Project - High Level Design

on

Finance Support Triage Agent

Course Name: Agentic AI

Institution Name: Medicaps University – Datagami Skill Based Course

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

Sr no	Student Name	Enrolment Number
1	Jidnyas Bhushan Bhonge	EN22CS301469
2	Khushi Mehra	EN22CS301504
3	Krati Patidar	EN22CS301514
4	Kashish Sharma	EN22CS301492
5	Kamal Yadav	EN22CS301476

Group Name: 03D5

Project Number: AAI-03

Industry Mentor Name: Mr.Suraj Nayak

University Mentor Name: Prof.Maya Yadav

Academic Year: 2025–2026

Table of Contents

1. Introduction

The Finance Support Triage Agent is an AI-driven automation system designed to streamline finance-related email support workflows. The system leverages Large Language Models (LLMs) to automatically classify incoming finance emails, determine urgency and intent, extract relevant financial entities, and generate structured draft responses.

By reducing manual intervention in email triage and response preparation, the system improves operational efficiency, response time, and consistency in finance support processes. The solution is implemented using Python, FastAPI, LangChain, Llama 3.3 (Groq API), PostgreSQL, and Streamlit.

1.1 Scope of the Document

This High-Level Design (HLD) document provides an architectural overview of the Finance Support Triage Agent system.

The scope includes:

- Overall system architecture
- Core modules and components
- Process and information flow
- External integrations (Gmail, LLM API)
- Data design and storage approach
- Key design considerations and non-functional aspects

This document does not include detailed code-level implementation or low-level technical specifications.

1.2 Intended Audience

This document is intended for:

- Industry Mentor (Datagami)
- University Mentor
- Evaluation Panel

- Development Team Members

It provides stakeholders with a clear understanding of the system structure, technology choices, and architectural decisions.

1.3 System Overview

The Finance Support Triage Agent processes incoming finance-related emails and converts them into structured support tickets with AI-generated draft responses.

At a high level, the system consists of the following layers:

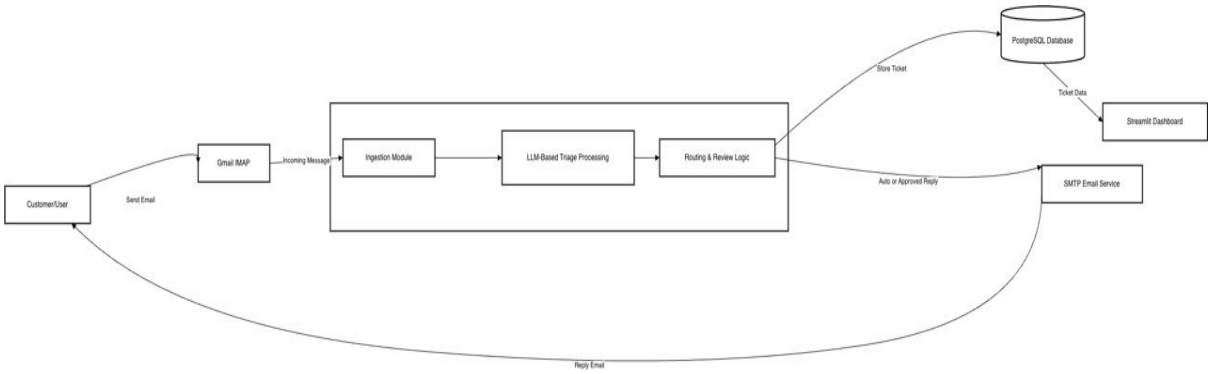
- **Email Integration Layer** – Retrieves emails using IMAP and sends approved responses via SMTP.
- **Backend Layer (FastAPI)** – Handles request processing and coordinates system operations.
- **AI Processing Layer (LangChain + LLM)** – Performs intent classification, priority detection, sentiment analysis, entity extraction, and draft generation.
- **Validation Layer (Pydantic)** – Ensures structured and consistent AI output.
- **Data Layer (PostgreSQL)** – Stores ticket details and processing results.
- **Presentation Layer (Streamlit Dashboard)** – Displays ticket information, extracted entities, and generated responses for review.

