**Email Classification & Key Number Extraction System**

This document outlines the overall application design, including system components, processing flow, and how data is handled from the initial email upload to final UI presentation.

**1. System Overview**

The system is designed to:

* **Receive and parse emails** (including attachments) uploaded via a frontend UI.
* **Extract key information** such as request types and important numerical values (e.g., loan amounts, customer IDs) using a Gemini LLM.
* **Apply customizable priority rules** defined in a JSON configuration file to guide the extraction process.
* **Return structured responses** that are then processed on the UI to display human-readable information.

**2. Architecture Components**

**Frontend (UI)**

* **File Upload:**  
  Users upload email files (or a directory of emails) through a web interface.
* **Response Processing:**  
  The UI receives JSON responses and converts fields like all\_extracted\_numbers into a comma-separated string for display.

**Backend (FastAPI)**

* **Email Processing Endpoint(s):**  
  Receives email files and processes them.
* **Email Parsing Module:**  
  Extracts components such as the email body, email chain text, and attachments.
* **Priority Rules Loader:**  
  Loads a configurable JSON file containing rules for:
  + Prioritizing extraction sources (e.g., email content versus attachments).
  + A list of standard banking numeric keys.
* **Extraction & Classification Module:**  
  Uses Gemini LLM to extract:
  + The primary request type.
  + Key numerical values from attachments based on the provided priority rules.
* **Duplicate Checker:**  
  Checks for duplicate emails based on content.
* **Response Serialization:**  
  Constructs a response using a Pydantic model (e.g., EmailData) that includes fields such as:
  + sender
  + subject
  + request\_type
  + sub\_request\_type
  + confidence\_score
  + duplicate\_flag
  + all\_extracted\_numbers

**External Services**

* **Gemini LLM:**  
  Receives a constructed prompt (with context and banking keys) and returns a JSON array containing key-value pairs for the important numeric fields.

**3. Detailed Data Flow**

**Step 1. Email Upload and Parsing**

1. **Upload:**
   * The user uploads one or more email files via the UI.
2. **Parsing:**
   * The backend receives the files and uses functions such as parse\_email\_bytes to extract:
     + **Email Body:** The main content of the email.
     + **Email Chain:** Text from email threads.
     + **Attachments:** Files attached to the email.
3. **Duplicate Check:**
   * The system checks for duplicate emails to avoid redundant processing.

**Step 2. Priority-Based Extraction of Key Numbers**

1. **Load Priority Rules:**
   * The system loads a JSON file (location and filename specified in the environment variables) that defines:
     + Whether to prioritize email content over attachments for request type identification.
     + A list of banking numeric keys (e.g., loan\_amount, customer\_id, account\_number).
2. **LLM Extraction:**
   * For each attachment:
     + The text is extracted via extract\_text\_from\_attachment.
     + A prompt is constructed using the loaded banking keys.
     + The Gemini LLM is called via extract\_key\_number\_with\_llm with the prompt.
     + The LLM returns a JSON array (or a JSON string) containing objects with key-value pairs.
3. **Combine Extraction Results:**
   * The backend loops over all attachments, parsing and combining the results into a single array.
   * The final array is converted back into a JSON string or stored as a structured array in the response.

**Step 3. Response Construction and Serialization**

* **Pydantic Model:**  
  Each processed email is stored in an EmailData model, which includes:
  + Standard email fields (sender, subject, etc.)
  + The extracted key numbers in the field all\_extracted\_numbers
* **Return:**  
  The backend returns a JSON array of these objects to the UI.

**Step 4. UI Response Processing and Presentation**

1. **Receive Response:**
   * The frontend (e.g., an Angular application) receives the response.
2. **Process Numeric Data:**
   * If the all\_extracted\_numbers field is a JSON string, the UI parses it into an array.
   * The UI maps over the array to convert each object into a "key: value" string.
   * The strings are then joined into a single comma-separated string for display.
3. **Display:**
   * The final human-readable data is shown on the UI.

**4. Visual Diagram**

Below is a simplified Mermaid sequence diagram that visualizes the flow:

mermaid

CopyEdit

sequenceDiagram

participant UI as Frontend UI

participant FE as FastAPI Endpoint

participant PP as Email Parsing & Processing

participant EX as Extraction Module (Gemini LLM)

participant RR as Priority Rules Loader

participant DB as Duplicate Checker

UI->>FE: Upload Email File(s)

FE->>PP: Parse Email (Body, Attachments, Chain)

PP->>DB: Check for Duplicates

PP->>EX: For each attachment, call extract\_key\_number\_with\_llm()

EX->>RR: Load Priority Rules (banking numeric keys, etc.)

RR-->>EX: Return Rules JSON

EX->>Gemini: Send prompt with context & banking keys

Gemini-->>EX: Return JSON array of key-value pairs

EX-->>PP: Return extracted numbers

PP->>FE: Construct EmailData object with all\_extracted\_numbers

FE-->>UI: Return JSON response (array of EmailData)

UI->>UI: Parse and format all\_extracted\_numbers for display

**5. Sample Priority Rules JSON File**

This is a sample JSON file (e.g., Priority\_Extraction\_Rules.json) that defines the rules for numerical extraction:

json

CopyEdit

{

"priority\_rules": {

"is\_prioritization\_extraction": true,

"request\_type\_identification": {

"order": ["email\_content", "document\_content"],

"fallback": "document\_content"

},

"numerical\_field\_extraction": {

"preferred\_source": ["attachments"],

"fallback": "email\_body"

},

"banking\_numeric\_keys": [

"loan\_amount",

"customer\_id",

"account\_number",

"transaction\_amount",

"balance",

"interest\_rate",

"credit\_limit",

"repayment\_amount",

"mortgage\_amount",

"fees",

"exchange\_rate",

"deposit\_amount",

"withdrawal\_amount",

"savings\_balance",

"overdraft\_limit",

"annual\_income",

"net\_worth",

"investment\_value",

"processing\_fee",

"penalty\_fee",

"service\_charge",

"minimum\_balance",

"maximum\_withdrawal",

"credit\_score",

"loan\_term\_years",

"monthly\_interest",

"annual\_percentage\_rate",

"reward\_points"

]

}

}

**6. Summary**

* **Backend Processing:**  
  Parses email files, loads configurable priority rules, and uses Gemini LLM for extracting key numeric values.
* **Response Serialization:**  
  Constructs structured responses (via Pydantic models) that include fields like all\_extracted\_numbers.
* **Frontend Processing:**  
  Parses and formats the JSON array (or string) of extracted numbers into a human-readable comma-separated string for display.
* **Configurable Rules:**  
  Priority rules (including banking numeric keys) are loaded from an external JSON file, allowing dynamic updates via an API endpoint.

This design flow provides a modular and maintainable approach to handle email processing and extraction of key numerical data in a banking context, ensuring that both backend logic and frontend presentation are aligned.