Portfolio Dashboard - User Manual and Engineering Handoff

Last updated: 09-02-2026

1. Purpose

This document is the full handoff manual for the Portfolio Dashboard project.
It covers:

1. What the system does.
2. How to run it locally (system setup) and on a server.
3. What each frontend page does.
4. What every route in `backend/app.py` does.
5. How parsing, deduplication, service requests, and admin workflows work.
6. Database structure, operational runbook, and production hardening notes.

---

2. System Overview

The project is a full-stack portfolio ingestion and analytics system:

1. Frontend: React + TypeScript + Vite + Tailwind.
2. Backend: Flask + PostgreSQL + Redis-backed session storage.
3. Ingestion: PDF statements (NSDL/CDSL/CAMS) parsed with PyMuPDF and regex parsers.
4. Analytics: Dashboard aggregates holdings, classifications, AMC/category charts, and Morningstar returns cache.
5. Roles: `user` and `admin`, with service-request workflow and admin approval flow.

High-level flow:

1. User registers (request is pending, not immediately active).
2. Admin approves registration.
3. User logs in with password + OTP email.
4. User uploads one or more statement PDFs.
5. Backend parses and stores holdings; cross-file duplicates go into `portfolio_duplicates`.
6. User views dashboard/history/snapshot/audit.
7. User can raise service requests; admin resolves and updates data.

---

3. Repository Map

Root-level structure:

1. `backend/`: Flask app, DB connector, OTP sender, parser modules.
2. `src/`: React app (pages, components, auth context, hooks).
3. `uploads/`: Stored uploaded files (runtime data, not source).

4. `portfolio_backup.dump`: PostgreSQL custom-format backup with schema + data.
5. `vite.config.ts`: Frontend proxy/build settings.

Key backend files:

1. `backend/app.py`: Main API and all routes.
2. `backend/db.py`: DB connection config (hardcoded).
3. `backend/ecasparser.py`: Dispatches file to NSDL/CDSL/CAMS parser.
4. `backend/nsdl_parser.py`, `backend/cdsl_parser.py`, `backend/cams_parser.py`: Statement-specific parsing + insert logic.
5. `backend/dedupe_context.py`: Request-scoped duplicate detection context.
6. `backend/morningstar.py`: Fetches and upserts trailing returns into `historic_returns`.
7. `backend/otpverification.py`: SMTP OTP email sender.

Key frontend files:

1. `src/App.tsx`: Frontend route map and role-protected routes.
2. `src/context/AuthContext.tsx`: Session restore/login/logout/OTP completion state.
3. `src/pages/*`: User/admin pages.
4. `src/components/*`: Shared layout, sidebar, holdings table, upload form, etc.

---

4. Local Startup (System Setup)

4.1 Prerequisites

Install:

1. Node.js 20+ and npm.
2. Python 3.10+.
3. PostgreSQL (project backup created from PG 17.7; PG 14+ recommended minimum).
4. Redis server (required for Flask session backend).

4.2 Clone and frontend install

```
git clone <repo-url>
cd Portfolio-Dashboard
npm install
```

4.3 Python backend environment

Install backend dependencies from committed requirements.

For local/dev (current baseline):

```
cd backend
python3 -m venv .venv
source .venv/bin/activate
pip install -r requirements.txt
```

For UAT server (Python 3.9 baseline):

```
cd backend
python3 -m venv .venv
source .venv/bin/activate
pip install -r requirements.txt(on the uat server)
```

Windows PowerShell equivalent (local/dev):

```
cd backend
python -m venv .venv
.venv\Scripts\Activate.ps1
pip install -r requirements.txt
```

Python 3.9 compatibility note:

1. UAT is pinned to Python 3.9 and uses `requirements.txt`.
2. Backend modules that rely on modern type-hint syntax include `from __future__ import annotations`.
3. Keep that import at the top of those files during future refactors.

4.4 PostgreSQL setup and restore

Create DB and restore backup:

```
cd /path/to/Portfolio-Dashboard
createdb portfolio_db
pg_restore --clean --if-exists --no-owner -d portfolio_db portfolio_bac
```

Notes:

1. Backup archive timestamp: 2026-02-02.
2. Roles table includes:
    1. `admin` (`role_id=1`)
    2. `user` (`role_id=2`)

Default expected by backend: `127.0.0.1:6379`.