The system follows a modular layered architecture to ensure scalability, maintainability, and future extensibility.

2. System Design

The Finance Support Triage Agent follows a modular, layered architecture to ensure scalability, maintainability, and reliability. The system integrates email services, AI processing, backend APIs, database storage, and a dashboard interface into a cohesive workflow.

The architecture separates responsibilities across distinct layers to minimize coupling and enable future enhancements.



2.1 Application Design

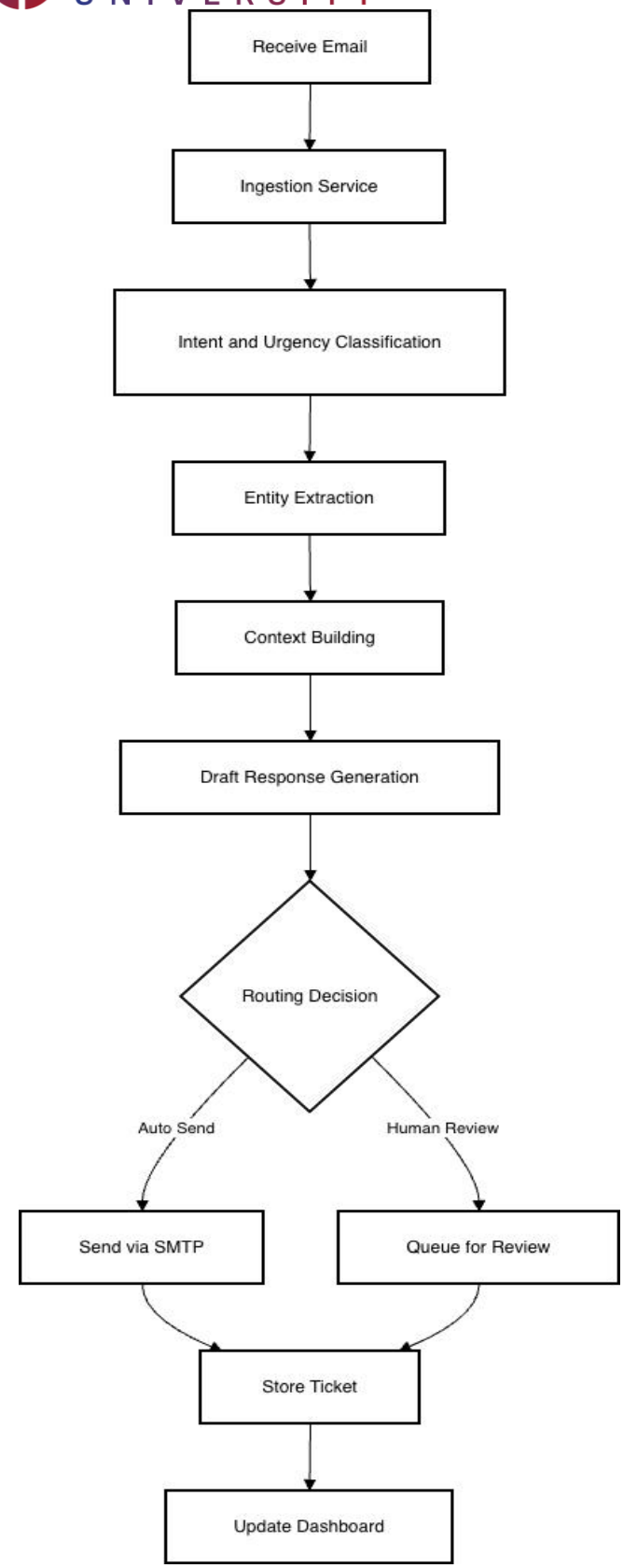
The application is designed using a layered architecture consisting of the following modules:

