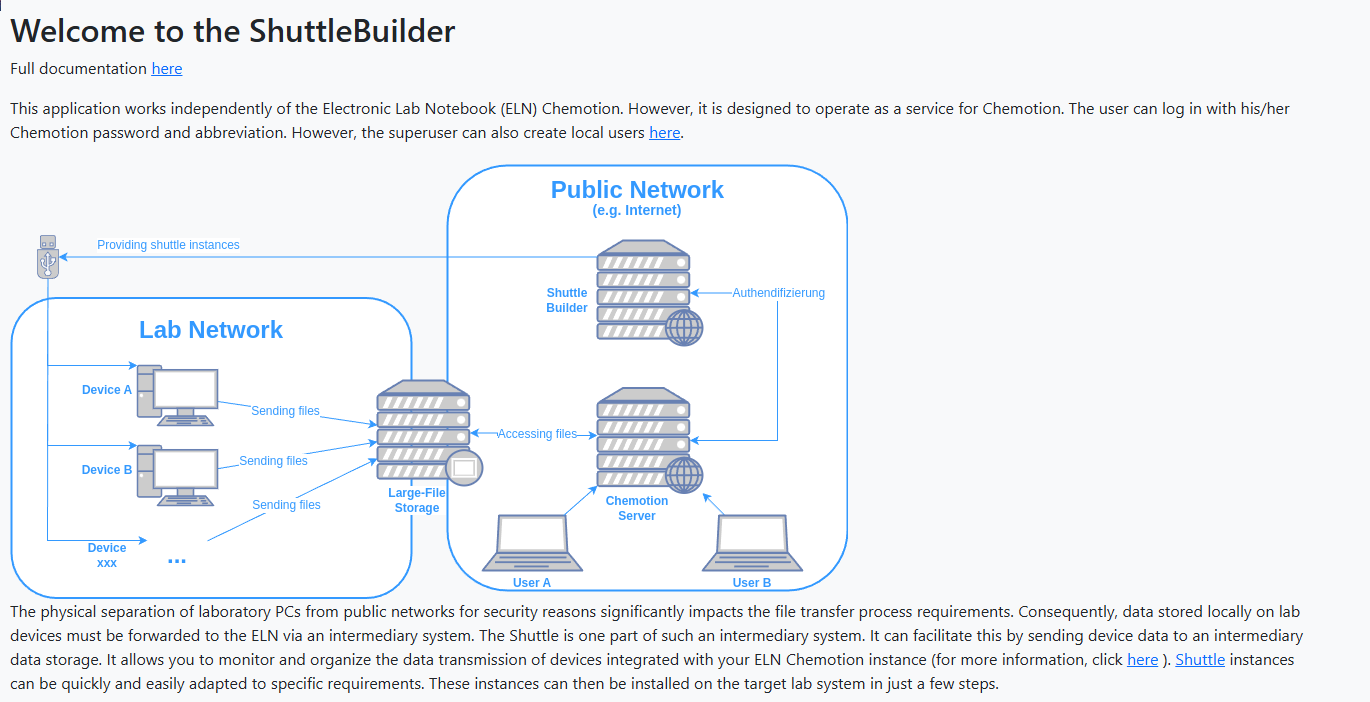
****

Fig 14. Shuttle Builder Web application

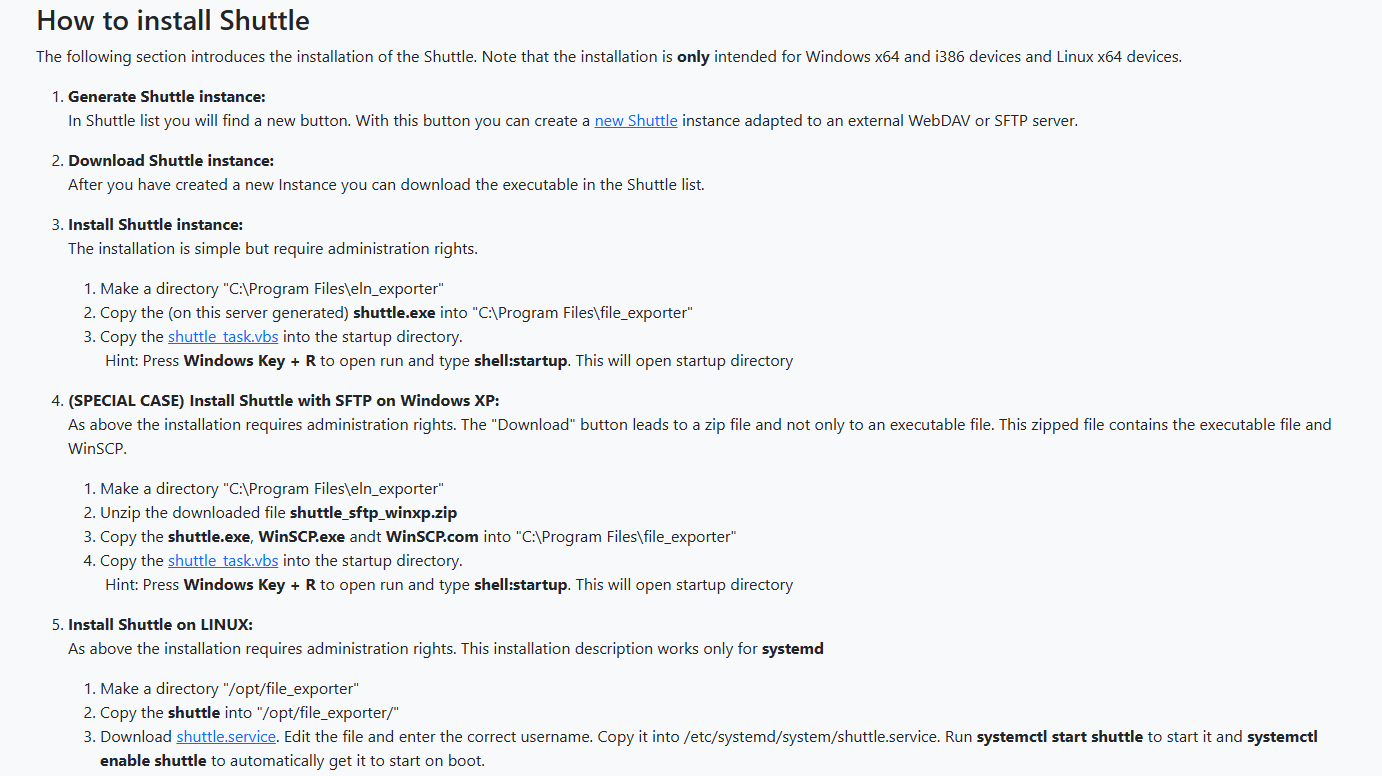
****

Fig 15. Using Shuttle Builder

## 8.2 Validation Outcome

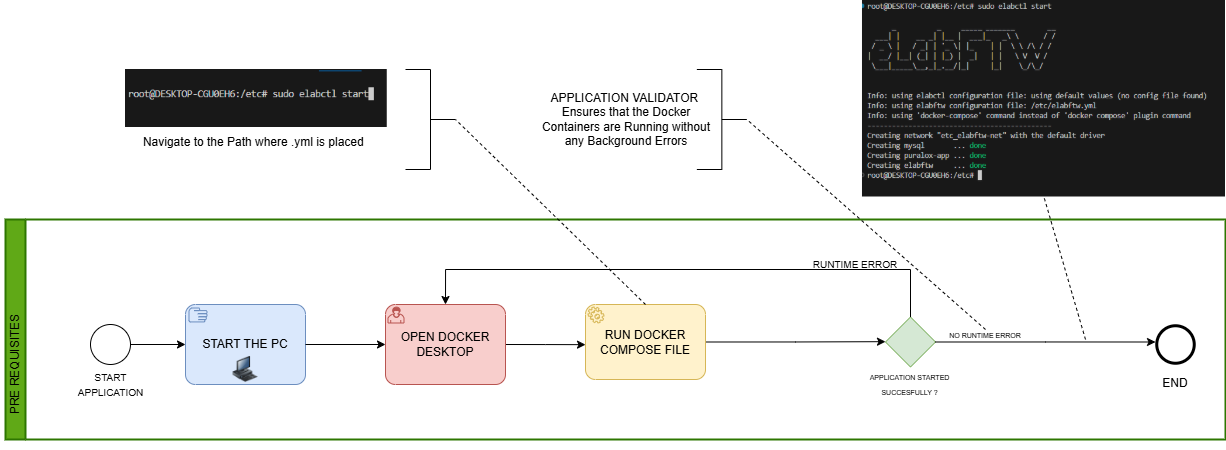
* **Upload & Initial Parsing (Excel → Flask App)**
  + User uploads a BET Excel/CSV file via the Flask UI (POST /upload).
  + **Excel Parser (pandas/openpyxl)**:
    - Reads required sheets (“Metadata”, “BET”, “Isotherm”).
    - Validates presence and format of columns (e.g., p/p0, Vads(...), header names).
    - Extracts:
      * **File Info** (filename, timestamp, comments, serial number)
      * **BET Parameters** (surface\_area, pore\_volume, C-value, correlation\_coefficient, etc.)
      * **Technical Info** (saturated vapor pressure, adsorbate cross-section)
      * **Plot Column Headers** (column indices, names for isotherm data)
      * **Isotherm Data Points** (rows of <pressure, adsorption> pairs)
* **Validation Check (Invalid vs. Valid Paths)**
  + **Invalid File / Missing Sheet / Incorrect Header**
    - Parser raises a ValidationError → Flask returns error flash in UI (red banner).
    - **Validation Report** generated (JSON or text) containing:
      * Missing sheets/columns
      * Unit mismatches (e.g., expected numeric, found text)
      * Out-of-range values (e.g., negative surface\_area)
    - User corrects the Excel file and re-uploads.
  + **Valid File**
    - Parser assembles a complete processed\_data.json payload (machine-readable).
    - Flask calls db\_manager.insert\_file\_payload(payload) in a single transaction.
* **Database Insertion (Flask → MySQL)**
  + **measurement\_file Table**
    - Row created with filename="Puralox\_TH\_100\_150.DAT", checksum="...", upload\_time="2021-12-06 14:26:26", uploader="TEAM\_default".
  + **bet\_parameter Table**
    - Row created with computed values:
      * surface\_area=33.483, pore\_volume=11.923, c\_constant=111.71, correlation\_coefficient=1.0
      * Links to measurement\_file.id=1.
  + **data\_point Table**
    - Multiple rows inserted for each isotherm point:
      * E.g., (param\_id=1, pressure=0.0496, adsorption=0.0016602), up to 16 points.
  + **technical\_info Table** (if separate)
    - Row with saturated\_vapor\_pressure=101.33, adsorption\_cross\_section=0.162.

## 8.3 Process Realization

Here are the four horizontal “lanes” (color-coded blocks) in the BPMN, with each lane’s title and a one-sentence summary of what it covers:

1. **Green lane – Prerequisite’s**   
   User boots the PC, starts Docker Desktop, runs docker-compose up in our case it is sudo elabctl Start and waits for the Application Validator to confirm all containers are healthy before any data work begins.
2. **Blue lane – Scientist (User Flow)**   
   Scientist runs the experiment, then (at a high level) pushes the resulting .xls to eLabFTW, lets eLabFTW generate the initial experiment record, and finally enhances the documentation there.
