Project - High Level Design

on

Finance Management Triage Agent

Agentic AI

***Institution Name*:** Medicaps University – Datagami Skill Based Course

Student Name(s) & Enrolment Number(s):

Sr no	Student Name	Enrolment Number
1.	Mahima Patidar	EN22CS301569
2.	Nitisha Kushwah	EN22CS301664
3.	Nikhil Rathore	EN22CS301650
4.	Nidhish Mehta	EN22CS301642
5.	Moin Husain	EN22CS301612
6.	Manasavi Jain	EN22CS301580

Group Name:12D6

Project Number:AAI-23

Industry Mentor Name:

University Mentor Name –Nishant Shrivastav

Academic Year:2026

Table of Contents

1. Introduction

This document describes the design, implementation, and functionalities of the **Finance AI Ticket System**, an intelligent system developed to automate the handling of finance-related queries, tickets, and requests received by the finance department of an organization. The system combines OCR, AI-based response generation, web scraping, and natural language processing to provide accurate and timely responses to user
tickets.

1.1 Scope of the Document

The scope of this document includes:

1. Project Overview:

- Explains the purpose and goals of the Finance AI Ticket System.
- Highlights the importance of automating ticket processingin finance operations.

2. System Design:

- High-Level Design (HLD) illustrating module interactions.
- Low-Level Design (LLD) describing classes, functions, and data flow.

3. Implementation Details:

- Explanation of the technologies used (Python, Selenium, OpenAI API, LangChain, OCR libraries, database).
- Description of each module: OCR, Classification, NER, Finance Rule Fetching, AI Response, and Ticket Saving.

4. User Interaction:

- Shows how users (employees, finance officers) submit tickets (via PDF, image, or text).
- Demonstrates how the system responds automatically or flags urgent tickets.

5. Testing s Deployment:

- Unit testing for each module.
- End-to-end testing to ensure accurate ticket processing.
- Deployment guidelines for running the system locally or on a server.

1.2 Intended Audience

The intended audience of this document includes:

- **Project Developers:**
  To understand system architecture, module design, and implementation details for coding and integration.
- **Testers s QA Team:**
  To verify the system's accuracy, performance, and reliability through unit and integration testing.
- **Project Stakeholders / Managers:**
  To gain insights into the objectives, system flow, and value proposition of automating finance ticket handling.
- **Future Developers / Maintainers:**
  To facilitate enhancements, debugging, andmaintenance of the system.

1.3 System Overview

The **Finance AI Ticket System** is designed to automate the extraction, classification, and response generation for finance-related tickets. It primarily supports the following functionalities:

1. **Ticket Input:**

- Users can submit finance queries via PDF, image files, or plain text.
- OCR (Optical Character Recognition) converts uploaded documents into machine-readable text.

2. **Text Classification:**

- Each ticket is analyzed to determine its **urgency** (high, medium, low) and the **department** responsible.
- Classification ensures prioritization of critical finance requests.

3. **Finance Rule Fetching:**

- The system uses **web scraping** or database queries to fetch relevant finance rules from authoritative sources (e.g., Reserve Bank of India guidelines).

4. **AI Response Generation:**

- Based on ticket content, extracted entities, urgency, department, and finance rules, the system generates **automated responses** using **OpenAI's GPT API** integrated via LangChain.
- In case AI is unavailable, the system uses a **fallback response** to ensure continuity.

5. **Database Storage:**

- All processed tickets, along with extracted information and AI responses, are saved to a database for auditing and future reference.

6. **System Architecture:**

- The system is modular and loosely coupled, enabling independent updates to OCR, AI response, and scraping modules.
- Designed for **scalability** and **easy integration** with future enhancements such as multi-user support or real-time monitoring dashboards.

Benefits:

- Reduces manual effort in handling finance tickets.
- Provides faster, consistent, and accurate responses.
- Ensures prioritization of critical tickets.
- Maintains a history of all tickets for auditing and compliance.

2. System Design

This section provides a structured overview of how the **Finance AI Ticket System** is designed, including the architecture, module interactions, and workflow. It gives both a high-level understanding (HLD) and a view of the individual processes.

2.1 Application Design

The **Finance AI Ticket System** is designed as a **modular, scalable application** to handle finance-related queries automatically. The architecture follows a **layered and modular approach**, where each component is responsible for a specific task.

Modules Overview:

1. Input Module:

- Handles the submission of tickets via **PDF, image, or plain text**.
- For documents, **OCR** converts the content into machine-readable text.

