Project Structure - Listldr (SQM Platform — Python Backend (batch + services)

1. Overview

Listldr is the Python/FastAPI backend of the **Sales Quote Management (SQM) platform**. It serves two roles:

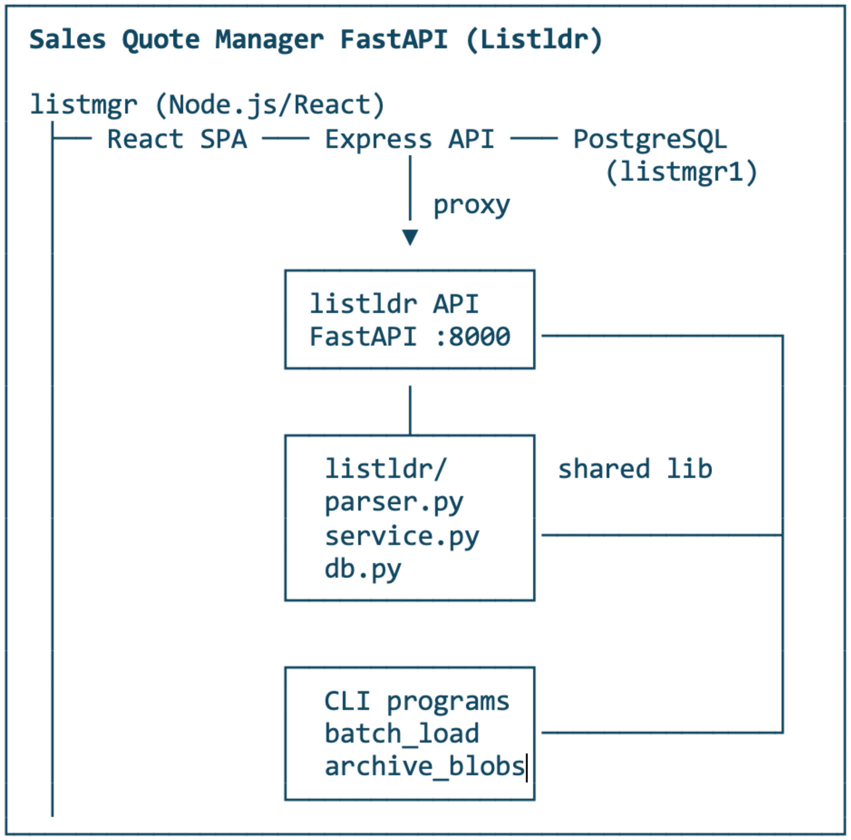
1. **Shared document processing library** — The `listldr/` package provides all .docx parsing, TOC-driven validation, section-type matching, and database loading logic. This is the single codebase for these capabilities, shared between the batch CLI programs and the REST API consumed by the **listmgr** frontend.

