
PharmaDocs-AI — Project Documentation

1. Overview

Project name: PharmaDocs-AI
Purpose: PharmaDocs-AI is a web application for creating, managing, verifying, and generating **Analytical Method Verification (AMV)** documents and related protocols used in pharmaceutical quality control (QC) and quality assurance (QA).

The system helps laboratories and pharma companies to:

- Create and manage AMV documents and verification protocols.
- Extract method parameters from uploaded PDF methods.
- Perform AMV-related calculations and ICH Q2(R1) checks.
- Generate DOCX reports and verification protocols.
- Generate chemical SMILES and structure images.
- Manage subscriptions and payments (via Razorpay).

1.1 Primary Users

- **QC Analysts** – prepare AMV documentation and verification protocols.
- **Method Owners** – maintain and update analytical methods.
- **QA Managers** – review, approve, and audit documents.
- **Administrators** – manage users, companies, and subscriptions.
- **Company Accounts** – manage organization-level data and plans.

1.2 Goals

- **Streamline AMV documentation & reporting** for ICH Q2(R1) compliance.
- **Automate numeric calculations** for validation and verification.
- **Provide reusable, auditable documents** with traceable changes.
- **Integrate with external services** for file storage, payments, and chemical utilities.

---

2. Tech Stack & Dependencies

2.1 Backend

- **Language:** Python
- **Framework:** Flask
- **ORM:** SQLAlchemy
- **Migrations:** Alembic
- **Templating:** Jinja2

2.2 Frontend

- Flask-rendered **HTML templates**
- **Jinja2** for server-side templating
- **Static JS/CSS** under static/
    - Feature-specific JS (e.g., AMV forms, file uploads, dashboard logic)

2.3 Storage & External Services

- **Database:** Relational DB (PostgreSQL/MySQL/SQLite via SQLAlchemy URI)
- **Local file storage:** uploads/ (for PDFs, signatures, etc.)
- **Generated documents:** reports/ (DOCX reports and verification protocols)
- **Cloud storage:** Cloudinary (via cloudinary_service.py)
- **Payments:** Razorpay integration (India-focused)
- **Auth/config (optional):** Firebase (via firebase_service.py)
- **Chemical utilities:** SMILES generation and structure rendering (may use external APIs or local tools like RDKit via smiles_service.py and chemical_structure_service.py)

2.4 Packaging & Containerization

- requirements.txt and/or pyproject.toml for Python dependencies
- Dockerfile for containerized deployment
- Procfile for process specification (e.g., for Heroku-like environments)

---

3. High-Level Architecture

PharmaDocs-AI follows a typical **layered architecture**:

1. **Presentation Layer (Flask Blueprints + Jinja2 templates)**
    - Handles HTTP requests, renders HTML, and returns JSON for AJAX requests.

2. **Service Layer (services/)**
    - Encapsulates business logic, parsing, calculations, and external integrations.
3. **Data Access Layer (models.py, database.py)**
    - SQLAlchemy models and DB session management.
4. **Integration Layer**
    - Cloudinary, Razorpay, Firebase, and chemical data/structure APIs.

3.1 Main Components

- **Flask App**
    - Entry-points: app.py / application.py / main.py (depending on deployment).
    - Registers blueprints from app_routes/.
    - Initializes database, configuration, and external services.
- **Database Layer**
    - database.py – SQLAlchemy instance and session.
    - models.py – ORM models (User, Company, Document, AMVDocument, etc.).
    - migrations/ – Alembic migrations.
- **Services Layer (services/)**
    - amv_report_service.py – AMV report generation (DOCX) and calculations.
    - method_extraction_service.py – method parameter extraction from PDF.
    - analytical_method_verification_service.py – verification calculations and ICH checks.
    - cloudinary_service.py – Cloudinary file handling.
    - razorpay_service.py / payment_service.py – payment and subscription flows.
    - smiles_service.py, chemical_structure_service.py – chemical utilities.
    - document_service.py – document-related helpers.
    - Other feature-specific helpers.
- **Routes / Blueprints (app_routes/)**
    - amv_routes.py – AMV creation, extraction, verification, report generation.
    - auth.py – login, register, logout.

