# Backend Web Developer Take-Home Challenge: Email Engine Core

## Welcome!

We're excited to have you participate in our take-home challenge for the Backend Web Developer position. This challenge will assess your skills and experience in building a core system for an email client.

**The Challenge:**

Your task is to develop a core system for an email client with essential functionalities. This system will connect with user email accounts (initially focusing on Outlook) and manage email data efficiently.

**Technical Requirements:**

- **Database:**
  - Implement a database, preferably using Elasticsearch, to store all user email address information.
  - Design separate indices for:
    - Email messages associated with each user address.
    - Mailbox details for each user address.
- **API Development:**
  - Create an API endpoint that allows users to create a local account and link it with their Outlook email address using OAuth.
  - Upon account creation, the API should:
    - Generate a URL for users to log in with their Outlook account.
    - Redirect users back to a specified callback URL after successful login.
    - Save the logged-in user's details and access token securely.

        - Don't forget, that the created account has a local ID. This ID will be attached to the email data and mailboxes later on.

- **Email Data Synchronization:**
    - Implement functionalities to:
        - Synchronize email data from Outlook to the local database upon successful account linking.
        - Handle rate limits and other potential challenges encountered during data fetching from Outlook.
        - Index all retrieved email data locally with unique identifiers for future reference.
        - Continuously monitor for changes in user email data (e.g., moved emails, read/unread status, flags, deletions, new emails) even during the initial sync.
        - Update the local data accordingly to reflect these changes.
- **Scalability and Performance:**
    - Design the system with scalability in mind. It should be able to handle a large number of user email addresses and allow for horizontal and vertical scaling as needed.
    - Prioritize code structure and optimization for efficient performance.
- **Best Practices:**
    - Adhere to best practices in software development, incorporating principles like:
        - DRY (Don't Repeat Yourself)
        - KISS (Keep It Simple, Stupid)
        - SOLID (Single responsibility, Open-closed, Liskov substitution, Interface segregation, Dependency inversion)
- **Extensibility:**
    - While the initial implementation focuses on Outlook, ensure the design can be easily extended to support other email providers (e.g., Gmail) using

IMAP protocols.

- Frontend showcase:
  - Design components are not important
  - Simple add account page (that reflect connection with outlook and the initial sync proccess after)
  - Data page:
    - Display real-time updates on the ongoing data synchronization process.
    - Present a list of fetched or updated emails in a simple format, including:
      - Subject line
      - Sender name/email address
    - This basic prototype aims to demonstrate the system's responsiveness to user actions performed within Outlook (e.g., marking emails as read, moving emails to different folders).
    - Upon such actions, the prototype should reflect the corresponding changes in the displayed email list.

**Deliverables:**

- A fully functional service showcasing the core functionalities.
- A simple interface to display ongoing account activity (during data sync and real-time updates).
- A Dockerized application with a docker-compose environment and clear documentation for running the project.
- Code that prioritizes best practices and maintainability.

**Evaluation Criteria:**

Your submission will be evaluated based on the following:

- Functionality: Completeness and correctness of implemented features.

- Scalability and Performance: Ability to handle large datasets and optimize for efficiency.
- Code Quality: Adherence to best practices, code clarity, and maintainability.
- Documentation: Clarity and completeness of instructions for running the application.

**Please Note:**

- Design elements are not a priority during development.
- Focus on demonstrating functionality and core system operations.

We look forward to reviewing your submission!