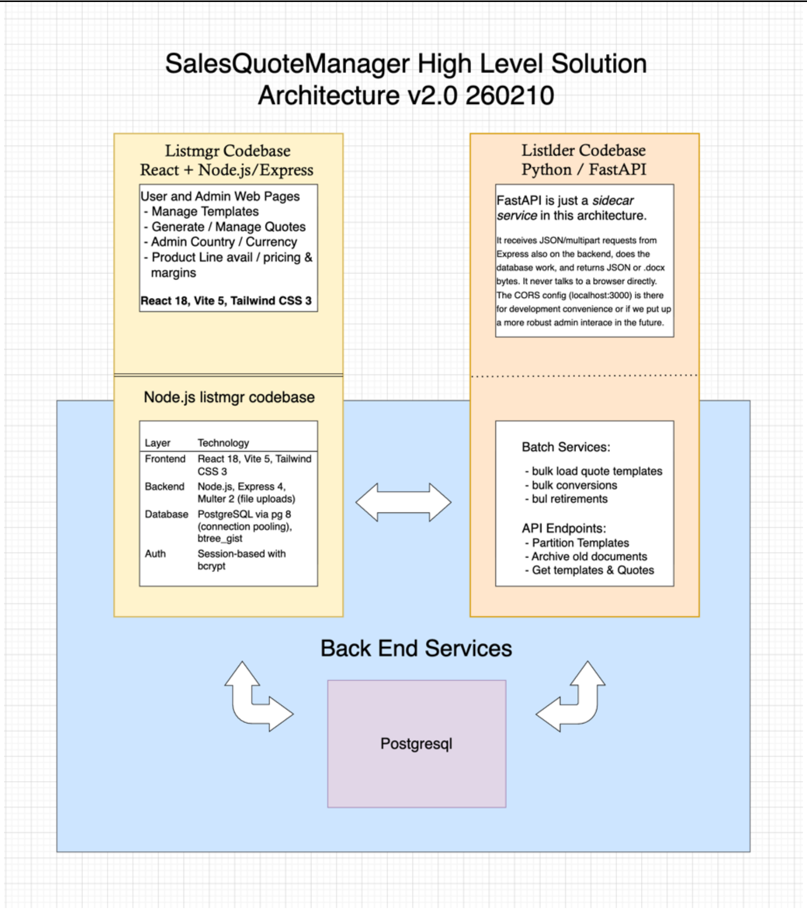
2. **Batch operations** — CLI programs for bulk-loading template folders and archiving/cleaning document blobs.

Both listldr and listmgr operate on the same PostgreSQL database (`listmgr1`). The listmgr Express backend proxies template-loading requests to listldr's FastAPI service, so there is only one code path for document parsing and ingestion.

2. System Architecture

(Note Combination of React / Express API and FastAPI for shared backend capabilities)


Sales Quote Manager FastAPI (Listldr)
listmgr (Node.js/React)
React SPA
Express API
PostgreSQL (listmgr1)
proxy
listldr API FastAPI :8000
listldr/ parser.py service.py db.py
shared lib
CLI programs batch_load archive_blobs


SalesQuoteManager High Level Solution Architecture v2.0 260210
Listmgr Codebase React + Node.js/Express
Listlder Codebase Python / FastAPI
Node.js listmgr codebase
Back End Services
Postgresql

User and Admin Web Pages
- Manage Templates
- Generate / Manage Quotes
- Admin Country / Currency
- Product Line avail / pricing & margins

React 18, Vite 5, Tailwind CSS 3

FastAPI is just a *sidecar service* in this architecture.

It receives JSON/multipart requests from Express also on the backend, does the database work, and returns JSON or .docx bytes. It never talks to a browser directly. The CORS config (localhost:3000) is there for development convenience or if we put up a more robust admin interace in the future.

Layer	Technology
Frontend	React 18, Vite 5, Tailwind CSS 3
Backend	Node.js, Express 4, Multer 2 (file uploads)
Database	PostgreSQL via pg 8 (connection pooling), btree_gist
Auth	Session-based with bcrypt

Batch Services:
- bulk load quote templates
- bulk conversions
- bul retirements

API Endpoints:
- Partition Templates
- Archive old documents
- Get templates & Quotes

3. Tech Stack

Layer	Technology	Version
Language	Python	3.12
Web framework	FastAPI	≥ 0.115
ASGI server	Uvicorn	≥ 0.32
Database	PostgreSQL	17.x
DB driver	psycopg2-binary	≥ 2.9
DB migrations	Alembic + SQLAlchemy	≥ 1.13 / ≥ 2.0
Document parsing	python-docx	≥ 1.1
Document assembly	docxcompose	≥ 1.4
Excel support	openpyxl	≥ 3.1
Form handling	python-multipart	≥ 0.0.9
Environment config	python-dotenv	≥ 1.0
Connection pooling	psycopg2 ThreadedConnectionPool	—

PostgreSQL extensions: `btree_gist` (date-range exclusion constraints), `pgcrypto` (SHA-256 blob deduplication).