4.5 Configure DB credentials

Edit `backend/db.py`:

1. `dbname` defaults to `portfolio_db`.
2. `user` defaults to `sohamathawale`.
3. `password` currently blank.
4. `host` defaults to `localhost`.
5. `port` defaults to `5432`.

4.6 Run backend and frontend

From repo root:

1. Backend:

```
cd backend
source .venv/bin/activate
python app.py
```

Backend runs at `http://127.0.0.1:8010`.

1. Frontend:

```
cd /path/to/Portfolio-Dashboard
npm run dev
```

Frontend runs at `http://localhost:5173`.

4.7 Quick smoke checks

1. Backend health:

```
curl http://127.0.0.1:8010/pmsreports/
```

1. Frontend load: Open `http://localhost:5173/login`.

2. Session check (after login):

```
curl -i http://127.0.0.1:8010/pmsreports/check-session
```

---

5. Server Startup and Deployment Guide

This project is currently coded for local use, so production deployment requires hardening and config edits.

5.1 Required production edits before deploy

1. Move hardcoded secrets to environment variables:
    1. Flask secret key.
    2. SMTP credentials.
    3. DB credentials.
2. Set secure session cookie:
    1. `SESSION_COOKIE_SECURE=True`
    2. `SESSION_COOKIE_SAMESITE` according to frontend domain strategy.
3. Restrict CORS origins to actual frontend domain(s).
4. Add auth decorators to currently open admin/data endpoints (see Section 13).

5.2 Backend process management with PM2

UAT service is run with PM2 using Python interpreter mode.

```
pm2 start app.py \
 --name PMS-reports \
 --interpreter python3 \
 --cwd /home/ndpms_user/PMS_reports/backend
```

Recommended PM2 persistence commands:

```
pm2 save
pm2 startup
```

Notes:

1. Backend app listens on port `8010` (`python app.py` behavior).
2. Keep Redis and PostgreSQL services available before PM2 starts this process.
3. Make sure PM2 uses the same Python environment where dependencies were installed.

5.3 Frontend build and serve

```
cd /opt/Portfolio-Dashboard
npm ci
npm run build
```

Build output: `dist/`.

5.4 Nginx reverse proxy example

```
server {
    listen 80;
    server_name your-domain.com;

    root /opt/Portfolio-Dashboard/dist;
    index index.html;

    location / {
        try_files $uri /index.html;
    }

    location /pmsreports/ {
        proxy_pass http://127.0.0.1:8010/pmsreports/;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
    }
}
```

After config:

```
sudo nginx -t
sudo systemctl reload nginx
```

5.5 Frontend API base in production

Set `VITE_API_URL` at build time if frontend and backend are on different origins.
If same-origin with Nginx proxy, keep using `/pmsreports`.

---

6. Configuration Reference

Area	Location	Current behavior	Handoff note
API base (frontend)	`import.meta.env.VITE_API_URL		'/pmsreports` in multiple files
Vite proxy	`vite.config.ts`	`/pmsreports -> 127.0.0.1:8010`	Keep backend on 8010 or update proxy
Backend process manager (UAT)	PM2 command	PM2 launches `app.py` via Python interpreter	Use PM2 command in Section 5.2
DB config	`backend/db.py`	Hardcoded dict	Move to env vars
Python requirements (UAT)	`backend/requirements.txt`	Separate dependency set for Python 3.9	Use this on UAT only
Python requirements (dev/local)	`backend/requirements.txt`	Default dependency set	Use for local development
Flask session store	`backend/app.py`	Redis session backend (`127.0.0.1:6379`)	Redis is mandatory
CORS	`backend/app.py`	Allows only localhost:5173 and 127.0.0.1:5173	Update for deploy domains
Session cookie	`backend/app.py`	`Lax`, secure false	Must set secure true on HTTPS
Upload folder	`backend/app.py`	Relative `"uploads"`	Use absolute path in production
OTP email	`backend/otpverification.py`	SMTP creds hardcoded	Move to secret manager/env

---

7. Database Handoff

Source of truth used here: `portfolio_backup.dump` schema.

