THOMAS MORE

Outlook Plugin Customer Service Backoffice

Realisatiedocument

Freek Boes
Student Bachelor in de Toegepaste Informatica – Applicatieontwikkeling

2025 - 2026

Inhoudsopgave

1. Inleiding

Dit realisatiedocument beschrijft de uitvoering van de stageopdracht bij H.Essers: het ontwikkelen van een Outlook-integratie met het Back Office platform. Waar het design document de architectuur en de functionele analyse uitwerkt, focust dit document op de effectieve realisatie: welke keuzes er gemaakt werden tijdens de implementatie, hoe de verschillende componenten technisch opgebouwd zijn, en welke uitdagingen er opgedoken zijn tijdens de ontwikkeling.

Het project liep van 23 februari tot 22 mei 2026 en werd opgedeeld in twee fases. In de eerste fase (week 1–3) stond analyse en architectuurontwerp centraal. De tweede fase (week 4–13) was gewijd aan de effectieve implementatie, integratie en afwerking van het systeem.

Het eindresultaat is een event-driven integratie bestaande uit drie hoofdcomponenten: een Outlook add-in waarmee gebruikers e-mails kunnen doorsturen naar het backoffice systeem, een Ingest API die de e-mailverwerking via Microsoft Graph en document storage afhandelt, en een Apache Kafka integratie waarmee de downstream CSBO-consumer automatisch taken aanmaakt op basis van de ontvangen events.

Dit document behandelt achtereenvolgens de Outlook add-in, de Gravitee API Gateway en de Ingest API, Mircorosft graph api, de document straoge en tot slot de Kafka-integratie. Per component wordt ingegaan op de technische opzet, de gemaakte implementatiekeuzes en relevante aandachtspunten.

2. Analyse

2.1. Outlook add-in

2.1.1. Inleiding

Voorafgaand aan de effectieve implementatie werd een grondige analyse uitgevoerd om de juiste technologische keuzes te maken. Hoewel de opdracht bij aanvang al een duidelijke richting aangaf, namelijk een Outlook-integratie met Office.js, was het toch zinvol om de beschikbare alternatieven te onderzoeken en de gemaakte keuzes te onderbouwen. De uiteindelijke keuzes lagen grotendeels voor de hand gezien de vereisten van de opdracht en de bedrijfsomgeving, maar worden in dit hoofdstuk toch toegelicht om aan te tonen waarom de gekozen aanpak de meest passende was.

De analyse focust op drie domeinen: de keuze van het add-in framework, de keuze van de frontend-technologie en de keuze van de authenticatiemethode. Elk van deze keuzes heeft een directe impact op de onderhoudbaarheid, veiligheid en schaalbaarheid van het eindproduct..

2.1.2. Add-in framework

2.1.2.1. Onderzochte opties

Voor de ontwikkeling van een Outlook-extensie zijn er historisch gezien meerdere technologieën beschikbaar geweest. Tijdens de onderzoeksfase werden de volgende opties in kaart gebracht:

Office.js (web-based add-in) is de huidige Microsoft-standaard voor het bouwen van Office-extensies. Een Office.js add-in is in essentie een webapplicatie (HTML, CSS, JavaScript) die door Outlook geladen wordt via een manifest. Dit maakt de add-in platformonafhankelijk: ze werkt in Outlook Web, Outlook for Windows en Outlook for Mac zonder extra installatie. Microsoft positioneert Office.js als de enige ondersteunde aanpak voor nieuwe add-in ontwikkeling.

VSTO (Visual Studio Tools for Office) is een oudere, COM-gebaseerde technologie waarmee Outlook-extensies als native Windows-applicaties gebouwd worden in C# of VB.NET. VSTO biedt diepere integratie met de Outlook-client, maar is strikt beperkt tot Windows en vereist installatie op het toestel van de gebruiker. Microsoft raadt nieuwe VSTO-ontwikkeling af ten voordele van Office.js.

Teams-app met Outlook-extensie is een nieuwere aanpak waarbij een Microsoft Teams-applicatie ook als Outlook add-in kan functioneren via het unified app manifest. Dit vereist echter dat de organisatie Microsoft Teams actief gebruikt en de Teams-infrastructuur beschikbaar is voor de add-in.

Conclusie: Office.js was de meest voor de hand liggende keuze. Het is de enige optie die volledig platformonafhankelijk is, actief ondersteund wordt door Microsoft en native aansluit op de authenticatieinfrastructuur die binnen H.Essers gebruikt wordt. VSTO valt af omdat het uitsluitend op Windows draait en door Microsoft als verouderd wordt beschouwd. De Teams-aanpak introduceert een onnodige afhankelijkheid van de Teams-infrastructuur voor een toepassing die louter binnen Outlook gebruikt wordt.

2.1.3. Frontend-technologie

2.1.3.1. Onderzochte opties

Binnen het Office.js framework is er vrije keuze in het gebruik van een frontend-bibliotheek of framework. De volgende opties werden onderzocht:

Vanilla JavaScript met HTML/CSS is de meest directe aanpak: geen extra frameworks, volledige controle over de DOM, en een minimale bundle-grootte. Dit vergt meer handmatig werk voor state management en DOM-manipulatie, maar reduceert de complexiteit van de toolchain.

React is een veelgebruikte JavaScript-bibliotheek voor het bouwen van componentgebaseerde interfaces. Microsoft biedt officiële ondersteuning voor React in Office.js via Fluent UI. React maakt het eenvoudiger om complexe, interactieve UI's te beheren via de componentenstructuur en hooks.

Angular is een uitgebreid frontend-framework dat een volledige structuur biedt voor grote applicaties. Het is zwaarder dan React en introduceert een steile leercurve, maar biedt sterke typeveiligheid via TypeScript.

Conclusie: Gekozen werd voor vanilla JavaScript, HTML en CSS, gebundeld via Webpack. De add-in UI is functioneel maar niet uitzonderlijk complex, er is geen nood aan geavanceerd state management of een uitgebreide componentenstructuur. Een lichtgewicht aanpak volstaat en vermijdt extra afhankelijkheden die de onderhoudbaarheid op termijn bemoeilijken. Dit sluit ook aan bij de bestaande kennis binnen het team en de relatief beperkte scope van de add-in UI.

2.1.4. Authenticatiemethode

2.1.4.1. Onderzochte opties

Een centrale vereiste is dat de gebruiker zich niet apart hoeft aan te melden in de add-in. De add-in moet de identiteit van de ingelogde Outlook-gebruiker automatisch kunnen doorgeven aan de backend. Hiervoor werden de volgende aanpakken onderzocht:

Nested App Authentication (NAA) is de huidige Microsoft-standaard voor authenticatie in Office.js add-ins. NAA maakt gebruik van de reeds aangemelde sessie van de gebruiker in Outlook en levert automatisch een Azure Entra ID access token op, zonder dat de gebruiker een extra aanmeldstap hoeft te doorlopen. Dit vermijdt pop-ups en cookie-problemen en ondersteunt moderne OAuth2-flows zoals de On-Behalf-Of flow. In de praktijk wordt NAA aangesproken via de ingebouwde Office.auth module die onderdeel is van Office.js.

getIdentityToken (legacy SSO) is de oudere SSO-methode in Office.js. Ze levert een identity token dat door de backend gevalideerd kan worden, maar vereist extra server-side verwerkingsstappen en ondersteunt de OBO-flow minder naadloos. Microsoft geeft aan dat NAA de aanbevolen vervanger is..

Manuele inlogpagina (OAuth2 popup) is een aanpak waarbij de gebruiker expliciet inlogt via een browservenster of dialoogvenster. Dit is technisch werkbaar maar introduceert extra gebruikersstappen en is conflicterend met de vereiste van een naadloze Single Sign-On ervaring.

Conclusie: NAA was de meest voor de hand liggende keuze. Omdat NAA via de ingebouwde Office.js authenticatiemodule aangesproken wordt, hoefde er geen externe bibliotheek aan het project toegevoegd te worden. Het verkregen token wordt rechtstreeks als Bearer token meegestuurd naar de Gravitee gateway, die het valideert via de publieke sleutels van Entra ID. De Ingest API gebruikt het token vervolgens om via de On-Behalf-Of flow namens de gebruiker Microsoft Graph aan te spreken. Dit maakt NAA de enige optie die de volledige authenticatieketen van add-in tot Microsoft Graph sluitend ondersteunt zonder extra complexiteit in de frontend.optie die de volledige authenticatieketen van add-in tot Microsoft Graph sluitend ondersteunt zonder extra complexiteit in de frontend.

2.2. Gravitee API Gateway

2.2.1. Inleiding

Voor de communicatie tussen de Outlook add-in en de Ingest API was een API Gateway nodig die instaat voor beveiliging, tokenvalidatie en routering van inkomende requests. Dit hoofdstuk beschrijft welke keuzes gemaakt werden rond de gateway-configuratie en hoe de authenticatieaanpak tot stand gekomen is. De keuze voor Gravitee als gateway lag op voorhand vast, aangezien H.Essers dit platform al intern in gebruik heeft. De analyse focust daarom vooral op de configuratie van de tokenvalidatie, waarbij verschillende aanpakken onderzocht en uitgeprobeerd werden voordat tot een werkende oplossing gekomen werd.

2.2.2. Keuze van de API Gateway

Voor het beveiligen en routeren van API-verkeer zijn er verschillende platformen beschikbaar. Binnen de context van dit project werden de volgende opties in kaart gebracht:

Gravitee is een open-source API Management platform dat binnen H.Essers al operationeel is als centraal gateway-platform. Gravitee draait binnen het interne Kubernetes-cluster en biedt ingebouwde ondersteuning voor authenticatie, rate limiting, logging, CORS en routering naar interne backend services.

Azure API Management is het API Management platform van Microsoft en sluit nauw aan bij de Azure-infrastructuur. Het biedt uitgebreide integratie met Azure Entra ID en Microsoft-diensten, maar vereist een afzonderlijke Azure-instantie en brengt extra beheerscomplexiteit met zich mee.

Kong is een veelgebruikte open-source API Gateway met een uitgebreid plugin-ecosysteem. Kong is flexibel en schaalbaar, maar vereist een aparte opzet en configuratie en is binnen H.Essers niet als standaardplatform in gebruik.

Conclusie: Gravitee was de enige logische keuze. De infrastructuur was reeds aanwezig binnen het bedrijf en het platform sluit aan bij de interne standaarden van H.Essers. Het opzetten van een alternatief platform zou buiten de scope van de stageopdracht vallen en onnodige complexiteit introduceren.

2.2.3. Tokenvalidatie

2.2.3.1. Onderzochte opties

Een van de centrale taken van de gateway is het valideren van het Azure Entra ID access token dat de Outlook add-in meestuurt bij elke request. Gravitee heeft hiervoor ingebouwde ondersteuning in de vorm van een JWT-validatiepolicy. Tijdens de implementatie werden verschillende aanpakken onderzocht en uitgeprobeerd.

JWKS_URL validatie is de aanbevolen aanpak waarbij de gateway de publieke sleutels van Azure Entra ID dynamisch ophaalt via de JWKS-endpoint van Microsoft. Hiermee kan Gravitee de handtekening van het JWT token automatisch valideren zonder dat de sleutels manueel beheerd moeten worden. Dit is de meest robuuste en toekomstbestendige oplossing.