    - admin.py – admin operations.
    - company.py, documents.py, dashboard.py, razorpay_routes.py, subscription_routes.py, etc.
- **Templates & Static**
    - templates/ – HTML templates for all pages and flows.
    - static/js/, static/css/ – frontend logic and styles.
- **File Storage**
    - uploads/ – user-uploaded PDFs, Excel, signatures.
    - reports/ – generated DOCX files.
- **Utilities (utils/)**
    - helpers.py, validators.py, subscription_middleware.py – shared helpers.

---

4. Folder & File Responsibilities

4.1 Root

- app.py / application.py / main.py
    - Flask initialization and app factory.
    - Registers blueprints from app_routes.
    - Binds DB and configuration.
- database.py
    - Creates SQLAlchemy db instance.
    - Provides DB initialization hooks.
- models.py
    - Defines SQLAlchemy models:
        - User, Company, Document
        - AMVDocument, AMVVerificationDocument
        - Equipment, Reagent, ReferenceProduct, GlassMaterial, OtherMaterial
        - Subscription/payment-related models.
- Dockerfile

- Procfile
- requirements.txt / pyproject.toml

4.2 app_routes/

Contains **Flask blueprints**. Key ones:

- amv_routes.py
    - AMV create/list/view.
    - Method extraction from PDFs.
    - AMV report generation (DOCX).
    - AMV verification upload/create/view.
    - Equipment/materials/reagents/reference product settings.
    - SMILES and chemical structure generation APIs.
    - ICH validation / mathematical calculation test endpoints.
    - Protocol generator endpoints (method-based validation plan).
- auth.py
    - Login, registration, logout.
    - Session management.
- admin.py
    - Admin dashboard and management screens.
- Other routes:
    - company.py – company profile, logo, metadata.
    - documents.py – generic document listing and detail.
    - dashboard.py – user dashboard overview.
    - razorpay_routes.py – payment callbacks, order creation, webhooks.
    - subscription_routes.py – subscription plans and status.
    - validation_routes.py, pdf_generator.py, pvr_templates.py, etc. – additional domain flows.

4.3 services/

- amv_report_service.py

    - Class AMVReportGenerator with methods to:
        - Create AMV DOCX reports.
        - Perform AMV calculations (system suitability, linearity, precision, recovery, etc.).
        - Validate results against ICH Q2(R1) criteria.
    - Utilities to extract methods from PDFs and process raw data files.
- method_extraction_service.py
    - PDFs → text → parsed method parameters.
    - Returns structured data and human-readable summary.
    - Suggests validation parameters based on extracted method and instrument type.
- analytical_method_verification_service.py
    - Generates AMV verification protocols.
    - Processes Excel raw data to compute verification statistics.
    - Performs ICH Q2(R1) compliance checks.
- cloudinary_service.py
    - Upload and manage files on Cloudinary.
- razorpay_service.py / payment_service.py
    - Plan creation, payment initiation, and success/failure handling.
    - Subscription linkage to User/Company.
- firebase_service.py
    - Firebase configuration (e.g., auth, messaging).
- smiles_service.py
    - Generate SMILES/fingerprint/metadata from compound name.
- chemical_structure_service.py
    - Generate structure image from SMILES.
- document_service.py
    - Abstractions around Document CRUD.

4.4 templates/

- Main pages:
    - index.html, dashboard.html
    - login.html, register.html
- AMV-related templates:
    - create_amv.html, view_amv.html, amv_list.html
    - amv_verification.html, view_amv_verification.html
    - amv_verification_protocol.html, protocol_generator.html
- Admin/company:
    - Admin dashboard templates, company settings templates, subscription pages.

4.5 static/

- static/css/style.css – global styling.
- static/js/ – feature-specific logic:
    - amv.js – AMV form behaviors, extraction & validation calls.
    - auth.js – login/registration helpers.
    - dashboard.js, document-creation.js, file-upload.js, subscription.js, utils.js, etc.
    - firebase-config.js – Firebase initialization (if used).