---

4. Directory Layout

```
1_listldr/
├── listldr/                # Shared library package (all business logic)
│   ├── __init__.py
│   ├── parser.py           #   Docx parsing, TOC extraction, section splitting,
│   │                       #     clone-and-strip section extraction (362 lines)
│   ├── service.py          #   load_template() orchestration: parse → validate
│   │                       #     → match → store → deduplicate (156 lines)
│   ├── db.py               #   SQMDatabase class — all DB operations:
│   │                       #     lookups, blob CRUD, template CRUD (443 lines)
│   ├── models.py           #   Dataclasses: SectionInfo, TemplateLoadResult
│   ├── config.py           #   DBConfig dataclass, db_config_from_env/ini()
│   ├── logger.py           #   SQMLogger — dual-output (file + console),
│   │                       #     timestamped, context-manager support
│   └── text_utils.py       #   longest_common_substring() — LCS algorithm
│
├── api/                    # FastAPI REST API (thin wrapper over listldr/)
│   ├── __init__.py
│   ├── app.py              #   App factory, lifespan manager (connection pool,
│   │                       #     cached section types, CORS), title v1.0.0
│   ├── routes.py           #   POST /load, GET /sections/{seqn}/docx (171 lines)
│   ├── schemas.py          #   Pydantic response models (4 classes)
│   └── dependencies.py     #   DI: get_db (pooled), get_section_types (cached)
│
├── cli/                    # CLI batch programs
│   ├── __init__.py
│   ├── batch_load.py       #   Batch template loader v2.1 — reads INI config,
│   │                       #     processes folder of .docx files (288 lines)
│   └── archive_blobs.py    #   Blob archive/cleanup by cutoff date (163 lines)
│
├── conf/                   # Configuration
│   └── listldr_sqt.ini     #   Batch loader config (paths, country, DB creds)
│
├── docs/                   # Documentation, specs, design analysis
├── templates_docx/         # Source .docx template files (CHE + USA samples)
├── templates_xlsx/         # Companion Excel files
├── inputs/                 # Batch processing input folder
├── outputs/                # Processing output folder
├── reports/                # Generated reports
├── log/                    # Log files (gitignored)
├── sql_files/              # Ad-hoc SQL scripts
├── alembic/                # DB migration framework (no versions yet —
│   └── versions/           #   schema managed via listmgr's schema.sql)
│
├── SQM_load_quote_template_docx_file_v2.0.py   # Entry shim → cli.batch_load
├── poc_section_swap.py     # POC: extract/replace sections at XML level
├── poc_docxcompose.py      # POC: assemble documents from section files
├── requirements.txt        # 10 packages
├── alembic.ini
├── .env / .env.example       # API environment variables
└── .gitignore
```

5. Shared Library — `listldr/`

All document processing and database logic lives in this package. Both the FastAPI API and the CLI programs are thin callers.

1. parser.py — Document Parsing Engine

Function	Purpose
`parse_docx_sections(source)`	Parse .docx into `Section` list (sequence, heading, content). Detects headings in paragraphs and table cells (single-cell rows only, avoids false positives in price tables). Pattern: `^\d\s*[–\-]\s*.+$` (max 80 chars).
`extract_toc_entries(sections)`	Extract TOC from cover page (section 0). Pattern: `(\d)\s*[–\-]\s*(.+?)`. Returns `[(section_number, title)]`.
`validate_section_sequence(sections, product_line_abbr)`	Compare parsed sections against TOC expectations. Returns `(is_valid, error_message)`.
`extract_section_docx(source_bytes, target_seqn)`	**Clone-and-strip**: Opens full .docx (preserving headers/footers/styles/images), maps elements to sections, removes everything except the target section. Returns formatted .docx bytes.

2. service.py — Template Loading Orchestration

Single public function: `load_template(file_bytes, filename, db, country_id, currency_id, section_types, ...)`

Processing pipeline: 1. Resolve product line from filename (or override) 2. Parse sections from .docx bytes 3. Validate section sequence against TOC 4. Match each heading to a section type via LCS (min 4-char match) 5. Store/deduplicate blob by SHA-256 hash 6. Insert or update template record 7. Archive old blob if document changed 8. Delete old sections, insert new ones 9. Return `TemplateLoadResult` dataclass

Does NOT commit — the caller (API or CLI) controls the transaction boundary.