GIVEN_KEY validatie is een alternatieve aanpak waarbij het publieke certificaat van Azure Entra ID statisch geconfigureerd wordt in de Gravitee JWT-policy. In plaats van de sleutels dynamisch op te halen, wordt er een hardcoded certificaat gebruikt om de tokenhandtekening te valideren. Dit vereist manueel beheer van het certificaat maar heeft geen uitgaande netwerkverbinding nodig.

API key validatie is een eenvoudigere aanpak waarbij geen JWT-validatie plaatsvindt in de gateway zelf. In plaats daarvan wordt een vaste API-sleutel meegegeven in de request header. Gravitee controleert enkel of deze sleutel aanwezig en geldig is. Dit biedt minder beveiliging dan JWT-validatie maar functioneert volledig onafhankelijk van externe systemen.

Conclusie: Tijdens de implementatie werd vastgesteld dat de Kubernetes-omgeving waarin Gravitee draait geen uitgaand verkeer toestaat naar externe domeinen zoals het Microsoft-authenticatieplatform. Dit is een interne netwerkpolicy bij H.Essers die ervoor zorgt dat de gateway-pod de JWKS-endpoint niet kan bereiken. Als gevolg hiervan kon de JWKS_URL aanpak niet werkend gemaakt worden. Vervolgens werd de GIVEN_KEY aanpak onderzocht als tussenoplossing, maar ook deze liep vast op dezelfde firewallbeperkingen tijdens de configuratiefase. Omdat de focus van het project lag op het realiseren van een werkende end-to-end flow, werd uiteindelijk gekozen voor een API key als pragmatische oplossing. Dit liet toe om de verdere ontwikkeling van de Ingest API en de Kafka-integratie voort te zetten zonder geblokkeerd te worden door infrastructuurproblemen die buiten de scope van de stageopdracht lagen. Voor een productieomgeving dient de JWT-validatie via JWKS_URL alsnog geïmplementeerd te worden, zodra de nodige outbound netwerkpermissies door de infrastructuurbeheerder zijn opengesteld.

2.3. Ingest API

2.3.1. Inleiding

De Ingest API vormt het hart van de backend-verwerking: ze ontvangt requests van de Outlook add-in via de Gravitee gateway, haalt de e-mail op via Microsoft Graph, slaat de bestanden op in document storage en publiceert een event naar Apache Kafka. Hoewel de keuze voor Java met Spring Boot op voorhand al vastlag gezien de bestaande technologiestandaarden binnen H.Essers, waren er tijdens de analyse en implementatie nog een reeks technische keuzes te maken rond authenticatie, HTTP-communicatie, e-mailformaat en de opbouw van de verwerkingsflow.

Dit hoofdstuk licht de belangrijkste keuzes toe die gemaakt werden voor de Ingest API: de authenticatiemethode voor de OBO-flow, de HTTP-client voor communicatie met document storage, en de keuze van het e-mailformaat dat opgeslagen wordt.

2.3.2. Authenticatiemethode voor de On-behalf-of flow

2.3.2.1. Onderzochte opties

De Ingest API moet namens de gebruiker Microsoft Graph kunnen aanspreken via de OAuth2 On-Behalf-Of flow. Hiervoor dient de API zich te authenticeren bij Azure Entra ID als vertrouwde applicatie. Twee aanpakken werden overwogen:

Client secret is de meest eenvoudige manier om een Azure-applicatieregistratie te authenticeren. Een gedeeld geheim wordt geconfigureerd in de Azure-portal en als omgevingsvariabele aan de applicatie meegegeven. Dit verlaagt de drempel voor initiële configuratie, maar brengt een beveiligingsrisico met zich mee: een gelekt geheim geeft onmiddellijk toegang tot de volledige applicatieregistratie zonder bijkomende verificatie. Microsoft raadt client secrets in productieomgevingen dan ook af ten voordele van certificaten.

Certificaatgebaseerde authenticatie maakt gebruik van een asymmetrisch sleutelpaar. De publieke sleutel wordt geregistreerd in Azure, terwijl de privésleutel veilig beheerd wordt buiten de codebase om. Dit is de aanbevolen aanpak voor productieomgevingen omdat een certificaat zonder de bijbehorende privésleutel nutteloos is voor een aanvaller. De MSAL-bibliotheek biedt ingebouwde ondersteuning voor deze aanpak en beheert bovendien zelf de token cache, zodat onnodige herhaalde aanvragen naar Entra ID vermeden worden.

Conclusie: Gekozen werd voor certificaatgebaseerde authenticatie, conform het beleid van H.Essers dat client secrets niet toegelaten zijn in productieve applicatieregistraties. Het certificaat wordt via omgevingsvariabelen aan de applicatie meegegeven, zodat gevoelige sleutelmateriaal nooit in de codebase terechtkomt. Bij opstart laadt de applicatie het certificaat en gebruikt het voor alle verdere communicatie met Entra ID.

2.3.3. HTTP-client voor communicatie met document storage

2.3.3.1. Onderzochte opties

De Ingest API moet bestanden uploaden naar de interne document storage via een REST API. Voor de HTTP-communicatie met deze service werden twee opties overwogen:

RestTemplate is de traditionele synchrone HTTP-client binnen het Spring-ecosysteem. Requests worden geblokkeerd totdat een antwoord ontvangen wordt, wat het eenvoudig maakt om te implementeren en te debuggen. Spring heeft RestTemplate echter als verouderd aangemerkt en raadt het gebruik ervan in nieuwe projecten af.

WebClient is de moderne HTTP-client van Spring en de aanbevolen opvolger van RestTemplate. WebClient biedt een vloeiende manier om HTTP-requests op te bouwen en verstuurt deze op een toekomstbestendige manier. Hoewel de Ingest API geen volledig reactieve architectuur heeft, laat WebClient toe om nieuwe code te schrijven die aansluit bij de huidige Spring-standaarden.

Conclusie: Gekozen werd voor WebClient. Omdat Spring RestTemplate als verouderd beschouwt, is WebClient de correcte keuze voor nieuwe implementaties. Alle communicatie met document storage verloopt via een geconfigureerde instantie met de juiste authenticatiegegevens en basis-URL, die als centrale configuratie beheerd wordt en door de betrokken services hergebruikt wordt.

2.3.4. E-mailformaat voor document storage

2.3.4.1. Onderzochte opties

Microsoft Graph levert e-mails standaard aan in het EML-formaat, een open standaard die de volledige inhoud van een e-mail beschrijft, inclusief headers, body en bijlagen. Voor de opslag in document storage moest bepaald worden in welk formaat de e-mail bewaard zou worden.

EML is het native uitvoerformaat van Microsoft Graph en eenvoudig te verkrijgen zonder extra conversie. Het is een breed ondersteunde open standaard en platformonafhankelijk. De document storage API van H.Essers kent echter geen geldig bestandstype toe aan het EML-formaat en weigert bestanden in dit formaat, waardoor opslag technisch niet mogelijk is.

MSG is het eigen Outlook-bestandsformaat en wordt door de document storage API wel geaccepteerd. Het formaat is volledig compatibel met Outlook, waardoor downstream systemen en medewerkers e-mails direct kunnen openen vanuit het archief. Een nadeel is dat MSG niet rechtstreeks door Microsoft Graph aangeleverd wordt en dus een conversie vereist vanuit het EML-formaat.

Conclusie: Gekozen werd voor het MSG-formaat, aangezien dit als enige geaccepteerd wordt door de document storage API. De conversie van EML naar MSG werd geïmplementeerd met behulp van de Apache POI bibliotheek. Een aandachtspunt dat tijdens de implementatie opdook, was dat geconverteerde MSG-bestanden door Outlook als concept herkend werden in plaats van als gewone verzonden berichten. Dit werd opgelost door bij de conversie de juiste berichtstatus mee te geven, zodat de e-mail bij openen correct als een gelezen bericht verschijnt en niet als een onafgewerkt concept.

2.4. Document Storage

2.4.1. Inleiding

Na het ophalen van de e-mail en bijlagen via Microsoft Graph moeten alle bestanden duurzaam opgeslagen worden in de interne document storage van H.Essers. Document storage fungeert als centrale bestandslaag: de Ingest API uploadt de bestanden, ontvangt als antwoord een unieke document-ID per bestand, en neemt die ID's mee in het Kafka-event. Downstream systemen zoals CSBO gebruiken die ID's om documenten op te halen en aan taken te koppelen, zonder ooit rechtstreeks met Microsoft Graph te communiceren.

Dit hoofdstuk beschrijft hoe de integratie met document storage technisch uitgewerkt is, welke keuzes gemaakt werden rond bestandstypes en foutafhandeling, en hoe selectieve verwerking van bijlagen gerealiseerd werd.

2.4.2. Bestandstypes en type-codes

2.4.2.1. Onderzochte opties

De document storage API van H.Essers vereist dat elk geüpload bestand voorzien wordt van een type-code die aangeeft wat voor soort document het betreft. Tijdens de integratie moest bepaald worden welke type-codes gebruikt zouden worden voor e-mails en bijlagen.

Zoals eerder toegelicht in het onderdeel over het e-mailformaat, werd vastgesteld dat het EML-formaat niet aanvaard wordt door de document storage API. De keuze voor het MSG-formaat was dan ook deels ingegeven door de beperkingen van de document storage. Voor bijlagen geldt dat elk bestandstype in principe aanvaard wordt, maar dat ook hier een correcte type-code opgegeven moet worden.

Conclusie: Na afstemming met de interne documentatie en de verantwoordelijke collega werd vastgelegd dat e-mails in MSG-formaat opgeslagen worden onder een specifieke type-code voor e-mailberichten, en dat bijlagen onder een afzonderlijke type-code vallen. Naast het bestandstype worden ook een klantcode en een padcode meegegeven bij elke upload, zodat de bestanden binnen document storage correct geclassificeerd en teruggevonden kunnen worden.

2.4.3. Foutafhandeling bij gedeeltelijk falen van bijlagen

2.4.3.1. Onderzochte opties

Een realistische situatie die zich kan voordoen is dat de upload van de e-mail zelf slaagt, maar dat één of meerdere bijlagen niet succesvol opgeslagen kunnen worden. Hiervoor moest een strategie bepaald worden.

Volledig afbreken bij fout is de meest strikte aanpak: als één bijlage faalt, wordt de volledige verwerking stopgezet en wordt er geen Kafka-event gepubliceerd. Dit garandeert volledige consistentie, maar betekent dat ook de succesvol opgeslagen documenten verloren gaan en de gebruiker het proces volledig opnieuw moet starten.

Doorgaan met beschikbare documenten is een meer pragmatische aanpak: de e-mail zelf wordt altijd opgeslagen en de bijlagen worden afzonderlijk verwerkt. Als een bijlage faalt, wordt dit gelogd maar wordt de verwerking niet geblokkeerd. Het Kafka-event bevat in dat geval enkel de document-ID's van de bestanden die wel succesvol opgeslagen werden.

Conclusie: Gekozen werd voor de tweede aanpak. De e-mail is het kernbestand en moet altijd beschikbaar zijn voor downstream verwerking. Bijlagen zijn aanvullend en rechtvaardigen niet dat de volledige flow afgebroken wordt bij een individuele fout. Als alle bijlagen falen, wordt alsnog een event gepubliceerd met enkel de e-mail. Als alle bijlagen én de e-mail falen, wordt de verwerking stopgezet en krijgt de gebruiker een foutmelding. Deze strategie zorgt ervoor dat het systeem robuust blijft zonder dat volledigheid ingeleverd wordt ten koste van beschikbaarheid.