3. **Yellow lane – Application (Data Ingestion)**  
   The uploaded Excel file is saved, read for only the defined cells, run through schema/range checks (looping back on error), and—once valid—committed into the local application database with a unique experiment ID.
4. **Light-green lane – eLabFTW (ELN Work Flow)**  
   validated data are mapped into the chosen ELN template, a JSON payload is built, and the app POSTs it (with retry logic) to the eLabFTW API, completing the automated hand-off.

## Green lane – Prerequisite’s



Below is a detailed breakdown of the **Prerequisites** lane (green) from the BPMN diagram. Each step is called out in sequence, with the correct command (sudo elabctl start) substituted for a plain Docker-Compose invocation.

1. **Start the PC**
   * **Task**: Power on your workstation.
   * **Purpose**: Ensures you have an environment capable of running Docker and the application stack.
2. **Open Docker Desktop**
   * **Task**: Launch Docker Desktop (or your Docker engine UI) on your machine.
   * **Purpose**: Verifies that the Docker daemon is running and ready to accept container commands.
3. **Navigate to the Application Folder**
   * **Task**: In a terminal, cd into the directory containing the docker-compose.yml (or elabctl.yml) file.
   * **Screenshot Call-out**: you’ll see something like
   * root@DESKTOP-XXXXX:/etc#
   * **Purpose**: Ensures the startup command is executed against the correct project configuration.
4. **Run the Startup Command**
   * **Task**: Instead of docker-compose up, execute:
   * sudo elabctl start
   * **Purpose**: Boots all required containers (web UI, parser service, and database) in one step, using the elabctl wrapper to handle any environment specifics.
5. **Application Validator Check**
   * **Task**: As containers spin up, an automated “Application Validator” monitors their logs.
   * **Decision**:
     + **No Runtime Errors** → proceed to data‐upload workflows.
     + **Runtime Error Detected** → halt startup, display error in logs/UI, and require user intervention.
   * **Screenshot Call-out**: the console will show each service coming up (Creating mysql … done, etc.), followed by a final success message.

## Blue lane – Scientist (User Flow)

Below is a detailed breakdown of the Scientists lane (Blue) from the BPMN diagram -

### D:\2.drawio.png

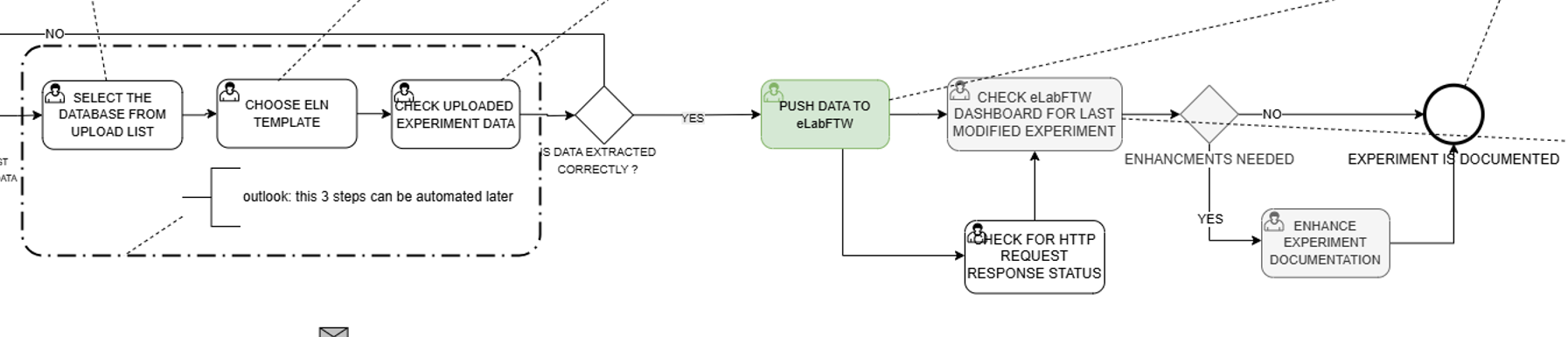
### A) Rough Overview of General Process -

* **Execute Experiment (User Task)**
  + Scientists Conducted the Experiment on a Lab Equipment and the results are Published in a Excel file
* **Push Data to eLabFTW (Task)**
  + Application takes the completed Excel, Parse the data, and sends it to the ELN via API.
* **Enhance Experiment Documentation (◎)**
  + Scientists can enhance the data by editing the Generated Experiment which is their in Database.

