Description:

INPUT: one HLD text or file (plain text, PDF, or doc).
TASK: Extract exactly the following sections. Return them as lists. Do not output anything else.

Sections to extract (exact headings must be used):

1. Functional Requirements (FR)
2. Non-Functional Requirements (NFR)
3. System Interfaces
4. Constraints
5. Data Requirements
6. Assumptions
7. Open Questions

RULES:
- DO NOT generate UML, PlantUML, class diagrams, sequence diagrams, API specs, database schemas, entities, flows, or any design artifacts.
- DO NOT invent or infer information that is not explicitly present.
- If a requested section has no content in the input, return an empty list for that section.
- Output only the structured sections, in the exact order above.
- Output format MUST be plain text following the “Expected Output” template below — no commentary, no explanations, no additional headings.
- If input is ambiguous, list specific clarifying questions under "Open Questions".

You have access to the the tool 'FileReaderTool'. It is advised to use this tool when you have access to it irrespective of whether it has been asked to be used explicitly.

Expected Output:

Functional Requirements (FR):
- FR1: <text>
- FR2: <text>
...

Non-Functional Requirements (NFR):
- NFR1: <text>
- NFR2: <text>
...

System Interfaces:
- Interface1: <description of interface, protocol, directionality, if present>
- Interface2: <...>

Constraints:
- Constraint1: <text>

- Constraint2: <text>

Data Requirements:
- DataElement1: <description, format, units if present>
- DataElement2: <...>

Assumptions:
- A1: <text>
- A2: <text>

Open Questions:
- Q1: <question to clarify>
- Q2: <question to clarify>

Summary:

INPUT: one HLD text or file (plain text, PDF, or...

Raw Output:

Functional Requirements (FR):
- FR1: Simulate SIP call establishment and teardown using a PJSIP stub.
- FR2: Simulate T.30 fax session management and page encoding using a SpanDSP stub.
- FR3: Transmit T.38-encapsulated fax data via simulated UDPTL.
- FR4: Orchestrate the end-to-end FOIP workflow in the Main Application, coordinating between SIP and fax layers.
- FR5: Accept target phone number (string) and sample page data as inputs.
- FR6: Output boolean status for call, transmission, and hangup, as well as logs to stdout.

Non-Functional Requirements (NFR):
- NFR1: No explicit throughput or latency targets; demo expected to complete workflow within a few seconds on typical host.
- NFR2: Basic error handling via boolean return values.
- NFR3: Automatic resource cleanup using RAII principles.
- NFR4: No safety-critical features; not intended for deployment in regulated environments.
- NFR5: No authentication or encryption in stub implementation.
- NFR6: Not optimized for low power; suitable for development and demonstration only.
- NFR7: Modular design; clear separation between SIP, fax, and application logic.
- NFR8: Easy replacement of stubs with real library implementations.
- NFR9: Unit testing not explicitly implemented; modules are simple and self-contained.
- NFR10: Integration testing via end-to-end workflow in main application.
- NFR11: Protocol compliance is simulated; not validated against real protocol stacks.
- NFR12: Regression testing via manual verification of console output.

System Interfaces:
- Interface1: Main Application to PJSIP Stub (simulated SIP signaling and UDPTL transport; direction: Main Application !' P J S I P S t u b) .
- Interface2: Main Application to SpanDSP Stub (simulated T.30 fax session management and page encoding;

direction: Main Application !’ SpanDSP Stub).
- Interface3: SpanDSP Stub to PJSIP Stub (encoded fax data passed for UDPTL transmission; direction: SpanDSP Stub !’ PJSIP Stub).
- Interface4: No direct hardware interfaces; all protocol interactions are stubbed and operate in software.
- Interface5: No physical communication buses (e.g., UART, SPI, I2C, CAN, Ethernet); all communication is simulated in software.

Constraints:
- Constraint1: Must run on any platform supporting C++17 and basic POSIX APIs.
- Constraint2: RAM usage expected "d 256 KB for demo operation.
- Constraint3: No explicit heap/stack constraints.
- Constraint4: No real-time or latency guarantees.
- Constraint5: Not intended for performance benchmarking.
- Constraint6: No explicit power or environmental constraints; suitable for desktop or embedded development boards.
- Constraint7: No hardware abstraction layer (HAL) required.
- Constraint8: IPv4 assumed for any network simulation.

