GCP GmaiL API Search Functions

Table of Contents

1. Introduction

In this assignment, the goal is to create a standalone Python Project that integrates with the Gmail API to perform rule-based operations on emails. The task is broken down into several key components:

1. **Authentication and Email Retrieval:** Using the official Google Python client, the script will authenticate to the Gmail API using OAuth and fetch a list of emails from the Inbox.

2. **Database Integration:** A relational database (e.g., Postgres, MySQL, SQLite3) will be used to store the retrieved emails, following a specific table representation.

3. **Rule-Based Email Processing:** A separate Python script will be developed to process emails based on predefined rules stored in a JSON file. The rules will consist of conditions and actions, such as marking emails as read/unread and moving messages.

4. **Rule Specification:** The JSON file will contain a collection of rules, each with field names, predicates, and values. The rules can have an overall predicate of "All" or "Any" to indicate the conditions' matching criteria.

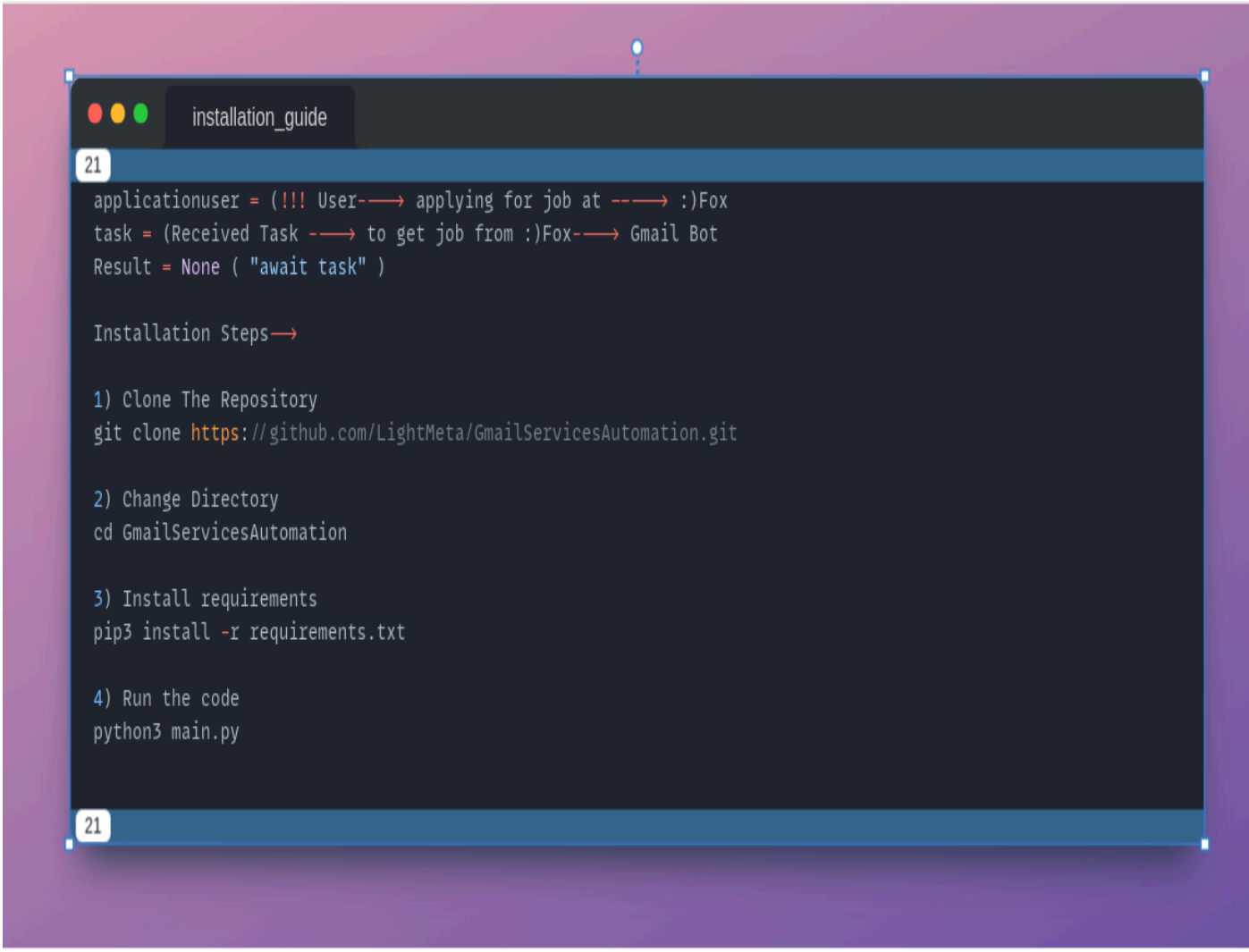
5. **Rule Implementation:** The rules will support various field types (e.g., From, Subject, Message, Received Date/Time) with corresponding predicates (e.g., Contains, Does not Contain, Equals, Does not equal, Less than, Greater than for date types).