2.4.4. Selectie verwerking van bijlagen

2.4.4.1. Onderzochte opties

Vanuit de Outlook add-in is het mogelijk dat een gebruiker niet alle bijlagen van een e-mail wil doorsturen, maar slechts een selectie. Hiervoor moest bepaald worden hoe de Ingest API weet welke bijlagen verwerkt moeten worden.

Alle bijlagen altijd verwerken is de eenvoudigste aanpak: de API haalt via Microsoft Graph alle bijlagen op en slaat ze allemaal op, ongeacht wat de gebruiker geselecteerd heeft in de add-in. Dit vereenvoudigt de logica aan de kant van de Ingest API, maar geeft de gebruiker geen controle over wat er opgeslagen wordt.

Selectie meesturen vanuit de add-in is een aanpak waarbij de add-in de ID's van de geselecteerde bijlagen meestuurt in het request. De Ingest API verwerkt dan enkel die bijlagen waarvan het ID in de selectielijst voorkomt. Dit geeft de gebruiker volledige controle en voorkomt dat ongewenste bestanden in document storage terechtkomen.

Conclusie: Gekozen werd voor de tweede aanpak. De add-in stuurt een lijst van geselecteerde bijlage-ID's mee in het request. De Ingest API filtert op basis van die lijst en verwerkt enkel de bijlagen die expliciet geselecteerd werden. Als de lijst leeg is of niet aanwezig is, worden geen bijlagen verwerkt. Een aandachtspunt dat hierbij opdook, was dat bijlage-ID's afkomstig van Microsoft Graph speciale tekens kunnen bevatten die problemen geven bij transport in een JSON-body. Dit werd opgelost door de ID's te normaliseren voor verzending en ze aan de ontvangende kant terug te converteren naar het originele formaat voordat de Graph API aangesproken wordt.

2.5. Apache Kafka

2.5.1. Inleiding

Apache Kafka fungeert als de communicatielaag tussen de Ingest API en het downstream CSBO-systeem. Nadat de e-mail en bijlagen succesvol opgeslagen zijn in document storage, publiceert de Ingest API een event naar Kafka. Dit event bevat geen binaire bestanden maar enkel metadata en de document-ID's die verwijzen naar de opgeslagen bestanden in document storage. Het CSBO-systeem luistert op het betrokken topic en gebruikt de informatie uit het event om automatisch een taak aan te maken.

Hoewel de keuze voor Apache Kafka op voorhand al vastlag gezien de bestaande infrastructuur binnen H.Essers, waren er tijdens de analyse nog een aantal technische keuzes te maken rond de opzet van de producer, de structuur van het event en de strategie voor foutafhandeling.

2.5.2. Keuze van het messaging platform

2.5.2.1. Onderzochte opties

Voor de asynchrone communicatie tussen de Ingest API en het CSBO-systeem zijn er verschillende platformen beschikbaar die event-driven communicatie ondersteunen.

Apache Kafka is een gedistribueerd event streaming platform dat ontworpen is voor hoge doorvoer en duurzame opslag van events. Kafka bewaart berichten gedurende een configureerbare retentieperiode, wat betekent dat events opnieuw geconsumeerd kunnen worden na een incident. Kafka is binnen H.Essers al operationeel als centraal messaging platform en wordt door meerdere systemen gebruikt.

RabbitMQ is een traditionele message broker die werkt op basis van een push-model. Berichten worden door de broker afgeleverd bij consumers en worden na verwerking verwijderd. RabbitMQ is eenvoudiger op te zetten dan Kafka maar biedt minder garanties rond retentie en herverwerking van berichten.

Azure Service Bus is het managed messaging platform van Microsoft en sluit nauw aan bij de Azure-infrastructuur. Het biedt ingebouwde ondersteuning voor dead letter queues en sessies, maar vereist een Azure-abonnement en introduceert een externe afhankelijkheid buiten de eigen infrastructuur van H.Essers.

Conclusie: Apache Kafka was de enige logische keuze. Het platform is reeds aanwezig binnen de infrastructuur van H.Essers en wordt intern als standaard gebruikt. Alternatieven opzetten zou buiten de scope van de stageopdracht vallen en onnodige beheerscomplexiteit introduceren. Bovendien biedt Kafka met zijn retentiemechanisme en offset-gebaseerde verwerking een sterkere basis voor betrouwbare en herstelbare event-verwerking dan de alternatieven.

2.5.3. Producerconfiguratie en betrouwbaarheid

2.5.3.1. Onderzochte opties

Bij het publiceren van events naar Kafka zijn er verschillende configuratiemogelijkheden die een directe impact hebben op de betrouwbaarheid van de berichtlevering. De centrale vraag is hoeveel garantie de producer geeft dat een bericht daadwerkelijk en slechts één keer aankomt.

Fire-and-forget is de meest eenvoudige aanpak waarbij de producer een bericht verstuurt zonder te wachten op een bevestiging van de Kafka-broker. Dit geeft de hoogste doorvoer maar biedt geen enkele garantie dat het bericht daadwerkelijk verwerkt is. Bij een tijdelijke fout in de broker kan een bericht ongemerkt verloren gaan.

Bevestiging van de leader is een tussenoplossing waarbij de producer wacht tot de broker-leader het bericht bevestigd heeft. Dit biedt meer zekerheid dan fire-and-forget, maar als de leader uitvalt voordat het bericht naar replica's gesynchroniseerd is, kan het bericht alsnog verloren gaan.

Bevestiging van alle replica's is de meest betrouwbare aanpak waarbij de producer wacht tot alle in-sync replica's het bericht ontvangen en bevestigd hebben. Dit verlaagt de doorvoer licht maar garandeert dat een bericht niet verloren gaat zolang minstens één replica beschikbaar blijft. In combinatie met idempotente publicatie wordt bovendien gegarandeerd dat bij retries geen dubbele berichten in het topic terechtkomen.

Conclusie: Gekozen werd voor de meest betrouwbare configuratie met bevestiging van alle replica's en idempotente publicatie. Omdat elke ingest-flow slechts één keer verwerkt mag worden en verlies van een event directe gevolgen heeft voor de taakcreatie in CSBO, weegt de lichte doorvoervermindering niet op tegen de garanties die deze aanpak biedt.

2.5.4. Foutafhandeling en dead letter queue

2.5.4.1. Onderzochte opties

Ondanks een robuuste producerconfiguratie kunnen er zich situaties voordoen waarbij een event niet correct verwerkt kan worden door de consumer. Hiervoor moest een strategie bepaald worden.

Onbeperkt blijven herproberen is een aanpak waarbij de consumer bij elke fout blijft retrien totdat de verwerking slaagt. Dit voorkomt dat events verloren gaan, maar kan de volledige verwerkingspipeline blokkeren als een event structureel fout is en nooit succesvol verwerkt zal worden.

Maximaal aantal retries met doorsturen naar een dead letter queue is een aanpak waarbij de consumer een beperkt aantal keren herprobeert bij een tijdelijke fout. Als de verwerking na het maximale aantal pogingen nog steeds niet geslaagd is, wordt het event doorgestuurd naar een apart topic dat dient als dead letter queue. Dit voorkomt blokkering van de pipeline en maakt het mogelijk om problematische events apart te analyseren en eventueel manueel of geautomatiseerd te herverwerken.

Conclusie: Gekozen werd voor de tweede aanpak met een dead letter queue. Een structureel foutief event mag nooit de verwerking van alle volgende events blokkeren. Door het event door te sturen naar een apart topic blijft de pipeline operationeel terwijl het problematische event bewaard blijft voor verdere analyse. Dit sluit ook aan bij de aanbevelingen uit het design document, waarin een dead letter queue expliciet als onderdeel van de architectuur beschreven staat.

3. Realisatie

3.1. Outlook add-in

3.1.1. inleiding

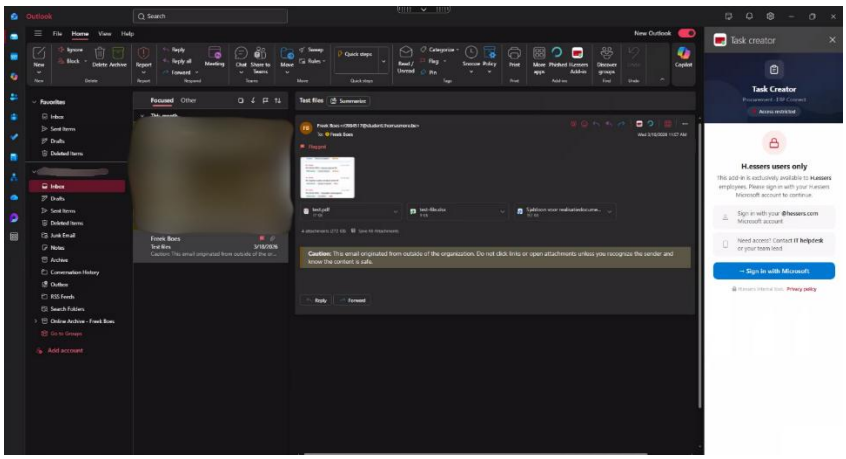
De Outlook add-in is het startpunt van de volledige end-to-end flow. Ze stelt gebruikers in staat om vanuit Outlook een geselecteerde e-mail samen met aanvullende gegevens door te sturen naar het CSBO-platform, zonder de mailclient te verlaten. Dit hoofdstuk beschrijft de opzet van de add-in, de voornaamste functionaliteiten en de technische principes die tijdens de implementatie toegepast werden.

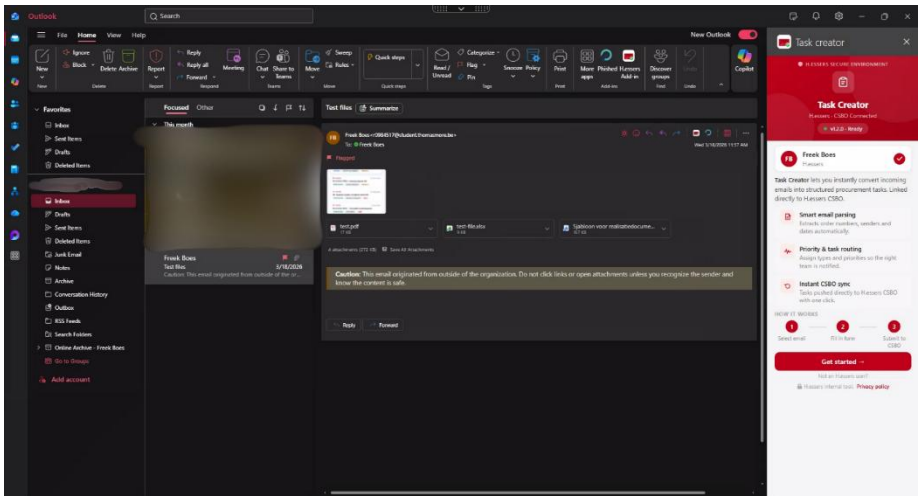
De add-in is gebouwd als een webapplicatie in vanilla JavaScript, HTML en CSS, gebundeld via Webpack. Office.js wordt gebruikt als brug tussen de webapplicatie en de Outlook-client. Dit geeft de add-in toegang tot e-maildata, gebruikersinformatie en de authenticatiemodule van Outlook. De keuze voor vanilla JavaScript zonder extra framework hield de bundlegrootte klein en de toolchain eenvoudig, wat belangrijk is voor een add-in die geladen wordt binnen de beperkte omgeving van het Outlook-taakvensterpaneel.

