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 stubbed PJSIP module.
- FR2: Manage T.30 fax session and encode page data using a stubbed SpanDSP module.
- FR3: Transmit T.38-encapsulated fax data via simulated UDPTL transport.
- FR4: Orchestrate the end-to-end FOIP workflow, coordinating between SIP and fax protocol layers.
- FR5: Accept target phone number and sample page data as inputs.
- FR6: Output workflow status and logs to console.

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 main application; manual verification via console output.
- NFR11: Protocol compliance simulated; not validated against real protocol stacks.

System Interfaces:
- Interface1: SIP Signaling Layer (PJSIP Stub): Simulated software interface for call setup/teardown and UDPTL transport; interacts with Main Application and receives encoded data from Fax Protocol Layer.
- Interface2: Fax Protocol Layer (SpanDSP Stub): Simulated software interface for T.30 fax session management and page encoding; interacts with Main Application.

- Interface3: Main Application: Software interface orchestrating FOIP workflow, coordinating between SIP and fax layers.
- 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 platforms 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: No explicit power or environmental constraints.
- Constraint6: Not intended for production use; lacks full security and error recovery.
- 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).
- 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 required.
- A4: IPv4 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) for production or field deployment?
- Q3: Should automated unit and protocol compliance tests be developed for each module?
- Q4: Is explicit measurement or enforcement of RAM usage required for production?
- Q5: Who are the author, date, and reviewers/approvers for the final document metadata?

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:
- Use PlantUML for all UML diagrams (sequence, class).
- Follow organization naming conventions for all classes, methods, and database objects.
- API specification must be OpenAPI 3.0 YAML compliant.
- Database schema must be in PostgreSQL format with explicit datatypes and constraints.
- Modular C++17 design with clear separation of SIP, fax, and application logic.
- Resource management via RAII principles.
- All protocol interactions are stubbed; no real network or telephony stack.
- No authentication, encryption, or safety-critical features in stub/demo implementation.
- Console output for workflow status and logs.
- Error handling via boolean return values only.

2. Entities:
- MainApplication: Orchestrates the FOIP workflow, coordinates SIP and fax layers.
- SIPSignalingStub: Simulates SIP call setup, teardown, and UDPTL transport.
- FaxProtocolStub: Simulates T.30 fax session management and page encoding.
- UDPTLTransportStub: Simulates T.38-encapsulated fax data transmission.
- WorkflowLogger: Outputs workflow status and logs to console.
- InputData: Holds target phone number and sample page data.
- WorkflowStatus: Tracks boolean status for call, transmission, hangup, and session.

3. Flows:
1. MainApplication receives target phone number and sample page data as input.
2. MainApplication invokes SIPSignalingStub to simulate SIP call establishment.
3. On successful call setup, MainApplication invokes FaxProtocolStub to start T.30 fax session.
4. FaxProtocolStub encodes the sample page data into T.30 frames.
5. FaxProtocolStub passes encoded frames to UDPTLTransportStub for T.38 encapsulation and simulated transmission.
6. UDPTLTransportStub returns transmission status to MainApplication.
7. MainApplication instructs SIPSignalingStub to simulate call teardown.
8. MainApplication collects all status booleans and logs workflow status via WorkflowLogger.
9. MainApplication ensures all resources are cleaned up automatically (RAII).

4. API Specification (OpenAPI 3.0 YAML):
```
openapi: 3.0.0
info:
  title: FOIP Simulation API
  version: 1.0.0
paths:
  /foip/sendFax:
```

```yaml
    post:
      summary: Simulate end-to-end FOIP fax transmission
      requestBody:
        required: true
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/SendFaxRequest'
      responses:
        '200':
          description: Fax transmission simulation completed
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/SendFaxResponse'
        '400':
          description: Invalid input data
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/ErrorResponse'
        '500':
          description: Internal simulation error
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/ErrorResponse'
components:
  schemas:
    SendFaxRequest:
      type: object
      required:
        - targetPhoneNumber
        - pageData
      properties:
        targetPhoneNumber:
          type: string
          description: Target phone number for fax transmission
        pageData:
          type: string
          format: byte
          description: Raw page data (base64-encoded)
    SendFaxResponse:
      type: object
```

```
    properties:
      callEstablished:
        type: boolean
      faxSessionStarted:
        type: boolean
      pageEncoded:
        type: boolean
      transmissionSuccess:
        type: boolean
      callTeardown:
        type: boolean
      logs:
        type: array
        items:
          type: string
  ErrorResponse:
    type: object
    properties:
      errorCode:
        type: string
      errorMessage:
        type: string
```

