Gmail Automation Project Setup & Execution Report

Step 1: Create a Google Cloud Project and Enable Gmail API

1. Go to the **Google Cloud Console**: https://console.cloud.google.com/
2. Click on **"Select Project"** → **"New Project"**.
3. Give your project a name, and click **"Create"**.
4. Once created, go to the left-hand sidebar, click **"APIs & Services"** > **"Library"**.
5. Search for **Gmail API**, select it, and click **"Enable"**.

Step 2: Set Up OAuth 2.0 Consent Screen and Credentials

1. In the **Cloud Console**, go to **"APIs & Services"** > **"OAuth consent screen"**.
2. Select **"External"** user type and click **"Create"**.
3. Fill in app name, user support email, and developer contact info.
4. Skip scopes for now (or add if needed), and complete the process.
5. Then go to **"Credentials"** > **"Create Credentials"** > **"OAuth client ID"**.
   - Choose **"Desktop App"** as application type.
   - Click **"Create"**.
   - Click **"Download"** to save the file `credentials.json`.

Important: Save this file securely in your project root or preferred location.
Do not commit this file to GitHub.

Step 3: Clone the GitHub Repository & Install Dependencies

3.1 Clone the GitHub repository:

```
git clone https://github.com/Aswinraj040/Happyfox_assignment
cd <your-project-folder>
```

Step 3A: Set Up a Python Virtual Environment

Using a virtual environment isolates your project's dependencies from the global Python installation — preventing version conflicts and ensuring reproducibility.

3A.1 Create the Virtual Environment

In the root directory of your project, run:

```
python -m venv venv
```

- This will create a folder named `venv/` which contains the isolated Python environment.

3A.2 Activate the Virtual Environment

Run the appropriate command for your OS:

- **On macOS/Linux**:

  ```
  source venv/bin/activate
  ```

- **On Windows (CMD)**:

  ```
  venv\Scripts\activate
  ```

- **On Windows (PowerShell)**:

  ```
  .\venv\Scripts\Activate.ps1
  ```

3A.3 Install Project Dependencies

After activation, install the requirements:

```
pip install -r requirements.txt
```

Folder Structure

```
<Folder Name>/
 ├── src/
|   ├── fetch_emails.py
|   ├── process_emails.py
|
 ├── tests/
|   ├── unit_tests.py
|   └── integration_tests.py
|
 ├── .env
 ├── .gitignore
 ├── credentials.json
 ├── token.json
 ├── EmailDatabase.db
 ├── requirements.txt
```

Note: When you run the fetch_emails.py script the EmailDatabase.db gets created automatically.

By default 50 emails are fetched from your inbox. If you want to increase or decrease the count change it in the .env

Step 4: Fetch Emails Using Gmail API

Once the project is configured:

1. Make sure `credentials.json` is in the correct path (usually `../credentials.json`).
2. Run the following command to execute the `fetch_emails.py` script:

```
python fetch_emails.py
```

This script will:

- Authenticate via Gmail API (generates `token.json`).
- Connect to your Gmail inbox.
- Fetch the latest `n` emails.
- Store them in a local SQLite database (`EmailDatabase.db`).

Step 5: Modify `rules.json` As Per Your Requirements

Open `rules.json` and update the following as needed:

- Sender email domain
- Subject keyword
- Email age (`less_than`, `greater_than`)
- Actions (`mark_as_read`, `move_to:trash`, etc.)

Example structure:

```
{
  "all_rules": [
    {
      "predicate": "all",
      "rules": [
        { "field": "sender", "predicate": "contains", "value":
"@domain.com" },
        { "field": "subject", "predicate": "contains", "value": "Invoice"
},
        { "field": "date", "predicate": "less_than", "value": "7_days" }
      ],
      "actions": [ "mark_as_read", "move_to:Invoices" ]
    }
  ]
}
```

Allowed rules are as follows

field : sender , recipient , subject , message , date
Overall predicate : all , any
predicate : contains , does_not_contain , equals , does_not_equal

actions : mark_as_read , mark_as_unread , move_to:starred , move_to:important , move_to:trash , move_to:starred , move_to:important , move_to:trash

Note: For queries involving date , kindly use this format 2_days , 3_days , 1_months , 2_months

-actions accept a list of actions like this [“mark_as_unread” , “move_to:starred” , “move_to:important”]

Step 6: Process Emails Based on Rules

To evaluate rules and apply actions (labels, read/unread, delete, etc.):

```
python process_emails.py
```

📊 The script will:

- Load emails from the database.
- Evaluate each rule from `rules.json`.
- Apply corresponding Gmail actions via API.

You will see logs confirming rule matches and applied actions.

Step 7: Run Unit & Integration Tests

7.1 Run Unit Tests (basic logic, rule matching etc.):

```
python -m unittest discover -s tests/unit
```

7.2 Run Integration Tests (Gmail API and DB interactions):

```
python -m unittest discover -s tests/integration
```

Make sure all tests pass to confirm system integrity.