EXPLANATION AND JUSTIFICATION

1) Data exploration : data_exploration.py

Purpose: one-off EDA. Produces dq_report/ (row counts, null rates, basic distributions) to understand the raw data before cleaning.
Notes: safe, read-only; helps tune thresholds and confirm columns present.

- **Found**: high missingness (terrace_area, has_garden), duplicates (listing_id), invalid years, outliers in price/area, and inconsistent boolean/category encodings.
- **Challenge**: Handling **true missing values**

Outcome: generated reports to guide cleaning rules

2) Data cleaning : cleaner.py

Purpose: turn raw CSV → **cleaned** CSV.
Highlights:

- **Standardized columns**: stripped spaces/namespaces (ITEM_TYPE.HOUSE.SINGLE_FAMILY_HOUSE → SINGLE_FAMILY_HOUSE).
- **Fail-fast decorators** (timing + logging + enforce non-None returns).
- **description_fr preserved** (never dropped/imputed).
- **Int columns as nullable Int64**: floor, room_count, balcony_count, terrace_count, build_year.
- **Deduplication**: kept latest row per listing_id using change_date/start_date.
- **Datatype normalization**:
    - Converted *zipcode* back to **5-digit strings** (fix for lost leading zeros).
    - Parsed numeric-like Converted booleans encoded as "yes/no", "0/1", "True/False".
- **Sanity checks**: replaced negative/invalid with null, capped build years outside [1800–2100].
- **Challenge**: deciding between **imputation** vs. **dropping** — e.g., terrace_area too sparse, so preserved but often null.
- **Artifacts produced**:
    - listings_cleaned.csv
    - cleaning_changelog.txt (step logs & counts of dropped/imputed values)

- Writes:
  - clean_artifacts_py*/listings_cleaned.csv
  - clean_artifacts_py*/cleaning_changelog.txt

✓ **Corrected Main Challenges in this dataset**

- **Many duplicate listing IDs**
- **Massive missingness** in terrace/garden/build_year fields
- **Zipcodes corrupted** (leading zeros lost, int instead of text)
- **Invalid years** (1, 19601970)
- **Outliers**
- **Invalid prices** (0 values)
- **Mixed encodings** in booleans and categoricals

3) **DB design (design.sql)**

Purpose: normalized 3NF + SCD2.
Highlights:

- Dimensions: transaction_type, item_type, item_subtype, city, zipcode, location(city_id, zipcode_id).
- Facts: listing (stable attributes) and listing_version (SCD2 history).
- **listing_current** = **materialized view** of "current" versions (fast queries).
- **listing_with_type** = **regular/convenience view** exposing derived item_type_id via item_subtype (always fresh).
- Constraints check build year, non-negatives, and formats.
- Indexes on common filters/sorts (location, type, price, date).

We're modeling **real estate listings**. Each record changes over time (price updates, renovations, description changes, etc.), so the design must support both **clean structure** and **historical tracking**.

Why 3NF

Third Normal Form = "each fact in only one place, avoid redundancy."

- **Dimensions split**:
    - transaction_type (SELL, RENT)
    - item_type (HOUSE, APARTMENT)
    - item_subtype (DUPLEX, LOFT, …)
    - city, zipcode, location

→ Instead of repeating "SELL" or "APARTMENT" in every row of listing, we reference small lookup tables.

- **Benefits**:
    - ✓ **Consistency**: only one place defines what "SELL" means.
    - ✓ **Smaller storage**: no repeated strings in the fact table.
    - ✓ **Flexibility**: if "DUPLEX" should map under "HOUSE", just update the dimension, not millions of rows.
    - ✓ **Referential integrity**: ensures listing.location_id always maps to a valid city + zipcode.

- **Trade-off**: requires more joins. But since real estate datasets are rarely “big data scale” (millions, not billions), joins are fine.
- **Challenge**:
    - Balancing normalization vs. query speed. We solved it by creating listing_current (denormalized snapshot).
    - listing_id from CSV is not guaranteed unique, so it became external_listing_id (TEXT) and mapped to internal surrogate id.

Why SCD-2 (Slowly Changing Dimension Type 2)

Listings change **over time**. Example:

Date	Price
Jan 2025	300,000
Mar 2025	285,000
Jun 2025	270,000

If we overwrite, we lose history.
SCD-2 pattern keeps **all versions** in listing_version, with valid_from/valid_to.

