SPAM SHIELD - Low Level Design - (CH21B009)

Introduction:

The **Spam Shield** application consists of **Frontend** and **Backend** components interacting with Gmail via IMAP, and uses a **Naive Bayes model** for spam prediction. This LLD includes component design, functions/methods, data flow, error handling, and logging details.

---

1. Frontend (Streamlit)

1.1 Structure

- **File**: *frontend.py*
- **Libraries**: *streamlit, requests, pandas, email*
- **User Interactions**:
    - **Email Input**: Can paste email content.
    - **CSV Upload**: Can Upload a CSV file containing emails.
    - **Gmail Integration**: Can fetch unread emails using IMAP.
    - **Prediction Display**: Shows spam or ham predictions.
    - **User Feedback**: Allows users to provide feedback for wrong predictions.
    - **Retrain Model**: Triggers backend retraining after collecting feedback.

1.2 Streamlit UI Components:

Input Fields:

- *st.text_area('Email')*: User pastes email content.
- *st.file_uploader('Upload CSV', type='csv'*): User uploads a CSV of emails.
- *st.button('Check for Spam'):* When clicked, the emails are sent to the backend for prediction.

Feedback Mechanism:

- *st.radio('Spam or Not Spam?', options=['Spam', 'Not Spam']):* For feedback on individual email predictions.
- *st.button('Submit Feedback')*: Submits feedback to backend for retraining.

Display Results:

- *st.table(prediction_results)*: Displays predictions for the uploaded CSV or pasted email.

1.3 Data Flow:

Email Paste or CSV Upload:

- If the user pastes an email, the frontend sends the email content to the backend's /predict API.
- If a CSV is uploaded, the frontend sends the file content (list of emails) to the backend.

Fetching Gmail Emails:

- The frontend calls the backend /get_gmail_emails API, passing the Gmail login credentials.

- The backend fetches unread emails using IMAP and returns them to the frontend.

Feedback Submission:

- When feedback is provided, new data points are created and are sent to the backend /train API for model retraining.

1.4 Functions used:

- get_emails() - taken from extract_email.py
- move_to_spam() - taken from extract_email.py

---

2. Backend (FastAPI)

2.1 Structure

- **File**: *backend_api.py*

- **Libraries**: *fastapi, pydantic, logging, email, sklearn*

- **User Interaction**:

  - **/predict (POST)**: Receives email content for spam prediction.

  - **/train (POST)**: Accepts feedback data and retrains the model.

2.2 API Endpoints:

2.2.1 /predict (POST):

URL: 0.0.0.0/predict

Method: POST

Parameters:

- emails: List[str]: List of email contents to predict.

Response:

- status: 'success' | 'error'
- predictions: List[str]: List of predicted labels (spam/ham).

Functions used:

(i). prediction() - taken from predict.py

- **Purpose**: Classify a list of emails as spam or ham.
- **Parameters**: emails: List[str]: List of email content to predict.
  Returns: List of predicted labels ('spam', 'ham').

Functionalities:

- Predicts using the Naive Bayes model.
- Return the predictions.

2.2.2 /train (POST):

URL: 0.0.0.0/train

Method: POST

Parameters:

- feedback: Dict[List]: Contains two lists of feedback where each list contains:
- emails: str
- labels: str ('spam' or 'ham')

Response:

- status: 'success' | 'error'

Functions used:

(i). retrain() - taken from retrain_model.py

- **Purpose**: Retrain the Naive Bayes model using new feedback data.
- **Parameters**: feedback: List[Dict]: List of feedback items, each containing:
  email: str: Email content.
  label: str: Label ('spam' or 'ham').

Functionalities:

- Append new feedback to the training dataset.
- Retrain the Naive Bayes model.
- Save the model to a file (naive_bayes_model.pkl).
- Log the retraining process.

---

3. Machine Learning Model

3.1 Model Structure:

- **Libraries**: *sklearn*
- **Model**: Naive Bayes classifier (MultinomialNB)
- **Model Files**:
  - ***naive_bayes_model.pkl:*** Trained model.
  - ***count_vectorizer.pkl:*** Fitted CountVectorizer.

---

4. Gmail Integration (IMAP Client)

4.1 IMAP Client Structure:

- **File**: extract_email.py
- **Libraries**: imaplib, email, logging

4.2 Functions:

(i). get_emails(mail_id: str, password: str):

Purpose: Fetch unread emails from Gmail using IMAP.

Parameters:

- mail_id: str: Gmail email address.
- password: str: Gmail password.

Returns: Tuple (mail_ids, bodies)

- mail_ids: List[str]: List of email IDs (UIDs).
- bodies: List[str]: List of email bodies.

Process:

- Establish IMAP connection.
- Search for unread emails.
- Parse emails and extract content and dates.
- Return email IDs and bodies.

(ii). move_to_spam(mail_id_list: List[str]):

Purpose: Move emails to the Gmail Spam folder.

Parameters: mail_id_list: List[str]: List of email UIDs to be moved.

Returns: None.

Process:

- Use the IMAP COPY command to move emails to the Spam folder.
- Mark the emails for deletion and expunge them.

5. Docker Structure:

5.1 Frontend:

- Created from python:3.10 image
- Simply runs frontend.py

5.2 Backend:

- Created from python:3.10 image
- Simply runs backend_api.py

5.3 cAdvisor (Container Monitoring):

- cAdvisor helps you monitor the resource usage (CPU, memory, disk I/O, network) of your containers in real-time.
- Exposes a UI at http://localhost:8080 where you can see the metrics for your containers.
- Volumes are mounted to capture system-level metrics.

5.4 Prometheus (Metrics Collection):

- Prometheus collects and stores metrics, including data from cAdvisor.
- Exposes a UI at http://localhost:9090 to query and visualize the metrics.
- The configuration for Prometheus is specified through the prometheus.yml file, which should be in the same directory as your Docker Compose file.

5.5 Grafana (Metrics Visualization):

- Grafana integrates with Prometheus and displays the collected metrics in visually appealing dashboards.
- Exposes a UI at http://localhost:3001 where you can access predefined dashboards or create your own.
- The default Grafana admin password is set to admin.

5.6 Access the services:

- Frontend (Streamlit): http://localhost:8501

- Backend API: http://localhost:6000

- cAdvisor (Monitoring): http://localhost:8080

- Prometheus (Metrics Collection): http://localhost:9090

- Grafana (Metrics Visualization): http://localhost:3001

---

6. Error Handling and Logging

Error Handling:

- **Frontend**: Displays user-friendly error messages for failed API requests or invalid inputs (e.g., file format, empty email content).
- **Backend**: Returns HTTP error codes (e.g., 400 Bad Request, 500 Internal Server Error) for invalid requests.
  Model Retraining: Ensures feedback data is valid (correct format, labels).

Logging:

- Logs are saved to both the console and a file.
- Important events like email fetching, predictions, feedback submission, and model retraining are logged at appropriate levels (INFO, WARNING, ERROR).

Conclusion

This detailed LLD provides a comprehensive breakdown of each component in the Spam Shield application. It covers the frontend UI elements, backend APIs, the machine learning model's structure and flow, the Gmail integration, error handling and logging. Each function and interaction is clearly defined, ensuring a smooth flow of data between components and robust error handling.