4.6 Other

- uploads/ – user uploads.
- reports/ – generated reports.
- migrations/ – Alembic migration scripts.
- scripts/ – setup scripts (e.g., Razorpay plan init).
- utils/ – shared utilities & middleware.

---

5. Data Model Overview

Note: Exact fields must be checked in models.py. Below is a high-level summary based on route usage.

5.1 Core Models

User

- **Typical fields:**
    - id, name, email, password_hash
    - role / is_admin
    - Subscription-related fields (plan, expiry, status)
- **Behavior:**
    - can_create_document() – checks subscription & role permissions.

Company

- **Typical fields:**
    - id, name, address
    - owner_user_id
    - logo_url
    - metadata JSON (optional)

Document

Generic document metadata.

- id
- user_id, company_id
- title
- document_type ('AMV', 'AMV_VERIFICATION', etc.)
- document_number
- document_metadata (JSON)
- generated_doc_url
- status
- created_at, updated_at

AMVDocument

Linked to Document (via document_id).

- product_name
- label_claim
- active_ingredient

- strength
- instrument_type (e.g., 'HPLC', 'UV', 'AAS', 'GC', UPLC, titration)
- val_params (JSON)
- parameters_to_validate (JSON)
- instrument_params (JSON)
- report_generated (bool)
- Helper methods:
    - get_validation_params()
    - get_parameters_to_validate()
    - get_instrument_params()

AMVVerificationDocument

- Verification-specific fields similar to AMVDocument.
- Links to Document and may reference uploaded Excel/raw data.

Equipment & Materials

- Equipment
- GlassMaterial
- Reagent
- ReferenceProduct
- OtherMaterial

Shared characteristics:

- Typically scoped by company_id.
- Used to build protocol content and materials sections.

Subscription & Payment Models

- Models for:
    - Subscription plans
    - Payment records
    - Subscription status per user/company

---

6. Key Flows & Data Flows

6.1 AMV Creation & Report Generation

1. **Create AMV Document**
    - User visits /amv/create (GET).
    - Fills form (AMV details, product info, instrument type, etc.).
    - Optionally uploads method PDF.
    - On submit (POST), system:
        - Validates input.
        - Creates a Document (type = AMV).
        - Creates associated AMVDocument.
        - Saves uploaded file to uploads/ and/or Cloudinary.
        - Commits DB transaction.
2. **Generate AMV Report (DOCX)**
    - User uploads signatures and triggers report generation via /amv/<id>/generate-report (POST).
    - Backend:
        - Loads Document, AMVDocument, Company, and user data.
        - Prepares form_data dict.
        - Instantiates AMVReportGenerator.
        - Generates DOCX, stores file in reports/.
        - Updates Document.generated_doc_url.
        - Sends file to user via send_file.

6.2 Method Extraction from PDF

1. User uploads method PDF and selects instrument type via /amv/api/extract-method.
2. Backend:
    - Reads file content.
    - Calls method_extraction_service.extract_method_parameters(file_content, instrument_type).
    - Receives structured parameters + summary.

- Returns JSON: {success, parameters, summary}.

3. Frontend:
    - Displays extracted parameters.
    - User can adjust and select validation parameters.

6.3 AMV Verification Protocol & Report Generation 🚧 (Partially Implemented)

Current status: core verification flow is implemented (upload, process, generate DOCX), but full parameter coverage & UI integration are in progress.