- **Email Integration Module**
  Handles email retrieval using Gmail IMAP and sends approved draft responses via SMTP.
- **Backend Processing Module (FastAPI)**
  Acts as the central orchestration layer. It receives email content, invokes AI processing, validates output, and manages database operations.
- **AI Processing Module (LangChain + LLM)**
  Uses structured prompts to interact with Llama 3.3 (Groq API).
  Performs:
  - Intent classification
  - Category detection
  - Priority assignment
  - Sentiment analysis
  - Named Entity Recognition (NER)
  - Draft response generation
- **Validation Module (Pydantic)**
  Ensures structured JSON output before database storage.
- **Data Storage Module (PostgreSQL + SQLAlchemy)**
  Stores ticket details, extracted entities, and generated responses.
- **Dashboard Module (Streamlit)**
  Displays ticket information and enables status tracking and response approval.

2.2 Process Flow

The system workflow is as follows:

1. A finance-related email is received.
2. The backend retrieves and processes the email content.
3. LangChain constructs a structured prompt.
4. The LLM processes the input and generates structured JSON output.
5. The output is validated using Pydantic schema.
6. The validated data is stored as a support ticket in PostgreSQL.
7. The Streamlit dashboard displays ticket details.
8. An approved draft response is sent via SMTP.

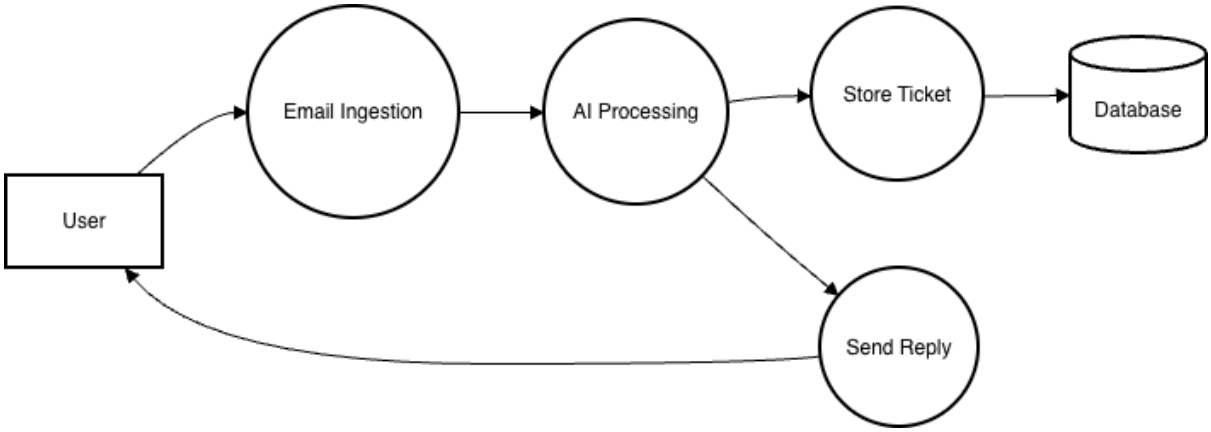

Receive Email
Ingestion Service
Intent and Urgency Classification
Entity Extraction
Context Building
Draft Response Generation
Routing Decision
Auto Send
Human Review
Send via SMTP
Queue for Review
Store Ticket
Update Dashboard

2.3 Information Flow

The information flow describes how data moves across system components:

Email Content
→ Backend API
→ AI Processing Layer
→ Validation Layer
→ Database Storage
→ Dashboard Interface
→ SMTP Response System

Each stage transforms or validates the data before passing it to the next component, ensuring reliability and consistency.



2.4 Components Design

The system consists of the following key components:

2.4.1 Email Service Component

Responsible for receiving and sending emails using IMAP and SMTP protocols.

2.4.2 AI Engine Component

Processes email content using LLM for classification, entity extraction, and draft generation.