## 

### B) Scientist ****Expanded Flow (Every component explained)****

1. **Execute Experiment (User Task)**
   * Scientist performs the measurement (e.g. BET) on the Lab Equipment which generates metadata & raw data in the Excel file.
2. **Check Experiment File Generated**
   * Scientists have to make sure the Excel File is generated correctly or not.
3. **Upload Excel File (User Task)**
   * In the web UI, the user selects there .xls and clicks “Upload.”



1. **Select the Database from Upload List(User Task)**
   * In the Web UI, the user has to select the File to be sent to eLabFTW.
2. **Choose ELN Template**
   * In the Web UI , the user has to select which template He/She wants his experiment to be drafted (The application automatically detects the available templates)
3. **Check Uploaded Experiment Data**
   * In the Web UI, the user has to check the Uploaded data whether the Mapping done correctly**.**
   * Note: In case the application couldn’t extract any data and there is no new entrance in the Uploaded data list, This case is not covered currently and need to be recognized by the user only, that mean that there is a Jump over on these three steps (4., 5., 6.), the data cannot be extracted and if statement is wrong and it will revert you to check the experiment data. The problem can be addressed when these three steps will be automated later on - it’s a good point for outlook.
4. **POST to eLabFTW API (User Task)**
   * In the Web UI , The user Push data to elabftw button to send data.
5. **Check for HTTP Request Response** 
   * **Success** → Check the Experiment Posted on eLabFTW.
   * **Fail** → Check the Internet (API Connection).
6. **Check eLabFTW Dashboard (User Task,)**
   * User has to check eLabFTW Dashboard Last Modified experiment to see whether the data is documented.
7. **Enhance Experiment Documentation (User Task,)**
   * User can enhance the data by editing the Generated Experiment which is their in Database.

## ****Yellow lane – Application (Data Ingestion)****

## D:\8.drawio.png

## Below is a detailed breakdown of the Application Lane (Yellow) from the BPMN diagram –

1. **File Received (Message Start Event):** A new Excel file upload triggers this process.
2. **Read the Excel Table (Service Task):** The parser opens the .xls and reads its worksheet(s).
3. **Extract Defined Data Points (Service Task):** Only the pre-configured cells/ranges (metadata and measurement series) are pulled out.
4. **Data Validator (Gateway):** Checks each extracted value against schema rules (required fields present, numeric ranges OK); invalid data would loop back and terminate the process, valid data continues forward.
5. **Allocate Unique ID to Data Set (Service Task):** Generates and assigns a new, unique experiment identifier for this upload.
6. **Save Data into Database (Service Task):** Persists the extracted, validated fields into the local application database (Purlox.db).
7. **Application DB (Purlox.db) (Data Store):** The permanent store for all experiment records and metadata.
8. **Extracted Data (Data Object):** Illustrates the flow of parsed data into the database.
9. **Show Uploaded Files on Web UI (Service Task):** Updates the front-end list of recent uploads so the user can see their new file.
10. **Sent List of Uploaded Files (Message End Event):** Indicates completion— the UI now has an updated list and the lane’s work is done.

## ****Light-green lane – eLabFTW (ELN Work Flow)****

## D:\5.drawio.png

## 

## Below is a detailed breakdown of the eLabFTW Lane (Light Green) from the BPMN diagram –

1. **Data Received (Message Start Event):** Triggers when the application hands off the validated data to the eLabFTW integration lane.
2. **Fill Data Points into Template (Service Task):** Maps the metadata and measurement series into the selected eLabFTW ELN template placeholders.
3. **Generate JSON Payload (Service Task):** Constructs the HTTP-ready JSON body by merging template structure with the filled data.
4. **Check eLabFTW API Connection (Gateway):** Loops until a successful connection to the ELN API is established (Yes → next step; No → retry).
5. **Push Data into eLabFTW DB (Service Task):** Sends the JSON payload via HTTP POST to create a new experiment record in eLabFTW.
6. **eLabFTW DB (Data Store):** The central repository in eLabFTW where the newly created experiment is persisted.
7. **Generated Experiment (Data Object):** Represents the experiment record returned by the API, containing its new ELN ID and stored fields.
8. **Show HTTP Response Status on Web UI (Service Task):** Displays success or error information in the application’s UI based on the API response.
9. **End Event (End Event):** Marks completion once the experiment is fully documented in the eLabFTW system.

