Microsoft Outlook Mailbox Integration

**Software Architecture Brief**

**Prepared By:** Shane Sandwith

**Role:** Software Architect

**Date:** 7th April 2024

Table of Contents

[1. Executive Summary 3](#_Toc195256788)

[2. Business Objectives 3](#_Toc195256789)

[3. Scope 3](#_Toc195256790)

[3.1. In Scope: 3](#_Toc195256791)

[3.2. Out of Scope (Phase 1): 3](#_Toc195256792)

[4. High-Level Architecture 4](#_Toc195256793)

[5. Authentication & Authorization 4](#_Toc195256794)

[6. Azure App Registration Responsibility 5](#_Toc195256795)

[6.1. Client Responsibilities 5](#_Toc195256796)

[6.2. Security Notes 5](#_Toc195256797)

[7. Security Considerations 5](#_Toc195256798)

[8. Technology Stack 5](#_Toc195256799)

[9. Success Criteria 6](#_Toc195256800)

[10. Error Handling Strategy 6](#_Toc195256801)

[10.1. Expected Error Scenarios 6](#_Toc195256802)

[11. Technical Deep Dive 6](#_Toc195256803)

[11.1. OAuth Flow 6](#_Toc195256804)

[11.2. Token Storage 7](#_Toc195256805)

[11.3. Microsoft Graph Calls 7](#_Toc195256806)

[11.4. Angular ↔ API Communication 7](#_Toc195256807)

[12. Data Flow Diagram 8](#_Toc195256808)

[13. User Flow Diagram 10](#_Toc195256809)

[14. System Diagram 11](#_Toc195256810)

# Executive Summary

This project involves integrating Microsoft Outlook mailboxes into our SaaS platform, allowing clients to view and interact with their Outlook emails directly through our application. This document outlines the architectural approach, technical requirements, and implementation plan for this feature.

# Business Objectives

- Enhance user productivity by centralizing email interactions within our app.

- Provide value-added features such as in-app messaging insights, email categorization, and search.

- Ensure secure and compliant integration with Microsoft 365 environments.

# Scope

## In Scope:

- OAuth 2.0 integration with Microsoft Identity platform.

- Secure token management and refresh lifecycle.

- Fetching emails from Microsoft Graph API.

- Support for both delegated and application-level permissions.

- User interface in Angular for email sync status and setup.

## Out of Scope (Phase 1):

- Sending or modifying Outlook emails.

- Calendar and contact integration.

- Offline or background syncing beyond current token scope.

# High-Level Architecture

**Frontend (Angular)**

- UI for "Authorize with Microsoft"

- Callback handler for OAuth redirect

- Securely calls backend with OAuth code

**Backend (C#/.NET Core API)**

- Token exchange with Microsoft using OAuth 2.0

- Secure token storage (encrypted database fields)

- API layer for fetching emails from Microsoft Graph

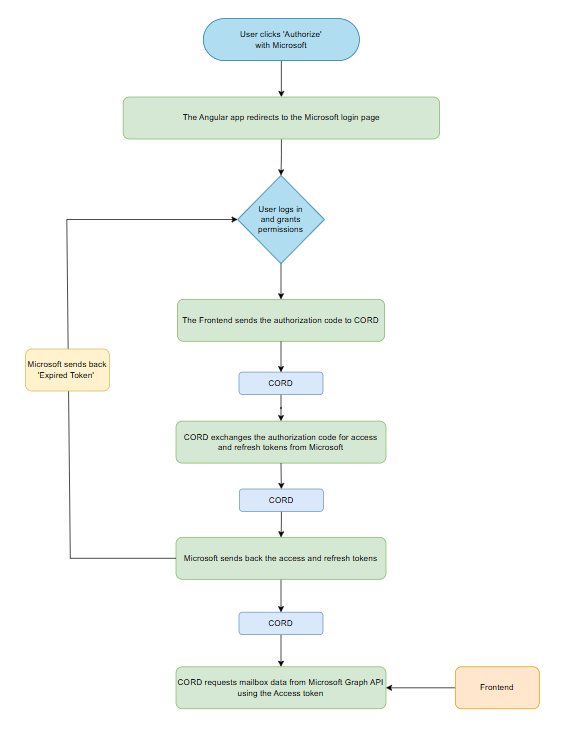
- Background service for token refresh

**External**

- Microsoft Identity Platform (Azure App Registration)

- Microsoft Graph API (https://graph.microsoft.com)

# User Flow Diagram



# Authentication & Authorization

- Microsoft OAuth 2.0 Authorization Code Grant Flow

- Uses client ID, secret, redirect URI from Azure App Registration

- Refresh tokens handled securely

- Optional: Admin consent flow for organization-wide access

# Azure App Registration Responsibility

To support a multi-tenant environment and ensure that each client retains control over their Microsoft 365 data, the Azure App Registration must be created and maintained by the client within their own Azure Active Directory (AAD) tenant.

## Client Responsibilities

Clients are responsible for:

- Registering the application in their Azure Active Directory.

- Providing the following configuration details to our team:

* **Client ID**
* **Client Secret**
* **Tenant ID**

- Granting appropriate API permissions to enable mailbox access via Microsoft Graph.

- Approving admin consent (if required for organization-wide access).

- Adding the provided **Redirect URI** to their app registration.

## Security Notes

- Rotation of secrets should be handled by the client, with a mechanism in place for updating credentials within our platform.

# Security Considerations

- All token data encrypted at rest.

- Access tokens never exposed to frontend.

- Use of secure HTTPS for all communication.

- Support for token revocation and integration expiry handling.