3.1.2. Opstart en welkomscherm

Bij de allereerste opstart toont de add-in een welkomstoverlay. Dit scherm geeft de gebruiker een kort overzicht van de functionaliteiten van de add-in en bevestigt dat de gebruiker correct aangemeld is via zijn H.Essers account. De naam en initialen van de aangemelde gebruiker worden automatisch ingevuld op basis van het Office-gebruikersprofiel, zodat de gebruiker direct kan zien met welk account hij werkt.

Eenmaal de gebruiker op "Get started" klikt, wordt de welkomstflag opgeslagen in de lokale opslag van de browser. Bij volgende opstart wordt de overlay overgeslagen en komt de gebruiker rechtstreeks op de hoofdinterface terecht. Via de "About" knop in de footer kan de welkomstoverlay altijd opnieuw geopend worden.



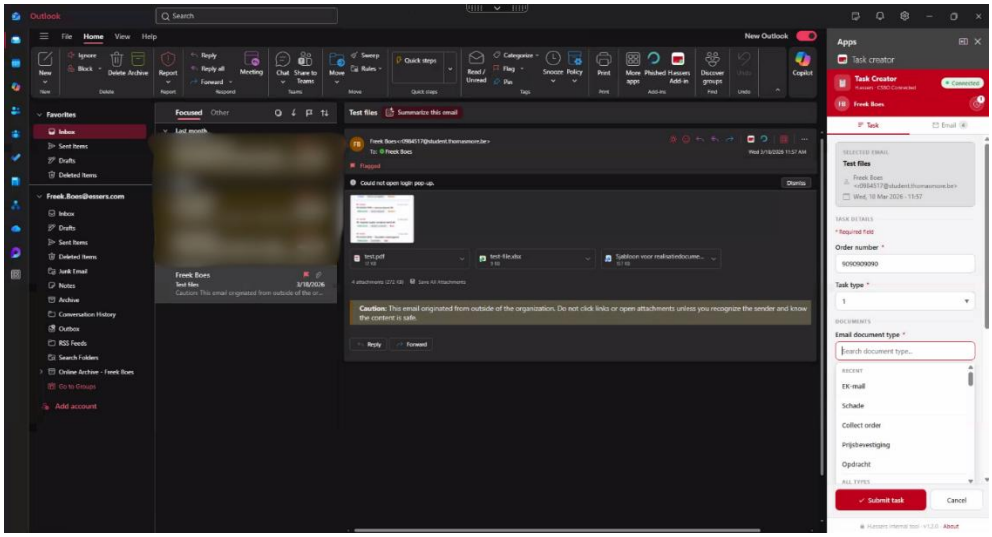


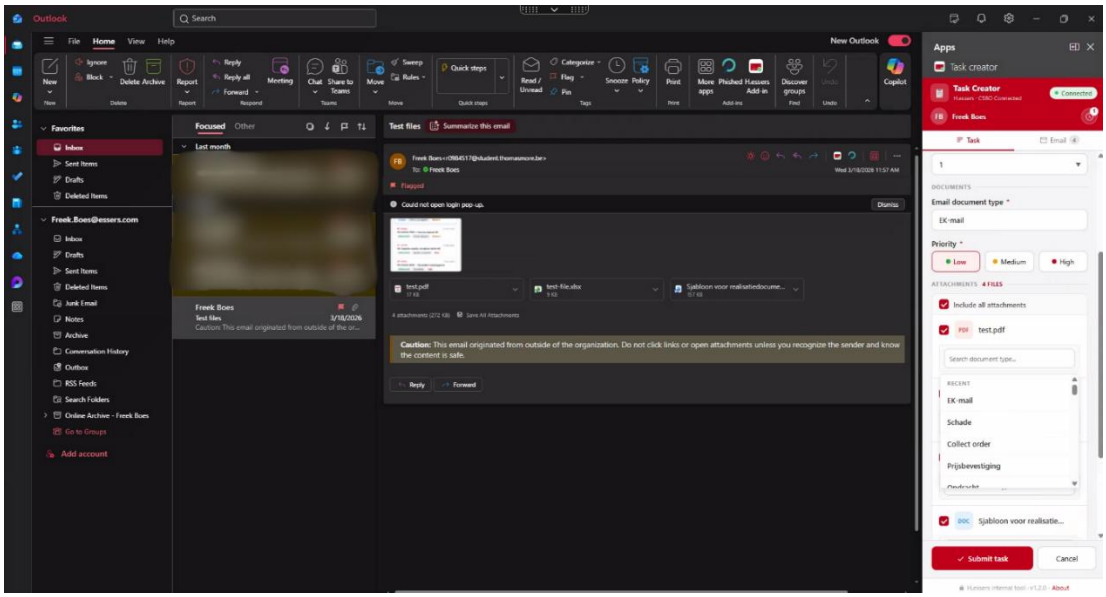
(Zie figuur 2)

3.1.3. Hoofdinterface en tabbladstructuur

De hoofdinterface is opgebouwd rond twee tabbladen: een Task-tab en een Email-tab. Bovenaan de interface bevindt zich een rode header in de huisstijl van H.Essers met de naam en initialen van de aangemelde gebruiker, een verbindingsindicator die aangeeft of de add-in online is, en een knop waarmee de inzendingsgeschiedenis geopend kan worden.

(Zie figuur 3 en 4)

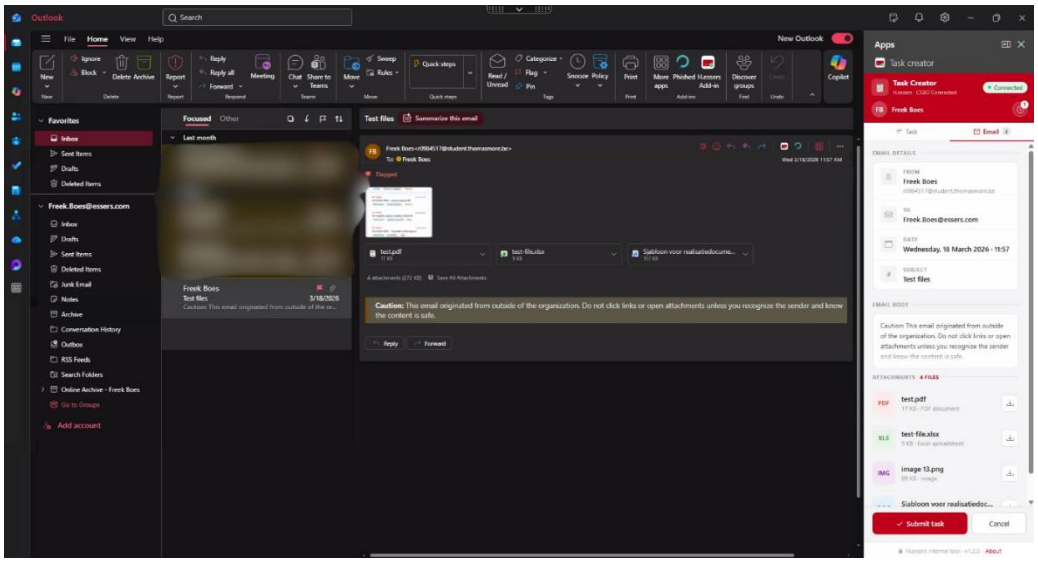




De Task-tab is de primaire werkruimte. Bovenaan wordt een compact overzicht getoond van de geselecteerde e-mail, met het onderwerp, de afzender en de datum. Hieronder bevindt zich het formulier dat de gebruiker invult voor hij de e-mail doorstuurt.

De Email-tab toont de volledige inhoud van de geselecteerde e-mail, inclusief afzender, ontvangers, datum en de volledige berichttekst. Langere e-mails worden ingekort weergegeven met een "Read full email" link waarmee de volledige tekst uitgevouwen kan worden. Bijlagen zijn in deze tab downloadbaar via een downloadknop per bestand. De inhoud van de Email-tab wordt lazy geladen, wat betekent dat de mailinhoud pas via Office.js opgehaald wordt op het moment dat de gebruiker de tab voor het eerst activeert. Dit vermijdt onnodige API-aanroepen wanneer de gebruiker de tab nooit opent.

(Zie figuur 5)



3.1.4. Formulier en validatie

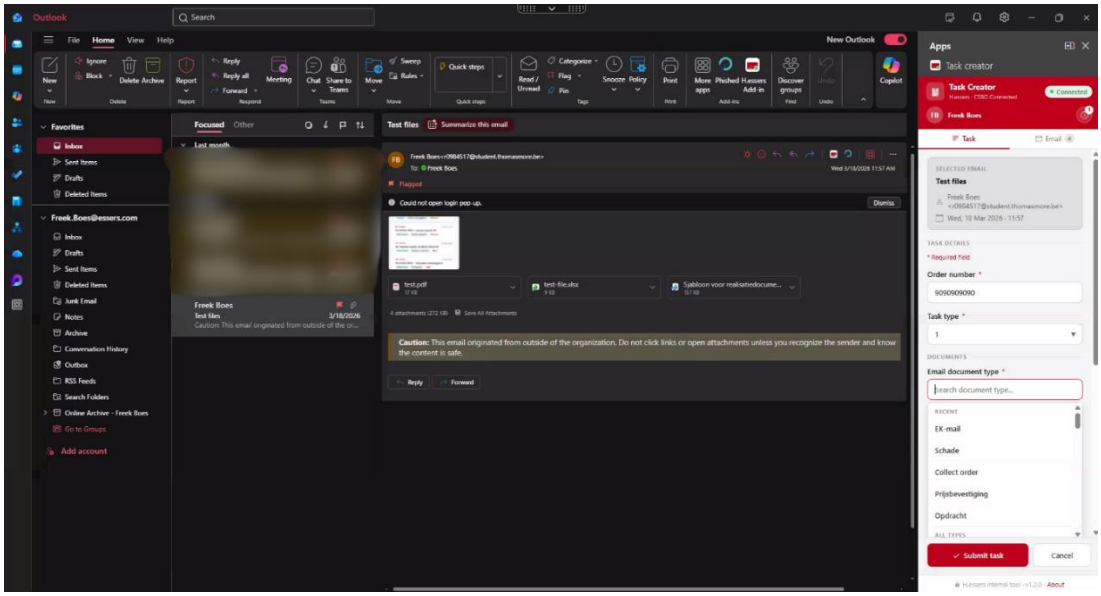
Het formulier op de Task-tab bestaat uit een ordernummer, een taaktype, een prioriteit en een optioneel commentaarveld. Het ordernummer en het commentaar zijn vrije tekstvelden. Het taaktype wordt gekozen via een dropdown. De prioriteit wordt ingesteld via drie knoppen: Low, Medium en High, elk voorzien van een gekleurde indicator.