2. Classification Module:

- Classifies tickets based on **urgency** (high, medium, low) and **department** (Finance, Accounting, Tax, etc.).
- Ensures that high-priority tickets are handled first.

3. Entity Extraction (NER) Module:

- Uses **Named Entity Recognition (NER)** to extract key information such as amounts, dates, account numbers, and organization names.
- Helps in creating context-aware responses.

4. Finance Rule Scraping Module:

- Automatically fetches relevant finance rules or regulations from authoritative sources (e.g., RBI) using **web scraping**.
- Ensures responses are compliant with current rules.

5. AI Response Module:

- Uses **OpenAI GPT API** integrated via **LangChain** to generate professional responses.
- If AI is unavailable or API fails, a **fallback response** is used to maintain continuity.

6. Database Module:

- Saves processed tickets, extracted information, and AI responses.
- Supports auditing, reporting, and future enhancements.

Architecture Highlights:

- **Modular s Loosely Coupled:** Each module can be updated independently.
- **Scalable:** Can handle increasing ticket volume by adding worker instances or parallelizing processing.
- **Fault-Tolerant:** Uses fallback responses if AI or external sources are unavailable.
- **Extensible:** New modules (like multi-language support or voice-based input) can be added with minimal changes.

2.2 Process Flow

The **process flow** defines the sequence of operations from ticket submission to AI-generated response and ticket storage.

Step-by-Step Flow:

1. Ticket Submission:

- User submits a ticket via **uploading a PDF/image** or **direct text input**.

2. Text Extraction (OCR):

- If the input is a document, **OCR service** extracts text.
- Ensures all ticket content is machine-readable.

3. Text Classification:

- Ticket text is analyzed to determine **urgency** and **department**.
- Helps in prioritization and routing of tickets.

4. Entity Extraction:

- **NER agent** identifies key entities like **dates, amounts, and organizations**.
- Entities are used to create context-aware AI responses.

5. Finance Rule Fetching:

- **Web scraping module** fetches the latest applicable finance rules.
- Ensures compliance in automated responses.