2. Installation

To run the Resume Builder Django application, follow these steps:

Prerequisites
- Python 3.x
- pip
~ FastApi
~Json

Installation Steps



3. Project Structure

```
HappyfoxAssignment/
|
├──json_rules
├──main.py
├── json_rules/
├─ requirements.txt
├── test_main.py
├── app — app.py
```

```
├──                —      test_main.py
└── venv

```

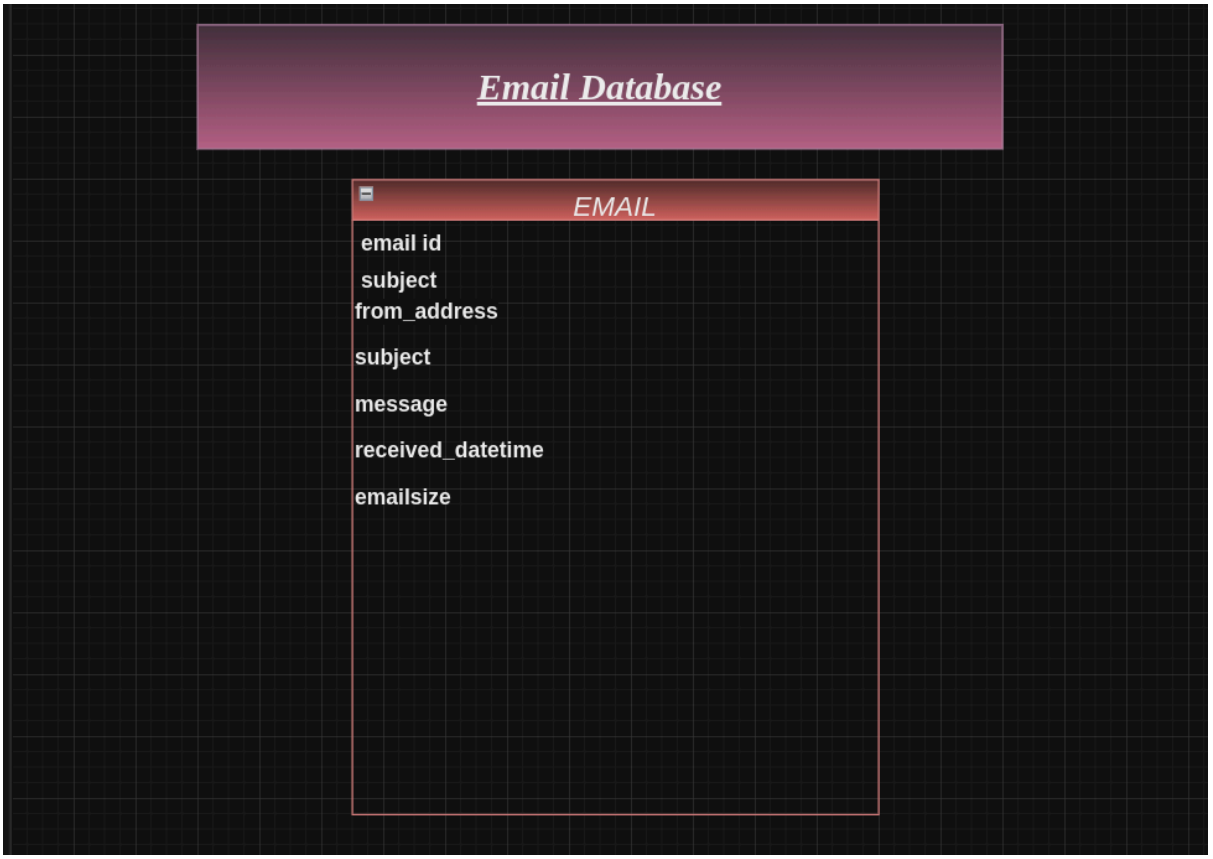
4. Configuration

- `DEBUG`: Set debug parameter to false in production.

5. Usage

1. Access the application at http://localhost:8000/docs
2. There are 5 API in total, kindly use them.
GET_DATA / LENGTH_DATA / INSERT_DATA/ CREATE_RULE / EXECUTE_RULE

4. Schema Design



There is an Email database based on sqlite3.

Database

EmailID
SUbject
From_address
Subject
Message
Received_datetime
Emailzise

5. API

GET_DATA
Fetch all data from Database. The Database is holding Emails.