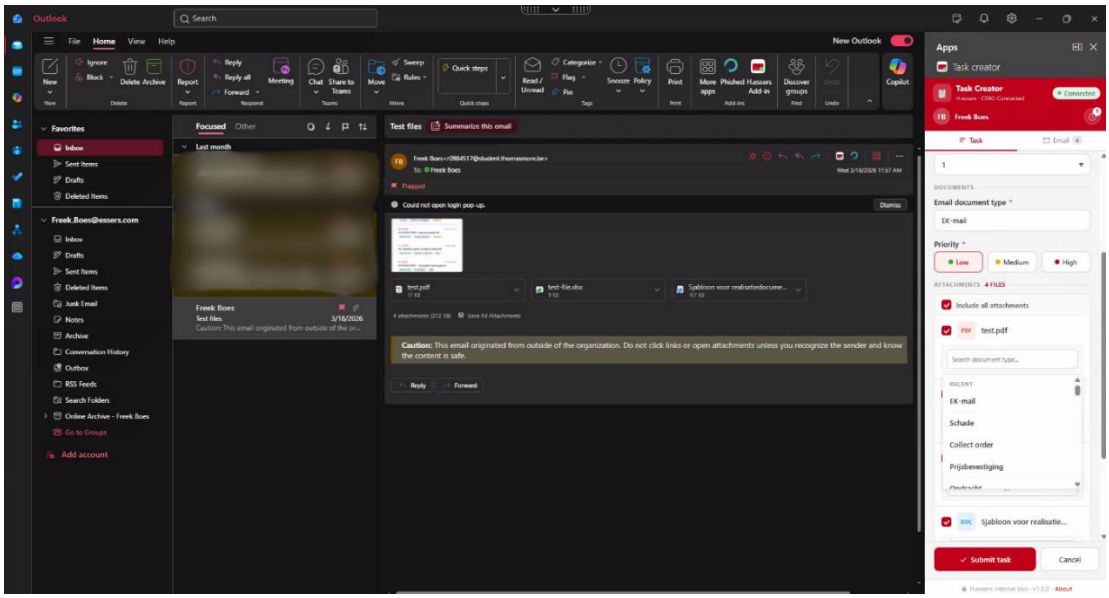
Bij het indienen van het formulier worden alle verplichte velden gevalideerd. Velden die leeg zijn of een ongeldige waarde bevatten worden rood omrand en krijgen een foutmelding onder het veld. Het formulier wordt pas ingediend als alle validaties slagen.

3.1.5. Bijlagenselectie

Wanneer de geselecteerde e-mail bijlagen bevat, verschijnt er op de Task-tab een bijlagensectie. Per bijlage wordt een gekleurde bestandsbadge getoond op basis van het bestandstype, samen met de bestandsnaam. Via een checkbox per bijlage kan de gebruiker kiezen welke bijlagen meegestuurd worden naar de Ingest API. Een "Include all attachments" checkbox bovenaan laat toe om alle bijlagen in één klik te selecteren of deselecteren. Wanneer slechts een deel van de bijlagen geselecteerd is, toont de teller in de sectieheader hoeveel bijlagen geselecteerd zijn ten opzichte van het totaal.

(Zie figuur 3 en 4)





3.1.6. Indienen en statusopvolging

Na het klikken op de "Submit task" knop wordt het formulier gevalideerd en wordt er een JWT access token opgehaald via de ingebouwde Office.js authenticatiemodule. Vervolgens wordt een POST-request verstuurd naar de Gravitee gateway met het token, de message ID van de geselecteerde e-mail, de geselecteerde bijlagen en de ingevulde formuliergegevens.

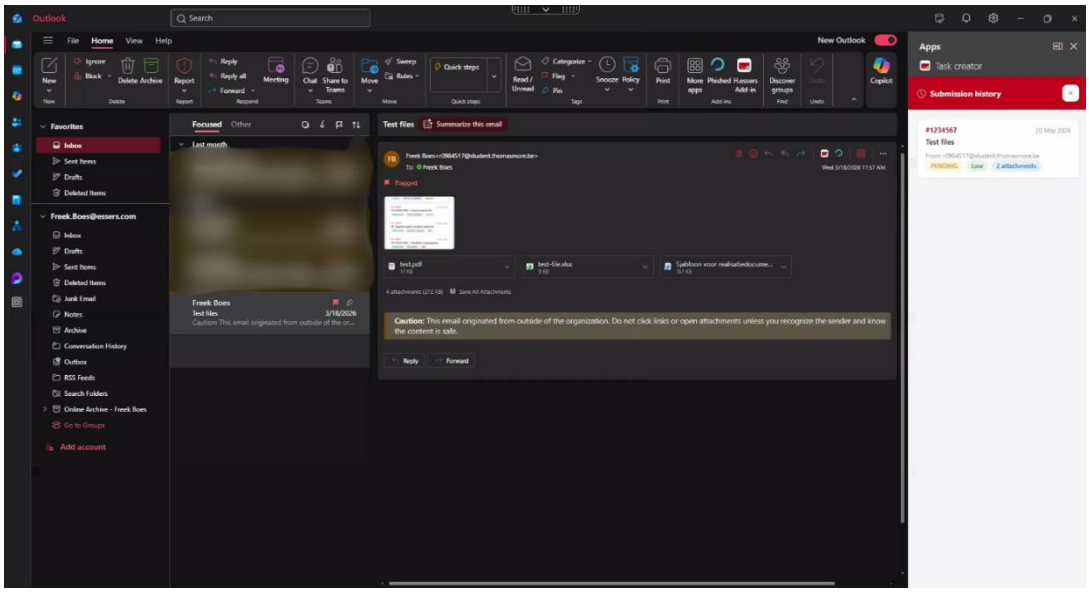
Na een succesvolle ontvangst door de gateway geeft de Ingest API een correlatie-ID terug. De add-in gebruikt dit ID om periodiek de status op te vragen via een polling-mechanisme. Zolang de verwerking bezig is, toont een laadmelding de huidige status aan de gebruiker. Zodra het CSBO-systeem de taak heeft aangemaakt, verschijnt er een groene bevestigingsbanner met een directe link naar de aangemaakte taak.

Bij een fout, bijvoorbeeld een verlopen token of een onbereikbare backend, toont de add-in een rode foutmelding met een beschrijving van het probleem. De gebruiker kan het formulier daarna opnieuw indienen.

3.1.7. Inzendingsgeschiedenis

Via de klokknop in de rechterbovenhoek van de header kan de gebruiker de inzendingsgeschiedenis openen. Deze overlay toont een overzicht van alle eerder ingediende formulieren, met per inzending het ordernummer, het onderwerp van de e-mail, het taaktype, de prioriteit en het tijdstip van indiening. De geschiedenis wordt lokaal opgeslagen en blijft beschikbaar tussen sessies in.

(Zie figuur 6)



3.1.8. Offline detectie

De add-in detecteert automatisch of er een internetverbinding beschikbaar is. De verbindingsindicator in de header wisselt tussen "Connected" en "Offline" naargelang de netwerkstatus. Wanneer de gebruiker het formulier probeert in te dienen zonder actieve verbinding, wordt de inzending geblokkeerd en verschijnt er een melding die de gebruiker vraagt de verbinding te controleren.

3.1.9. Jenkins pipeline en containerisatie

Dockerfile
Ook de Outlook add-in wordt gecontaineriseerd via een multi-stage Dockerfile, zij het met een andere opzet dan de Ingest API. De add-in is een statische webapplicatie die geen server-side runtime vereist.

In de eerste fase wordt de applicatie gebouwd via een Node.js 20 Alpine image. De dependencies worden eerst apart geïnstalleerd via npm ci op basis van het package-lock.json bestand, wat reproduceerbare builds garandeert. Pas daarna worden de bronbestanden gekopieerd. Net als bij de Ingest API zorgt deze volgorde ervoor dat Docker de geïnstalleerde node modules cachet zolang de package bestanden niet wijzigen. Het .env bestand wordt bewust niet gekopieerd naar het image. Gevoelige configuratiewaarden zoals de Gravitee API key worden via Webpack's DefinePlugin tijdens het bouwen als omgevingsvariabele ingebakken in de browser-bundle.

In de tweede fase worden de gegenereerde statische bestanden uit de dist map gekopieerd naar een nginx Alpine image. nginx dient de browser-bundle op poort 80. HTTPS-terminatie gebeurt buiten de container, hetzij via een Kubernetes ingress controller met cert-manager, hetzij via Azure Front Door. Dit is een bewuste architectuurkeuze: de container zelf houdt zich enkel bezig met het serveren van statische bestanden en delegeert TLS-beheer aan de omgeving eromheen.

Jenkin pipeline
De Jenkins pipeline voor de Outlook add-in is opgezet als een parameterized pipeline waarbij bij elke build gekozen wordt naar welke omgeving gedeployed wordt: development, test of productie. Dit verschilt van de Ingest API pipeline waar de omgeving bepaald wordt door de branch. De keuze voor een parameterized aanpak geeft meer flexibiliteit bij het uitrollen naar een specifieke omgeving zonder dat hiervoor een aparte branch aangemaakt hoeft te worden.

Na de checkout wordt een versiesleutel samengesteld op basis van het buildnummer en de korte Git commit hash. Deze sleutel maakt elke build uniek traceerbaar. Vervolgens worden de dependencies geïnstalleerd, wordt de code gelint en worden de Jest-testen uitgevoerd. Testresultaten worden als JUnit-rapport gepubliceerd in Jenkins en de coverage-rapporten worden als HTML-artefact bewaard zodat ze via de Jenkins interface raadpleegbaar zijn.

Bij de bouwstap injecteert Webpack automatisch de juiste API base URL, hosting URL en Gravitee API key op basis van de gekozen omgeving. Het manifest wordt eveneens door Webpack aangepast zodat de hosting URL erin correct ingesteld is per omgeving. De gegenereerde dist map inclusief het aangepaste manifest wordt als Jenkins artefact opgeslagen.

De webapplicatie wordt gedeployed naar een Azure Static Web App via de Azure CLI. Per omgeving is er een aparte Azure Static Web App instantie voorzien. Voor development en test wordt het manifest als sideloading-bestand beschikbaar gesteld via Jenkins artefacten, zodat het manueel ingeladen kan worden in Outlook. Voor de productieomgeving is een manuele goedkeuringsstap voorzien waarna het manifest automatisch uitgerold wordt via Microsoft 365 Centralized Deployment met behulp van een PowerShell script. Bij een mislukte pipeline wordt automatisch een e-mail verstuurd naar de verantwoordelijke collega.

3.1.10. Documentatie

Naast de technische implementatie werden voor de Outlook add-in twee afzonderlijke documenten opgesteld om de overdraagbaarheid van het project te garanderen.
De setup guide beschrijft stap voor stap hoe een nieuw Outlook add-in project opgebouwd kan worden met dezelfde technologiestack. Het document behandelt de volledige configuratieketen: het aanmaken van een nieuw project via Yeoman scaffolding, het configureren van de Azure Entra ID applicatieregistratie voor SSO via NAA, en het instellen van de Gravitee API Gateway voor JWT-validatie en routering. Het document is bewust technologie-generiek opgesteld zodat het als referentie kan dienen voor elk toekomstig add-in project dat dezelfde stack gebruikt. Projectspecifieke waarden worden consequent aangeduid met punthaken zodat duidelijk is wat vervangen moet worden bij een nieuwe implementatie.
De deployment guide beschrijft hoe de add-in gebouwd, verpakt en uitgerold wordt. Het document maakt een duidelijk onderscheid tussen de deployment van de webapplicatie naar Azure Static Web Apps en de uitrol van het manifest via Microsoft 365 Centralized Deployment. Per omgeving wordt beschreven welke stappen de Jenkins pipeline uitvoert en hoe het manifest correct geconfigureerd moet zijn. Voor de productieomgeving wordt ook de goedkeuringsstap en de automatische manifestuitrol via PowerShell toegelicht.