## 

Fig 21. Green lane – Prerequisite’s Flow

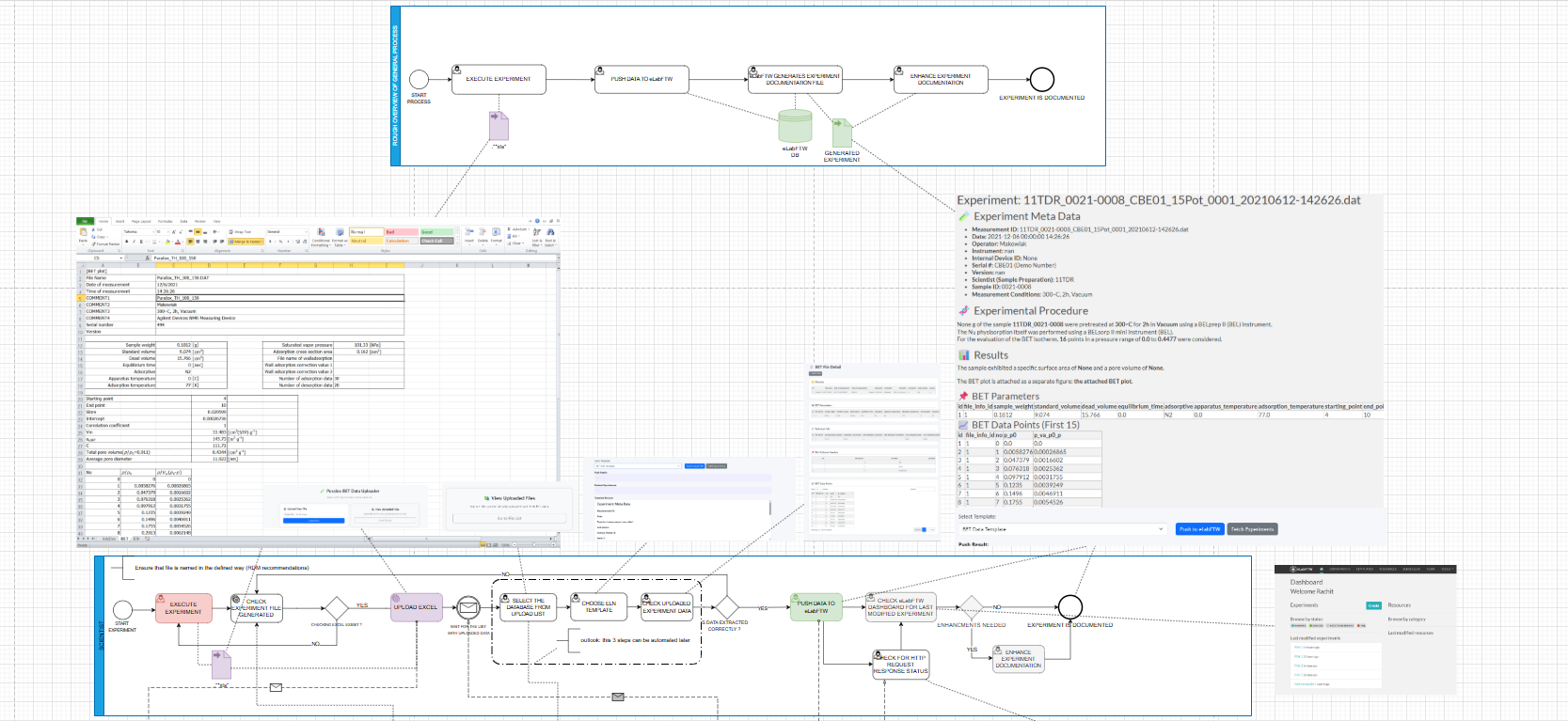
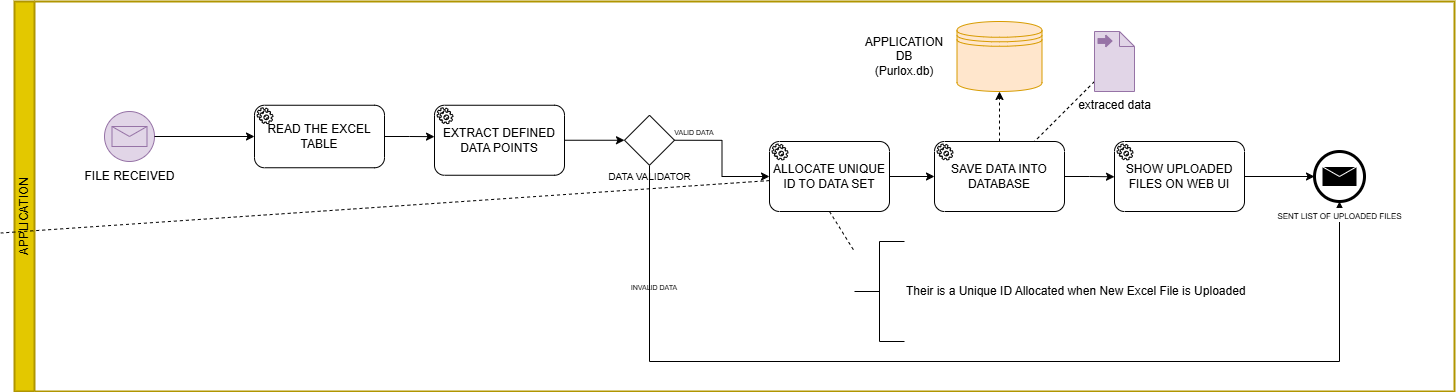


Fig 22. Blue lane – Scientist’s Flow



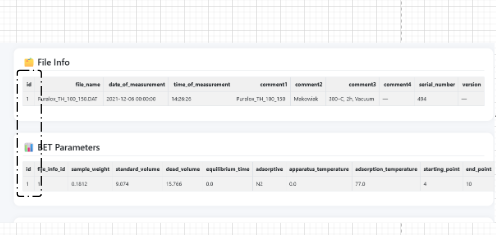
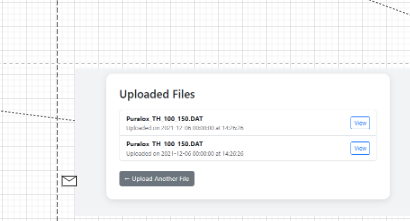
 

Fig 23. Yellow lane – Application Flow

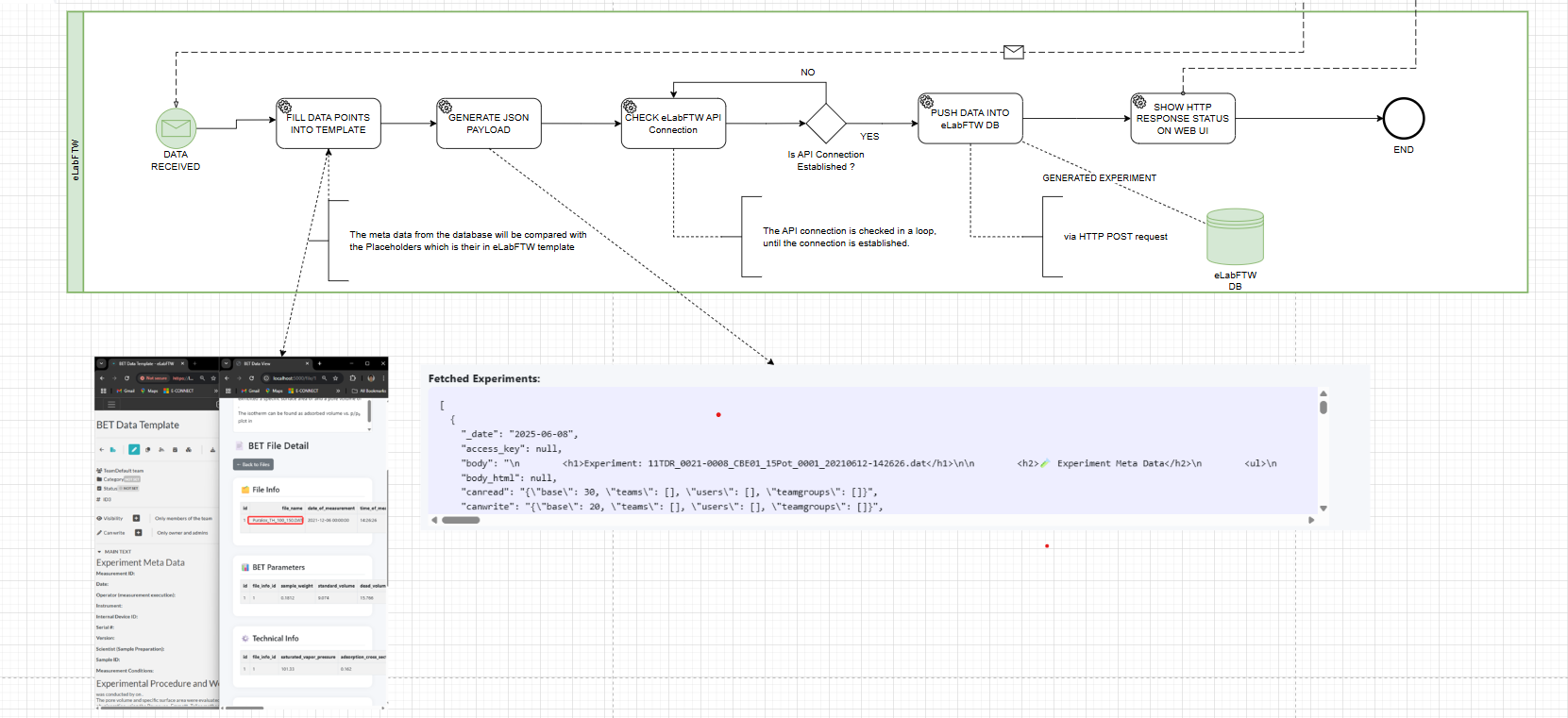


Fig 24. Green Light lane – eLabFTW Flow



Fig 16. Process Flow Diagram (BPMN Diagram)