1. **Initiate AMV Verification**
    - **User visits /amv/verification (GET).**
    - **Chooses an existing AMV or product/instrument context.**
    - **Provides:**
        - **Reference to method.pdf:**
            - **Either by upload, or**
            - **By selecting/storing a method link / method document URL instead of a raw file upload.**
        - **Uploads raw data (Excel) for verification (e.g., recovery, precision, linearity data).**
2. **Upload & Process Verification Data**
    - **POST /amv/verification/upload**
    - **Inputs:**
        - **excel_file (raw data measurements).**
        - **method_pdf or method_pdf_link (planned: support URL-based linkage, not just upload).**
        - **Additional metadata (instrument type, batch info, analyst, date range, etc.).**
    - **Backend:**
        - **Reads Excel and parses structured datasets (replicates, concentrations, areas).**
        - **Reads method information either:**
            - **Directly from PDF OR**

        - From a stored method.pdf link that points to an already registered method.
    - Passes parsed data + method metadata to analytical_method_verification_service.

3. Verification Parameter Handling (🚧 Pending Enhancements)
    - Current status:
        - Core verification calculations (e.g., basic linearity/precision) are implemented.
        - Some parameters are not yet fully integrated into:
            - UI parameter selection,
            - Back-end mapping from UI → verification engine.
    - Planned:
        - Complete mapping of all ICH Q2(R1) verification parameters (e.g., accuracy, intermediate precision, robustness, LOQ, LOD, etc.) into:
            - Form configuration,
            - Validation rules,
            - Displayed results in the verification doc.
        - Ensure that each configured parameter appears in:
            - Input form,
            - Calculations,
            - Final verification protocol/report.
4. Generate AMV Verification Report
    - Once data is processed:
        - analytical_method_verification_service returns:
            - Verification statistics (R^2, %RSD, bias, recovery, etc.).
            - Pass/fail decisions against ICH criteria.
        - A verification DOCX is generated similar to AMV report:
            - Includes product/method details.
            - Displays each evaluated parameter with:

        - **Acceptance criteria,**
        - **Observed result,**
        - **Pass/Fail status.**
    - **Includes signature sections, dates, and versioning details.**
- **Document is:**
    - **Saved under reports/,**
    - **Linked to a AMVVerificationDocument + Document entry,**
    - **Downloadable via /amv/verification/<document_id>/download.**

---

7. Routes & APIs

Note: Exact URL names may vary slightly; check app_routes/amv_routes.py & others.

7.1 Authentication

- **POST** /auth/login
    - **Input:** email, password
    - **Output:** Session cookie; redirect to dashboard or error.
- **POST** /auth/register
    - Register new user.
- **GET** /auth/logout
    - Clear session and redirect to login.

7.2 AMV Documents

- **GET** /amv/create
    - Render AMV creation form.
- **POST** /amv/create
    - Create new AMV document + AMVDocument.
    - **Input:** Form fields, optional uploaded method_pdf.
- **GET** /amv/<int:document_id>
    - View AMV document details.

- **GET** /amv/list
    - List AMV documents for logged-in user / company.
- **POST** /amv/<int:document_id>/generate-report
    - Generate AMV DOCX report.
    - **Input (multipart):**
        - Optional signature files: engineer_signature, manager_signature, approved_signature.
- **GET** /amv/<int:document_id>/download
    - Download generated AMV report (DOCX).

7.3 AMV Settings (Equipment & Materials)

- **GET/POST** /amv/settings/equipment
- **GET/POST** /amv/settings/materials
- **GET/POST** /amv/settings/reagents
- **GET/POST** /amv/settings/reference-products

Used for managing company-specific equipment & materials lists.

7.4 Protocol Generator APIs

- **GET** /amv/protocol
    - Render protocol generator UI.
- **GET** /amv/protocol/api/methods
    - Returns supported methods (e.g., ["uv", "aas", "hplc", "uplc", "gc", "titration"]).
- **GET** /amv/protocol/api/parameters/<method>
    - Returns list of parameters to validate for given method.
- **POST** /amv/protocol/api/generate_protocol
    - **Input (JSON):** extractedData, selectedParams, selectedMethod
    - **Output (JSON):** Generated protocol structure.
- **POST** /amv/protocol/api/download_protocol
    - **Input (JSON):** Protocol content.
    - **Output:** DOCX/TXT stream for download.