3.1.11. Codekwaliteit en testen

Om de kwaliteit van de codebase te bewaken werd SonarQube ingezet als tool voor statische codeanalyse. SonarQube analyseert de broncode op mogelijke problemen zoals code smells, beveiligingskwetsbaarheden en kwaliteitsissues. Concrete verbeteringen die op basis van SonarQube-rapporten doorgevoerd werden, waren onder andere het vervangen van innerHTML door veilige DOM-methoden voor het renderen van gebruikersgestuurde content zoals bestandsnamen en e-mailonderwerpen. Dit voorkomt potentiële XSS-kwetsbaarheden waarbij kwaadaardige inhoud in de DOM geïnjecteerd zou kunnen worden. Daarnaast werden ontbrekende null-checks en andere kwaliteitsissues opgespoord en opgelost.

Naast de statische analyse werden er ook unit- en integratietesten geschreven met behulp van Jest. De unit testen focussen op de afzonderlijke functies van de add-in, zoals formuliervalidatie, prioriteitsselectie, datumopmaak en het beheer van de inzendingsgeschiedenis. De integratietesten controleren of de samenwerking tussen de verschillende onderdelen correct verloopt, bijvoorbeeld het volledige submit-proces van formuliervalidatie tot het versturen van het request. Bij het schrijven van de testen werd gefocust op het testen van logica met echte waarden in plaats van het opblazen van de testdekking met oppervlakkige mocks. Branch coverage werd hierbij als de meest waardevolle metric beschouwd, omdat het aantoont of alle mogelijke paden door de code effectief getest worden.

3.1.12. Toegepaste principes en patronen

Tijdens de ontwikkeling van de add-in werden een aantal concrete principes en patronen toegepast om de kwaliteit, veiligheid en onderhoudbaarheid van de code te bewaken.

Single Responsibility. Elke functie in de codebase heeft één duidelijk afgebakende verantwoordelijkheid. Zo is er een aparte functie voor het valideren van het formulier, voor het ophalen van het access token, voor het versturen van het request en voor het beheren van de inzendingsgeschiedenis. Dit maakt de code gemakkelijker te begrijpen, te testen en aan te passen zonder onbedoelde neveneffecten.

Gecentraliseerde event registratie. Alle event listeners worden geregistreerd op één centrale plaats, namelijk binnen de Office.onReady callback die uitvoert zodra Office.js volledig geladen is. Er worden nergens inline event handlers in de HTML gebruikt. Om te voorkomen dat event listeners meerdere keren geregistreerd worden wanneer Outlook het script opnieuw laadt, wordt een data-initialized vlag op de submitknop gezet na de eerste registratie. Dit is een concreet voorbeeld van defensief programmeren in de context van een Office.js add-in, waar het lifecycle-gedrag anders is dan in een gewone webpagina.

Beveiliging via DOM-methoden. Alle dynamisch gegenereerde inhoud zoals bestandsnamen, e-mailonderwerpen en gebruikersnamen wordt via DOM-methoden aan de pagina toegevoegd in plaats van via innerHTML. Dit voorkomt dat kwaadaardige inhoud in de DOM geïnjecteerd kan worden, een principe dat ook door SonarQube als aandachtspunt werd geïdentificeerd en aanleiding gaf tot concrete aanpassingen in de code.

Guard clauses en vroege terugkeer. Functies die afhankelijk zijn van externe toestand, zoals de aanwezigheid van een mail item of een DOM-element, controleren dit aan het begin van de functie en keren vroeg terug indien de verwachte toestand niet aanwezig is. Dit vermijdt diep geneste if-structuren en maakt de controlestroom van de code duidelijker leesbaar.

Asynchrone verwerking met async/await. De communicatie met de Gravitee gateway en de statusopvolging via polling verlopen volledig asynchroon. Door gebruik te maken van async/await in combinatie met try/catch blokken blijft de code leesbaar en wordt foutafhandeling op een consistente manier toegepast doorheen de volledige verwerkingsketen.

Lazy loading. De inhoud van de Email-tab wordt pas geladen wanneer de gebruiker de tab voor het eerst activeert. Een vlag houdt bij of de tab al geladen is, zodat de mailinhoud niet opnieuw opgehaald wordt bij elk wisselen van tab. Dit beperkt het aantal Office.js API-aanroepen en verbetert de responsiviteit van de add-in.

3.2. Gravitee Gateway API

3.2.1. Inleiding

De Gravitee API Gateway fungeert als de centrale toegangspoort tussen de Outlook add-in en de interne Ingest API. Alle requests die de add-in verstuurt verlopen verplicht via de gateway, die instaat voor beveiliging, validatie en routering. De Ingest API zelf is niet rechtstreeks bereikbaar van buitenaf, wat het aanvalsoppervlak beperkt en het beveiligingsbeheer centraliseert.

3.2.2. Opzet en configuratie

De gateway is bereikbaar via een intern endpoint dat geconfigureerd werd binnen de bestaande Gravitee-instantie van H.Essers, die draait binnen het interne Kubernetes-cluster. De configuratie omvat een API-definitie met de bijhorende policies voor beveiliging en routering.

Binnen de gateway werd een policy geconfigureerd die inkomende requests valideert op de aanwezigheid van een geldige API-sleutel in de request header. Requests die geen geldige sleutel bevatten worden door de gateway geweigerd voordat ze de Ingest API bereiken. Enkel requests met een correcte sleutel worden doorgestuurd naar de interne backend.

3.2.3. Routering naar de Ingest API

Na succesvolle validatie routeert de gateway het inkomende request door naar de Ingest API die intern binnen het Kubernetes-cluster draait. Een belangrijk aandachtspunt hierbij was dat het adres localhost in een Gravitee target URL niet verwijst naar de ontwikkelmachine van de ontwikkelaar, maar naar de Kubernetes-server zelf. Het interne service-adres van de Ingest API binnen het cluster diende daarom correct geconfigureerd te worden als doelendpoint.

Wanneer de gateway het request succesvol doorstuurt maar de Ingest API niet bereikbaar is, retourneert Gravitee een 502 Bad Gateway. Dit onderscheid was nuttig tijdens de ontwikkeling: een 502 betekent dat de gateway-configuratie en de policies correct zijn, maar dat de backend niet bereikbaar is. Dit maakte het eenvoudiger om problemen te diagnosticeren en te scheiden van gateway-configuratiefouten.

3.2.4. JWT-validatie en tijdelijke oplossing

Tijdens de configuratiefase werd uitgebreid onderzocht hoe het Azure Entra ID access token dat de Outlook add-in meestuurt gevalideerd kon worden via de ingebouwde JWT-validatiepolicy van Gravitee. De aanbevolen aanpak hiervoor is dat de gateway de publieke sleutels van Azure Entra ID dynamisch ophaalt via de JWKS-endpoint van Microsoft, waarna de handtekening van elk binnenkomend token automatisch gecontroleerd wordt.

In de praktijk bleek dit niet realiseerbaar binnen de bestaande infrastructuur. De Kubernetes-omgeving waarin Gravitee draait heeft geen uitgaand verkeer naar externe domeinen, waardoor de gateway-pod de JWKS-endpoint van Microsoft niet kon bereiken. Een statische configuratie van het publieke certificaat als tussenoplossing liep op dezelfde beperking vast tijdens de configuratiefase.

Omdat dit een infrastructuurprobleem betrof dat buiten de scope van de stageopdracht lag, werd in overleg gekozen voor een API key als werkbare tijdelijke oplossing. De JWT-validatie blijft hierdoor voorlopig beperkt tot de Ingest API zelf, die het token verder verwerkt via de On-Behalf-Of flow richting Microsoft Graph. Voor een productieomgeving dient de volwaardige JWT-validatie in de gateway alsnog geactiveerd te worden zodra de nodige outbound netwerkpermissies door de infrastructuurbeheerder zijn opengesteld.

3.3. Ingest API

3.3.1. Opbouw van het project

De Ingest API is gebouwd als een Spring Boot applicatie in Java 21 en draait achter de Gravitee API Gateway. Het project volgt een gelaagde structuur die opgedeeld is in duidelijk afgebakende pakketten: controller, service, dto, event, model, exception en util. Elk pakket heeft een specifieke verantwoordelijkheid en de lagen communiceren enkel via duidelijke interfaces en dataobjecten, zonder directe afhankelijkheden over meerdere lagen heen.

De configuratie van externe afhankelijkheden zoals MSAL, document storage en Kafka is centraal ondergebracht in het config-pakket. Dit zorgt ervoor dat alle Spring beans op één plaats geconfigureerd worden en de serviceklassen zelf geen configuratielogica bevatten. Gevoelige configuratiewaarden zoals certificaatpaden, wachtwoorden en service-URL's worden via omgevingsvariabelen meegegeven en zijn niet opgenomen in de codebase.

3.3.2. Endpoints en controller

De IngestController is het enige toegangspunt van de API en beheert vier endpoints onder het basispad **/v1/ingest**. Het hoofdendpoint **POST /v1/ingest/messages** start de volledige verwerkingsflow: Graph-ophaling, opslag in document storage en publicatie naar Kafka. Naast dit hoofdendpoint zijn er twee preview-endpoints ontwikkeld die gebruikt worden tijdens de integratie en het testen. Het eerste preview-endpoint voert de volledige flow uit inclusief opslag in document storage maar publiceert nog geen event naar Kafka. Het tweede preview-endpoint haalt de e-mail enkel op via Microsoft Graph en slaat het resultaat lokaal op, zonder interactie met document storage of Kafka. Deze twee endpoints hebben het mogelijk gemaakt om elke integratiestap afzonderlijk te verifiëren voordat de volledige flow in gebruik genomen werd. Ten slotte is er een statusendpoint **GET /v1/ingest/status/{correlationId}** waarmee de Outlook add-in de verwerkingsstatus kan opvragen nadat een request ingediend werd.

Bij elke binnenkomende request genereert de controller een unieke correlationId die doorheen de volledige verwerkingsketen meegegeven wordt. Deze identifier maakt het mogelijk om een specifieke ingest-aanvraag te volgen over alle logregels en systemen heen. De controller extraheert ook de JWT-claims uit de Authorization-header en geeft zowel de ruwe token als de geparste claims door aan de service. De validatie van het token zelf gebeurt bij Gravitee; de controller decodeert enkel de payload om de gebruikersinformatie beschikbaar te maken voor de verdere verwerking.

3.3.3. JWT-verwerking

De JwtClaimsExtractor decodeert de payload van het Bearer token zonder dit opnieuw te valideren, aangezien die validatie al door Gravitee uitgevoerd werd. Uit de payload worden drie waarden geëxtraheerd: de object-ID van de gebruiker, het UPN-adres en de tenant-ID. Voor het UPN-adres worden meerdere claims geprobeerd in volgorde van betrouwbaarheid: eerst preferred_username, dan upn en tot slot unique_name. Als geen van deze claims beschikbaar is, wordt een weergavenaam afgeleid uit het e-mailadres. Dit zorgt ervoor dat de API robuust omgaat met tokens die afhankelijk van de configuratie van de Azure-applicatieregistratie licht kunnen verschillen in structuur.

