Here is a concise Software Design Description (SDD) in plain text for the RFQ Quotation Agent implemented in VS Code using Python and OpenAI API.

Title: RFQ Quotation Agent – Software Design Description (SDD)

1. Overview

* Purpose: Automate processing of incoming plain-text “email” files, detect RFQ intent, extract requested services and quantities using an LLM, match against a local catalog, compute totals, and generate a quote file per RFQ.
* Scope: Single-user, local filesystem app run from VS Code terminal; reads data/emails/*.txt, catalog JSON, writes output/*.txt; uses environment-stored API key for LLM calls.
* Stakeholders: Individual developer/operator; potential hand-off to a small sales/support team.

1. System Context

* Inputs: Plain-text emails in data/emails; Product/Service catalog in data/catalog/catalog.json; Environment variable for API key.
* Outputs: Text quotations saved to output; console logs for each processed email.
* External Services: OpenAI chat completion API for information extraction (structured output).

1. Architecture

* Pattern: Modular script with functional pipeline (ingest → classify → extract → match → price → render → persist).
* Key Modules:
  + I/O Layer: Directory loading (emails), JSON catalog loading, quote file writing.
  + NLP Layer: RFQ rule-based classifier; LLM-based structured extraction with Pydantic schema.
  + Matching & Pricing: SKU/name matching, totals computation (subtotal, tax, total).
  + Orchestration: Iterates emails and executes the pipeline with robust logging and fallbacks.

A diagram of a system

AI-generated content may be incorrect.

1. Directory & Files

* Root
  + .venv/ (virtual environment; do not place project code here)
  + data/
    - emails/ (plain .txt email files: rfq1.txt, rfq2.txt, rfq3.txt, nonrfq1.txt, rfq4\_unavailable.txt)
    - catalog/
      * catalog.json (service name, sku, unit\_price)
  + output/ (generated quotes)
  + main.py (application entry)
  + requirements.txt (langchain, langchain-openai, openai, python-dotenv, langchain-community)
  + .env (OPENAI\_API\_KEY=sk-\*\*\*)

A screenshot of a computer

AI-generated content may be incorrect.

1. Data Models

* Catalog Item (JSON):
  + name: string
  + sku: string
  + unit\_price: number
* Extraction Schema (Pydantic):
  + LineItem: product (string), quantity (int)
  + EmailExtraction: items (List[LineItem]), notes (Optional[str])

1. Control Flow

* Startup:
  + Load environment variables.
  + Ensure output directory exists.
  + Load catalog JSON.
  + Initialize LLM client with deterministic params (temperature 0).
* For each email file:
  + Read text (UTF-8 forced).
  + Classify RFQ via regex keywords; skip non-RFQs.
  + Structured extraction via LLM into EmailExtraction (items list).
  + Match items to catalog:
    - Exact SKU match; else case-insensitive name contains/equality.
    - Accumulate matched lines and record missing items.
  + Compute totals: subtotal, tax (8%), total.
  + Render quote text; write to output/<email\_id>.txt.
  + Print console summary per email.

1. LLM Integration (OpenAI)

* Model: Chat model suitable for structured output (e.g., gpt-4o-mini); temperature=0 for consistent parsing.
* Prompting: ChatPromptTemplate with system role (“Extract requested products and integer quantities…”) and human slot for email text.
* Parsing: with\_structured\_output(EmailExtraction) to enforce schema and reduce post-processing.
* Error Handling:
  + Rate limits/insufficient quota: surface error and skip that email.
  + Extraction returns empty items: log and skip quote generation.

1. Error Handling & Resilience

* Virtual environment issues: guide to select interpreter and install requirements.
* Encoding issues: Directory/Text loader forces encoding="utf-8" to avoid cp1252 “charmap” errors on Windows.
* Missing packages: instructions to pip install required libraries in venv.
* Authentication: .env with OPENAI\_API\_KEY; if missing/invalid, extraction raises and is reported.
* Unmatched catalog items: listed under “Unavailable/Unmatched items” in the quote.

1. Security

* Secrets: API key stored in .env; .gitignore excludes .env.
* Key Handling: Loaded at runtime; never logged; follow safe API key handling practices.
* Filesystem: Operates within project directories; no arbitrary file execution.

1. Performance & Limits

* Batch size: Processes files sequentially; sufficient for small folders (dozens to a few hundred).
* Token usage: Minimal prompts; deterministic extraction; quick, low-cost runs.
* I/O: Lightweight filesystem operations.

1. Testing Strategy

* Unit-like manual tests:
  + Non-RFQ email is skipped.
  + RFQ with valid SKUs generates a quote with correct totals.
  + RFQ with unknown item shows it under “Unavailable/Unmatched items.”
  + Encoding edge case: special characters in emails load successfully.
* Integration checks:
  + Run end-to-end with 4–5 sample emails and one invalid/unavailable item file.
  + Force temporary API failure to validate error logging and continuation.

1. Deployment & Runbook

* Setup:
  + Create and select venv in VS Code.
  + pip install -r requirements.txt
  + Add OPENAI\_API\_KEY to .env
  + Ensure directory structure and sample files exist.
* Run:
  + python main.py
* Outputs:
  + Console shows per-email outcome.
  + Quotes in output/\*.txt.

1. Change Log (Key Activities Completed)

* Created project folder and virtual environment; fixed shell command separators for PowerShell.
* Corrected folder placement (kept data/ and output/ outside .venv).
* Auth: Added .env with OPENAI\_API\_KEY and loaded via python-dotenv.
* Implemented catalog.json (IT services with SKU and unit\_price).
* Authored five sample emails (RFQ and non-RFQ) aligned to the catalog.
* Resolved import and environment path issues (pip installing into correct venv).
* Fixed Windows encoding error by forcing UTF-8 in DirectoryLoader/TextLoader.
* Implemented LLM extraction with structured output and Pydantic schema.
* Implemented matching, pricing, and quote rendering; wrote files to output/.
* Addressed API quota/401 errors by ensuring valid OpenAI API key in environment.

1. Future Enhancements

* Replace regex RFQ detection with LLM classification for higher recall.
* Add currency/tax configuration and multi-currency display.
* Support richer input formats (EML/MSG) via specialized loaders.
* Persist runs and items to a lightweight DB and add CLI flags.
* Add unit tests and tracing/telemetry.

A diagram of a company

AI-generated content may be incorrect.