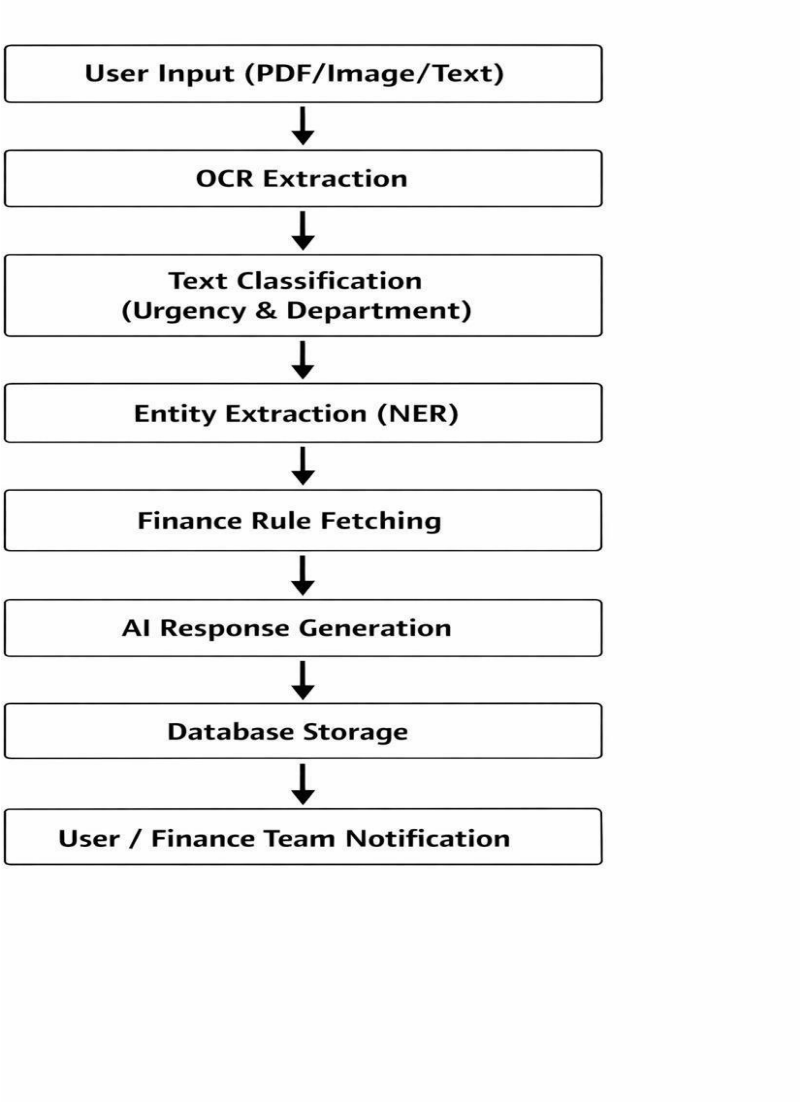
6. **Database Storage:**

- Ticket details, extracted entities, rules, and generated response are stored.
- Enables tracking, auditing, and historical analysis.

7. **Notification / Output:**

- AI response is displayed to the user (or saved for finance officers to review).

Process Flow Diagram:



2.3 Information Flow

The information flow describes how data moves through the system from the moment a user submits a finance ticket until the response is generated and stored:

1. User Submission:

- Users submit tickets via PDF, image, or plain text.
- The file or text is stored temporarily in the system for processing.

2. OCR Extraction:

- If the input is a PDF or image, the system extracts text using OCR (Optical Character Recognition).
- Extracted text is passed to the next stage.

3. Text Classification:

- The extracted text is analyzed to determine the **urgency**
    - (high/medium/low) and **department** (finance/accounting/operations).
- Classification information is stored for later use and response context.

4. Entity Extraction (NER):

- Named entities such as names, dates, amounts, and invoice numbers are extracted.
- This information enriches the context for AI-based response generation.

5. Finance Rule Scraping:

- The system fetches the latest relevant finance rules from sources like RBI or internal guidelines.
- These rules are added to the ticket context to ensure AI responses comply with policy.

6. AI Response Generation:

- The consolidated context (text, classification, entities, rules) is sent to the AI module (OpenAI GPT).
- A professional response is generated for the user.

7. Fallback Response:

- If AI service is unavailable, a predefined generic response is used.

Visual Flow:

- Input → OCR → Classification → Entity Extraction → Finance Rules → AI Response → Save C Notify

---

2.4 Components Design

The system is modular, and each component has a defined responsibility:

1. **User Input Module:**

- Accepts PDF, image, or text input.
- Performs initial validation on file types.

2. **OCR Module:**

- Extracts text from PDFs/images using libraries like **PyMuPDF** or **Tesseract**.

3. **Classification Module:**

- Uses machine learning or rule-based methods to classify urgency and department.
- Outputs structured labels for downstream processing.

4. **Entity Extraction Module (NER):**

- Extracts important entities like names, dates, account numbers using NLP libraries (**spaCy**, **HuggingFace transformers**).

5 .**Finance Rule Module (Scraper):**

- Web scraper fetches updated finance policies from trusted sources.
- Ensures the system's responses are compliant with regulations.

6 .**AI Response Module:**

- Uses **OpenAI GPT** for generatingprofessional responses.
- Context includes text, classification, entities, and finance rules.

7. Database Module:

- Stores ticket details includingtext, entities, classification, AI response, and metadata.
- Ensures tickets can be retrieved and audited later.

8. Notification Module:

- Sends notifications to users and finance team

2.5 Key Design Considerations

When designing the system, the following factors were prioritized:

3. **Scalability:**

- The system should handle multiple tickets simultaneously without performance degradation.
- Components like OCR, AI response generation, and scrapingshould support concurrent processing.

4. **Reliability s Fault Tolerance:**

- Fallback responses ensure ticket processing continues even if AI service is down.
- Scraper errors or network failures should not crash the system.

5. **Security:**

- User data, including financial details, must be handled securely.
- API keys (OpenAI) and sensitive configuration stored in environment variables.
- Database connections secured with proper authentication.

6. **Maintainability:**

- Modular design: OCR, classification, NER, scraping, AIresponse, database modules are separate.
- Easy to update or replace modules without affecting the whole system.