3.3.4. On-Behalf-Of flow

De MsalOboService is verantwoordelijk voor het omwisselen van het gebruikerstoken naar een Microsoft Graph access token via de OAuth2 On-Behalf-Of flow. Het ontvangen SSO-token wordt als UserAssertion meegegeven aan de MSAL-bibliotheek, die op basis daarvan een nieuw delegated token aanvraagt bij Azure Entra ID met de Graph-scope als doelscope. MSAL beheert intern een token cache, waardoor herhaalde token-aanvragen voor dezelfde gebruiker vermeden worden. De ConfidentialClientApplication die door MSAL gebruikt wordt, is geconfigureerd met een certificaat in .pfx-formaat conform het beveiligingsbeleid van H.Essers.

3.3.5. E-mail ophalen via Microsoft Graph

De GraphService gebruikt het delegated Graph-token om de e-mail op te halen. Per request worden drie afzonderlijke Graph-calls uitgevoerd: een eerste call haalt de berichtmetadata op zoals afzender, ontvangers, onderwerp en tijdstempels, een tweede call haalt de volledige MIME-inhoud op als binaire byte-stroom in EML-formaat, en een derde call haalt de bijlagen op als deze aanwezig zijn op basis van het hasAttachments-veld uit de metadata.

De service ondersteunt zowel de primaire mailbox van de gebruiker als gedeelde mailboxen. Wanneer een mailboxHint aanwezig is in het request, worden de Graph-calls omgeleid naar het /users/{mailbox}-pad in plaats van het standaard /me-pad. Omdat message-ID's van Microsoft Graph speciale tekens kunnen

bevatten die problemen veroorzaken in URL's, worden deze ID's geëncodeerd voor gebruik in de Graph-endpoints.

De Graph SDK vereist een TokenCredential-implementatie als authenticatiemechanisme. Omdat het token echter al via OBO verkregen werd en niet opnieuw aangevraagd hoeft te worden, is een eenvoudige StaticTokenCredential geïmplementeerd die het reeds beschikbare token teruggeeft zonder een nieuwe aanvraag te doen.

3.3.6. Bijlageverwerking en filtering

Nadat de e-mail opgehaald is, worden de bijlagen gefilterd op basis van de selectedAttachmentIds die de Outlook add-in meestuurt in het request. Dit veld kent drie mogelijke waarden: wanneer het veld afwezig is worden alle bijlagen verwerkt, wanneer het een lege lijst is worden geen bijlagen verwerkt, en wanneer het specifieke ID's bevat worden enkel de overeenkomende bijlagen verder verwerkt.

Een aandachtspunt dat tijdens de implementatie opdook, was een inconsistentie in de Base64-codering van bijlage-ID's. Microsoft Graph levert ID's aan in URL-safe Base64-formaat waarbij koppeltekens en underscores gebruikt worden, terwijl het JavaScript-deel van de Outlook add-in de standaard Base64-tekens met plussen en slashes kan produceren. Beide varianten worden voor de vergelijking genormaliseerd naar hetzelfde formaat, zodat de filtering correct werkt ongeacht welk formaat de add-in aanlevert.

3.3.7. Opslag in document storage

De DocumentStorageServiceImpl verzorgt de uploads naar de document storage API van H.Essers. Zoals eerder toegelicht in de analysebespreking ondersteunt de document storage API het EML-formaat niet. Elke e-mail wordt daarom eerst omgezet van EML naar MSG via de EmlToMsgConverter voordat de upload plaatsvindt. Bijlagen worden zonder conversie opgeslagen met hun originele bestandsnaam en content-type.

Elke upload wordt verstuurd als een multipart/form-data request via WebClient. Naast het bestand worden ook een type-code, een padcode en een bestandseigenaar als queryparameters meegegeven. Voor e-mails wordt type-code 024 gebruikt, voor bijlagen type-code 212. Als antwoord geeft de document storage API een bestandsidentifier terug die als referentie opgenomen wordt in het Kafka-event.

Na elke upload berekent de ChecksumCalculator een SHA-256 hash van de bestandsbytes. Deze checksum wordt opgenomen in het Kafka-event zodat downstream systemen de integriteit van opgeslagen bestanden kunnen verifiëren.

Wanneer een individuele bijlage-upload mislukt, wordt de fout gelogd maar wordt de verwerking niet onderbroken. De overige bijlagen en de e-mail zelf worden gewoon verder verwerkt. Enkel wanneer de e-mail zelf niet opgeslagen kan worden, stopt de verwerking volledig.

3.3.8. EML naar MSG-conversie

De EmlToMsgConverter zet een EML-bestand om naar het MSG-formaat dat door Outlook gebruikt wordt. De MIME-inhoud van de e-mail wordt ingelezen via de angus-mail bibliotheek, waarna de relevante velden zoals afzender, ontvangers, onderwerp en berichttekst uitgelezen worden. Deze gegevens worden vervolgens weggeschreven als MAPI-properties in een OLE2 Compound Document structuur via de Apache POI bibliotheek, wat het native containerformaat is van MSG-bestanden.

Een specifiek aandachtspunt tijdens de implementatie was dat Outlook geconverteerde MSG-bestanden als concept opende in plaats van als ontvangen bericht. Dit werd veroorzaakt door een ontbrekende berichtstatus in de MAPI-properties. Door de MSGFLAG_READ property expliciet in te stellen wordt het bericht bij openen correct als een gelezen ontvangen bericht herkend en niet als een onafgewerkt concept.

3.3.9. Deduplicatie

De DeduplicationService detecteert dubbele ingest-aanvragen op basis van een combinatie van tenant-ID, mailboxadres en message-ID. Bij elke nieuwe aanvraag wordt gecontroleerd of deze combinatie al eerder verwerkt werd. Als dit het geval is, wordt een DuplicateMessageException gegooid en wordt de verwerking stopgezet zonder dat er een event gepubliceerd wordt. Als de combinatie nieuw is, wordt ze geregistreerd als verwerkt.

De huidige implementatie slaat de verwerkte sleutels op in een gesynchroniseerde in-memory map met een maximum van tienduizend entries. Oudste entries worden automatisch verwijderd wanneer dit maximum bereikt wordt. Dit is een tijdelijke oplossing die voldoende is voor de huidige fase van het project. Voor een productieomgeving dient deze implementatie vervangen te worden door een Redis-backed oplossing zodat de deduplicatiestatus ook na een herstart van de applicatie bewaard blijft en gedeeld kan worden over meerdere instanties.

3.3.10. Jenkins pipeline en conatinerasatie

Dockerfile
De Ingest API wordt gecontaineriseerd via een multi-stage Dockerfile. Dit betekent dat het buildproces en de uiteindelijke runtime-omgeving strikt gescheiden worden in twee afzonderlijke fases.

In de eerste fase wordt de applicatie gebouwd met behulp van een Maven-image met JDK 21. De afhankelijkheden worden in een aparte stap gedownload voordat de bronbestanden gekopieerd worden. Dit zorgt ervoor dat Docker de gedownloade afhankelijkheden cachet zolang het pom.xml bestand niet wijzigt, wat de bouwtijd bij herhaalde builds aanzienlijk verkort. De Maven settings.xml met de Nexus-credentials wordt via een BuildKit secret gemount tijdens het bouwen en wordt nooit opgenomen in het uiteindelijke image. Dit is een bewuste beveiligingskeuze: gevoelige credentials horen niet thuis in een Docker image dat mogelijk in een register opgeslagen wordt.

In de tweede fase wordt enkel de gecompileerde JAR gekopieerd naar een slanker runtime-image op basis van Eclipse Temurin JRE 21. De volledige Maven-installatie en de bronbestanden zijn niet aanwezig in het uiteindelijke image, wat het image kleiner en veiliger maakt. Er wordt een niet-root gebruiker aangemaakt waaronder de applicatie draait, conform de beveiligingsrichtlijnen voor containerisatie. Er wordt ook een aparte map voorzien voor het MSAL-certificaat, bedoeld om in Kubernetes als een secret volume gemount te worden zodat het certificaat nooit in het image zelf terechtkomt.

Jenkins pipeline
De Jenkins pipeline voor de Ingest API bestaat uit een aantal opeenvolgende stappen. Na het ophalen van de broncode via de checkout-stap wordt de applicatie gebouwd met Maven waarbij de tests overgeslagen worden. De testresultaten worden in een aparte stap uitgevoerd en gepubliceerd als JUnit-rapporten in Jenkins. Vervolgens wordt het Docker image gebouwd waarbij de Maven settings.xml via een BuildKit

secret meegegeven wordt zodat de Nexus-dependencies opgehaald kunnen worden zonder de credentials in het image op te slaan.

De pipeline ondersteunt drie omgevingen: development, test en productie. De deployment naar development wordt getriggerd vanuit de develop branch, de deployment naar test vanuit de test branch en de deployment naar productie vanuit de main branch. Voor de productieomgeving is een manuele goedkeuringsstap voorzien waarbij expliciet bevestigd moet worden voor de deployment doorgaat. Alle gevoelige configuratiewaarden zoals MSAL-credentials, Kafka-bootstrapservers en document storage-gegevens worden via Jenkins credentials geïnjecteerd als omgevingsvariabelen en zijn nergens hardcoded in de pipeline aanwezig.

De deployment naar het Kubernetes-cluster en het pushen naar het Azure Container Registry staan klaar in de pipeline maar zijn voorlopig uitgecommentarieerd in afwachting van de infrastructuurconfiguratie door de verantwoordelijke collega. Na elke pipeline-run wordt het lokale Docker image verwijderd om schijfruimte op de Jenkins agent vrij te maken.

3.3.11. Documentatie

Ook voor de Ingest API werden twee afzonderlijke documenten opgesteld.

De setup guide beschrijft stap voor stap hoe de Ingest API opgebouwd kan worden met dezelfde technologiestack. Het document behandelt de volledige configuratieketen: het opzetten van het Maven en Spring Boot project, het configureren van de MSAL OBO-flow met een certificaat in .pfx-formaat, de integratie met Microsoft Graph voor het ophalen van e-mails en bijlagen, de EML naar MSG-conversie voor opslag in document storage en het publiceren van events naar Apache Kafka. Net als bij de add-in setup guide is dit document technologie-generiek opgesteld en zijn projectspecifieke waarden consequent aangeduid met punthaken.

De deployment guide beschrijft hoe de Ingest API gebouwd, gecontaineriseerd en uitgerold wordt. Het document maakt een duidelijk onderscheid tussen het bouwen en pushen van het Docker image naar het Azure Container Registry en de uitrol naar het Kubernetes-cluster. Per omgeving wordt beschreven welke Jenkins pipeline-stappen uitgevoerd worden, hoe de Kubernetes Secrets geconfigureerd moeten worden voor het MSAL-certificaat en de overige gevoelige configuratiewaarden, en hoe de samenwerking met de infrastructuurbeheerder verloopt voor de onderdelen die buiten de scope van de pipeline vallen.

3.3.12. Toegepaste principes en patronen

Tijdens de ontwikkeling van de Ingest API werden een aantal concrete principes en patronen toegepast om de kwaliteit, betrouwbaarheid en onderhoudbaarheid van de code te bewaken.