2.4.3 Backend API Component

Handles request routing, processing coordination, and database interaction.

2.4.4 Database Component

Stores structured ticket data including intent, category, priority, extracted entities, and draft reply.

2.4.5 Dashboard Component

Provides visualization of tickets, extracted information, and workflow status.

2.5 Key Design Considerations

The following design principles were considered:

- **Structured Output Control**
  LangChain structured prompts reduce hallucination and enforce JSON formatting.
- **Data Validation**
  Pydantic ensures reliable and consistent data before storage.
- **Modularity**
  Layered architecture allows independent scaling and maintenance.
- **Scalability**
  Cloud-based PostgreSQL supports growing ticket volumes.
- **Security**
  API keys and credentials are managed using environment variables.
- **Extensibility**
  The system can be extended to integrate CRM, ERP, or analytics modules.

2.6 API Catalogue

The backend exposes the following APIs:

Endpoint	Method	Description
/	GET	Health check endpoint to verify that the API is running
/analyze	POST	Analyses email text and returns structured triage data (intent, sentiment, entities, priority, category, summary)
/process_ticket	POST	End-to-end processing: analyse email, generate draft response, save ticket to database
/process_ticket_image	POST	Processes uploaded email image using OCR, then analyses and saves ticket
/tickets	GET	Retrieves all stored tickets (optional status filtering)
/tickets/{ticket_id}	GET	Retrieves a single ticket by ID
/tickets/{ticket_id}/approve	PATCH	Marks ticket as "In Progress"

/approve_ticket/{ticket_id}	POST	Sends draft response via email and closes the ticket
/tickets/{ticket_id}/reject	PATCH	Rejects draft and closes the ticket
/fetch_emails	POST	Connects to Gmail via IMAP, fetches emails, processes them, and saves tickets

These APIs enable structured interaction between the backend, AI processing layer, and dashboard interface.

3. Data Design

The Finance Support Triage Agent uses a structured relational database to store and manage processed finance support tickets. PostgreSQL is used as the primary database to ensure reliability, scalability, and structured data handling.

The database schema is designed to support AI-generated outputs including classification, sentiment detection, entity extraction, and draft response generation.

ENUM types are used to enforce controlled values for ticket status, priority, and category, ensuring data consistency.

Primary Table: tickets

Field Name	Description
id	Unique ticket identifier (UUID – Primary Key)
customer_name	Name of the customer
email_body	Original email content
status	Ticket lifecycle state (New, In Progress, Resolved, Closed)
priority	Urgency level (High, Medium, Low)
category	Issue classification (Fraud, Payment Issue, General)
sentiment	Detected sentiment of email
intent	Identified user intent
summary	AI-generated concise issue summary
transaction_id	Extracted transaction reference
amount	Extracted transaction amount
draft_response	AI-generated reply
created_at	Timestamp of ticket creation

Design Characteristics

- UUID is used as the primary key for uniqueness and scalability.
- ENUM types enforce controlled values for status, priority, and category.
- Indexes are created on frequently queried fields (status, priority, category, created_at) to improve performance.

3.2 Data Access Mechanism

The system uses **SQLAlchemy ORM** to interact with the PostgreSQL database. This ensures abstraction from raw SQL queries and improves maintainability.

The data access mechanism supports:

- Inserting new tickets after AI processing and validation
- Retrieving tickets for dashboard display
- Updating ticket status (e.g., New → In Progress → Resolved)
- Filtering tickets by priority, category, or status

Indexed columns (status, priority, category, created_at) optimize query performance for dashboard filtering and sorting operations.

Database credentials are managed securely using environment variables.

3.3 Data Retention Policies

The system stores support tickets for operational tracking, monitoring, and audit purposes.

Retention considerations include:

- Each ticket includes a timestamp (created_at) for traceability.
- Sensitive financial details are stored securely within controlled access environments.
- Future enhancements may include configurable retention periods or archival mechanisms.