3. db.py — Database Operations

Class `SQMDatabase` accepts either a `DBConfig` or a pre-existing connection (for pool-based usage).

Category	Methods
Lookups	`lookup_country()`, `lookup_currency()`, `lookup_product_line()`, `fetch_all_section_types()`, `lookup_section_type_by_lcs()`
Blob ops	`get_or_create_blob()` (SHA-256 dedup), `archive_blob()`, `get_blob_bytes()`
Template ops	`get_template_by_id/name()`, `insert_template()`, `update_template()`, `get_section_info()`
Section ops	`insert_section()`, `delete_template_sections()`
Transaction	`connect()`, `close()`, `commit()`, `rollback()`, context manager

4. text_utils.py — LCS Algorithm

`longest_common_substring(s1, s2)` — Case-insensitive, O(mn) dynamic programming. Used to fuzzy-match parsed headings against the 17 known section type names.

6. API Endpoints

Base: `http://localhost:8000/api/v1/templates`

Method	Path	Description
POST	`/load`	Upload and parse a .docx template. Multipart form: `file` (required), `country`, `currency`, `product_line` (optional override), `dry_run` (bool). Returns `LoadSuccessResponse` with template details and section list.
GET	`/{plsqt_id}/sections/{seqn}/docx`	Extract a single section as a formatted .docx file (clone-and-strip). Returns binary attachment with `X-Section-Count` and `X-Content-Length` headers.

Startup: `uvicorn api.app:app --reload`

Lifespan manager creates a `ThreadedConnectionPool(1–10)`, pre-fetches and caches section types, and initializes a shared `SQMLogger`.

CORS: Configurable via `LISTLDR_CORS_ORIGINS` env var (comma-separated). Default: `http://localhost:3000`. Allows GET and POST.

Integration with listmgr: The Express backend in listmgr has a `loadTemplate.js` route that proxies requests to this FastAPI service (default `http://127.0.0.1:8000`), forwarding multipart uploads and section extraction requests.

---

7. CLI Programs

5. Batch Template Loader (v2.1)

```
python SQM_load_quote_template_docx_file_v2.0.py          # via shim
python -m cli.batch_load                                    # direct
python -m cli.batch_load --ini conf/listldr_sqt.ini --country CHE --currency CHF
```

Reads `conf/listldr_sqt.ini`, discovers `.docx` files in the configured input folder (skipping ~-prefixed and numeric-only filenames), sorts by name, and processes each file through the shared `load_template()` pipeline with per-file commit/rollback.

CLI arguments: `--ini`, `--path-root`, `--input-folder`, `--country`, `--currency`, `--skip`, `--process`, `--noupdate` (dry-run), `--no-continue` (halt on first error), `--silent`.

6. Blob Archive Utility (v1.0)

```
python cli/archive_blobs.py YYMMDD [--entity-type template|quote|both] [--dry-run]
```