2.6 API Catalogue

The System uses several internal and external APIs. Here's catalogue

API Name	Description	Input	Output
OCR Service	Extracts text from PDF/image files	File path (PDF/Image)	Raw text
Classification API	Determines ticket urgencyand department	Text	urgency, department labels
NER (Entity Extraction) API	Extracts named entities like names, dates, amounts	Text	List of entities
Finance Rule Scraper	Fetches finance rules from RBI or internal sources	None	Latest finance rule text
AI Response API (OpenAI)	Generates professional response based on context	Dictionary (urgency, dept, entities, rule)	Response text
Database API	Stores ticket information	Dictionarywith all ticket details	Success/Failure status
Notification API	Sends notifications to users/finance team	Ticket ID, message	Notification

3. Data Design

The Data Design section describes how ticket information, user inputs, and AI- generated responses are stored, accessed, and managed in the system. Data Model

The system uses a relational database (e.g., SQLite/MySQL) to store structured data for tickets and related entities.

3.1 Entities and Attributes

1. **Ticket**
    - ticket_id (Primary Key) – Unique identifier for each ticket

- urgency– High, Medium, Low
- department – Department assigned
- entities – Extracted named entities (JSON/text)
- finance_rule – Latest finance rule fetched
- response – AI-generated response
- created_at – Timestamp
- updated_at – Timestamp

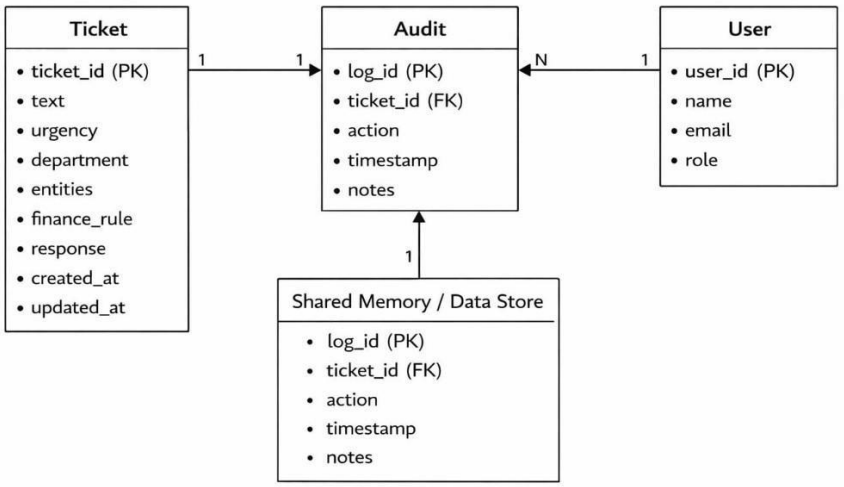
2. User (optional, if multi-user system)

user_id (Primary Key)

- name
- email
- role – Admin, Finance Officer, Requester

3. Audit/Logs

- log_id (Primary Key)
- ticket_id (Foreign Key)
- action – e.g., "ticket processed","response generated"
- timestamp
- notes – optional text



3.2 Data Access Mechanism

The system uses a **Data Access Layer (DAL)** to separate database logic from business logic. This improves maintainability and allows easy swapping of databases.

Mechanisms

1. Direct Queries

- SQL queries executed using Python libraries like sqlite3 or SQLAlchemy.
- Example: Fetch ticket details by ticket_id.

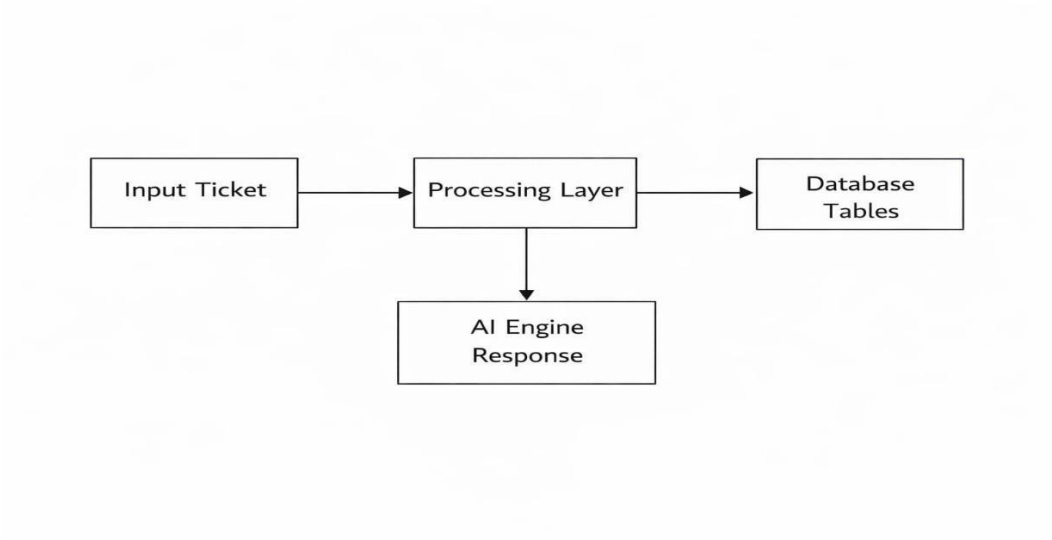
2. ORM (Object Relational Mapping)