- **Schema split**:
  - listing = **stable identity** (external id, location, type, etc.).
  - listing_version = **time-varying attributes** (price, area, floor, counts, terrace, etc.).
- **Benefits**:
  - ✓ **Historical analysis**: see price evolution, detect trends, measure time-on-market.
  - ✓ **Auditability**: regulators or business may ask "what did we publish on March 1st?"
  - ✓ **Current snapshot**: a listing_current materialized view gives "latest only" for APIs.
- **Trade-off**:
  - Slightly more complex loader (need to check deltas, close/open versions).
  - Queries must sometimes filter valid_to IS NULL for current data.

Why 3NF + SCD-2 Together

They complement each other:

- **3NF** ensures clean, normalized structure.
- **SCD-2** ensures temporal history.
- 3NF avoids redundancy: cities/zipcodes/dimensions normalized.
- SCD-2 allows **history tracking** (e.g., price changes over time).
- listing_current materialized view gives latest snapshot for API queries.

Without 3NF: you'd repeat "SELL", "APARTMENT", "Paris" in each row and blow up storage / risk inconsistencies.
Without SCD-2: you'd only see the latest state, no history.

Alternatives & Why We Didn't Choose Them

- **Flat denormalized table**: simpler, but huge redundancy + hard to change dimension definitions.
- **SCD-1 (overwrite)**: simpler loader, but history lost. Not acceptable in real estate.
- **SCD-3 (extra columns)**: only tracks *one previous value*. Not enough for long-term analysis.

Star schema / Data Vault: good for analytics warehouse, but our pipeline is closer to **OLTP + API exposure** where 3NF is natural.

4) **Loader (loader.py)**

Purpose: bulk load the **cleaned CSV** into Postgres, **upsert** dimensions + listings, then do **SCD2**: close prior current rows and insert new versions (trigger-free).
Highlights:

- **Ingests cleaned CSV** into staging (stg_clean_csv).
- **Upserts dimensions**: transaction_type, item_type, item_subtype, city, zipcode, location.
- **Staging (TEXT-only)** → typed temp → delta detection.
- Handles **upserts**, **SCD2**, and **listing_current refresh (CONCURRENTLY)**.
- Ensures zipcode is 5 digits, description_fr goes into listing_description.body, and terrace_area is numeric.
- Verbose logs for COPY counts, deltas, affected rows.
- **Creates listings** with external key = external_listing_id.
- **SCD-2 logic**: tracks changes in **price, area, floor, room counts, terrace..**. If values differ from latest version → close old row (valid_to) and insert a new one.
- **Challenge**:
    - Many duplicate listing_ids → loader needed to reconcile them correctly.
    - Extreme values caused type/cast errors → had to sanitize in cleaner first.
    - Refreshing materialized view concurrently sometimes fails (fallback to non-concurrent refresh).

5) **API (FastAPI)**

Purpose: expose the processed data: **filter/sort/paginate** current listings.
Structure:

- api/main.py (app),
- api/deps.py (DB connection),
- api/sql.py (Core metadata/selects), api/repository/listings.py (query builder), api/routers/listings.py (HTTP layer), api/models.py (Pydantic response models).
- Queries use listing_current + joins to dims; supports filters like price range, area, city, type, etc.

6) **Data Quality (dq/)**
Purpose: detect anomalies & enforce guarantees.

- dq_constraints.sql – **CHECK constraints** (e.g., zipcode format, non-negative terrace_area, price/area > 0, build_year bounds).
- dq_profile.sql – **daily metrics** into dq_daily_metrics (medians/p95s, coverage, counts).
- dq_check.py – lightweight **Python checker** that reads listing_current + dq_daily_metrics and exits non-zero on issues (good for CI/cron).

<u>How they connect (flow)</u>

1. **exploration → cleaner**
   Tune the cleaner with insights from data_exploration.py.
2. **cleaner → loader**
   Cleaner writes listings_cleaned.csv. Loader reads it, stages, upserts dims/listings, runs SCD2, refreshes listing_current.
3. **loader → API**
   API reads from listing_current (fast) with joins to dims for human-readable filters.
4. **DQ**
   - Constraints prevent bad data from entering.
   - Daily profiling captures trends.
   - dq_check.py alerts on anomalies (empty set, zip coverage < 95%, terrace all null, description empty, large day-over-day jumps).