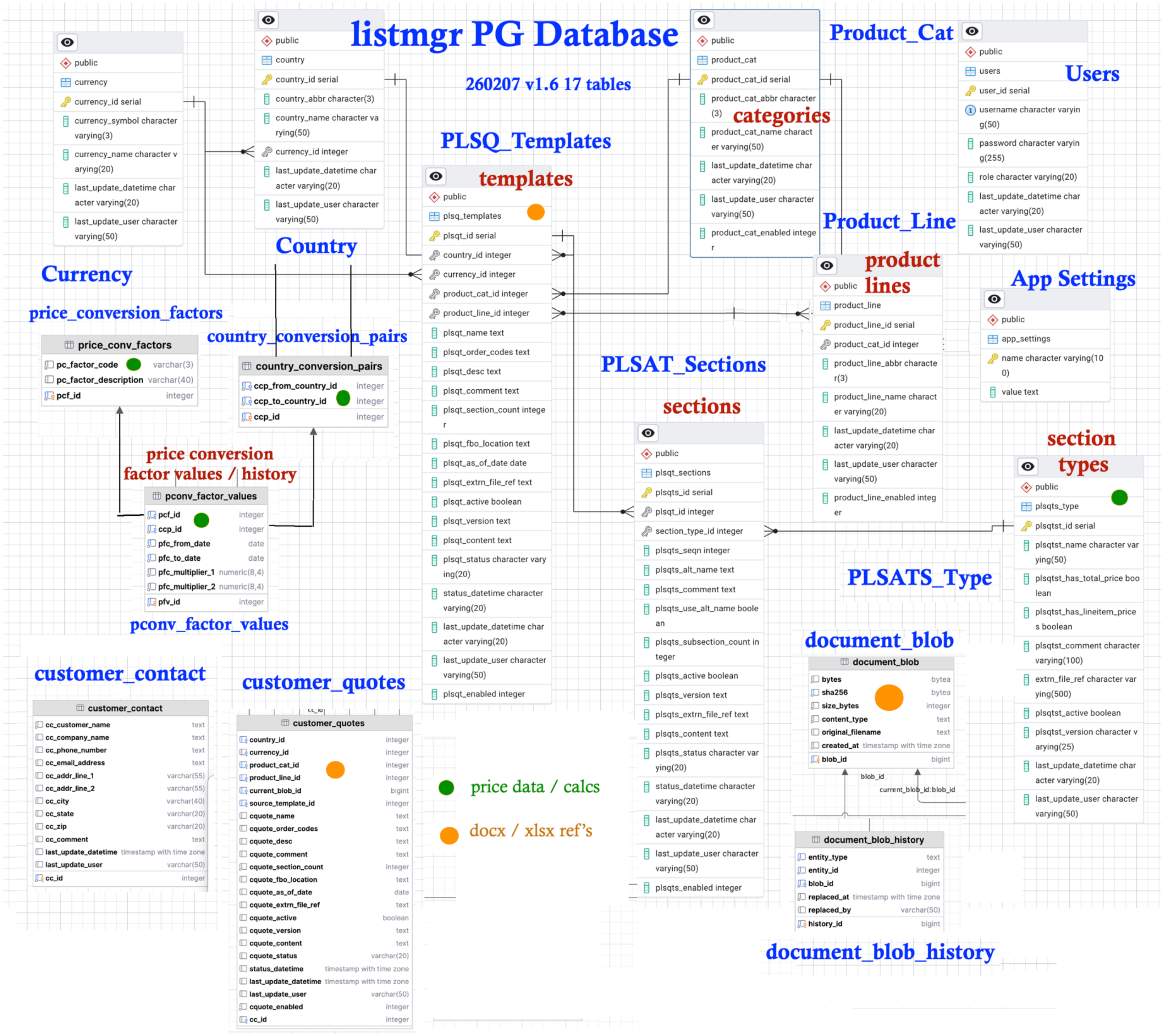
Deletes `document_blob_history` rows older than the cutoff date, then removes orphaned blobs not referenced by any template, quote, or remaining history record. Reports counts and bytes freed.

8. Database — `listmgr1`

Shared with listmgr. PostgreSQL 17.x, 15 tables. Schema managed via `listmgr/backend/db/schema.sql` and numbered migrations.