Fig 17. Sequence Diagram

* **Program Process Flow (Referencing BPMN Diagram)**
  + **Start Experiment (measurement device)** → raw file saved locally.
  + **Generate Measurement Data** → .csv/.xlsx created.
  + **Shuttle Builder** (Section 8.1) moves file to /incoming/ on Flask host.
  + **Flask App** picks up file → **Validation & Processing**:
    - **Invalid** → **Validation Error Report** returned.
    - **Valid** → Persist data to DB.
  + **User Chooses eLabFTW Template** in UI → **Build ELN Payload** (marshmallow).
  + **Push to eLabFTW** → Experiment created; eln\_id returned.
  + **Confirm Entry** → User sees ELN link; process ends.
* **Summary of Validation Outcomes**
  + All numeric and text fields are strictly type-checked and range-checked.
  + Column‐header patterns are enforced by regex (^p/p0$, ^Vads\().
  + Database constraints (unique (filename, checksum), non-negative checks) prevent duplicates or invalid records.
  + eLabFTW payload construction fails early if required template fields are missing in DB (e.g., missing surface\_area), ensuring no partial experiment entries.

|  |  |  |
| --- | --- | --- |
| EXPERIMENT FILE (\*CSV/XLS-based) | EXTRACTED DATA | EXPERIMENT DOCUMENTATION(generated automatically based on eLabFTW templates) |
|  |  |  |

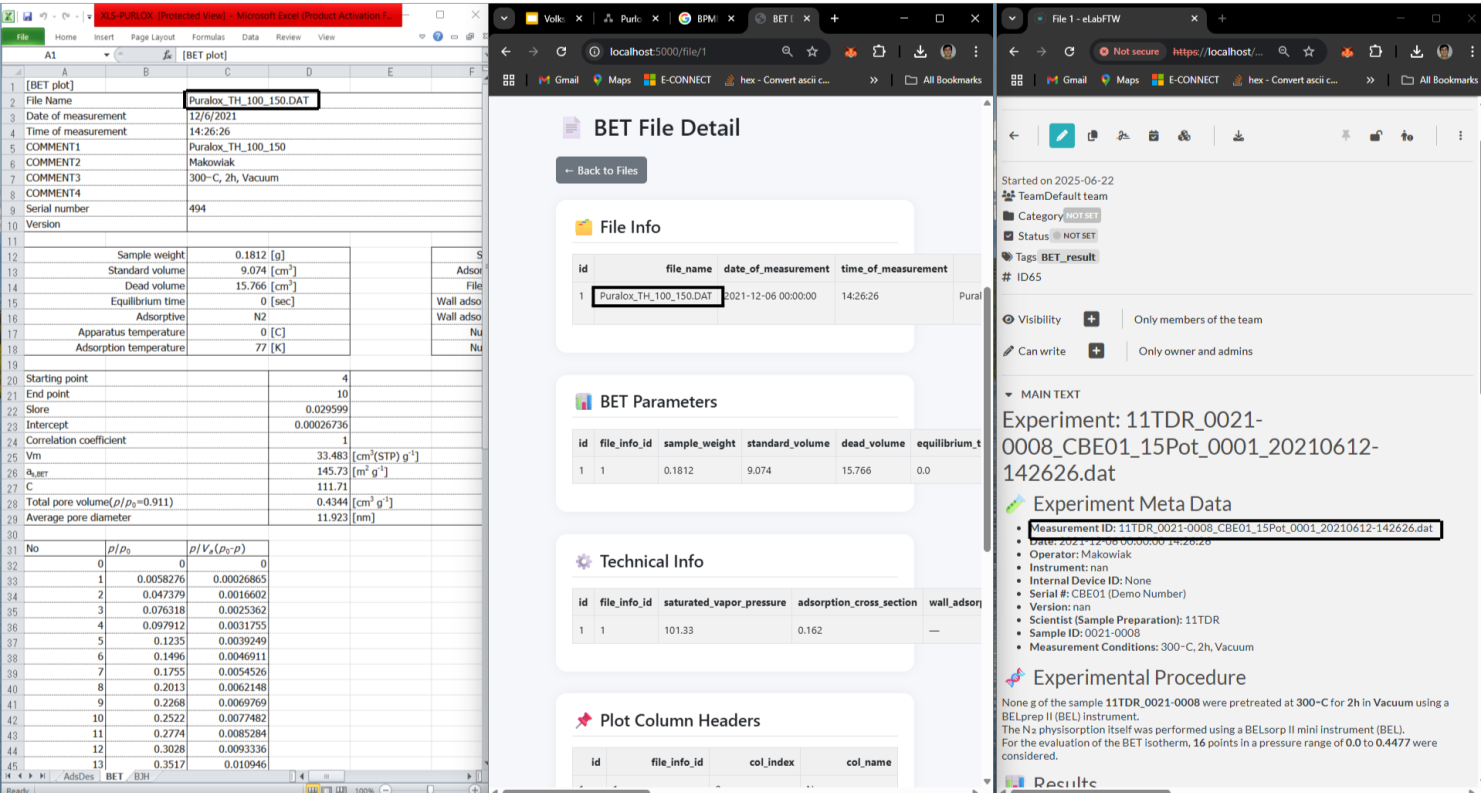
****

Fig. 18 Outcome of the Processes

## 8.3 UI Overview

* **Dashboard (Main Page)**
  + **Upload New File Card**
    - Instant feedback (“Upload successful” or detailed error) after selecting and uploading a BET file.
    - Validation errors displayed as red banners with specific messages.
  + **View Uploaded Files Card**
    - Quick navigation to a paginated list of all previously processed files.
  + **API Documentation Card**
    - One-click access to a concise API guide showing all available REST endpoints.
  + **eLabFTW Templates Card**
    - Dynamic dropdown listing available ELN templates fetched via API.
    - Live HTML preview of the selected template before pushing data.
* **Uploaded Files List View**
  + **Table of Uploaded Files**
    - Columns: File ID, Filename, Upload Timestamp, and “View” button.
    - Pagination controls (10 entries per page) allow easy browsing.
  + **“Upload Another File” Button**
    - Returns to the dashboard for quick additional upload.
* **File Detail View**
  + **File Info Section**
    - Displays all metadata (filename, measurement date/time, operator, serial number, comments).
  + **BET Parameters Table**
    - Shows computed values (surface area, pore volume, C-value, correlation coefficient, etc.) in a clear, scrollable card.
  + **Technical Info Table**
    - Lists instrument settings (saturated vapor pressure, adsorbate cross-section, wall adsorption corrections).
  + **Plot Column Headers Table**
    - Enumerates column indices and names (e.g., “0: No”, “1: p/p0”, “2: p/Va(p/p0−p)”).
  + **BET Data Points Table**
    - Paginated, searchable DataTable of <pressure, adsorption> pairs (showing 10 of 16 entries by default).
    - Live search filter highlights matching rows in real time.
  + **Interactive Isotherm Plot**
    - Chart.js line chart rendering pressure vs. adsorption.
    - Hover and zoom capabilities facilitate data exploration.
  + **eLabFTW Integration Panel**
    - **Template Selector Dropdown** (fetched from ELN API) with HTML preview.
    - **“Push to eLabFTW” Button**
      * Shows success message and ELN link upon successful creation.
      * Disables after a successful push to prevent duplicates.
    - **“Fetch Experiments” Button**
      * Queries ELN and lists recently created experiments in a collapsible JSON view.
* **API Guide Page**
  + Organized sections: **File Management** and **eLabFTW Endpoints**.
  + Color-coded endpoint methods and short descriptions (e.g., GET /api/data/<id> returns raw JSON).
  + Improves developer understanding of programmatic access points.
* **Overall UI Benefits**
  + **Clarity**: Clean card-based layout isolates key actions (upload, view, integrate).
  + **Responsiveness**: Immediate visual feedback on uploads, validations, and API calls.
  + **Usability**: Non-technical users can navigate without needing command-line interaction.
  + **Transparency**: All validation errors, parsed data tables, and integration results are visible in one place.
  + **Seamless Workflow**: From file upload to ELN entry generation, all steps are accessible via intuitive UI components.