5. Sequence Diagram (PlantUML):

```
@startuml
actor User
participant MainApplication
participant SIPSignalingStub
participant FaxProtocolStub
participant UDPTLTransportStub
participant WorkflowLogger

User -> MainApplication: Provide targetPhoneNumber, pageData
MainApplication -> SIPSignalingStub: establishCall(targetPhoneNumber)
SIPSignalingStub --> MainApplication: callEstablished (bool)
MainApplication -> FaxProtocolStub: startFaxSession(pageData)
FaxProtocolStub --> MainApplication: faxSessionStarted (bool)
FaxProtocolStub -> FaxProtocolStub: encodePageData()
FaxProtocolStub --> MainApplication: pageEncoded (bool)
FaxProtocolStub -> UDPTLTransportStub: transmitT38(encodedFrames)
UDPTLTransportStub --> FaxProtocolStub: transmissionSuccess (bool)
FaxProtocolStub --> MainApplication: transmissionSuccess (bool)
MainApplication -> SIPSignalingStub: teardownCall()
SIPSignalingStub --> MainApplication: callTeardown (bool)
```

```
MainApplication -> WorkflowLogger: log(status, logs)
WorkflowLogger --> MainApplication: logComplete
MainApplication -> User: return status/logs
@enduml

6. Class Diagram (PlantUML):
@startuml
class MainApplication {
  +sendFax(targetPhoneNumber: string, pageData: vector<byte>): WorkflowStatus
}
class SIPSignalingStub {
  +establishCall(targetPhoneNumber: string): bool
  +teardownCall(): bool
}
class FaxProtocolStub {
  +startFaxSession(pageData: vector<byte>): bool
  +encodePageData(pageData: vector<byte>): vector<byte>
  +transmitT38(encodedFrames: vector<byte>): bool
}
class UDPTLTransportStub {
  +transmitT38(encodedFrames: vector<byte>): bool
}
class WorkflowLogger {
  +log(status: WorkflowStatus, logs: vector<string>): void
}
class InputData {
  +targetPhoneNumber: string
  +pageData: vector<byte>
}
class WorkflowStatus {
  +callEstablished: bool
  +faxSessionStarted: bool
  +pageEncoded: bool
  +transmissionSuccess: bool
  +callTeardown: bool
  +logs: vector<string>
}

MainApplication --> SIPSignalingStub
MainApplication --> FaxProtocolStub
FaxProtocolStub --> UDPTLTransportStub
MainApplication --> WorkflowLogger
MainApplication --> InputData
MainApplication --> WorkflowStatus
```

@enduml

7. Database Schema (PostgreSQL):

```
CREATE TABLE foip_workflow_log (
    id SERIAL PRIMARY KEY,
    timestamp TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
    target_phone_number VARCHAR(32) NOT NULL,
    call_established BOOLEAN NOT NULL,
    fax_session_started BOOLEAN NOT NULL,
    page_encoded BOOLEAN NOT NULL,
    transmission_success BOOLEAN NOT NULL,
    call_teardown BOOLEAN NOT NULL,
    logs TEXT[],
    status VARCHAR(16) NOT NULL CHECK (status IN ('SUCCESS', 'FAILURE'))
);
```

8. Error Handling Strategy:
- All module APIs return boolean status; false indicates failure at that stage.
- MainApplication aggregates status booleans and logs error context.
- API returns errorCode and errorMessage in case of input validation or internal errors.
- No exceptions are thrown; all errors are handled via return values and logging.
- Input validation: targetPhoneNumber must be non-empty, pageData must be non-empty.
- Resource cleanup is automatic via RAII; no explicit error recovery for resource leaks.
- No authentication, encryption, or protocol compliance errors are handled in stub.

9. Configuration Parameters:
- Max page data size: 1 MB (for demo, not enforced in stub).
- Log output: Console only.
- Supported protocol: T.30 over T.38 (simulated).
- Network simulation: IPv4 only.
- RAM usage: "d 256 KB (expected, not enforced).
- No authentication or encryption.
- No real-time or latency constraints.
- No hardware abstraction layer required.
- Demo only; not for production or regulated environments.