7.1 Core tables and roles

1. `users`: User identity, contact, password hash, family mapping.
2. `families`: Family grouping for one primary user + family members.
3. `family_members`: Member records with two IDs:
    1. `id` (global canonical primary key).
    2. `member_id` (per-family sequence shown to client).

4. `roles`: Role master (`admin`, `user`).
5. `user_roles`: Maps user to role.
6. `pending_registrations`: Staging table for signup approval flow.
7. `login_otps`: One-time login OTP records.
8. `portfolios`: Canonical holdings table.
9. `portfolio_duplicates`: Duplicate candidates captured across multi-file uploads.
10. `historic_returns`: Morningstar returns cache keyed by ISIN.
11. `service_requests`: User-raised change/support requests.

7.2 Key triggers/functions

1. `create_family_for_user()` + trigger `auto_create_family`:
    1. On user insert with `family_id` null, auto-creates family.
2. `assign_service_request_id()` + trigger `trg_assign_service_request_id`:
    1. Assigns per-user incremental `request_id` in `service_requests`.

7.3 Important constraints/indexes

1. `unique_family_member` on `(family_id, member_id)`.
2. `uniq_duplicate_link` unique index on `portfolio_duplicates.linked_portfolio_entry_id` when not null.
3. Portfolio lookup indexes:
    1. `idx_portfolios_user`
    2. `idx_portfolios_portfolio_id`
    3. `idx_portfolios_member`
    4. `idx_portfolios_isin`

7.4 Role seed

`roles` table data in backup:

1. `role_id=1`, `role_name='admin'`
2. `role_id=2`, `role_name='user'`

---

8. API Reference (`backend/app.py`)

Auth legend:

1. `Public`: no session check/decorator.
2. `Session`: requires user session (manual check or `@login_required`).
3. `Admin`: requires admin session role.
4. `Open-admin`: admin/data endpoint currently missing admin/session guard in code.

8.1 Health, auth, registration, session

Method	Path	Auth	Purpose	Input	Output

GET	/pmsreports/	Public	Health check	None	{message}
POST	/pmsreports/register	Public	Create pending registration	JSON: email, phone, password	201 with pending_id
GET	/pmsreports/admin/pending-registrations	Admin	List pending signups	None	Pending rows
POST	/pmsreports/admin/approve-registration/<pending_id>	Admin	Approve pending user + create family + assign user role	Path param	New user_id, family_id
DELETE	/pmsreports/admin/reject-registration/<pending_id>	Admin	Reject pending signup	Path param	{message}
GET	/pmsreports/admin/approved-accounts	Admin	List approved users	None	User list
POST	/pmsreports/login	Public	Password check + OTP generation/email	JSON: email, password	{otp_required: true, user_id, role, email}
POST	/pmsreports/verify-otp	Public	Validate OTP and create session	JSON: email, otp	Session user payload
POST	/pmsreports/check-email	Public	Email availability check (users + pending_registrations)	JSON: email	{exists, pending?}
POST	/pmsreports/check-phone	Public	Phone availability check (users + pending_registrations)	JSON: phone	{exists}
POST	/pmsreports/logout	Public (session-aware)	Clears session	None	{message}
GET	/pmsreports/check-session	Public	Session presence and role summary	None	{logged_in, user_id, email, role...}
GET	/pmsreports/session-user	Session	Returns user + role by session user ID	None	{user}

8.2 Upload, dashboard, history, portfolio

Method	Path	Auth	Purpose	Input	Output

POST	/pmsreports/upload	Session	Upload one or more PDFs for self; creates new portfolio_id	Multipart: email, files[], file_types[], passwords[]	Per-file summa + totals
GET	/pmsreports/dashboard-data	Session	Dashboard analytics for selected user/ member filters	Query: include_user=true/false, members=1,2,...	Summary + charts + holdi
GET	/pmsreports/history-data	Session	List historical portfolios across user + family	None	Portfolio histor list
GET	/pmsreports/portfolio/<portfolio_id>/members	Session	Snapshot analytics by member and all-members aggregate	Path param	{portfolio_ members:[..
DELETE	/pmsreports/delete-portfolio/<portfolio_id>	Session	Delete portfolio rows for current user and portfolio ID	Path param	{message}
GET	/pmsreports/portfolio/latest	Session	Latest portfolio ID for current user	None	{portfolio_