3.4 Data Migration

The system is designed to support future scalability and migration requirements.

Key considerations include:

- Use of PostgreSQL ensures compatibility with enterprise and cloud environments.
- Schema-based design enables version control and incremental upgrades.
- UUID-based primary keys prevent conflicts during data migration.
- Backup and export mechanisms can be implemented for disaster recovery and replication.

The modular architecture ensures minimal impact on application logic during database upgrades or migration to other managed database services.

4. Interfaces

The Finance Support Triage Agent interacts with multiple external and internal interfaces to enable automated email processing, AI-based classification, and ticket management.

The system integrates with email services, AI APIs, database systems, and a user-facing dashboard interface.

4.1 External Interfaces

4.1.1 Gmail IMAP Interface

The system uses the IMAP protocol to retrieve incoming finance-related emails.
This interface allows:

- Reading email content
- Extracting sender details
- Triggering processing workflow

IMAP integration ensures real-time ingestion of customer queries into the backend system.

4.1.2 SMTP Interface

SMTP is used to send approved draft responses back to customers.

This interface supports:

- Sending AI-generated draft responses

- Confirming communication with the customer
- Completing the support workflow

4.1.3 LLM API Interface (Groq API – Llama 3.3 70B)

The system integrates with the Groq API to access the Llama 3.3 Large Language Model.

This interface is responsible for:

- Intent classification
- Priority detection
- Sentiment analysis
- Named Entity Recognition
- Draft response generation

LangChain is used to structure prompts and enforce JSON-formatted responses from the LLM.

4.1.4 Database Interface (PostgreSQL)

The backend interacts with the PostgreSQL database using SQLAlchemy ORM.

This interface supports:

- Inserting processed tickets
- Updating ticket status
- Retrieving tickets for dashboard display
- Filtering and sorting operations

4.2 Internal Interfaces

4.2.1 Backend–AI Interface

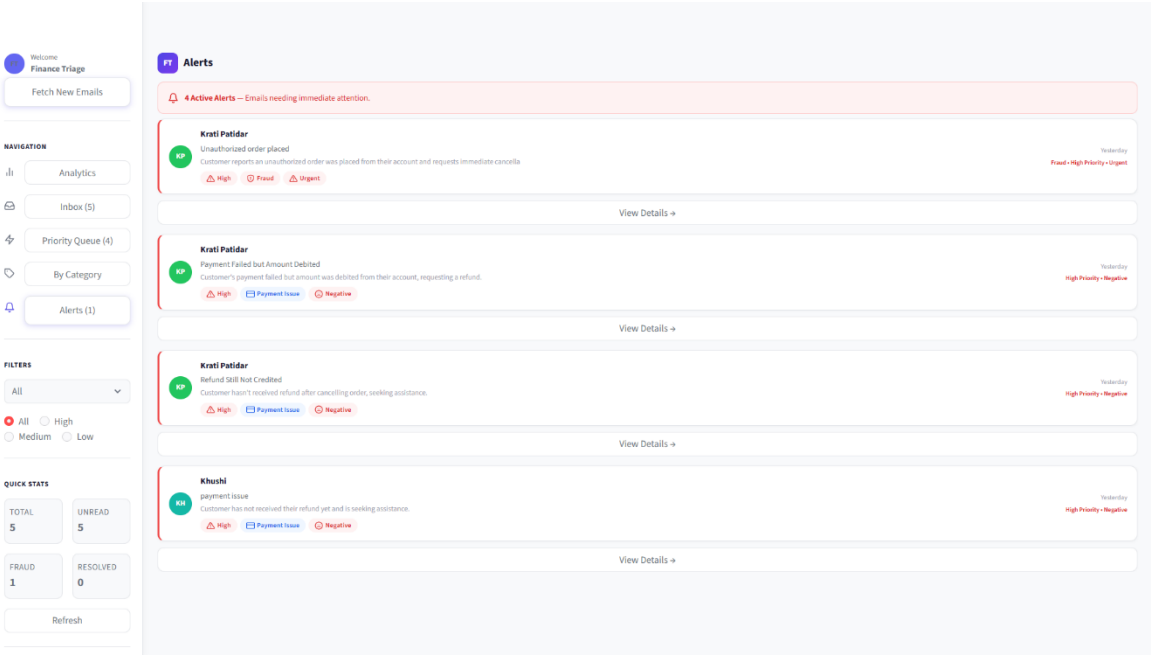
The FastAPI backend communicates with the AI processing layer via structured prompts.
This ensures controlled input and validated output between application logic and LLM processing.