listmgr PG Database

260207 v1.6 17 tables


public
currency
currency_id serial
currency_symbol character varying(3)
currency_name character varying(20)
last_update_datetime character varying(20)
last_update_user character varying(50)
Currency
public
country
country_id serial
country_abbr character(3)
country_name character varying(50)
currency_id integer
last_update_datetime character varying(20)
last_update_user character varying(50)
Country
PLSQ_Templates
templates
public
plsq_templates
plsqt_id serial
country_id integer
currency_id integer
product_cat_id integer
product_line_id integer
plsqt_name text
plsqt_order_codes text
plsqt_desc text
plsqt_comment text
plsqt_section_count integer
plsqt_fbo_location text
plsqt_as_of_date date
plsqt_extrn_file_ref text
plsqt_active boolean
plsqt_version text
plsqt_content text
plsqt_status character varying(20)
status_datetime character varying(20)
last_update_datetime character varying(20)
last_update_user character varying(50)
plsqt_enabled integer
Product_Cat
categories
public
product_cat
product_cat_id serial
product_cat_abbr character(3)
product_cat_name character varying(50)
last_update_datetime character varying(20)
last_update_user character varying(50)
product_cat_enabled integer
Users
public
users
user_id serial
username character varying(50)
password character varying(255)
role character varying(20)
last_update_datetime character varying(20)
last_update_user character varying(50)
Product_Line
product lines
public
product_line
product_line_id serial
product_cat_id integer
product_line_abbr character(3)
product_line_name character varying(20)
last_update_datetime character varying(20)
last_update_user character varying(50)
product_line_enabled integer
App Settings
public
app_settings
name character varying(100)
value text
price_conversion_factors
price_conv_factors
pc_factor_code varchar(3)
pc_factor_description varchar(40)
pcf_id integer
country_conversion_pairs
country_conversion_pairs
ccp_from_country_id integer
ccp_to_country_id integer
ccp_id integer
price conversion factor values / history
pconv_factor_values
pcf_id integer
ccp_id integer
pfc_from_date date
pfc_to_date date
pfc_multiplier_1 numeric(8,4)
pfc_multiplier_2 numeric(8,4)
pfv_id integer
pconv_factor_values
PLSAT_Sections
sections
public
plsqt_sections
plsqts_id serial
plsqt_id integer
section_type_id integer
plsqts_seqn integer
plsqts_alt_name text
plsqts_comment text
plsqts_use_alt_name boolean
plsqts_subsection_count integer
plsqts_active boolean
plsqts_version text
plsqts_extrn_file_ref text
plsqts_content text
plsqts_status character varying(20)
status_datetime character varying(20)
last_update_datetime character varying(20)
last_update_user character varying(50)
plsqts_enabled integer
section types
public
plsqts_type
plsqtst_id serial
plsqtst_name character varying(50)
plsqtst_has_total_price boolean
plsqtst_has_lineitem_prices boolean
plsqtst_comment character varying(100)
extrn_file_ref character varying(500)
plsqtst_active boolean
plsqtst_version character varying(25)
last_update_datetime character varying(20)
last_update_user character varying(50)
PLSATS_Type
customer_contact
customer_contact
cc_customer_name text
cc_company_name text
cc_phone_number text
cc_email_address text
cc_addr_line_1 varchar(55)
cc_addr_line_2 varchar(55)
cc_city varchar(40)
cc_state varchar(20)
cc_zip varchar(20)
cc_comment text
last_update_datetime timestamp with time zone
last_update_user varchar(50)
cc_id integer
customer_quotes
cc_id
customer_quotes
country_id integer
currency_id integer
product_cat_id integer
product_line_id integer
current_blob_id bigint
source_template_id integer
cquote_name text
cquote_order_codes text
cquote_desc text
cquote_comment text
cquote_section_count integer
cquote_fbo_location text
cquote_as_of_date date
cquote_extrn_file_ref text
cquote_active boolean
cquote_version text
cquote_content text
cquote_status varchar(20)
status_datetime timestamp with time zone
last_update_datetime timestamp with time zone
last_update_user varchar(50)
cquote_enabled integer
cc_id integer
● price data / calcs
● docx / xlsx ref's
document_blob
document_blob
bytes bytea
sha256 bytea
size_bytes integer
content_type text
original_filename text
created_at timestamp with time zone
blob_id bigint
blob_id
current_blob_id:blob_id
document_blob_history
entity_type text
entity_id integer
blob_id bigint
replaced_at timestamp with time zone
replaced_by varchar(50)
history_id bigint
document_blob_history