8.3 Family management

Method	Path	Auth	Purpose	Input	Output
POST	/pmsreports/upload-member	Session	Upload member statements tied to selected member and latest user portfolio ID	Multipart: member_id (per-family), files[], file_types[], passwords[]	Per-file summary + totals
POST	/pmsreports/family/add-member	Session	Add family member with auto-incremented per-family member_id	JSON: name, email?, phone?	Created member
DELETE	/pmsreports/family/delete-member/<member_id>	Session	Delete member by per-family member_id	Path param	{message}

GET	`/pmsreports/family/members`	Session (`@login_required`)	List members for current user family	None	Member list

8.4 User service requests

Method	Path	Auth	Purpose	Input	Output
POST	`/pmsreports/service-requests`	Session (`@login_required`)	Create request (`Change Email`, `Change Phone`, `Portfolio Update`, `General Query`)	JSON: `request_type`, `description?`, `member_id?` (per-family)	Created request
GET	`/pmsreports/service-requests`	Session (`@login_required`)	List own requests with member name	None	Request list
DELETE	`/pmsreports/service-requests/<req_id>`	Session (`@login_required`)	Delete own pending request only	Path param	`{message}`

8.5 Admin service requests

Method	Path	Auth	Purpose	Input	Output
GET	`/pmsreports/admin/service-requests`	Admin	List all requests, optional type filter	Query: `type?`	Request list
PUT	`/pmsreports/admin/service-requests/<req_id>`	Admin	Update status and/or admin note	JSON: `status?`, `admin_description?`	`{message}`
DELETE	`/pmsreports/admin/service-requests/<req_id>`	Admin	Delete request	Path param	`{message}`
POST	`/pmsreports/admin/service-requests/<req_id>/perform`	Admin	Execute request-specific business update and mark completed	JSON varies by `request_type`	Completion result

PATCH	/pmsreports/admin/service-requests/<req_id>/add-note	Admin	Add/update note without status change	JSON: admin_description	Updated note

perform payload variants:

1. Change Email: new_email, optional admin_description.
2. Change Phone: new_phone, optional admin_description.
3. Portfolio Update: portfolio_entry_id, fields object, optional admin_description.
4. General Query: optional admin_description only.

8.6 Admin analytics and user detail

Method	Path	Auth	Purpose	Input	Output
GET	/pmsreports/admin/stats	Open-admin	Global admin stats (users, families, portfolios, requests)	None	Aggregated stats
GET	/pmsreports/admin/user/<user_id>	Open-admin	Detailed user profile, holdings, monthly uploads, allocation	Path param	User detail payload
GET	/pmsreports/admin/user/<user_id>/portfolio-ids	Admin	Distinct portfolio IDs for selected user	Path param	portfolio_ids
GET	/pmsreports/admin/user/<user_id>/portfolios	Admin	Portfolio entries for selected user + portfolio_id query	Query: portfolio_id	Portfolio rows

8.7 Duplicate audit endpoints

Method	Path	Auth	Purpose	Input	Output
POST	/pmsreports/portfolio/duplicates/<dup_id>/accept	Session	Insert duplicate row into portfolios and link duplicate record	Path param	{status:'accepted'}
DELETE	/pmsreports/portfolio/duplicates/<dup_id>/remove	Session	Remove linked portfolio row and unlink duplicate record	Path param	{status:'removed'}
GET	/pmsreports/portfolio/<portfolio_id>/duplicates/detail	Public	Detailed duplicate rows	Path param	Duplicate detail list

GET	/pmsreports/portfolio/<portfolio_id>/duplicates/summary	Public	Grouped duplicate counts by ISIN/fund	Path param	Summary list
GET	/pmsreports/portfolio/<portfolio_id>/entries	Public	Final portfolio rows	Path param	Portfolio entries

---

9. End-to-End Functional Flows

9.1 User registration and admin approval

1. User submits signup (`/register`) -> row goes to `pending_registrations`.
2. Admin reviews pending list (`/admin/pending-registrations`).
3. Admin approves:
    1. Creates family.
    2. Inserts user with stored password hash.
    3. Assigns default role `user` in `user_roles`.
    4. Deletes pending row.
4. User can now authenticate.