Data Requirements:
- DataElement1: Target phone number (string).
- DataElement2: Sample page data (raw page data, byte vector).
- DataElement3: Encoded fax data (byte array for transmission).
- DataElement4: Encoded T.30 frames (byte vector).
- DataElement5: Boolean status for call, transmission, hangup, and session.

Assumptions:
- A1: Only sender role is implemented (no fax reception).
- A2: All protocol interactions are simulated; no real network or telephony stack.
- A3: No hardware abstraction layer (HAL) required.
- A4: IPv4 is assumed for any network simulation.

Open Questions:
- Q1: What are the requirements and use cases for receiving faxes and handling multi-page documents?
- Q2: What are the specific requirements for SIP authentication and encrypted transport (TLS/SRTP) if moving to production?
- Q3: Should automated unit and protocol compliance tests be implemented, and if so, what are the acceptance criteria?
- Q4: Is there a need to profile and document actual RAM usage on intended embedded targets for production use?
- Q5: Who are the author, date, and reviewers for the metadata section?

Description:

Given structured requirements, generate a complete Low-Level Design document containing all required engineering components.

You must include:

1. Architecture Standards
   - List all standards required for the system using consistent structure.

2. Entities
   - List all entities/classes/components inferred strictly from requirements.
   - Provide a clear one-line purpose for each entity.

3. Flows
   - Describe Step-by-step flows for each requirement-driven process.
   - 100% aligned to the requirements.

4. API Specification (OpenAPI 3.0)
   - Produce YAML format.
   - Include request body, response structure, status codes, and schemas.

5. Sequence Diagram (PlantUML)
   - Proper actor !’ component interactions.
   - Steps must match flows.

6. Class Diagram (PlantUML)
   - All classes, public methods, relations, and dependencies.

7. Database Schema (PostgreSQL)
   - Proper table design with datatypes and constraints.
   - Should reflect entities and API requirements.

8. Error Handling Strategy
   - Error codes, exception paths, validity checks.

9. Configurations & Deployment Notes
   - Thresholds, intervals, system constants.

Rules:
- DO NOT guess requirements.
- DO NOT generate requirements.
- DO NOT omit any mandatory LLD part.
- Keep the final output engineering-grade, deterministic, and standards compliant.

You have access to the KnowledgeRAGTool. It is advised to use this tool when you have access to it irrespective of whether a knowledge retrieval task has been assigned explicitly.

Expected Output:

The output must strictly follow this structure:

```
1. Architecture Standards:
   - …

2. Entities:
   - EntityName: description
   - …

3. Flows:
   1. …
   2. …

4. API Specification (OpenAPI 3.0 YAML):
   openapi: 3.0.0
   info:
     title: …
     version: …
   paths:
     …

5. Sequence Diagram (PlantUML):
   @startuml
   …
   @enduml

6. Class Diagram (PlantUML):
   @startuml
   …
   @enduml

7. Database Schema (PostgreSQL):

CREATE TABLE …
8. Error Handling Strategy:
- …

9. Configuration Parameters:
- …
```

Summary:

Given structured requirements, generate a complete Low-Level Design document containing...

Raw Output:

1. Architecture Standards:
   - C++17 compliance for all source code.
   - POSIX API usage for platform abstraction.
   - Modular separation: SIP, Fax, Application logic in distinct modules.
   - RAII (Resource Acquisition Is Initialization) for resource management.
   - RESTful JSON API for external control.
   - OpenAPI 3.0 for API specification.
   - PostgreSQL for persistent data modeling (demo purposes).
   - PlantUML for UML diagrams.
   - All protocol interactions simulated; no real network stack.
   - IPv4 assumed for any simulated network transport.

2. Entities:
   - MainApplication: Orchestrates the FOIP workflow and coordinates modules.
   - PJSIPStub: Simulates SIP signaling and UDPTL transport.
   - SpanDSPStub: Simulates T.30 fax session management and page encoding.
   - FaxSession: Represents a single fax transmission session.
   - CallStatus: Tracks boolean status for call, transmission, hangup.
   - LogManager: Outputs logs to stdout.
   - Config: Holds configuration parameters and constants.

