THE MAIL GPS

Send mails to the right person

Thibault Goutorbe

thibault.goutorbe@aivancity.education

—

1. Project Preparation and Configuration

1.1 Choose the necessary tools and libraries:

- Choose programming languages (Python, Node.js, etc.)
- Identify libraries for email manipulation and sending (e.g., `smtplib`, `email`, `imaplib`, `flask`, etc.)
- Install dependencies (e.g., `pip install smtplib`, `imaplib`, `requests`, etc.)

1.2 Set up the development environment:

- Set up a virtual environment to isolate dependencies (e.g., `virtualenv`, `venv`)
- Manage sensitive configurations (e.g., environment variables for mail server login credentials)
- Create a clear project structure with folders for tests, configuration, and main code files.

—

2. Implementation of Automatic Email Sending

- 2.1 Set up email sending scripts

- Use the SMTP library (e.g., `smtplib` for Python) to configure email sending.
- Structure the message using the `email` library (create MIME objects for attachments, email body, etc.)

- 2.2 Add email customization features:

- Create email templates based on request types:
- Email for school program requests

- Email for schedule requests
- Other common requests
- Use variables to personalize emails based on content (e.g., insert the recipient's name or specific information).

—

3. Connecting to the Mistral API (Experimental Version)

- 3.1 Study the Mistral API:

- Read the API documentation to understand available endpoints, request, and response formats.

3.2 Implement connection to the Mistral API:

- Use an HTTP library to make API calls (e.g., `requests` for Python).
- Create a function that interacts with the API to send data from received emails.
- Handle API errors and implement retry mechanisms if necessary.

—

4. Incoming Mail Handling and Text Extraction

4.1 Configure IMAP to receive emails:

- Use the `imaplib` library to configure receiving emails from an inbox (or a specific folder).
- Set up secure access (SSL/TLS) to ensure the connection is protected.

4.2 Extract content from emails:

- Extract text and attachments from emails.
- Use libraries like `email.parser` to parse the email body.
- Filter relevant emails (those related to program, schedule, or other inquiries).
- Store the extracted email text in a local database or file for future analysis.

—

5. Sending Requests to the API and Retrieving Responses

5.1 Send extracted information to the Mistral API:

- After extracting the relevant data from the email, send this data to the Mistral API through HTTP POST/GET requests.
- Ensure the email's body and subject are passed correctly to the API for processing.
- Implement error handling to retry requests in case of failures.

5.2 Process the API's responses to determine the email's topic:

- Receive and parse the API's responses, which will contain the detected topic or classification of the email (e.g., "School Schedule Inquiry," "Program Information Request," etc.).
- Store or log the response for further action.

—

6. Email Routing Based on the Topic

6.1 Define the routing logic:

- Based on the topic identified by the API, establish rules for routing the email to the correct department or person in the school.
- For example:
  - Emails about schedules → Send to the administrative office.
  - Emails about programs → Forward to the academic office.
  - General inquiries → Send to the school’s front desk.

6.2 Implement the routing mechanism:

- Create a function that, based on the topic detected, forwards the email to the appropriate email address within the school.
- Use the same email-sending scripts developed in **2.1** to re-send the original email to the correct recipient.
- Add a "forwarded by" note in the email or adjust the email headers to indicate that the email was automatically routed.

—

7. Testing and Validation

7.1 Create test emails:

- Simulate different email subjects and bodies representing common inquiries (e.g., schedule requests, program inquiries, etc.).
- Ensure your system can successfully extract the text, send it to the API, and route the email to the correct department based on the response.

7.2 Unit testing and continuous integration:

- Test individual modules for text extraction, API connection, and email forwarding.
- Implement tests to ensure the email is routed to the correct recipient based on various topics.
- Use a CI/CD pipeline to automate testing for every code update (e.g., with GitLab or GitHub Actions).

—

8. Optimization and Security

8.1 Secure sensitive information

- Encrypt sensitive information such as email credentials and API keys (using `dotenv` or environment variable management).
- Ensure that email routing follows secure practices to avoid data breaches (e.g., use TLS for email forwarding).

8.2 Improve request management:

- If multiple emails are processed simultaneously, implement asynchronous handling (e.g., using Celery or RabbitMQ) to ensure efficient processing and routing.
- Add a priority system in case some email types (e.g., urgent inquiries) need to be processed and routed faster.

8.3 Activity tracking and logging:

- Implement logging to track email processing, API requests, and routing decisions.
- Create a dashboard or reports that show email traffic, topics detected, and routing history.
- Set up alerts for any errors in email routing (e.g., if an email fails to be forwarded or if the API returns an error).

—

9. Documentation and Delivery

9.1 Document the project:

- Document each module with clear comments, especially around the routing rules and interaction with the API.
- Write a user guide explaining how the system detects topics, routes emails, and how it can be configured for different departments in the school.

9.2 Prepare for deployment:

- Create deployment scripts (e.g., Docker or server configurations) for the project.
- Test the system in a simulated production environment with real email flows.
- Ensure that the deployment environment is secured and tested for performance.