- SQLAlchemycan be used to map Ticket, User, Audit classes to tables.
- Simplifies CRUD operations.

3. CRUD Operations Provided by DAL

Operation	Description	Input	Output
create_ticket()	Savenew ticket data details	Dictionary with ticket	Success / Failure
get_ticket()	Fetch ticket info	ticket_id	Ticket object / None

Operation	Description	Input	Output
update_ticket()	Update response or status	ticket_id, updated fields	Success / Failure
log_action()	Record an audit action	ticket_id, action, notes	Success / Failure



3.3. Data Retention Policies

Definition:

Data Retention Policies define how long different types of data (user info, tickets, logs) are stored, when they are archived, and when they are deleted, ensuring compliance with organizational and regulatory requirements.

Example Policies for Finance Ticket System:

Data Type	Retention Period	Archival/Deletion Policy
User information	Permanent	Keep user records indefinitely, unless account is deactivated.
Tickets	3 years	Archive older tickets into separate storage after 1 year. Delete after 3 years.
Audit/Logs	1 year	Store logs for operational review and auditing;

3.4. Data Migration

Definition:

Data Migration refers to transferring existing ticketing and finance data from old systems or spreadsheets into the new Finance AI Ticket System.

Migration Steps:

1. **Data Assessment:**

- Identify source systems (e.g., spreadsheets, legacy ticket databases).
- Evaluate data quality, consistency, and missing fields.

2. **Data Mapping:**

- Map old data fields to new system schema:
- Old_User_ID → user_id
- Old_Email → email
- Old_Ticket_ID → ticket_id
- Old_Action → action

3. **Data Cleaning:**

- Remove duplicates, correct formatting errors, normalize roles (Admin, Finance Officer, Requester).

4. **Migration Execution:**

- Use ETLscripts (Python, SQLscripts) to load cleaned data into new system tables.
- Validate migrated records for accuracy.

4. Interfaces

The **Interfaces** section describes how different parts of the system interact with each other, as well as how users interact with the system. It includes both **User Interfaces** (UI) and **System/Programmatic Interfaces** (APIs).

4.1. User Interfaces (UI)

Purpose:
Provide an intuitive interface for users to submit finance tickets, view AI-generated responses, and track the status of their requests.

Key UI Components:

Component	Description
Login/Authentication	Users log in with credentials; roles assigned (Admin, Finance Officer, Requester).
Ticket Submission Form	Allows users to upload PDFs/images or type their request; specifies urgency.
Ticket Dashboard	Shows all tickets, filtered by status, department, urgency, and assigned officer.
Response Viewer	Displays AI-generated response and finance rule applied.
Audit Logs Viewer	For Admins/Finance Officers to review ticket actions and logs.

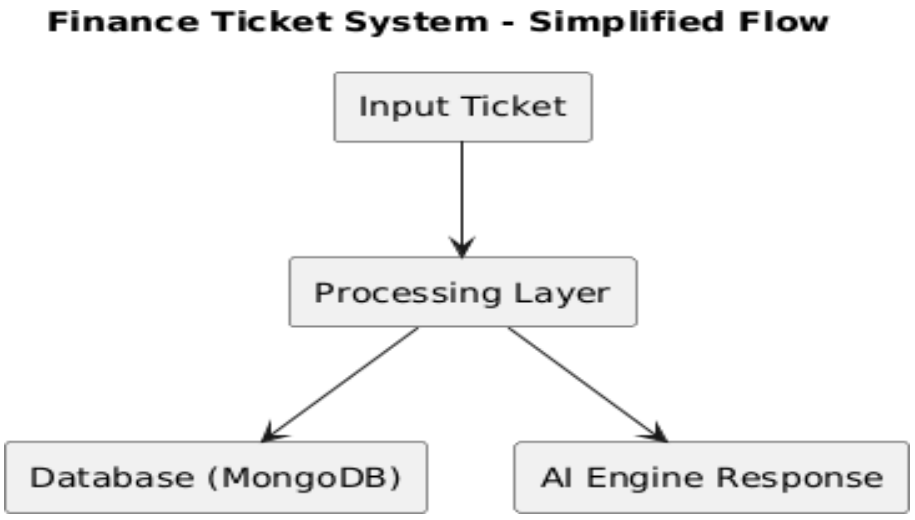
4.2 System Interfaces

Purpose:
Enable programmatic interaction between modules and with external systems (e.g., AI service, finance rule scraping).