7.5 AMV Method Extraction & Utilities

- **POST** /amv/api/generate-number
    - **Input (JSON):** {company_id}
    - **Output:** {document_number}
        - Format: <COMPANY_PREFIX>/AMV/<YEAR>/<COUNT+1:04d>
- **POST** /amv/api/extract-method
    - **Input (multipart):**
        - method_file (PDF)
        - instrument_type
    - **Output (JSON):** {success, parameters, summary}
- **POST** /amv/api/suggest-validation
    - **Input (JSON):**
        - extracted_params
        - parameters_to_validate
    - **Output (JSON):** Suggestions for validation plan.
- **POST** /amv/api/test-mathematical-calculations
    - Test endpoint for calculations in AMVReportGenerator.
- **POST** /amv/api/validate-ich-criteria
    - **Input (JSON):** results, parameter
    - **Output (JSON):** validation_status, overall_compliance
- **POST** /amv/api/generate-smiles
    - **Input (JSON):** ingredient_name
    - **Output (JSON):** smiles, inchi, formula, etc.
- **POST** /amv/api/generate-structure
    - **Input (JSON):** smiles
    - **Output:** chemical structure image URL or binary file.
- **POST** /amv/extract-method

    - Alternate/legacy extraction endpoint using inline MethodPDFExtractor (duplicate logic with /amv/api/extract-method).

7.6 AMV Verification

- **POST** /amv/verification/upload
    - **Input (multipart):** excel_file, method_pdf, possibly other fields.
    - **Output:** DOCX verification protocol stream.
- **POST** /amv/verification/create
    - Create AMVVerificationDocument and related Document record.
- **GET** /amv/verification/<int:document_id>
    - View verification details.
- **GET** /amv/verification/<int:document_id>/download
    - Download verification protocol DOCX.
- **GET** /amv/verification
    - Render verification protocol UI.

7.7 Payments & Subscriptions

- Endpoints under:
    - razorpay_routes.py
    - subscription_routes.py

Typical operations:

- Create Razorpay order/session.
- Handle success/failure callbacks/webhooks.
- Update user/company subscription status.

---

8. Services — Contracts & Responsibilities

8.1 AMVReportGenerator (in amv_report_service.py)

Inputs:

- form_data (dict) **must include**:
    - document_title

    - product_name
    - label_claim
    - active_ingredient
    - strength
    - instrument_type
    - document_number
    - val_params (list/dict)
    - parameters_to_validate
    - instrument_params
    - Optional: signature file paths / image bytes.
- company_data (dict, optional):
    - company_name, address, logo_path/url, etc.

Core Responsibilities:

- Generate AMV DOCX with:
    - Company header (logo, name, address).
    - Method details & validation parameters.
    - Insertion of calculation results and tables.
    - Signature blocks using provided images.
- Implement mathematical & statistical functions:
    - System suitability (e.g., RSD, plate count).
    - Linearity, accuracy, precision, recovery.
    - ICH Q2(R1) criteria checks (pass/fail).

Outputs:

- Path to generated DOCX file
  OR BytesIO buffer to be returned directly via send_file.

Error Modes:

- Missing or malformed numeric inputs → ValueError or custom exceptions.
- File IO failures → propagate or wrap with meaningful messages.

- Should not rely on external network calls for core calculations.

8.2 method_extraction_service.py

Inputs:

- file_content: raw bytes of uploaded PDF.
- instrument_type: string ('hplc', 'uv', 'gc', etc.).

Responsibilities:

- Extract text from PDF.
- Use heuristics (regex, patterns) to find:
    - Wavelength, flow rate, injection volume, column details.
    - Mobile phase composition.
    - Standard/sample concentration, volume, and weights.
- Build:
    - Structured parameters dictionary.
    - Human-readable summary string of method.

Outputs:

- { "parameters": {...}, "summary": "..." } (or equivalent dict)

Error Modes:

- Unreadable/malformed PDF (return success: false with message).
- Unsupported MIME type.
- Insufficient text content.

8.3 analytical_method_verification_service.py

Inputs:

- Raw data (from Excel or JSON).
- Selected instrument & validation parameters.
- Additional configuration from the form.