9.2 Login with OTP

1. User submits email/password (`/login`).
2. Backend validates password hash.
3. Backend generates OTP, stores in `login_otps`, sends email.
4. User submits OTP (`/verify-otp`).
5. Backend validates OTP and expiration, deletes OTP, sets Flask session:
    1. `user_id`
    2. `user_email`
    3. `phone`
    4. `role_id`
    5. `role`

9.3 Self upload and parsing

1. Frontend upload form sends multipart to `/upload`.
2. Backend resolves user by provided email, creates new `portfolio_id` for that user.
3. Each file is validated by selected type and parsed with matching parser.
4. Parser output holdings:
    1. Unique rows inserted in `portfolios`.
    2. Cross-file duplicates inserted into `portfolio_duplicates`.
5. Response includes per-file and overall totals.

9.4 Member upload

1. User picks family member and uploads to `/upload-member`.
2. Backend maps per-family `member_id` -> global `family_members.id`.
3. Backend uses latest existing user `portfolio_id` (does not create a new one).
4. Parsed rows are inserted with `member_id` set.

9.5 Dashboard analytics

1. Frontend sends `/dashboard-data?include_user=...&members=...`.
2. Backend maps selected per-family IDs to canonical IDs.
3. Backend pulls latest portfolio per selected entity (user and/or member).
4. Backend computes:
    1. MF invested/current/profit/return%.
    2. Equity/NPS/Govt Security/Corporate Bond values.
    3. Asset allocation.
    4. Top AMC and top categories.
5. For MF ISINs, backend refreshes Morningstar returns if missing/stale (30+ days) and returns cached values.

9.6 History and snapshot

1. `/history-data` returns portfolio timeline.
2. Opening an item fetches `/portfolio/<id>/members`.
3. Snapshot modal shows:
    1. Member-specific and all-member allocation.
    2. Top AMC/categories.
    3. Holdings table with optional returns.
    4. PDF download via html2canvas + jsPDF.

9.7 Service request processing

1. User creates request (`/service-requests`).
2. Admin views queue (`/admin/service-requests`).
3. Admin can:
    1. Update status.
    2. Add notes.
    3. Perform request (including data changes) via `/perform`.
4. On successful perform, request is marked `completed`.

9.8 Duplicate audit and reconciliation

1. Deduped rows are stored in `portfolio_duplicates`.
2. Audit screen loads:
    1. Final entries.

2. Duplicate summary.
3. Duplicate details.

3. Admin/user can accept duplicate into final portfolio or remove previously accepted duplicate.

---

10. Frontend Route and Page Guide

Defined in `src/App.tsx`.

Frontend Path	Role Access	Page Component	Purpose
`/login`	Public	`Login`	Sign in, sign up, OTP verify
`/upload`	`user`	`Upload`	Upload statements
`/history`	`user`	`History`	Historical portfolio list + snapshot modal
`/portfolio/:portfolio_id`	`user`	`Dashboard`	User dashboard (legacy direct route)
`/portfolio-audit/:portfolio_id`	`user`, `admin`	`PortfolioAudit`	Duplicate audit and reconciliation
`/service-requests`	`user`	`ServiceRequests`	Create/list own requests
`/admin/service-requests`	`admin`	`AdminServiceRequests`	Manage all requests
`/dashboard`	`user`, `admin`	`RoleBasedDashboard`	User dashboard or admin dashboard
`/pmsreports/profile`	`user`, `admin`	`ProfilePage`	User profile and family management (admin redirected)
`/admin`	`admin`	`AdminProfilePage`	Admin request summary page

`/admin/edit-portfolio/:userId/:requestId`	admin	`AdminPortfolioEditor`	Portfolio-update request execution page
`/admin/user/:userId`	admin	`AdminUserDetail`	Per-user admin analytics/ detail
`/admin/pending-registrations`	admin	`AdminPendingRegistrations`	Approve/ reject pending signups