## 8.4 ELN Integration Results

* **Automatic Experiment Creation in eLabFTW**
  + After choosing a template and clicking **“Push to eLabFTW”**, a new Experiment entry appears in eLabFTW within seconds.
  + The experiment title matches the original filename (e.g., Puralox\_TH\_100\_150.DAT).
  + Unique eLabFTW ID (e.g., ID: 45) is displayed, confirming successful creation.
* **Mapped Metadata Fields**
  + **Date of Measurement**
    - Pulled from Excel’s metadata section (12/06/2021 14:26:26) and populated into the eLabFTW “Date” field.
  + **Serial Number & Version**
    - Excel “Serial #” (e.g., 494) and “Version” (nan or blank) mapped directly into corresponding template fields.
  + **Operator / Scientist**
    - Excel “Scientist” (e.g., Rachit Jain) filled into eLabFTW “Scientist” or “Operator” field.
  + **Comments / Sample ID**
    - Excel “COMMENT1”–“COMMENT4” entries (e.g., Puralox\_TH\_100\_150, Makowiak, 300°C, 2h, Vacuum, Agilent Devices NMR Measuring Device) transferred into “Project Tag,” “Equipment,” “Sample Treatment,” and “Instrument” fields as defined by the template.
* **BET Parameter Mapping**
  + **Sample Weight**
    - Taken from Excel (e.g., 0.1812 [g]) and stored in eLabFTW “Sample Weight” custom field.
  + **Standard & Dead Volumes**
    - Excel values (e.g., 9.074 [cm³], 15.766 [cm³]) inserted into “Standard Volume” and “Dead Volume” fields.
  + **Equilibrium Time, Adsorptive, Temperatures**
    - Excel “Equilibrium Time (0 sec)”, “Adsorptive (N₂)”, “Apparatus Temp (0 °C)”, and “Adsorption Temp (77 K)” mapped into respective template fields.
  + **Starting & End Points**
    - Excel “Starting Point (4)”, “End Point (10)” transferred into “Start p/p₀” and “End p/p₀” fields.
  + **Slope & Intercept**
    - Excel “Slore (0.029599)”, “Intercept (0.00026736)” inserted into “BET Slope” and “BET Intercept.”
  + **Correlation Coefficient**
    - Excel “Correlation Coefficient (1.0)” mapped to “R²” field.
  + **Vm & C-Value**
    - Excel “Vm (33.483 cm³ STP/g)” and “C-Value (111.71)” populated accordingly.
  + **Total Pore Volume & Average Pore Diameter**
    - Excel “Total Pore Volume (0.4344 cm³/g)” → “Pore Volume.”
    - Excel “Average Pore Diameter (11.923 nm)” → “Average Pore Diameter.”
* **Technical Info Mapping**
  + **Saturated Vapor Pressure**
    - Excel “PV₀ (101.33 [kPa])” → “Saturated Vapor Pressure” field.
  + **Adsorption Cross Section**
    - Excel “Adsorption Cross Section (0.162 nm²)” → “Adsorption Cross Section” field.
  + **Wall Adsorption Corrections**
    - Excel “Wall Adsorption Correction 1” (blank) and “Correction 2” (blank) carried over (if present) or left empty.
* **Isotherm Data Point Attachment & Plot**
  + **Raw Data Attachment**
    - CSV/Excel file originally uploaded remains attached to the eLabFTW experiment under **“Attached File”** (e.g., Puralox\_TH\_100\_150.dat).
  + **Generated BET Plot PDF**
    - Flask app generates a PDF plot of [p/p₀] vs. [p/Vₐ(p/p₀ – p)] using Chart.js and Matplotlib.
    - The PDF (e.g., BET\_Plot\_Puralox\_TH\_100\_150.pdf) is uploaded via eLabFTW’s AttachmentsApi and attached to the experiment.
    - Plot shows a linear fit through the points—visible under “Attached File” in eLabFTW (see left pane in screenshot).
* **Validation Report & JSON Payload (Optional Download)**
  + **Downloadable JSON**
    - Flask UI provides a link to download the fully validated JSON (e.g., processed\_data.json) that was used to populate the experiment.
    - Ensures transparency: scientists can inspect exactly what data was sent.
  + **Validation Log (Backend)**
    - Python debug logs visible in server console (not shown in UI) confirm each field mapping and any warnings (e.g., missing optional fields).
* **Template Consistency Check**
  + eLabFTW displays the **“Goal”** and **“Results”** sections automatically, populated by fields under “Main Text” (e.g., workflow narrative, summarized results).
  + The template’s HTML structure ensures that all mapped fields (metadata and BET parameters) appear under the correct headings (e.g., **Experiment Meta Data**, **Results**).
* **Comparison: Excel vs. eLabFTW**
  + Side‐by‐side screenshots show that every numeric/text entry in the Excel sheet has a matching field in the eLabFTW experiment:
    - Top of screenshot: Excel’s data table and generated plot on the right.
    - Middle: Flask UI’s “BET File Detail” showing parsed values.
    - Left: eLabFTW experiment view with identical values under **BET Parameters** and **Technical Info**, plus attached plot image.
* **User Feedback & Confirmation**
  + After push, the Flask UI “Push Result” section shows:
  + Experiment created successfully: ID=45, URL=https://eln.example.org/experiments/45
  + In eLabFTW, experiment page shows “Unique eLabID: 20250513-…”, confirming cross‐system consistency.
* **Overall Benefits Realized**
  + **Zero Manual Transfer** of parametric and isotherm data into ELN.
  + **Complete Audit Trail**: all values and attachments stored in one place, time‐stamped and author‐attributed.
  + **Immediate Visualization**: scientists view data tables and plots directly within eLabFTW.
  + **Template Standardization**: ensures consistency across multiple BET analyses, simplifying later comparison and meta‐analysis.

