### Project Overview: Job Tracker Extension using Outlook Emails

This project involves integrating multiple technologies like Microsoft Graph API, Apache Airflow, PostgreSQL, and a Chrome extension to track job applications. The goal is to streamline the process of tracking job applications by automating data extraction from emails and building a browser extension that checks job board listings against a database of applied companies.

---

### Components of the Project:

1. \*\*Outlook API Integration\*\*:

- Use Microsoft Graph API to fetch email data.

- Filter and process job application emails.

2. \*\*Airflow for Scheduling\*\*:

- A DAG (Directed Acyclic Graph) in Airflow to fetch email data daily.

- Process the emails and extract relevant information (e.g., applied company names).

3. \*\*PostgreSQL Database\*\*:

- Store the company names and other relevant details like date applied.

4. \*\*Chrome Extension\*\*:

- Fetch the company name from job board pages.

- Compare it against the database and notify the user if they have already applied or not.

---

### Detailed Design Document

#### 1. \*\*Outlook API Integration\*\*

- \*\*Objective\*\*: Fetch emails related to job applications from the Outlook inbox using Microsoft Graph API.

\*\*Steps\*\*:

1. \*\*Authentication\*\*:

- Use OAuth 2.0 to authenticate and gain access to the user's Outlook inbox.

- The API endpoint to retrieve the emails is `https://graph.microsoft.com/v1.0/me/messages`.

2. \*\*Filtering Emails\*\*:

- Fetch all emails and filter based on specific job boards or application emails.

- Example: Filter emails where the sender's email address matches job alert services (e.g., LinkedIn, Indeed).

```json

{

"sender": {

"emailAddress": {

"name": "LinkedIn Job Alerts",

"address": "jobalerts-noreply@linkedin.com"

}

}

}

```

3. \*\*Extracting Company Names\*\*:

- Process the email body to extract company names using regex or natural language processing (NLP).

- Output: List of company names for further processing.

---

#### 2. \*\*Airflow for Scheduling & Processing\*\*

- \*\*Objective\*\*: Automate the process of fetching and processing emails daily.

\*\*Steps\*\*:

1. \*\*Setting up the DAG\*\*:

- Install Airflow and create a DAG that runs daily.

- Schedule the DAG to run once every 24 hours to fetch and process new emails.

2. \*\*Fetching Emails\*\*:

- In a Python task, use the Outlook API to fetch new emails.

- Save the emails locally or in-memory for further processing.

3. \*\*Processing the Data\*\*:

- A Python task in the DAG to parse the fetched emails, extracting the company names.

- Store the company names in a list.

4. \*\*Storing Data in PostgreSQL\*\*:

- Another task in the DAG to insert the extracted company names and any other relevant information (date, job role) into a PostgreSQL database.

```sql

CREATE TABLE applied\_companies (

id SERIAL PRIMARY KEY,

company\_name VARCHAR(255) NOT NULL,

date\_applied DATE,

UNIQUE(company\_name)

);

```

---

#### 3. \*\*PostgreSQL Database Design\*\*

- \*\*Objective\*\*: Design a database to store applied companies.

\*\*Steps\*\*:

1. \*\*Database Schema\*\*:

- Table name: `applied\_companies`.

- Columns:

- `id`: A unique identifier for each application.

- `company\_name`: The name of the company the application was sent to.

- `date\_applied`: The date the application was submitted.

2. \*\*Connecting Airflow to PostgreSQL\*\*:

- Use the `psycopg2` Python library or an Airflow operator to connect and push data into the database.

Example code snippet for inserting records:

```python

def insert\_company\_to\_db(company\_name, date\_applied):

conn = psycopg2.connect("dbname=test user=postgres")

cursor = conn.cursor()

cursor.execute(

"INSERT INTO applied\_companies (company\_name, date\_applied) VALUES (%s, %s) ON CONFLICT (company\_name) DO NOTHING",

(company\_name, date\_applied)

)

conn.commit()

cursor.close()

conn.close()

```

---

#### 4. \*\*Chrome Extension\*\*

- \*\*Objective\*\*: Build a Chrome extension that checks if a company from job board listings has already been applied to.

\*\*Steps\*\*:

1. \*\*Fetching Company Name from Job Boards\*\*:

- Use JavaScript to scrape the company name from the job board page (e.g., LinkedIn, Indeed).

- Parse the HTML content of the job board to identify and extract the company name.

2. \*\*API Integration\*\*:

- Use a background script to send a request to the backend (which connects to the PostgreSQL database).

- Check if the company name is present in the `applied\_companies` table.

3. \*\*Displaying Results\*\*:

- Use the Chrome extension's content script to display a notification in the browser (e.g., "Already Applied" or "Yet to Apply").

Example: Popup notification in Chrome.

4. \*\*Backend (Python)\*\*:

- Create a lightweight Flask or FastAPI app in Python to expose an API that the Chrome extension can call.

- This API would connect to the PostgreSQL database to check if the company name exists and return the result.

Example endpoint:

```python

from flask import Flask, request, jsonify

import psycopg2

app = Flask(\_\_name\_\_)

@app.route("/check\_company", methods=["POST"])

def check\_company():

company\_name = request.json['company\_name']

conn = psycopg2.connect("dbname=test user=postgres")

cursor = conn.cursor()

cursor.execute("SELECT \* FROM applied\_companies WHERE company\_name = %s", (company\_name,))

result = cursor.fetchone()

conn.close()

if result:

return jsonify({"status": "already\_applied"})

else:

return jsonify({"status": "yet\_to\_apply"})

if \_\_name\_\_ == "\_\_main\_\_":

app.run(debug=True)

```

---

### Tech Stack:

- \*\*Backend\*\*:

- Python with FastAPI or Flask for the API.

- PostgreSQL for database storage.

- \*\*Scheduler\*\*:

- Apache Airflow to automate email fetching and processing.

- \*\*Frontend\*\*:

- Chrome Extension using JavaScript, HTML, and CSS.

- \*\*Integration\*\*:

- Microsoft Graph API for fetching emails from Outlook.

- OAuth 2.0 for authentication.

---

### Chrome Extension in Python:

Chrome extensions themselves are built with HTML, CSS, and JavaScript, so you can't write the actual extension in Python. However, you can integrate Python on the backend (e.g., Flask API) for the extension to interact with when checking company names in the database.

---

### Next Steps:

1. \*\*Start with the API\*\*: Implement the Microsoft Graph API integration to fetch Outlook emails.

2. \*\*Set up Airflow\*\*: Create a DAG that runs daily to fetch and process the emails.

3. \*\*Build the Database\*\*: Design the PostgreSQL schema and create the table for storing applied companies.

4. \*\*Develop the Chrome Extension\*\*:

- Focus on scraping the job board pages for company names.

- Implement the API calls to check the database.

5. \*\*Testing\*\*: Integrate all components and test the functionality end-to-end.

Let me know if you'd like further details or code implementation guidance!