Hackathon Email Assistant – Detailed Documentation

1. Introduction

Emails are one of the most widely used forms of communication. However, with the increasing volume of emails, it becomes challenging to manage, prioritize, and respond efficiently.

The Hackathon Email Assistant was built to:

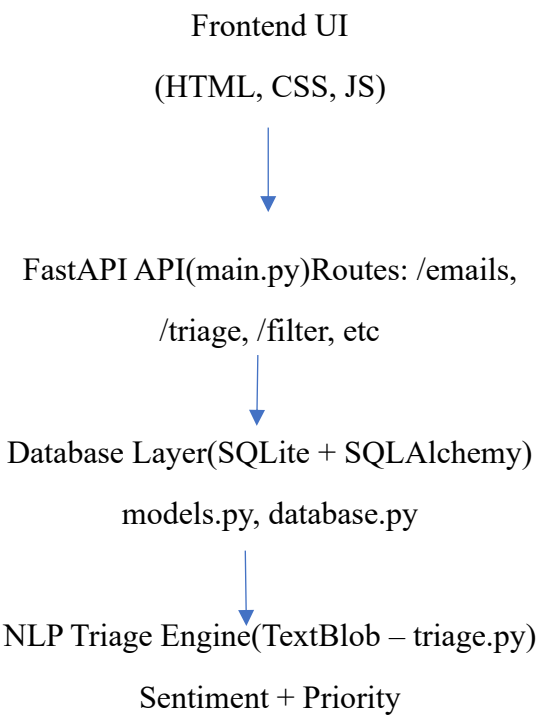
- Automatically analyze emails
- Detect urgency and sentiment
- Prioritize them in a structured way
- Provide a simple frontend dashboard for users to view and filter messages

This system provides end-to-end management of emails using FastAPI (Backend), SQLite (Database), TextBlob (NLP), and HTML/JS (Frontend).

2. Objectives

1. Build a backend service for email storage and retrieval.
2. Use NLP to triage (classify) emails based on sentiment and urgency.
3. Provide a database layer to persist emails.
4. Develop a frontend interface to visualize, filter, and manage emails.
5. Keep the system extendable for real-world integration (Gmail, Outlook, Auto-replies).

3. System Architecture



4. File-by-File Description

1. main.py – Core Backend Service

- Creates a FastAPI app.
- Defines routes (endpoints):
  - / → Health check
  - /emails/dummy → Returns dummy hardcoded emails
  - /emails → Returns real emails from DB
  - /triage → Applies NLP triage to emails
  - /emails/filter → Filters by priority/sentiment
- Mounts frontend (/static) so index.html works.
- Connects backend with database and models.

2. models.py – Database Schema

Defines the structure of the Email table in SQLite:

- id → Unique ID
- sender → Email sender
- subject → Subject line
- priority → Urgent / Not urgent
- sentiment → Positive / Neutral / Negative
- status → Pending / Replied / Ignored

This ensures consistency in storing email data.

3. database.py – Database & Dummy Data

- Provides helper functions for:
  - Fetching dummy emails (to test UI quickly)
  - Querying real emails stored in emails.db
- Acts as the data access layer (keeps DB logic separate from business logic).

4. triage.py – NLP Triage Engine

- Uses TextBlob for Natural Language Processing.
- For each email:
  - Sentiment Analysis → Positive, Neutral, or Negative
  - Urgency Check → If email contains urgent words (e.g., *immediately, ASAP*), it's marked as Urgent.

- Returns structured data:


```
{
 "id": 1,
 "sender": "boss@example.com",
 "subject": "Project deadline today",
 "priority": "urgent",
 "sentiment": "negative"
}
```


5. reply.py – (Optional Auto-Reply Module)

- Generates suggested responses for emails.
- Example: If an email is urgent and negative, it can suggest:

“Acknowledged. We will address this issue immediately.”

This makes the project extensible for real-world scenarios.

6. static/index.html – Frontend Dashboard

- Simple HTML + JS interface.
- Fetches data from backend APIs (/emails, /triage).
- Displays emails in a table.
- Provides filters for priority (urgent / not urgent) and sentiment (positive / negative / neutral).

This makes the tool user-friendly and easy to interact with.

7. emails.db – SQLite Database

- Stores email data persistently.
- Allows you to restart the app without losing records.
- Works with SQLAlchemy ORM for easier queries.

5. Features Implemented

1) Email Retrieval – Dummy & real DB emails.
2) NLP Triage – Sentiment + Urgency detection.
3) Database Storage – SQLite with SQLAlchemy ORM.
4) Frontend Dashboard – Web UI to view emails.
5) Filtering – By sentiment & priority.
6) Auto-Reload Server – uvicorn --reload detects changes instantly.

6. Example Workflows

1)Viewing Emails

1. User opens http://127.0.0.1:8000/static/index.html
2. UI fetches /emails → Displays email list

2) Running Triage

1. User clicks “Triage” button
2. UI calls /triage
3. Emails now show priority & sentiment

3) Filtering

1. User selects priority=urgent and sentiment=negative
2. UI calls /emails/filter with query parameters
3. Only urgent + negative emails are shown

7. Future Enhancements

1)Email Integration – Connect to Gmail/Outlook via IMAP/SMTP.
2)Smarter NLP – Use HuggingFace / spaCy for better sentiment analysis.
3)Auto-Reply System – Integrated with Gmail API.
4)Analytics Dashboard – Trends of sentiment over time.
5) Authentication – Multi-user login and permissions.

8. Why This Approach Works

- FastAPI → Lightweight, high-performance API.
- SQLite + SQLAlchemy → Simple but powerful database.
- TextBlob → Quick NLP for hackathon use.
- Frontend in HTML/JS → Easy to set up, no heavy frameworks.
- Separation of Concerns → Each file has a dedicated responsibility.

9. Conclusion

The Hackathon Email Assistant is a fully working prototype that automates email management using NLP + Database + FastAPI + Frontend UI.

It solves the problem of overwhelming email inboxes by:

- Detecting urgency,
- Classifying sentiment,
- Prioritizing communication,
- Giving users a simple way to filter & manage their inbox.

This project is ready for demo and can be extended into a real-world system with minor additions like Gmail/Outlook integration.