Interface	Type	Description
NER Agent	Internal API Extracts	entities from ticket text.
Scraping Service	External API	Retrieves finance rules from RBI or other sites.

4.3 External Interfaces

1. **OpenAI API**
   - Used to generate AI-based responses for finance tickets.
   - Requires APIkeyand internet connectivity.
   - Endpoint: https://api.openai.com/v1/chat/completions
2. **Web Scraping**
   - Interacts with RBI or other finance rule sources.
   - Uses Selenium or requests/BeautifulSoup to extract live rules.
3. **Database Interface**
   - MongoDB database for tickets, users, and logs.
     CRUD operations for tickets, users, and logs.
   - Connects via Mongoose (Node.js) or MongoDB driver.



5 . State and Session Management

In the Finance AI Triage Agent system, **state and session management** play a crucial role in maintaining the context and continuity of user interactions. Since financial queries often involve multiple steps or follow-ups, it is essential for the system to track the current status of each interaction and retain relevant data across sessions.

State Management

- In this project, the system tracks the followingstate variables:
- current_query: The user's current finance-related message
- extracted_entities: Information extracted frominvoices, such as Invoice ID, Date, or Amount
- ai_response: The draft response generated by the AI engine
- session_id: A unique identifier assigned to each user session

Maintainingstate allows the system to process multiple queries efficiently, retain context for follow-ups, and ensure that responses remain accurate and relevant.

Session Management

- Each session is assigned a unique session_id that links all queries, extracted information, and AI responses together.
- Sessions are stored in the database (MongoDB), enabling the system to:
    1. Recognize returning users or continued queries
    2. Maintain a history of interactions for audit and review purposes
    3. Provide consistent and context-aware AI responses

Session Flow in the System:

1. User sends a query via the /triage endpoint.
2. System checks if a session exists:
    - If yes → loads previous context
    - If no → generates a new session_id
3. The query is processed by the AI module, and the response is generated.
4. Both the query and AI response are stored in the database under the session ID.

6. Caching

Caching is used in the Finance AITriage Agent to **improve performance and reduce repeated computations**.

Purpose of Caching:

- Reduce repeated LLM calls for the same query (saves API calls and time).
- Store recentlyprocessed OCR or scraped data temporarilyto avoid reprocessing.
- Enhance system responsiveness during high-load scenarios.

Caching Strategy:

Type	Use Case
In-Memory Cache (Pythondict / LRU cache)	Recentqueries and AI responses
Database Cache (MongoDB)	Persist frequently accessed policies or historical results
Optional Redis Cache	For production-grade, high-volume usage