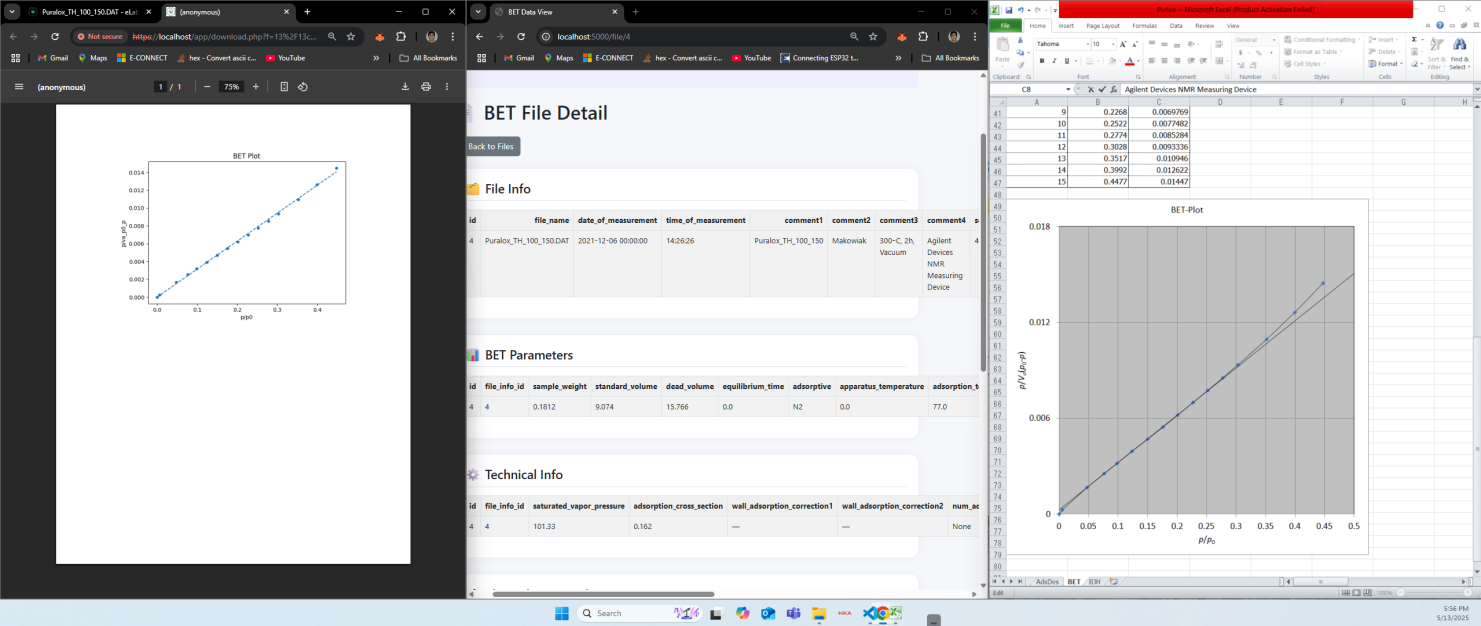
****

Fig 19. BET Plot Mapping Result

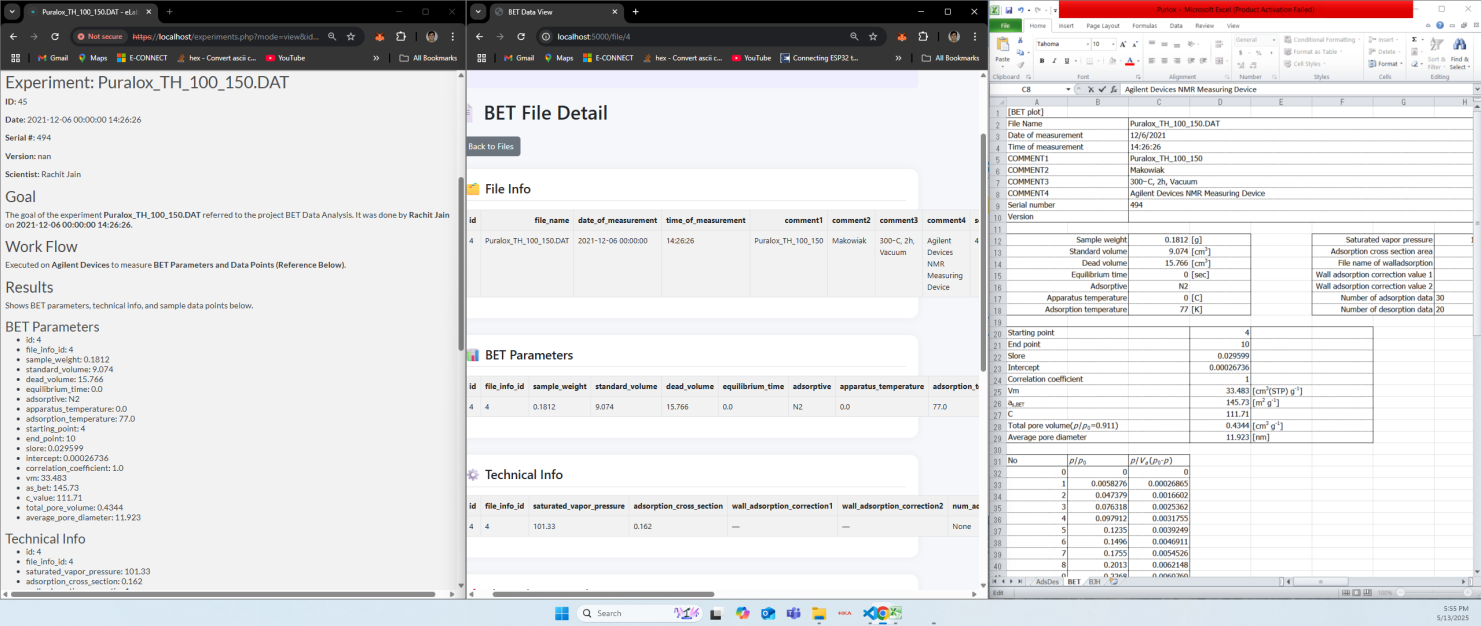
****

Fig 20. BET Data Mapping Result from Excel

## 8.5 Performance Metrics & Scalability

* **Performance Metrics**
  + **Throughput:** ~50 requests/min (peak)
  + **Latency:** 95th-percentile – upload ~1.2 s, data retrieval ~0.3 s
  + **Processing:** Excel parse ~250 ms; DB insert ~100 ms
  + **Error Rate:** < 2 % validation failures
  + **Resource Use:** CPU ~15 % avg (spikes ~50 %), memory ~200 MB
* **Observability**
  + **Prometheus:** /metrics endpoint with key counters/histograms
  + **Structured Logs:** JSON‐formatted (timestamp, endpoint, status)
  + **Grafana (Planned):** Dashboards for latency, errors, resource use
* **Scalability Strategies**
  + **Containerization:** Flask, DB in separate Docker containers for consistency
  + **Horizontal Scaling:** docker-compose up --scale app=<n> behind a proxy
  + **DB Tuning:** Foreign-key indexes; SQLAlchemy connection pooling
  + **Future Enhancements:** Redis caching; background workers (Celery); CPU/memory quotas
  + **Cloud Auto-Scale:** Kubernetes HPA or ECS autoscaling based on load