LENGTH_DATA
How many rows in the database. Majorly emails in databas
INSERT_DATA
Creating a Oath connection using pickled credentials

Functions Provided:

1. **Authentication Functionality:**
   - Authenticates Gmail credentials securely using Google Cloud Platform.
   - Handles token refresh and creation for seamless authentication.

2. **Email Information Extraction:**
   - Extracts email information like sender, subject, date, message content, and email size in KB.
   - Retrieves emails in batches of 50 from the Inbox label.

3. **Database Interaction:**
   - Creates a SQLite database connection and cursor to interact with the database.
   - Creates an "emails" table if it doesn't exist to store email information.

4. **Data Storage Functionality:**
   - Saves the fetched email data into the SQLite database.
   - Manages the insertion of email information rows into the database table.

Current Status:
- The system has successfully inserted the retrieved email data into the database.

For detailed information, please refer to the [specific documentation](https://www.googleapis.com/auth/gmail.readonly) regarding the Gmail API's `gmail.read only` permission.

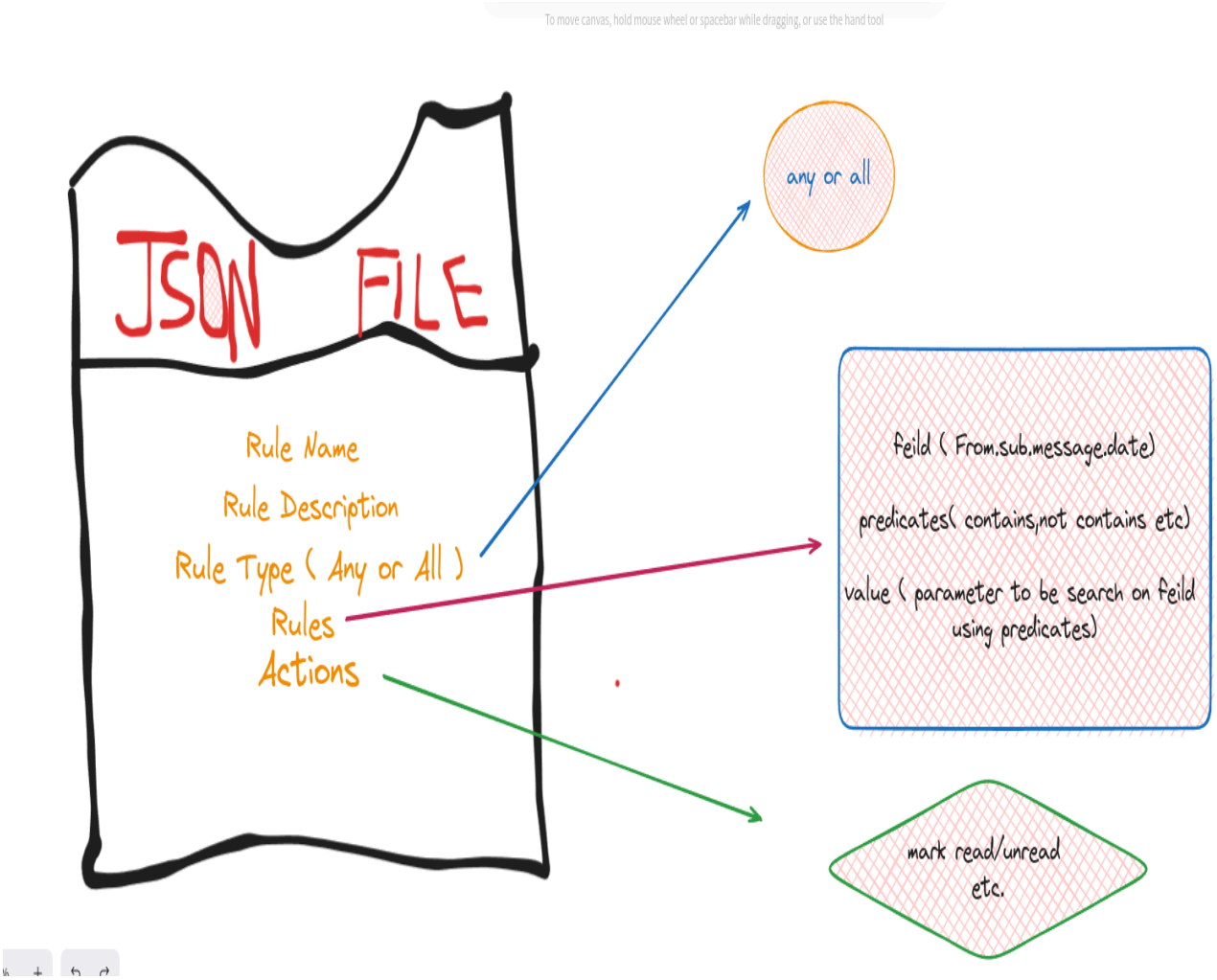

To move canvas, hold mouse wheel or spacebar while dragging, or use the hand tool
Start
Authenticates credentials using GCP
Token valid?
No
Refresh/Create token
Use existing token
Retrieve emails in batches of 100 from Inbox label
Extract email details
Create SQLite DB connection
Create 'emails' table if not exists
Save email data into database
Manage email info insertion into table
End