Responsibilities:

- Process raw data (e.g., peak areas, concentrations).
- Calculate AMV verification statistics:
    - Linearity, precision, accuracy, robustness, etc.
- Perform ICH Q2(R1) compliance checks.
- Generate verification protocol content (DOCX or structured data).

Outputs:

- Verification results dict (with pass/fail, statistics).
- Optional: DOCX buffer for protocol.

Error Modes:

- Missing data.
- Insufficient number of replicates.
- Invalid numeric entries.

8.4 Cloud & Payment Services

cloudinary_service.py

- Upload files to Cloudinary.
- Return their secure URLs.
- Handle deletion if needed.

razorpay_service.py / payment_service.py

- Create Razorpay orders.
- Verify payment signatures.
- Mark subscription as active/expired.
- Store payment records for audit.

---

9. Security, Validation & Error Handling

9.1 Authentication & Authorization

- Uses Flask session (e.g., session['user_id']).
- Most protected routes check user presence in session.

- Recommended:
    - Use decorators (e.g., @login_required) to centralize auth checks.
    - Implement role-based access control for admin-only endpoints.

9.2 Input Validation

- Validate all form and JSON inputs.
- For file uploads:
    - Check extension (.pdf, .xlsx, etc.).
    - Use secure_filename.
    - Enforce MAX_CONTENT_LENGTH and content-type checks.
- Use schema-based validation (e.g., Marshmallow/Pydantic) for complex payloads where possible.

9.3 SQL Injection & XSS

- SQLAlchemy ORM protects against SQL injection when used correctly.
- Jinja2 auto-escapes templates; only use |safe when absolutely necessary.
- Escape or sanitize any rich text stored and displayed from user inputs.

9.4 File Storage & Security

- Store files in uploads/ & reports/ (non-public path in production).
- Use Cloudinary for persistent/served assets.
- Ensure proper permissions and regular cleanup of temporary files.

9.5 Secrets Management

- All API keys, DB credentials, and secret keys must be stored in environment variables:
    - SECRET_KEY
    - DATABASE_URL
    - CLOUDINARY_*
    - RAZORPAY_*
    - FIREBASE_*
- Never commit secrets to the repository.

---

10. Running the Project

10.1 Local Development

1. **Clone & Setup Environment**

```
python -m venv venv
source venv/bin/activate  # Windows: venv\Scripts\activate
pip install -r requirements.txt
```

2. **Configure Environment Variables**

Create .env (not committed):

```
FLASK_APP=app.py
FLASK_ENV=development
SECRET_KEY=your-secret-key
DATABASE_URL=postgresql://user:pass@localhost/pharmadocs_ai
CLOUDINARY_CLOUD_NAME=...
CLOUDINARY_API_KEY=...
CLOUDINARY_API_SECRET=...
RAZORPAY_KEY_ID=...
RAZORPAY_KEY_SECRET=...
UPLOAD_FOLDER=./uploads
PORT=5000
```

3. **Run Migrations**

```
alembic upgrade head
# or if using Flask-Migrate:
# flask db upgrade
```

4. **Start the App**

```
flask run
# or
python app.py
```

10.2 Docker

1. **Build**

```
docker build -t pharmadocs-ai .
```

2. **Run**

```
docker run -p 5000:5000 \
  -e SECRET_KEY=your-secret \
  -e DATABASE_URL=... \
  -e CLOUDINARY_CLOUD_NAME=... \
  -e CLOUDINARY_API_KEY=... \
  -e CLOUDINARY_API_SECRET=... \
  -e RAZORPAY_KEY_ID=... \
  -e RAZORPAY_KEY_SECRET=... \
  pharmadocs-ai
```

3. **Production**

- Use Gunicorn (e.g., web: gunicorn application:app from Procfile).
- Put behind Nginx / reverse proxy.
- Use HTTPS.

