**Email Engine Core App Documentation**

**Overview**

This is a Node.js application built using Express.js, Socket.io, and Microsoft Authentication Library (MSAL) for authentication with Azure Active Directory (AAD). The app provides a RESTful API for user management, email synchronization, and webhook notifications.

**Requirements**

NodeJS v16 or higher on your system.

Microsoft Azure account with active directory.

Need to register a new application there on azure and from the Secret tokens tab and create new token then you need to add that to the environment variables.

Also, you will need to add the redirect URI while registering the app on azure.

If the **NOTIFICATION\_URL** variable is not set or is set to a URL that is not exposed to the internet, then the app will work fine but notification won’t be received by the app.

**Features**

User authentication with Azure Active Directory (AAD) using MSAL.

RESTful API for user management (create, read, update, delete).

Email synchronization with Microsoft Outlook using Microsoft Graph API.

Webhook notifications for email events (e.g., new email, email update).

Real-time communication using Socket.io.

Error handling and logging using Morgan and error handler middleware.

**Running the App**

**To run the app, follow these steps:**

Clone the code from the repo: git clone <https://github.com/walee275/email_engine_core.git> .

Install the required dependencies using npm install or yarn install.

Set up the environment variables by changing the .env.example file to .env file and then add then set the variables values.

Start the server using node server.js or npm start.

The app will start listening on the specified port, and you can access the API endpoints using your browser. For example: <http://localhost:3000/>.

Frontend is only for logging in and checking emails. Users' management is based on API’s.

**Environment Variables**

The app uses the following environment variables:

**DB\_CONNECTION\_STRING:** MongoDB connection string.

**PORT**: Port number of the server default will be 3000

**OAUTH\_CLIENT\_ID**: Client ID of the Azure AD application

**OAUTH\_REDIRECT\_URI**: Redirect URL you have setup in your azure app registration. Default will be <http://localhost:3000/auth/callback>.

**OAUTH\_SCOPES**: These are basically the permissions your azure app will be asking from the user. Default ('user.read, calendars.readwrite, mailboxsettings.read, mail.read, offline\_access,mail.readwrite').

**OAUTH\_AUTHORITY**: Authority URL of the Azure AD tenant

**OAUTH\_CLIENT\_SECRET**: Client secret of the Azure AD application

**NOTIFICATION\_URL**: Webhook URL for email notifications. On local you may use ngrok to expose your app and add that ngrok app URL here.

**API Endpoints**

**User Management**

GET /api/users: Retrieve a list of all users

GET /api/users/:id: Retrieve a user by ID

POST /api/users: Create a new user

PUT /api/users/:id: Update a user

DELETE /api/users/:id: Delete a user

Email Synchronization

GET /sync-mails: Synchronize emails with Microsoft Outlook (requires authentication)

GET /refresh-token: Refresh the access token for email synchronization (requires authentication)

**Webhook Notifications**

POST /webhook: Handle webhook notifications from Microsoft Graph API

**MSAL Configuration**

The app uses MSAL for authentication with Azure Active Directory (AAD). The MSAL configuration is set up using the following environment variables:

OAUTH\_CLIENT\_ID: Client ID of the Azure AD application

OAUTH\_AUTHORITY: Authority URL of the Azure AD tenant

OAUTH\_CLIENT\_SECRET: Client secret of the Azure AD application

**Microsoft Graph API**

The app uses the Microsoft Graph API to synchronize emails with Microsoft Outlook. The Microsoft Graph API is configured using the following environment variables:

NOTIFICATION\_URL: Webhook URL for email notifications

**Socket.io Configuration**

The app uses Socket.io for real-time communication. The Socket.io configuration is set up using the following code:

**Error Handling**

The app uses error handler middleware to catch and log errors. The error handler middleware is set up.

**Cron Job**

The app uses a crone job to create or renew subscriptions with Microsoft Graph API every hour.