Description:

You will always receive some text input from Agent 2.
You must never assume the input is missing.
You must continue processing it even if:
- the LLD is incomplete
- the LLD is missing sections
- the LLD has formatting issues
- the LLD has partial diagrams
- the LLD only contains some sections
You must never return errors like “No LLD provided” or “file not found.”
If a section is missing!’ generate a refined version of that section.
If a section exists!’ refine and improve it.
You must always output a complete refined LLD in the fixed format.

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:

Must follow exactly:
1. Architecture Standards:
   - …

2. Entities:
   - …

3. Flows:
   1. …
   2. …

4. API Specification (OpenAPI 3.0 YAML):
   …

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

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

7. Database Schema (PostgreSQL):
   …

8. Error Handling Strategy:

…

9. Configuration Parameters:

…

RULES:

- Never output errors.
- Never ask for files or directories.
- Never say “No LLD provided.”
- Never return markdown formatting.
- If the input LLD is incomplete!’ fill missing parts based on architectural log
- If the input LLD is present!’ refine it.
- Maintain PlantUML syntax exactly.
- Maintain OpenAPI 3.0 format exactly.
- Maintain PostgreSQL schema format exactly.

Summary:

You will always receive some text input from Agent 2.

You...

Raw Output:

1. Architecture Standards:

- Use PlantUML for all UML diagrams (sequence, class).
- API specification must be OpenAPI 3.0 YAML compliant.
- Database schema must be in PostgreSQL format with explicit datatypes and constraints.
- Modular C++17 design with clear separation of SIP, fax, and application logic.
- Resource management via RAII principles.
- All protocol interactions are stubbed; no real network or telephony stack.
- No authentication, encryption, or safety-critical features in stub/demo implementation.
- Console output for workflow status and logs.
- Error handling via boolean return values only.
- Follow SOLID principles, especially Interface Segregation and minimum dependencies between modules.
- Use organization naming conventions for all classes, methods, and database objects.

2. Entities:

- MainApplication: Orchestrates the FOIP workflow, coordinates SIP and fax layers.
- SIPSignalingStub: Simulates SIP call setup, teardown, and UDPTL transport.
- FaxProtocolStub: Simulates T.30 fax session management and page encoding.
- UDPTLTransportStub: Simulates T.38-encapsulated fax data transmission.
- WorkflowLogger: Outputs workflow status and logs to console.
- InputData: Holds target phone number and sample page data.
- WorkflowStatus: Tracks boolean status for call, transmission, hangup, and session.

3. Flows:

1. MainApplication receives target phone number and sample page data as input.
2. MainApplication invokes SIPSignalingStub to simulate SIP call establishment.
3. On successful call setup, MainApplication invokes FaxProtocolStub to start T.30 fax session.
4. FaxProtocolStub encodes the sample page data into T.30 frames.
5. FaxProtocolStub passes encoded frames to UDPTLTransportStub for T.38 encapsulation and simulated transmission.
6. UDPTLTransportStub returns transmission status to MainApplication.
7. MainApplication instructs SIPSignalingStub to simulate call teardown.
8. MainApplication collects all status booleans and logs workflow status via WorkflowLogger.
9. MainApplication ensures all resources are cleaned up automatically (RAII).

4. API Specification (OpenAPI 3.0 YAML):

```
openapi: 3.0.0
info:
  title: FOIP Simulation API
  version: 1.0.0
paths:
  /foip/sendFax:
    post:
      summary: Simulate end-to-end FOIP fax transmission
      requestBody:
        required: true
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/SendFaxRequest'
      responses:
        '200':
          description: Fax transmission simulation completed
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/SendFaxResponse'
        '400':
          description: Invalid input data
          content:
            application/json:
              schema:
                $ref: '#/components/schemas/ErrorResponse'
        '500':
          description: Internal simulation error
          content:
            application/json:
```

```
            schema:
              $ref: '#/components/schemas/ErrorResponse'
components:
  schemas:
    SendFaxRequest:
      type: object
      required:
        - targetPhoneNumber
        - pageData
      properties:
        targetPhoneNumber:
          type: string
          description: Target phone number for fax transmission
        pageData:
          type: string
          format: byte
          description: Raw page data (base64-encoded)
    SendFaxResponse:
      type: object
      properties:
        callEstablished:
          type: boolean
        faxSessionStarted:
          type: boolean
        pageEncoded:
          type: boolean
        transmissionSuccess:
          type: boolean
        callTeardown:
          type: boolean
        logs:
          type: array
          items:
            type: string
    ErrorResponse:
      type: object
      properties:
        errorCode:
          type: string
        errorMessage:
          type: string
```