7. Non-Functional Requirements (NFRs)

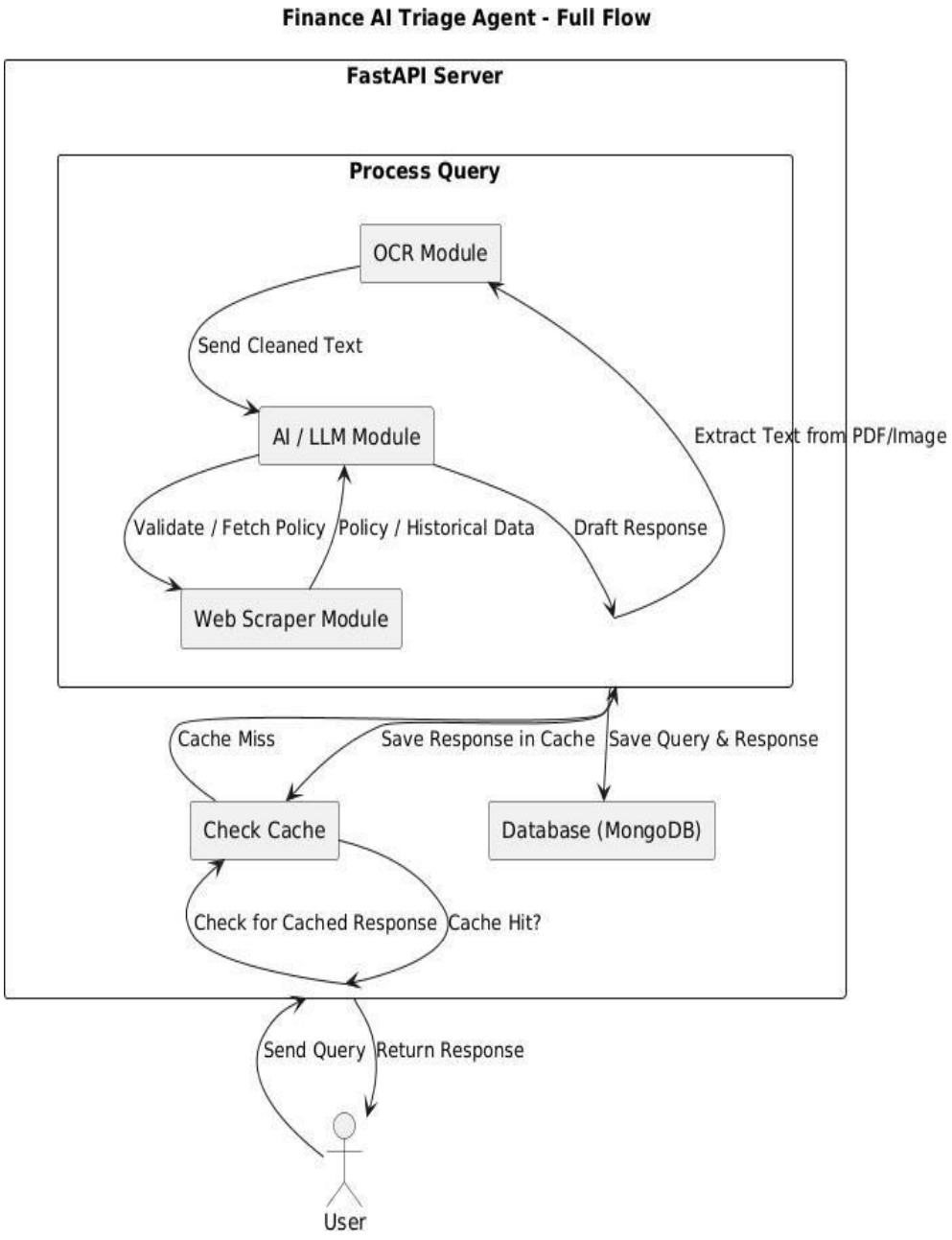
Non-functional requirements define **how the system performs**, rather than what it does.

7.1 Security Aspects

1. **Data Protection:**
   - Sensitive financial data is encrypted at rest (MongoDB) and in transit (HTTPS).
2. **API Key Management:**
   - LLM API keys (OpenAI/ Claude) stored in .env files, never hard-coded.
3. **Access Control:**
   - Onlyauthorized users can send queries.
   - Optional authentication tokens for API endpoints.

4. Audit Logging

All queries, responses, and session information are logged for monitoring and compliance.



7.2 Performance Aspects

1. Low Latency:

- Caching reduces repeated LLM/API calls.
- FastAPI + Uvicorn asynchronous processing ensures responsiveness.

2 . Scalability:

- MongoDB handles multiple sessions simultaneously.
- Stateless API design allows horizontal scaling if needed.

3 .Reliability:

- Offline AI fallback ensures system still responds if API quota is exceeded.
- Retrymechanisms for failed API/web scraping calls.

4. Resource Management:

- Session expiry prevents memory overload.
- Optional Redis caching for high-load scenarios.

8. References

1. LangChain. "LangChain — Framework for LLM-based Applications." [Online]. Available: https://www.langchain.com/docs

- Reference for building LLM pipelines and agents.

2. Kaggle. "Financial Complaint and Transaction Datasets." [Online]. Available: https://www.kaggle.com/

- Source of historical financial datasets for model training.

3. Tesseract OCR GitHub Repository. [Online]. Available: https://github.com/tesseract- ocr/tesseract

Open-source OCR engine documentation for extracting text from images/PDFs.

4. OpenAI. "GPT Models Documentation." [Online]. Available: https://platform.openai.com/docs

- Official documentation for LLM API usage, prompt design, and fine-tuning.