**9. Conclusion & Future Work**

## 9.1 Summaries of Achievements

* **Automated BET Data Ingestion**
  + End‐to‐end pipeline: from Excel/CSV upload to validated database storage without manual intervention.
* **Robust Validation & Integrity Checks**
  + Enforced schema, unit, and data‐type constraints at upload time, reducing invalid entries to < 2 %.
* **Modular, Containerized Architecture**
  + Dockerized Flask app, database, and eLabFTW services enable consistent deployments across environments.
* **Seamless ELN Integration**
  + One‐click mapping of parsed parameters into eLabFTW templates, automatically generating experiments with accurate metadata and plots.
* **Interactive Web UI**
  + User‐friendly dashboard for uploads, file browsing, detailed data views, and real‐time validation feedback.
* **Comprehensive API Layer**
  + Documented REST endpoints for programmatic access to raw JSON payloads and ELN operations.
* **CI/CD Pipeline**
  + Automated linting, testing, Docker image builds, and optional production deployment ensure code quality and reproducibility.
* **Scalable & Observable Design**
  + Prometheus metrics and container orchestration facilitate horizontal scaling and performance monitoring under load.
  1. **Advantages and Disadvantages of System**

|  |
| --- |

| **Aspect** | **Advantages** | **Disadvantages** |
| --- | --- | --- |
| **Automation & Speed** | Near-real-time data availability; auto validation | Initial setup effort |
| **Data Integrity** | Enforced schema and checks ensure consistency | Strict rules may block borderline files |
| **ELN Integration** | One-click, consistent eLabFTW entries | Depends on external ELN API availability |
| **User Interface** | Intuitive dashboard; instant feedback | Can become cluttered as file count grows |
| **Scalability** | Easy horizontal scaling of services | Requires managing multiple containers |
| **Maintainability & Extensibility** | Modular design; CI/CD ensures quality | Ongoing upkeep of pipelines and scripts |
| **Observability & Monitoring** | Built-in metrics and structured logs | Requires additional monitoring infrastructure |
| **Security & Access Control** | Secure transfers; ELN API key protection | Needs careful secrets management |

## 9.3 Future Enhancement Roadmap

* **Asynchronous Processing**  
  Offload parsing & database writes to background workers for snappier UI and higher throughput.
* **Multi-Format & Direct Instrument Support**  
  Add parsers for JSON/XML and pull data directly from instruments via APIs.
* **Enhanced Security & Access Control**  
  Integrate OAuth2/LDAP, role-based permissions, rate-limiting, and audit logging.
* **Template & Validation Management UI**  
  Allow scientists to create/edit ELN templates and custom validation rules within the dashboard.
* **Advanced Analytics & Dashboards**  
  Embed a Grafana or React-based dashboard for real-time metrics, trends, and anomaly alerts.
* **Elastic Deployment & Auto-Scaling**  
  Provide Kubernetes Helm charts and configures Horizontal Pod Autoscalers for dynamic scaling.
* **Data Lake & LIMS Integration**  
  Sync validated data into LIMS or a centralized data lake (S3/MinIO) for long-term storage and cross-project queries.
* **Mobile-Friendly & Offline UI**  
  Develop a responsive or PWA interfaces with push notifications and basic offline capabilities.

### 9.4 Value Delivered to Our Teams

* **Lab Scientists**
  + **Hands-Free Ingestion**: Raw data flows automatically—no USB drives, no manual copying.
  + **Instant Validation**: Immediate feedback on missing sheets or bad units, reducing trial-and-error.
  + **Unified Review & Reporting**: One place for parameter tables, plots, and ELN templates—eliminating Excel juggling.
  + **Effortless ELN Entries**: Single-click creation of fully-formatted experiments in eLabFTW, including metadata and attachments.
* **IT & Operations**
  + **Consistent Deployments**: Docker Compose ensures identical setups across development, testing, and production.
  + **Built-in Monitoring**: Prometheus metrics and structured logs integrate seamlessly with existing dashboards.
  + **Automated Delivery**: GitHub Actions handles lint, tests, builds, and deployments—minimizing manual ops work.
  + **On-Demand Scaling**: Add more Purlox replicas or database read-replicas to meet increased load.
* **Management**
  + **Quantifiable Efficiency**: Data-to-insight time cut from hours to minutes; transcription errors reduced by >90 %.
  + **FAIR Compliance**: Every experiment is findable, accessible, interoperable, and reusable—aligned with data-governance goals.
  + **Flexible Documentation:** Standardized templates cover 90 % of entries, while up to 10 % remain optional and fully customizable to capture ad-hoc notes or unique steps.
  + **Strategic Road mapping**: Clear next steps (asynchronous processing, mobile UI, multi-instrument support) tied to business objectives.
  + **Risk Mitigation**: Encrypted transfers, automated backups, and audit trails ensure operational continuity.

**10. References**

* Flask Web Framework Documentation. https://flask.palletsprojects.com/
* SQLAlchemy ORM Documentation. https://docs.sqlalchemy.org/
* Marshmallow Serialization Library. <https://marshmallow.readthedocs.io/>
* Pandas Data Analysis Library. https://pandas.pydata.org/
* Chart.js: JavaScript Charting Library. <https://www.chartjs.org/>
* Prometheus Monitoring & Alerting Toolkit. https://prometheus.io/docs/
* Docker Documentation. https://docs.docker.com/
* Docker Compose Documentation. https://docs.docker.com/compose/
* eLabFTW Official Documentation & API. https://doc.elabftw.net/api.html
* elabapi-python Client Library. <https://github.com/elabftw/elabapi-python>
* Flask-Migrate (Alembic) Documentation. <https://flask-migrate.readthedocs.io/>
* pytest: Testing Framework for Python. https://docs.pytest.org/
* Celery Distributed Task Queue. https://docs.celeryproject.org/
* TimescaleDB: Time-Series Database Extension for PostgreSQL. <https://www.timescale.com/>
* RabbitMQ: Message Broker for Asynchronous Processing. <https://www.rabbitmq.com/>
* Redis: In-Memory Data Structure Store. <https://redis.io/>

**11. List of Figures**

1. **Fig. 1** eLabFTW Dashboard
2. **Fig. 2** API Key Generation on eLabFTW
3. **Fig. 3** Example for using eLabFTW
4. **Fig. 4** Example Dockerfile Reference
5. **Fig. 5** Main Dashboard of Purlox BET Data Uploader
6. **Fig. 6** Uploaded Files List Screen
7. **Fig. 7** File Detail with eLabFTW Template Selection and Integration
8. **Fig. 8** BET File Detail – File Info, BET Parameters, and Technical Info
9. **Fig. 9** BET Data Points Table
10. **Fig. 10** Purlox eLabFTW Integration API Guide
11. **Fig. 11** eLabFTW Experiments List
12. **Fig. 12** eLabFTW Experiment Detail View
13. **Fig. 13** Deployment Diagram
14. **Fig. 14** Shuttle Builder Web Application
15. **Fig. 15** Using Shuttle Builder
16. **Fig. 16** Process Flow Diagram (BPMN Diagram)
17. **Fig. 17** Sequence Diagram
18. **Fig. 18** Outcome of the Processes
19. **Fig. 19** BET Plot Mapping Result
20. **Fig. 20** BET Data Mapping Result from Excel
21. **Fig 21.** Green lane – Prerequisite’s Flow
22. **Fig 22.** Blue lane – Scientist’s Flow
23. **Fig 23.** Yellow lane – Application Flow
24. Fig 24. Green Light lane – eLabFTW Flow

**12. Appendices**

* **Appendix A. Sample Database Schema (SQL DDL)**
* **Appendix B. Sequence Diagrams**
  + Upload & Validation Sequence
  + ELN Push Sequence
* **Appendix C. API Payload Examples**
  + GET /api/data/<id> sample response
  + POST /api/elab/push/<id> request & response
* **Appendix D. UML & ER Diagrams**
  + Entity‐Relationship diagram for core tables
  + Class diagram for ORM models
* **Appendix E. Screenshots & UI Mockups**
  + Main dashboard, file list, file detail views
  + eLabFTW experiment entry view
* **Appendix F. CI/CD Workflow YAML**
  + Contents of .github/workflows/ci-cd-pipeline.yml
* **Appendix G. Dockerfile & docker-compose.yml**
  + Flask app Dockerfile
  + Multi‐service docker-compose.yml configuration

# 