3. Flows:
   1. FOIP Transmission Flow:
      - MainApplication receives target phone number and sample page data.
      - MainApplication initiates SIP call via PJSIPStub.
      - Upon call establishment, MainApplication starts fax session via SpanDSPStub.
      - SpanDSPStub encodes page data into T.30 frames.
      - Encoded fax data is passed to PJSIPStub for UDPTL transmission.
      - MainApplication monitors transmission status.
      - After transmission, MainApplication tears down SIP call via PJSIPStub.
      - MainApplication outputs boolean status and logs to stdout.
   2. Error Handling Flow:
      - Each module returns boolean status for success/failure.
      - On failure, MainApplication logs error and ensures resource cleanup.

4. API Specification (OpenAPI 3.0 YAML):

```yaml
openapi: 3.0.0
info:
  title: FOIP Demo API
  version: 1.0.0
paths:
  /fax/send:
    post:
      summary: Simulate FOIP fax transmission
      requestBody:
        required: true
        content:
          application/json:
            schema:
              type: object
              properties:
                phone_number:
                  type: string
                  description: Target phone number
                page_data:
                  type: string
                  format: byte
                  description: Raw page data (base64-encoded)
      responses:
        '200':
          description: Transmission result
          content:
            application/json:
              schema:
                type: object
                properties:
                  call_status:
                    type: boolean
                    description: SIP call establishment status
                  transmission_status:
                    type: boolean
                    description: Fax transmission status
                  hangup_status:
                    type: boolean
                    description: SIP call teardown status
                  logs:
                    type: array
                    items:
                      type: string
                    description: Workflow logs
        '400':
          description: Invalid input
          content:
            application/json:
              schema:
                type: object
                properties:
                  error:
                    type: string
        '500':
          description: Internal error
          content:
            application/json:
              schema:
                type: object
                properties:
                  error:
                    type: string
components:
  schemas:
    FaxSendRequest:
      type: object
      properties:
        phone_number:
          type: string
        page_data:
          type: string
          format: byte
    FaxSendResponse:
      type: object
      properties:
        call_status:
          type: boolean
        transmission_status:
          type: boolean
        hangup_status:
          type: boolean
        logs:
          type: array
          items:
            type: string
```

5. Sequence Diagram (PlantUML):

```
@startuml
actor User
```

```
participant MainApplication
participant PJSIPStub
participant SpanDSPStub

User -> MainApplication: Submit phone number, page data
MainApplication -> PJSIPStub: Establish SIP call
PJSIPStub --> MainApplication: Call status (bool)
MainApplication -> SpanDSPStub: Start fax session, encode page
SpanDSPStub --> MainApplication: Encoded fax data, session status (bool)
SpanDSPStub -> PJSIPStub: Transmit T.38 fax data via UDPTL
PJSIPStub --> SpanDSPStub: Transmission status (bool)
MainApplication -> PJSIPStub: Teardown SIP call
PJSIPStub --> MainApplication: Hangup status (bool)
MainApplication -> User: Return statuses, logs
@enduml
```

6. Class Diagram (PlantUML):

```
@startuml
class MainApplication {
  +sendFax(phoneNumber: string, pageData: byte[]): FaxSendResponse
  +log(message: string): void
}

class PJSIPStub {
  +establishCall(phoneNumber: string): bool
  +transmitUDPTL(data: byte[]): bool
  +teardownCall(): bool
}

class SpanDSPStub {
  +startSession(pageData: byte[]): bool
  +encodePage(pageData: byte[]): byte[]
  +getEncodedFaxData(): byte[]
}

class FaxSession {
  +phoneNumber: string
  +pageData: byte[]
  +encodedData: byte[]
  +status: bool
}
```

```
class CallStatus {
  +callEstablished: bool
  +transmissionSuccess: bool
  +hangupSuccess: bool
}

class LogManager {
  +log(message: string): void
}

class Config {
  +SIP_TIMEOUT: int
  +FAX_MAX_PAGE_SIZE: int
}

MainApplication --> PJSIPStub
MainApplication --> SpanDSPStub
MainApplication --> LogManager
MainApplication --> Config
MainApplication --> FaxSession
MainApplication --> CallStatus
SpanDSPStub --> PJSIPStub : <<uses>>
@enduml
```