# Technology Stack

- Frontend: Angular 18+

- Backend: .NET 6+ Web API

- Identity Provider: Microsoft Azure AD

- API Integration: Microsoft Graph

- Storage: MongoDB (encrypted fields for tokens)

# Success Criteria

- 100% secure and working integration with at least 1 real Microsoft tenant

- Ability to fetch latest 50 emails per mailbox

- Admin consent flow working for multi-user access

- All tokens securely stored and managed

# Error Handling Strategy

A robust error handling strategy is essential for ensuring reliability, user trust, and debuggability of the Outlook integration feature. This section outlines the expected error scenarios and recommended handling approaches.

## Expected Error Scenarios

|  |  |  |
| --- | --- | --- |
| Category | Example | Handling Approach |
| Authentication Errors | Invalid or expired OAuth token, revoked client credentials | 401 Unauthorized |
| Authorization Errors | Missing API permissions (e.g., Mail.Read not granted) | 403 Forbidden |
| Token Expiry | Access or refresh token has expired | Attempt automatic refresh where possible; if not, notify user to reauthenticate with a 401 Unauthorized |
| Consent Errors | User denied access during Microsoft login | Frontend should handle gracefully and allow retry. |
| Invalid Configuration | Incorrect Client ID, Secret, or Tenant ID | 500 Internal Server Error |

# Technical Deep Dive

## OAuth Flow

- Frontend redirects or opens popup to Microsoft login with required scopes.

- Microsoft returns code to frontend.

- Code sent to CORD for token exchange using client credentials.

- Access and refresh tokens are returned and stored securely.

## Token Storage

- Tokens are encrypted before being saved to the database.

- Refresh token expiration time is tracked to proactively refresh before expiry.

- If refresh token is revoked, backend notifies frontend to trigger re-auth.

## Microsoft Graph Calls

- Access token used to call “/me/messages” or “/users/{id}/messages” endpoints.

- Email results are paginated; backend handles @odata.nextLink pagination.

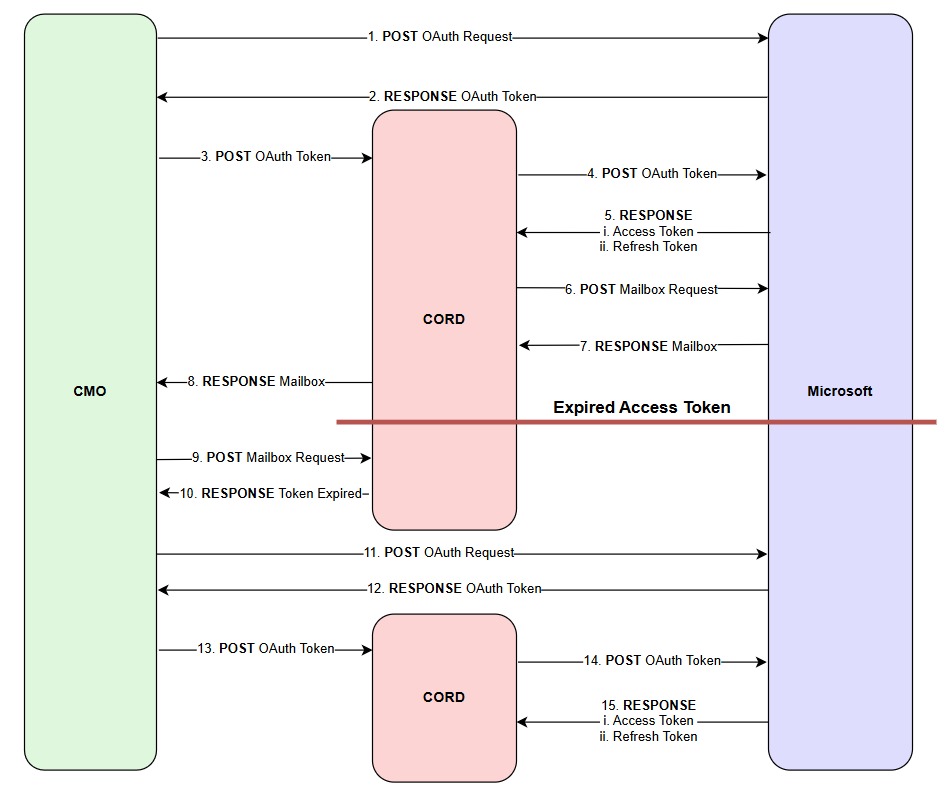
- Graph API rate limits are respected with exponential backoff retries.

## Angular ↔ API Communication

- Uses HTTP-only cookies or bearer tokens for session management.

- OAuth code never stored on frontend, only transiently used.

# Data Flow Diagram



1. The Angular frontend initiates user authentication by submitting an OAuth authorization request to Microsoft’s identity server.
2. Microsoft responds with an OAuth authorization code upon successful user authentication and consent.
3. The frontend forwards the authorization code to the CORD backend.
4. CORD exchanges the authorization code with Microsoft for access and refresh tokens.
5. Microsoft returns the access token and refresh token to CORD.
6. CORD uses the access token to request the user's mailbox data via the Microsoft Graph API.
7. Microsoft responds with the user’s mailbox content.
8. CORD returns the mailbox data to the Angular frontend for display.

|  |
| --- |
| **Token Expiry and Reauthorization Flow** |

1. The frontend attempts to retrieve mailbox data after the tokens have expired.
2. CORD detects the token expiration and responds with an authentication error.
3. The frontend prompts the user to reauthenticate, initiating a new OAuth authorization request to Microsoft.
4. Microsoft returns a new authorization code.
5. The frontend sends the updated authorization code to the CORD backend.
6. CORD exchanges the new authorization code for a fresh set of access and refresh tokens.
7. Microsoft returns the updated tokens to CORD.
8. CORD resumes the standard flow to request and return the user's mailbox data.

# System Diagram