10.1 User pages

1. `Login.tsx`
    1. Includes signup live checks (`/check-email`, `/check-phone`).
    2. Validates password rules on signup.
    3. Handles OTP step (`/verify-otp`) and completes auth context.
2. `Upload.tsx` + `UploadForm.tsx`
    1. Multi-row file upload with per-file type and optional password.
    2. Optional member target (`/upload-member` vs `/upload`).
3. `Dashboard.tsx`
    1. Member filter dropdown (self + family members).
    2. Summary cards, charts, holdings table, MF category table.
    3. PDF export.
4. `History.tsx`
    1. Lists uploads from `/history-data`.
    2. Opens `PortfolioSnapshot` modal.
    3. Allows deleting portfolio.
5. `ServiceRequests.tsx`
    1. Create request with optional family member target.
    2. View previous requests and statuses.
6. `ProfilePage.tsx`
    1. Shows family member count and upload count.
    2. Add/delete family members.

10.2 Admin pages

1. `AdminDashboard.tsx`

    1. Pulls `/admin/stats`.
    2. Shows cards/charts/table and links to user detail.

2. `AdminPendingRegistrations.tsx`

    1. Pulls pending + approved lists.
    2. Approve/reject actions.

3. `AdminServiceRequests.tsx`

    1. Status actions, perform action, note editing.

4. `AdminPortfolioEditor.tsx`

    1. For `Portfolio Update` requests:

        1. Select user portfolio ID.
        2. Select entry.
        3. Edit fields.
        4. Save through `/perform`.

5. `AdminUserDetails.tsx`

    1. Pulls `/admin/user/<id>`.
    2. Shows profile/totals/charts/family/holdings.

6. `AdminProfilePage.tsx`

    1. Summary cards and list view of all service requests.

10.3 Shared components and notes

1. `Layout.tsx` wraps pages with desktop sidebar and mobile bottom nav.
2. `Sidebar.tsx` fetches latest portfolio for audit link.
3. `HoldingsTable.tsx` supports searching and sortable return columns.
4. `Toast.tsx` and `useToast.ts` exist but are not central in current page flows.
5. `ChartCard.tsx` and `HistoryList.tsx` exist as utility components but current pages use custom layouts.

---

11. Backend Module Guide

11.1 `backend/app.py`

Responsibilities:

1. Flask app/session/CORS setup.
2. Authentication and authorization decorators.
3. Registration, login, OTP verification.
4. File upload endpoints and parser integration.
5. Dashboard/history/snapshot/service/admin APIs.
6. Duplicate reconciliation APIs.

Key helper behavior:

1. `find_user(email)` reads from `users`.
2. `assign_default_role` inserts role into `user_roles`.
3. `resolve_per_family_member_to_canonical` maps member IDs from frontend to canonical DB ID.

11.2 Parser pipeline

1. `ecasparser.py`
    1. Validates PDF first page content against selected type.
    2. Dispatches to parser module.
2. `nsdl_parser.py`
    1. Extracts shares, mutual funds/folios, govt securities, NPS, corporate bonds.
    2. Classifies categories/subcategories.
    3. Deduplicates in-file and cross-file.
3. `cdsl_parser.py`
    1. Reads text blocks preserving order.
    2. Parses mutual funds and equities.
    3. Classifies with detailed category logic.
4. `cams_parser.py`
    1. Parses two-column block layout.
    2. Extracts folio/scheme/ISIN/units/NAV/invested/value.

11.3 Deduplication model (`dedupe_context.py`)

1. Global in-process context keyed by holdings signature.
2. Dedupe key:
    1. For ISIN instruments: `(isin, units, valuation)`.
    2. For non-ISIN (for example NPS): `(type, fund_name, units, valuation)`.
3. Duplicate only if same key appears from another source file in same request context.
4. Request handlers call `reset_dedup_context()` once per upload request.

11.4 Morningstar returns cache (`morningstar.py`)

1. Fetch endpoint: Morningstar trailing total return API (XML).
2. Normalizes ISIN and parses 1Y/3Y/5Y/10Y returns.
3. Upserts into `historic_returns`.
4. Dashboard refresh policy: stale if `updated_at` older than 30 days.