5. Sequence Diagram (PlantUML):

```
@startuml
```

```
actor User
participant MainApplication
participant SIPSignalingStub
participant FaxProtocolStub
participant UDPTLTransportStub
participant WorkflowLogger

User -> MainApplication: Provide targetPhoneNumber, pageData
MainApplication -> SIPSignalingStub: establishCall(targetPhoneNumber)
SIPSignalingStub --> MainApplication: callEstablished (bool)
MainApplication -> FaxProtocolStub: startFaxSession(pageData)
FaxProtocolStub --> MainApplication: faxSessionStarted (bool)
FaxProtocolStub -> FaxProtocolStub: encodePageData()
FaxProtocolStub --> MainApplication: pageEncoded (bool)
FaxProtocolStub -> UDPTLTransportStub: transmitT38(encodedFrames)
UDPTLTransportStub --> FaxProtocolStub: transmissionSuccess (bool)
FaxProtocolStub --> MainApplication: transmissionSuccess (bool)
MainApplication -> SIPSignalingStub: teardownCall()
SIPSignalingStub --> MainApplication: callTeardown (bool)
MainApplication -> WorkflowLogger: log(status, logs)
WorkflowLogger --> MainApplication: logComplete
MainApplication -> User: return status/logs
@enduml

6. Class Diagram (PlantUML):
@startuml
class MainApplication {
  +sendFax(targetPhoneNumber: string, pageData: vector<byte>): WorkflowStatus
}
class SIPSignalingStub {
  +establishCall(targetPhoneNumber: string): bool
  +teardownCall(): bool
}
class FaxProtocolStub {
  +startFaxSession(pageData: vector<byte>): bool
  +encodePageData(pageData: vector<byte>): vector<byte>
  +transmitT38(encodedFrames: vector<byte>): bool
}
class UDPTLTransportStub {
  +transmitT38(encodedFrames: vector<byte>): bool
}
class WorkflowLogger {
  +log(status: WorkflowStatus, logs: vector<string>): void
}
```

```
class InputData {
  +targetPhoneNumber: string
  +pageData: vector<byte>
}
class WorkflowStatus {
  +callEstablished: bool
  +faxSessionStarted: bool
  +pageEncoded: bool
  +transmissionSuccess: bool
  +callTeardown: bool
  +logs: vector<string>
}

MainApplication --> SIPSignalingStub
MainApplication --> FaxProtocolStub
FaxProtocolStub --> UDPTLTransportStub
MainApplication --> WorkflowLogger
MainApplication --> InputData
MainApplication --> WorkflowStatus
@enduml
```

7. Database Schema (PostgreSQL):

```
CREATE TABLE foip_workflow_log (
    id SERIAL PRIMARY KEY,
    timestamp TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
    target_phone_number VARCHAR(32) NOT NULL,
    call_established BOOLEAN NOT NULL,
    fax_session_started BOOLEAN NOT NULL,
    page_encoded BOOLEAN NOT NULL,
    transmission_success BOOLEAN NOT NULL,
    call_teardown BOOLEAN NOT NULL,
    logs TEXT[],
    status VARCHAR(16) NOT NULL CHECK (status IN ('SUCCESS', 'FAILURE'))
);
```

8. Error Handling Strategy:
- All module APIs return boolean status; false indicates failure at that stage.
- MainApplication aggregates status booleans and logs error context.
- API returns errorCode and errorMessage in case of input validation or internal errors.
- No exceptions are thrown; all errors are handled via return values and logging.
- Input validation: targetPhoneNumber must be non-empty, pageData must be non-empty.
- Resource cleanup is automatic via RAII; no explicit error recovery for resource leaks.
- No authentication, encryption, or protocol compliance errors are handled in stub.

9. Configuration Parameters:

- Max page data size: 1 MB (for demo, not enforced in stub).
- Log output: Console only.
- Supported protocol: T.30 over T.38 (simulated).
- Network simulation: IPv4 only.
- RAM usage: "d 256 KB (expected, not enforced).
- No authentication or encryption.
- No real-time or latency constraints.
- No hardware abstraction layer required.
- Demo only; not for production or regulated environments.