7. Core Tables

Table	Purpose
`plsq_templates`	Template master: name, country, currency, product line, section count, status workflow (not started → in process → in review → approved → cloned), `current_blob_id` FK to document_blob
`plsqt_sections`	Template sections: sequence number, section type FK, content text, alt name, status. CASCADE DELETE from parent template.
`plsqts_type`	17 section type definitions (e.g. “Product Pump”, “Cover Page - CH/EU”) with pricing flags
`document_blob`	Binary .docx storage (BYTEA), deduplicated by SHA-256 hash (UNIQUE). Size constraint enforced.
`document_blob_history`	Blob version archive: entity type (template/quote), entity ID, prior blob reference, replaced_at, replaced_by
`customer_quotes`	Customer quote records (schema in place, not yet exposed in UI)

8. Reference Tables

Table	Purpose
`country`	Country codes (CHE, USA) with default currency FK, enable/disable flag
`currency`	Currency codes (CHF, USD, EUR) with enable/disable flag
`product_cat`	Product categories with enable/disable flag
`product_line`	Product lines (UBM, ECM, KD, etc.) within categories, with enable/disable flag

9. Price Conversion Tables

Table	Purpose
`country_conversion_pairs`	Directional migration paths (CHE→USA), UNIQUE on pair
`price_conv_factors`	Factor types: FX (currency exchange), MU (markup/duties)
`pconv_factor_values`	Factor values by date range and conversion pair. Two multiplier fields. `btree_gist` EXCLUDE constraint prevents overlapping date ranges.

10. Other Tables

Table	Purpose
`customer_contact`	Customer address book (name, company, phone, email, address)
`users`	Application users with role (admin/user) and enable/disable
`app_settings`	Key-value settings (theme color, app version, client name)

9. Key Design Patterns

- **Shared library (`listldr/`)**: Single codebase for all document processing — both the FastAPI API (serving listmgr) and the batch CLI call into the same functions
- **Connection injection**: `SQMDatabase` accepts an optional conn parameter — pooled connection from API, standalone from CLI
- **Caller-managed transactions**: `load_template()` does not commit; the API dependency or CLI loop controls commit/rollback
- **LCS section-type matching**: Fuzzy matching (min 4-char common substring) handles section title variations across product lines without hardcoded rules
- **TOC-driven validation**: Section sequence validated against the cover page's table of contents rather than per-product-line expectations
- **Clone-and-strip extraction**: Preserves full .docx formatting (headers, footers, styles, images) while isolating a single section
- **SHA-256 blob deduplication**: Identical documents share one blob row; `document_blob_history` tracks version lineage

10. Configuration

11. Batch CLI — `conf/listldr_sqt.ini`

```
[paths]         # PATH_ROOT, TEMPLATE_INPUT_FOLDER, LOGFILE_DIR_PATH
[template]      # TEMPLATE_COUNTRY_IN, TEMPLATE_CURRENCY_IN, LOG_FILENAME_SLUG
[processing]    # NUM_TO_SKIP, NUM_TO_PROCESS, NOUPDATE, CONTINUE_ON_ERRORS, SILENT
[database]      # host, port, user, password, database
```

12. API — `.env`

```
LISTLDR_DB_HOST, LISTLDR_DB_PORT, LISTLDR_DB_USER,
LISTLDR_DB_PASSWORD, LISTLDR_DB_NAME, LISTLDR_CORS_ORIGINS
```

11. Current State and Roadmap

B. Implemented

- CHE template import pipeline (batch CLI + API)
- Section extraction and database storage
- Clone-and-strip section-as-docx extraction endpoint
- Blob versioning and archive
- Price conversion factor infrastructure (DB tables populated, no application code yet)
- POC code for section swap and document assembly
- Integration with listmgr via proxied API calls

C. In Progress

- CHE→USA template migration analysis (6 sample templates compared across all sections)
- Identifying generic vs product-line-specific transformation rules

D. Planned

- Template migration function (text/format conversion, then price conversion)
- Section-level text replacement rules engine
- Document assembly from transformed sections

---

12. Development Notes

This project was developed manually (not AI-generated - but with AI support). The companion project **listmgr** — which provides the primary user-facing UI/UX — was entirely generated by Claude Code and autoForge (formerly Autocoder).