---

12. Operational Runbook

12.1 Approve new user

1. Login as admin.
2. Open `/admin/pending-registrations`.
3. Click approve.
4. Confirm new user appears in approved list.

12.2 Create a new admin account manually (SQL)

If you need a direct admin bootstrap without pending-approval flow, use this.

Step 0: Generate a scrypt password hash.

```
python -c "from werkzeug.security import generate_password_hash; print(
```

Example output format:

```
scrypt:32768:8:1$<salt>$<hash>
```

Step 1: Insert admin user and save returned `user_id`.

```
INSERT INTO users (email, phone, password_hash)
VALUES (
  'admin@yourapp.com',
  '9999999999',
  'scrypt:32768:8:1$MUH37DgsjeKx2jGq$5f1ecb9418fc029eb4019081c4e3b40821
)
RETURNING user_id;
```

Trigger behavior after Step 1:

1. `auto_create_family` trigger creates a family automatically.
2. The new `users.family_id` is attached automatically.
3. No manual family insert is needed.

Step 2: Assign admin role.

```
INSERT INTO user_roles (user_id, role_id, scope)
SELECT
  u.user_id,
  r.role_id,
  'global'
FROM users u
JOIN roles r ON r.role_name = 'admin'
WHERE u.email = 'admin@yourapp.com';
```

Step 3: Verify mapping.

```
SELECT
  u.user_id,
  u.email,
  r.role_name,
  ur.scope
FROM users u
```

```
JOIN user_roles ur ON ur.user_id = u.user_id
JOIN roles r ON r.role_id = ur.role_id
WHERE u.email = 'admin@yourapp.com';
```

Expected result:

```
admin@yourapp.com | admin | global
```

12.3 Promote an existing user to admin (SQL)

```
INSERT INTO user_roles (user_id, role_id, scope)
SELECT
  u.user_id,
  r.role_id,
  'global'
FROM users u
JOIN roles r ON r.role_name = 'admin'
WHERE u.email = '<existing-user-email>'
ON CONFLICT DO NOTHING;
```

Optional: remove old user role if needed.

12.4 Resolve service requests

1. Go to admin service requests page.
2. For profile updates, use `Perform` and provide new value.
3. For portfolio updates, open editor and save.
4. Add admin notes for auditability.

12.5 Database backup and restore

Backup:

```
pg_dump -Fc -d portfolio_db -f portfolio_backup.dump
```

Restore:

```
createdb portfolio_db
pg_restore --clean --if-exists --no-owner -d portfolio_db portfolio_bac
```

12.6 Clear stale OTPs/sessions

1. OTP cleanup is automatic by replacement on login, but old rows can be pruned manually.
2. Redis session keys use prefix `pms:`.
   If emergency logout-all is required, flush matching keys.

12.7 Upload storage maintenance

1. Uploads are stored under process working directory in `uploads/`.
2. Establish retention/archive policy for old files.

3. Ensure filesystem permissions allow backend write access.

---

13. Security and Reliability Gaps (Current Code)

These are critical for handoff awareness.

1. Secrets in source code:
    1. Flask secret key is hardcoded.
    2. SMTP email/app-password are hardcoded.
2. Open admin/data endpoints:
    1. `/pmsreports/admin/stats` has no auth decorator.
    2. `/pmsreports/admin/user/<user_id>` has no auth decorator.
3. Open portfolio data endpoints:
    1. `/pmsreports/portfolio/<id>/duplicates/detail`
    2. `/pmsreports/portfolio/<id>/duplicates/summary`
    3. `/pmsreports/portfolio/<id>/entries` These currently have no session/ownership checks.
4. `/pmsreports/upload` does not enforce logged-in session and accepts user email from form data.
5. Session cookie is configured with `SESSION_COOKIE_SECURE=False`.
6. CORS allowlist is localhost-only; deploy will fail until updated.
7. Backend runs `debug=True` in `app.run`.
8. Frontend sidebar logout calls hardcoded `http://127.0.0.1:5000/logout` before context logout, which does not match backend port/path (`8010 + /pmsreports/logout`).
9. Python dependencies are controlled by `requirements.txt`, but no hash-locked constraints file is maintained.
10. Upload folder path is relative; behavior changes with process working directory.
11. Dashboard holding `amc` field assignment currently uses a leaked variable from previous loop scope (data consistency risk for per-holding AMC value).