7. Database Schema (PostgreSQL):

```sql
CREATE TABLE fax_sessions (
    id SERIAL PRIMARY KEY,
    phone_number VARCHAR(32) NOT NULL,
    page_data BYTEA NOT NULL,
    encoded_data BYTEA,
    call_status BOOLEAN NOT NULL,
    transmission_status BOOLEAN NOT NULL,
    hangup_status BOOLEAN NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE logs (
    id SERIAL PRIMARY KEY,
    fax_session_id INTEGER REFERENCES fax_sessions(id) ON DELETE CASCADE,
    log_message TEXT NOT NULL,
    log_time TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

8. Error Handling Strategy:
- Boolean return values for all major operations (call, transmission, hangup).
- Invalid input (e.g., missing phone number, malformed page data) returns HTTP 400.
- Internal errors (e.g., stub failure, resource allocation issues) return HTTP 500.
- All errors logged to stdout via LogManager.
- RAII ensures automatic resource cleanup on error paths.
- No exception propagation beyond module boundaries; errors handled locally.

9. Configuration Parameters:
- SIP_TIMEOUT: 5000 ms (default call setup/teardown timeout)
- FAX_MAX_PAGE_SIZE: 1 MB (maximum allowed page data size)
- LOG_LEVEL: "INFO" (stdout logging verbosity)
- UDPTL_SIM_PORT: 4000 (simulated UDPTL port)
- IPV4_BIND_ADDR: "127.0.0.1" (simulated network bind address)
- RAM_USAGE_LIMIT: 256 KB (soft limit for demo operation)
- SESSION_CLEANUP_INTERVAL: 10 seconds (resource cleanup timer)

This LLD strictly follows the provided requirements and constraints, ensuring modularity, standards compliance, and deterministic engineering structure.

Description:

You receive LLD from Agent 2 in a fixed structure:

Architecture Standards

Entities

Flows

API Specification

Sequence Diagram

Class Diagram

Database Schema

Error Handling Strategy

Configuration Parameters

Your job is to:

Detect missing sections

Fix formatting

Improve clarity

Normalize naming conventions

Improve class names and method names (without altering meaning)

Make diagrams consistent

Ensure database schema is valid PostgreSQL

Ensure OpenAPI is valid

Fix indents, syntax, PlantUML formatting

Remove noise

Improve quality

You MUST output the same sections in same order.

You MUST NOT:

'L Add new entities
'L Add new flows
'L Add new APIs
'L Modify system behavior
'L Invent new features

You are ONLY polishing and correcting.

You have access to the KnowledgeRAGTool. It is advised to use this tool when you have access to it irrespective of whether a knowledge retrieval task has been assigned explicitly.

Expected Output:

1. Architecture Standards:
(Refined, cleaned, consistent)

2. Entities:
- EntityName: clear, concise description
…

3. Flows:
- Refine text, ensure clarity and correct numbering.

4. API Specification (OpenAPI 3.0 YAML):
- Correct indentation
- Correct value types
- Validate YAML

```
openapi: 3.0.0
info:
  title: ...
  version: ...
paths:
  ...
```

5. Sequence Diagram (PlantUML):

```
@startuml
...
@enduml
```

Ensure:
- Participant names consistent

- Arrow directions correct
- Flow meaningful

6. Class Diagram (PlantUML):
@startuml
...
@enduml

7. Database Schema (PostgreSQL):
Ensure:
- Valid SQL
- Constraints correct
- Keys properly defined
- CREATE TABLE ...

8. Error Handling Strategy:
- Improve clarity, consistency, formatting.

9. Configuration Parameters:
- List structured parameters cleanly.

Ø=Ý7 R U L E S
- DO NOT modify functional requirements.
- DO NOT invent new domain items.
- Maintain exact section order.
- Maintain exact section titles.
- Fix only quality, accuracy, formatting, and consistency.
- Output only the refined LLD, no explanations.

Summary:

You receive LLD from Agent 2 in a fixed structure:

Architecture...

Raw Output:

No LLD file was found in the provided directory. Please provide the LLD content in the required structure for review and refinement.