4.2.2 Backend–Dashboard Interface

The Streamlit dashboard interacts with the backend APIs to:

- Display ticket details

- Show classification results
- Allow status updates
- Trigger response sending



5. State and Session Management

The Finance Support Triage Agent follows a stateless backend architecture using FastAPI, where each request is processed independently without maintaining server-side session data. All processed information, including ticket status, classification results, extracted entities, and draft responses, is stored in the PostgreSQL database to ensure persistence and reliability.

Ticket lifecycle management is handled through the status field in the database, allowing transitions such as New, In Progress, Resolved, and Closed. This ensures workflow continuity without relying on in-memory session storage.

The Streamlit dashboard uses session state only for temporary user interactions such as viewing selected tickets or applying filters. Long-term data persistence is handled entirely at the database level.

This design improves scalability, maintainability, and fault tolerance by separating application state from persistent data storage.

6. Caching

The current implementation of the Finance Support Triage Agent does not use an explicit caching mechanism. All email processing requests are handled in real time, and results are directly stored in the PostgreSQL database.

Since each incoming email represents a unique support request, response-level caching is not mandatory in the current design. The system ensures efficiency through optimized database indexing on frequently queried fields such as status, priority, category, and created_at.

However, caching can be introduced in future enhancements to improve performance and reduce API usage costs. Potential caching strategies include:

- Caching frequently accessed dashboard queries
- Caching repeated AI responses where applicable
- Using in-memory caching systems such as Redis

The system architecture is modular and allows easy integration of caching mechanisms if scalability requirements increase.

7. Non-Functional Requirements

The Finance Support Triage Agent is designed to ensure reliability, security, scalability, and performance in handling finance-related email workflows.

The key non-functional aspects are outlined below.

7.1 Security Aspects

The system handles financial communication and therefore implements the following security measures:

- **Secure Credential Management** – API keys and email credentials are stored using environment variables.
- **Database Protection** – PostgreSQL access is controlled through authenticated connections.

- **Input Validation** – Pydantic ensures validated and structured data before storage.
- **Encrypted Email Communication** – SMTP uses TLS for secure email transmission.
- **Controlled AI Output** – Structured prompts reduce unpredictable model responses.

These measures ensure confidentiality, integrity, and secure system behavior.

7.2 Performance Aspects

The system is optimized for efficient processing and scalability.

- **Indexed Database Columns** improve dashboard filtering and retrieval speed.
- **Stateless Backend Design** enhances scalability and reduces memory overhead.
- **Single LLM Call Processing** minimizes response latency and API cost.
- **Modular Architecture** allows future integration of caching or load balancing mechanisms.

These design choices ensure smooth operation under increasing ticket volumes.

8. References

The following resources and documentation were referred to during the design and implementation of the Finance Support Triage Agent:

1. FastAPI Documentation – https://fastapi.tiangolo.com/
2. LangChain Documentation – https://python.langchain.com/
3. Groq API Documentation – https://groq.com/
4. PostgreSQL Documentation – https://www.postgresql.org/docs/
5. SQLAlchemy Documentation – https://docs.sqlalchemy.org/
6. Streamlit Documentation – https://docs.streamlit.io/
7. Pydantic Documentation – https://docs.pydantic.dev/
8. Gmail IMAP and SMTP Protocol Documentation