---

11. Testing & CI (Recommended)

11.1 Tests

Suggested test areas:

- **Unit tests:**
  - Method extraction parsing (method_extraction_service).
  - AMV calculations in AMVReportGenerator (e.g., RSD, plate count).
  - ICH validation logic in analytical_method_verification_service.
  - Document number generation logic.
- **Integration tests:**
  - Authentication (login/logout).
  - AMV create → generate report workflow.

    - AMV verification upload → DOCX generation.

Use **pytest** as the recommended test runner.

11.2 CI Pipeline

- GitHub Actions workflow:
    - Setup Python.
    - Install dependencies.
    - Run flake8 or pylint.
    - Run pytest.
- Optional:
    - Add mypy for type checking.
    - Add black/isort for formatting.

---

12. Known Caveats & Improvements

12.1 Duplicated Endpoints

- Both /amv/api/extract-method and /amv/extract-method exist with overlapping functionality.
- **Recommendation:** Standardize on one endpoint with a shared service.

12.2 Long-Running Tasks

- PDF parsing and DOCX generation can be heavy.
- **Recommendation:**
    - Move to background tasks using Celery + Redis / RQ.
    - Return job IDs and have the frontend poll for status.

12.3 File Size & Limits

- Enforce max upload file size.
- Add friendly error messages for large PDF/Excel files.

12.4 Validation & Error Handling

- Centralize validation & error responses.
- Use consistent JSON error structures and HTTP status codes.

---

13. Roadmap & Future Enhancements

- **High Priority**
    - Background job queue for heavy operations.
    - Consolidate method extraction services & endpoints.
    - Add proper RBAC and @login_required decorators.
    - Introduce automated tests and CI.
- **Medium Priority**
    - Full Swagger/OpenAPI documentation for APIs.
    - Rate limiting & DOS protection.
    - Central logging & monitoring (e.g., Sentry).
- **Low Priority / Nice-to-Have**
    - Versioning for documents and reports.
    - Full audit trail (who changed what and when).
    - ML-based method extraction (NLP-driven) while keeping numeric calculations deterministic and non-AI as per domain constraints.

---

14. Developer Guidelines

- Keep **domain logic** in services/, not in routes.
- Keep **routes** thin: authenticate, validate, call services, format response.
- When updating models:
    - Create a new Alembic migration.
    - Update any affected services and tests.
- When adding new endpoints:
    - Consider if it should be HTML-rendering, JSON-only, or both.
    - Update docs and, if using Swagger, the OpenAPI spec.

15. Current Implementation Status Summary

1. **Analytical Method Verification (AMV)**

- **AMV Creation & Report Generation**:
    - Implemented end-to-end (UI + backend + DOCX).
    - Supports method metadata, validation parameters, signatures, and ICH-style calculations.
- **Method PDF Handling**:
    - Currently supports direct upload.
    - Planned: robust support for **method.pdf link / referenced document**, stored against AMV so that verification and related flows always reference the same method version.

2. **AMV Verification**
    - Core flow implemented:
        - Upload of verification raw data (Excel).
        - Processing via verification service.
        - DOCX generation with verification protocol.
    - **Pending Enhancements**:
        - Full parameter mapping for all verification items.
        - Auto-population of parameters in the verification report.
        - Clean method linkage via method.pdf URL/ID rather than repeated raw upload.
3. **Stability Studies**
    - **Planned**:
        - Service definition done at design level.
        - Will handle multi-condition, multi-timepoint stability evaluations and reporting.
4. **Degradation / Stress Studies**
    - **Planned**:
        - Design focuses on proving specificity and stability-indicating capability of methods.
        - Will structurally integrate stress conditions, observed degradants, and chromatographic performance.

Link of method.pdf file which I used in this project
https://drive.google.com/file/d/1t9hD3FOaKJS_wYr7yjJYB1pvzX71s_Fi/view?usp=drivesdk

system generated documents
https://drive.google.com/drive/folders/1y7al5aAbJ358JHDawdZMvEAcm8f2q1s0?usp=drive_link

reference documents
https://drive.google.com/drive/folders/1FfiGy8TqsTnLK06-bWDkt1gaPZFeaNqL?usp=drive_link