ChatGPT - Research

📑 Proposal: Documentation Guidelines for Code Base

1. Purpose
   To establish a clear and consistent documentation standard across the code base. Proper documentation ensures that all contributors can quickly understand, maintain, and extend the system without relying on tribal knowledge.
2. Scope
   This guideline applies to all source code files (.py, .sql, .js, .ts, etc.) and all modules/packages in the repository.
3. File-Level Documentation (Comment Block)

3.1 Requirement

Every file must begin with a comment block describing:

- File name & purpose (what the file does, why it exists)
- Author/owner (optional if tracked in version control)
- Dependencies (other modules, datasets, APIs it relies on)
- Key functions/classes defined in the file
- Usage examples (if applicable)
- Change log (optional)

3.2 Format

- Use the native comment style for the file type (# for Python/SQL, // for JS/TS, /* */ for block comments if needed).
- Keep concise but informative (5–15 lines typical).

3.3 Example (Python)

python

```python
"""
File: subscription_loader.py
Purpose: Handles extraction of raw subscription data from CRM API
         and loads into the raw_layer dataset in BigQuery.

Dependencies:
- requests (HTTP calls)
- google.cloud.bigquery
- utils/logger.py

Key Functions:
- fetch_subscriptions()
- transform_payload()
- load_to_bq()

Usage:
    python subscription_loader.py --date 2025-09-22
"""
```

3.4 Example (SQL)

```sql
-- File: customer_summary.sql
-- Purpose: Aggregates customer revenue and service counts
--          for the profitability dashboard.
--
-- Dependencies: raw_layer.subscriptions, raw_layer.customers
-- Output: transformation_layer.customer_summary
--
-- Change Log:
-- 2025-09-15 (KO): Initial version
-- 2025-09-20 (NT): Added margin calculation
```

4. Module-Level Documentation (.md file)

4.1 Requirement

Each module (logical directory/package) must include a README.md file describing:

- Module purpose (why it exists, business context)
- Directory structure (list of files/submodules)
- Data flow / architecture diagram (if applicable)
- Key dependencies
- Setup/installation instructions (if needed)
- Examples of usage
- Known limitations / TODOs

4.2 Example Structure

```markdown
```

~~~
# Subscription Module

## Purpose
This module manages subscription lifecycle data from multiple CRM sources,
ensuring consistency in the normalization and transformation layers.

## Structure
- subscription_loader.py — Extracts & loads raw subscription data
- subscription_cleaner.sql — Cleans raw subscription records
- subscription_mapper.py — Applies normalization rules

## Data Flow
CRM API → raw_layer.subscriptions → transformation_layer.subscriptions_clean

## Dependencies
- Python: requests, google.cloud.bigquery
- SQL: BigQuery standard SQL

## Usage
Run loader:
```bash
python subscription_loader.py --date YYYY-MM-DD
~~~

Limitations

Currently only supports CRM type A; CRM type B integration in progress.

````
yaml



---


## 5. Style Conventions

- **Consistency > perfection**: follow the format even if minimal.
- **Plain language**: avoid jargon unless defined.
- **Versioning**: use version control for history; keep only high-level change notes in docs.
- **Diagrams encouraged**: for complex modules, add a simple flowchart or architecture image.

---
````

```markdown
## 6. Enforcement

- Add documentation checks to **PR reviews**.
- Optionally enforce file headers via a **pre-commit hook or linter**.
- Missing/poor documentation = request changes.

---

✅ With this, you’ll have a **two-layer doc system**:
- **File-level (quick understanding)** → comment blocks.
- **Module-level (big picture)** → README.md files.
```