CREATE RULE

1. Extract Rule Structure:
   - Extracts the rule's name, description, type, list of rules, and list of actions from the input JSON.

2. Convert to JSON and Save:
   - Converts the rule structure into a JSON-formatted string with an indentation of 2 spaces.
   - Generates a file name based on the rule name, replacing spaces with underscores and adding a `.json` extension.
   - Saves the JSON-formatted rule to a file in the "json_rules" folder with the generated file name.

3. Return Message:
   - Returns a message confirming the successful saving of the rule in the specified file.



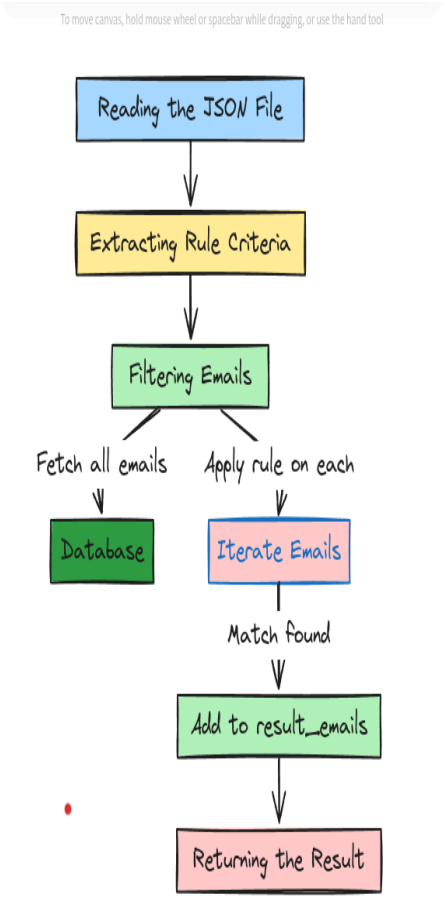
APPLY RULE

1. **Reading the JSON File:**
   - Reads the contents of a JSON file to extract the rule criteria for filtering emails.

2. **Extracting Rule Criteria:**
   - Retrieves the rule type and the specific rule criteria from the JSON data, such as the field to search on, the predicate/scenario to apply, and the value to compare with.

3. **Filtering Emails:**
   - Fetches all email data from the database.
   - Iterates through each email and applies the rule criteria based on the specified field, scenario, and value.
   - If the email matches the criteria, it is added to the result_emails list along with its associated ID for potential actions like deletion.

4. **Returning the Result:**
   - Returns a dictionary containing the result_emails list, which includes the emails that passed the specified rules.



6. Urls

http://localhost:8000/docs/

7. **Features**

- Scalable
    (various apis can be integrated)
    (api endpoints can directly be used in other applications)
    (secure , usage of pickle file)
    (Schema Selection , because models are given in Api)
    (Rules can be created again with additional features in JSON)
~ Secure
~ User Friendly
.

7. **Unit Tests**

The project has a feature of automated testing instead of manual testing.
Fast Api integration Testing has been used due to speed and reliability instead of the Pytest module.
The Basic idea is we will directly hit the API links , Either we can have status code 200 else any other status code like 404 (error or not).

If anyone wants to check whether emails are inserted , there is an endpoint which gives rows/email of the database. So the user does not have to open the database again and again to check whether new emails are there.
.

8. Appendix and Additions

1. Login System can be integrated
2. Response jsons from Api can be used in other applications.
3. Data can be converted to various formats (json to pdf etc)
4. New features such as email sending etc can be automated.
5. Instead of relational Database , we can move to Doc based such as ElasticSearch , HBase to help in Faster search with growing data.
6. Script for cron jobs could be there which runs at midnight like 12 am on server and adds up all mails that are inbox in the last 24 hours.
7. Important trigger Mail sender can be a feature, that as soon as we get mail from a particular person or Organisation such as Naukri,Official company or Team Members.That mail directly moves to the important section in Gmail.
8. Automated blocking system for particular text mails , such as women harassment , child harassment.