INDIA SENIOR BACKEND DEVELOPER— INTERVIEW TASK (INTERNAL USE)

Project Title: Microsoft Graph API Integration for Email Sending and Retrieval

Objective

Create a Python-based service that:

1. **Sends emails** via the Microsoft Graph API.
2. **Retrieves new emails** from the Microsoft Graph API and stores them in a MongoDB database.
3. **Runs periodically** to automatically capture any new messages from the last 24 hours—without manual user triggers.

Requirements & Specifications

1. **API Endpoints**
    - **Send Email Endpoint**
        Input: recipient's email address, subject, body, attachments (optional).
        Task: Uses Microsoft Graph API to send an email.
    - **Retrieve Email Endpoint** (Periodic)
        Fetches all new emails from the user's inbox within the past 24 hours.
        Stores the email data (sender, subject, body, timestamps) in MongoDB.
2. **Database**
    - Use **MongoDB** to persist email records.
    - Schema design is up to you, but please store essential email fields (sender, subject, body, timestamps).
3. **Scheduling**
    - The service should **periodically** (e.g., every 30 minutes or hourly) auto-fetch new emails without any user action required.
    - You can use any scheduling mechanism (e.g., cron job, celery, APScheduler, or a simple loop with sleep for demonstration).
4. **Testing**
    - **Create a free Outlook account** for testing. You'll use its credentials to authenticate with Microsoft Graph API.
    - Create Free Entra ID account under your outlook account to register your app (we will not give you AAD application).
    - Ensure your solution can demonstrate the sending of at least one email and the retrieval of at least one email automatically within the past day.
5. **Local or Cloud Deployment**
    - The solution should be runnable locally on a developer machine.
    - If you prefer to deploy a demo version in the cloud (Heroku, AWS, etc.), please provide relevant links and instructions.
6. **Security & Configuration**

    - Use **environment variables** to store and retrieve sensitive information (e.g., Outlook credentials, client secret).
    - Do **not** commit credentials to the repository.
7. **Documentation**
    - Include a **README.md** with:
        Setup instructions for installing dependencies, environment variables, and running the service.
        Clear explanation of your approach to scheduling, API structure, and how to test sending/retrieving emails.
        If deployed, provide the base URL and any required endpoints to run a live test.
8. **Use of Generative AI Tools**
    - Document how you utilized GenAI coding tools (e.g., Cursor, ChatGPT, GitHub Copilot) to assist in code generation, debugging, test creation, or documentation.
    - This can be a brief section in your README (e.g., “How I Used AI Coding Tools”).
9. **Submission**
    - Commit your complete solution to a **GitHub repository**.
    - Send a **GitHub invitation** to **asa-sabsono** granting access to your repo.
    - Your final repo should contain all relevant source code, tests, the README, and any scripts or configurations needed to demonstrate the project.
    - Example of successful task such as sending email and retrieve email periodically should be attached to the Github repository

Suggested Technology & Tools

- **Language**: Python (3.8+ preferred).
- **Framework**: Flask, FastAPI, or Django (your choice).
- **Database**: MongoDB (local or hosted).
- **Scheduling**: Built-in schedulers (e.g., APScheduler), Celery, or cron (any approach to show periodic tasks).
- **Version Control**: Git (GitHub for submission).
- **Testing**: Pytest or the built-in unittest library.
- **Documentation**: Markdown-based README and docstrings in code.
- **Generative AI**: ChatGPT, GitHub Copilot, etc.

What We’re Evaluating

1. **Correctness & Functionality**
    - Does the service send and retrieve emails via Microsoft Graph API as required?
    - Are emails correctly stored in MongoDB?
2. **Code Quality & Structure**
    - Is the project well-organized with clear separation of concerns (controllers, database layer, etc.)?
    - Are best practices used for error handling, logging, and configuration (environment variables)?
3. **Use of Generative AI Tools**
    - How effectively were AI tools employed to expedite coding, testing, and documentation?
    - Did you maintain code readability and quality while using AI-generated snippets?

4. **Security & Best Practices**
    - Credentials handling via environment variables.
    - Proper use of gitignore and safe commits.
5. **Documentation & Ease of Setup**
    - Is the README sufficiently detailed for another developer to install, configure, and run the service?

Bonus Points

- **Automated Tests**: Unit or integration tests that verify both sending and receiving of emails.
- **Dockerization**: Provide a Dockerfile or docker-compose setup for easy build and deploy.
- **Advanced Features**:
    - Handling attachments in emails.
    - Graceful handling of rate limits or API errors.
    - Using advanced AI-based code generation or test coverage reports.

Good Luck & Have Fun!

We look forward to seeing how you tackle this assignment—especially how you integrate generative AI coding tools to streamline the process. Feel free to be creative and add any features you believe will showcase your skills.