---

14. Recommended Immediate Hardening Tasks

1. Introduce `.env`-driven backend config and remove hardcoded secrets.
2. Add `@login_required + @admin_required` on all admin/data-sensitive routes.
3. Enforce ownership checks on duplicate/entries endpoints.
4. Require session identity on `/upload` and stop taking arbitrary email.
5. Fix frontend logout endpoint mismatch in `Sidebar.tsx`.
6. Keep backend `requirements.txt` updated and pinned as dependencies change.
7. Move uploads to absolute configured path.
8. Run backend under PM2 in production and disable Flask debug mode.
9. Add audit logging for admin mutation endpoints.
10. Add automated tests for:
    1. Auth/authorization.

2. Upload parser contracts.
3. Service-request perform workflows.
4. Duplicate accept/remove behavior.

---

15. Handoff Checklist

Use this checklist before final transfer:

1. Local startup verified on fresh machine.
2. DB restore tested from `portfolio_backup.dump`.
3. Admin login path and pending approval flow validated.
4. Sample NSDL/CDSL/CAMS uploads validated.
5. Dashboard + history + snapshot + audit paths validated.
6. Service request create and admin perform paths validated.
7. Production config values externalized.
8. Security hardening items in Section 14 planned or completed.
9. Monitoring/logging ownership assigned.
10. This manual reviewed by incoming maintainer.

---

16. Quick File-by-File Responsibility Index

Backend

1. `backend/app.py`: All HTTP API routes, role checks, session flow, dashboard calculations.
2. `backend/db.py`: PostgreSQL connection config.
3. `backend/ecasparser.py`: File type validation + parser dispatch.
4. `backend/nsdl_parser.py`: NSDL extraction + DB insert + dedupe handling.
5. `backend/cdsl_parser.py`: CDSL extraction + classification + insert + dedupe.
6. `backend/cams_parser.py`: CAMS two-column extraction + insert + dedupe.
7. `backend/dedupe_context.py`: In-memory duplicate context.
8. `backend/morningstar.py`: Returns API fetch + DB upsert.
9. `backend/otpverification.py`: Email OTP send.
10. `backend/checkhash.py`, `backend/hashgenerate.py`, `backend/passhash.py`: password/hash utilities.
11. `backend/camstest.py`: CAMS parser debug utility.

Frontend

1. `src/App.tsx`: Route tree and role access.
2. `src/context/AuthContext.tsx`: auth/session state.
3. `src/components/Layout.tsx`, `src/components/Sidebar.tsx`, `src/components/Navbar.tsx`: shell/navigation.
4. `src/components/UploadForm.tsx`: upload UI and multipart payload building.
5. `src/components/HoldingsTable.tsx`: holdings search/sort rendering.
6. `src/pages/Login.tsx`: login/signup/OTP flow.
7. `src/pages/Dashboard.tsx`: analytics and charts.
8. `src/pages/History.tsx`: portfolio timeline and snapshot modal open.
9. `src/pages/PortfolioSnapshot.tsx`: historical member breakdown modal and PDF export.

10. `src/pages/PortfolioAudit.tsx`: duplicates audit and accept/remove actions.
11. `src/pages/ServiceRequests.tsx`: user request creation/list.
12. `src/pages/ProfilePage.tsx`: user profile + family member management.
13. `src/pages/AdminDashboard.tsx`: admin aggregate analytics.
14. `src/pages/AdminPendingRegistrations.tsx`: registration approvals.
15. `src/pages/AdminServiceRequests.tsx`: admin request operations.
16. `src/pages/AdminPortfolioEditor.tsx`: request-driven portfolio row editing.
17. `src/pages/AdminUserDetails.tsx`: user deep-dive page.
18. `src/pages/AdminProfilePage.tsx`: admin request summary.