Single Responsibility. Elke klasse heeft één duidelijk afgebakende verantwoordelijkheid. De IngestController ontvangt requests en geeft responses terug, de IngestService coördineert de verwerkingsstappen, de GraphService communiceert met Microsoft Graph, de MsalOboService wisselt tokens uit en de JwtClaimsExtractor decodeert enkel de JWT-payload. Dit zorgt ervoor dat elke klasse onafhankelijk aangepast of getest kan worden zonder impact op de rest van de applicatie.

Constructor-injectie. Alle afhankelijkheden tussen klassen worden via de constructor meegegeven. Dit maakt de afhankelijkheden van een klasse onmiddellijk zichtbaar en vereenvoudigt het schrijven van unit tests, omdat een testinstantie aangemaakt kan worden door de gewenste afhankelijkheden rechtstreeks mee te geven zonder de volledige Spring-context op te starten.

Optionele Kafka-afhankelijkheid. De KafkaPublisherService wordt als Optional geïnjecteerd in de IngestService. Dit zorgt ervoor dat de volledige verwerkingsflow inclusief Graph-ophaling en document

storage-opslag functioneel blijft wanneer Kafka niet beschikbaar is, wat het lokaal ontwikkelen en testen aanzienlijk vereenvoudigt.

Record-gebaseerde dataobjecten. Alle dataobjecten zoals IngestRequest, FormFields, EmailData, StorageResult en IngestEvent zijn geïmplementeerd als Java records. Records zijn onveranderlijk, genereren automatisch een constructor en getters en maken de intentie van de klasse onmiddellijk duidelijk. Omdat deze objecten enkel data transporteren en geen gedrag bevatten, zijn records de meest passende keuze.

Correlatie-ID voor traceerbaarheid. Bij elke binnenkomende request genereert de controller een unieke correlationId die doorheen de volledige verwerkingsketen meegegeven wordt. Deze identifier wordt opgenomen in elke logregel, meegegeven aan document storage en opgenomen in het Kafka-event. Dit maakt het mogelijk om een specifieke ingest-aanvraag end-to-end te volgen over alle systemen heen op basis van één identifier.

Fail-fast validatie. Inkomende requests worden gevalideerd via Jakarta Bean Validation annotaties op de IngestRequest en FormFields records. Als een request ongeldige of ontbrekende gegevens bevat, wordt de verwerking onmiddellijk stopgezet met een HTTP 400-response voordat er enige businesslogica uitgevoerd wordt. Dit voorkomt dat ongeldige data de verwerkingsketen inkomt en op een later moment voor moeilijk te debuggen fouten zorgt.

Gelaagde architectuur. De applicatie is opgedeeld in een controller-, service-, configuratie- en util-laag waarbij elke laag enkel communiceert met de aangrenzende laag. Dit houdt de codebase overzichtelijk en voorkomt dat logica op de verkeerde plaats terechtkomt.

3.4. State database

3.4.1. Inleiding

De state database is een interne component van de Ingest API die de verwerkingsstatus van elke inzending bijhoudt. Ze vervult twee doeleinden. Ten eerste dient ze als basis voor de statusopvolging in de Outlook add-in: nadat een gebruiker een e-mail indient, pollt de add-in periodiek de status van de verwerking en toont deze in de interface. Ten tweede implementeert ze het deduplicatiemechanisme op basis van het internetMessageId, zodat dezelfde e-mail niet meerdere keren verwerkt kan worden. De database werd opgezet met Spring Data JPA en H2 als in-memory database.

3.4.2. Technologiekeuze

Voor de persistentielaag werd gekozen voor Spring Data JPA. Dit sluit aan bij de bestaande Spring Boot-structuur van de Ingest API en biedt een eenvoudige manier om entiteiten te definiëren en op te slaan zonder boilerplate SQL te schrijven.

Voor de database zelf werd gekozen voor H2, een lichtgewicht in-memory database die zonder externe installatie of configuratie opstart als onderdeel van de Spring Boot applicatie. De oorspronkelijke bedoeling was om een volwaardige relationele database te gebruiken, maar de installatie daarvan werd tegengehouden door de firewallconfiguratie binnen de VDI-omgeving. H2 was in dat geval de meest praktische oplossing om toch de volledige functionaliteit te kunnen realiseren en testen binnen de beschikbare infrastructuur. Voor een productie-uitrol dient H2 vervangen te worden door een persistente database zoals PostgreSQL.

3.4.3. Databasemodel

De database bestaat uit vier tabellen die samen de volledige context van een inzending bijhouden.

De centrale tabel is ingest_submission. Elke rij stelt één inzending voor en bevat de correlationId, de tenant- en gebruikersinformatie uit het JWT-token, het Graph messageId, het mailboxadres en de verwerkingsstatus. De status volgt de levenscyclus van een inzending en kan de waarden PENDING, PROCESSING, COMPLETED of FAILED aannemen. Daarnaast bevat de tabel een task_url voor de link naar de aangemaakte taak in het downstream systeem en een error_message voor het geval de verwerking mislukt.

De tabel mail_metadata slaat de RFC-headers van de opgehaalde e-mail op, zoals afzender, onderwerp en verzend- en ontvangsttijdstip. Ze heeft een één-op-één relatie met ingest_submission.

De tabel submission_form_fields bevat de velden die de gebruiker invulde in de Outlook add-in, zoals ordernummer, prioriteit, taaktype en een optioneel commentaar. Ook deze tabel heeft een één-op-één relatie met ingest_submission.

De tabel submission_document bevat de documentreferenties van de e-mail en bijlagen die succesvol werden opgeslagen in document storage. Per document worden de document-ID, het type, de bestandsnaam, het content-type, de bestandsgrootte, de SHA-256 checksum en een vlag voor inline afbeeldingen bijgehouden. Deze tabel heeft een één-op-veel relatie met ingest_submission.

3.4.4. Deduplicatie

Het deduplicatiemechanisme is geïmplementeerd als een gedeeltelijke unique index op de combinatie van tenant_id, mailbox_address en internet_message_id. Deze index is enkel actief wanneer internet_message_id niet null is, omdat niet elke e-mail een RFC-conform Message-ID heeft. Wanneer dezelfde e-mail opnieuw wordt ingediend binnen dezelfde tenant en mailbox, faalt de database-insert op deze constraint en wordt de inzending geblokkeerd zonder dat de verdere verwerkingsflow gestart wordt. Naast deze database-constraint wordt in de Ingest API ook vooraf gecontroleerd of het internetMessageId al bestaat, zodat een duidelijke foutmelding teruggegeven kan worden aan de gebruiker in plaats van een onverwachte databasefout.

3.4.5. Statusopvolging

De verwerkingsstatus wordt doorheen de volledige ingest-flow bijgewerkt. Bij ontvangst van een request wordt een nieuwe inzending aangemaakt met status PENDING. Zodra de verwerking start, wordt de status bijgewerkt naar PROCESSING. Na succesvolle publicatie naar Kafka en opslag van alle documentreferenties wordt de status COMPLETED. Bij een fout in een van de verwerkingsstappen wordt de status FAILED en wordt de foutmelding opgeslagen in het error_message veld.

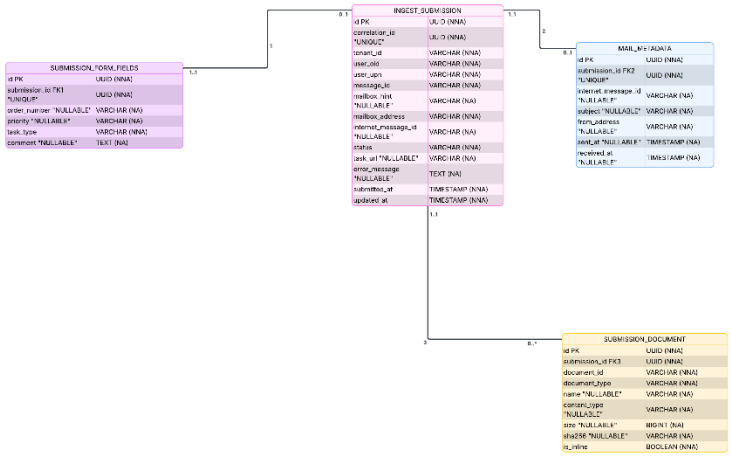
De Outlook add-in bevraagt via polling een statusendpoint van de Ingest API. Dit endpoint leest de huidige status op uit de database en geeft deze terug aan de add-in, die de gebruiker op de hoogte houdt van de voortgang.

3.4.6. Toegepaste principes en patronen

Interface-gebaseerd ontwerp. De database-integratie verloopt via Spring Data JPA repositories die als interfaces gedefinieerd zijn. De IngestService kent enkel deze interfaces en heeft geen weet van de onderliggende implementatie. Dit maakt het eenvoudig om in tests alternatieve implementaties te injecteren en vereenvoudigt een eventuele migratie naar een andere database.

Scheiding van verantwoordelijkheden. Het datamodel is opgesplitst in vier afzonderlijke tabellen die elk één specifiek aspect van een inzending bevatten. De inzending zelf, de e-mailmetadata, de formuliervelden en de documentreferenties worden afzonderlijk opgeslagen. Dit voorkomt brede tabellen met veel nullable kolommen en maakt het datamodel uitbreidbaar zonder bestaande structuren te wijzigen.

Infrastructuuronafhankelijke configuratie. De databaseverbinding wordt volledig via omgevingsvariabelen geconfigureerd. Voor de H2 in-memory database volstaat de standaardconfiguratie van Spring Boot, maar de configuratie is zo opgezet dat een overstap naar een externe database enkel een aanpassing van de omgevingsvariabelen vereist zonder codewijzigingen.



3.5. Document Storage

3.5.1. Inleiding

De document storage API is een bestaande interne dienst binnen H.Essers die gebruikt wordt door het backoffice systeem om bestanden op te slaan en te beheren. In het kader van dit project werd geen nieuwe document storage service gebouwd, maar werd de integratie ermee vanuit de Ingest API volledig uitgewerkt. Concreet betekent dit dat de DocumentStorageServiceImpl geschreven werd om uploads te verzorgen, dat de conversie van EML naar MSG geïmplementeerd werd omdat het EML-formaat niet ondersteund wordt, en dat de nodige parameters en type-codes afgestemd werden met de verantwoordelijke collega.

3.5.2. Verbinding en authenticatie

De verbinding met de document storage API wordt geconfigureerd via een centrale DocumentStorageConfig klasse. Hierin wordt een WebClient bean aangemaakt met de basis-URL van de service en een basisauthenticatiefilter die bij elke request automatisch de juiste credentials toevoegt. De URL, gebruikersnaam en het wachtwoord worden via omgevingsvariabelen meegegeven en zijn niet opgenomen in de codebase. Dit zorgt ervoor dat de configuratie per omgeving aangepast kan worden zonder aanpassingen aan de code zelf.

Door de WebClient als een centrale bean te configureren en deze te injecteren in de DocumentStorageServiceImpl, is er één gedeelde instantie die hergebruikt wordt voor alle uploads. Dit is efficiënter dan per request een nieuwe client aan te maken en sluit aan bij de aanbevolen aanpak binnen het Spring-ecosysteem.

3.5.3. Uploadflow

Elke upload naar de document storage verloopt als een multipart/form-data request naar het /transport/upload endpoint. Naast het bestand zelf worden vier queryparameters meegegeven: een type-code die het soort document aangeeft, een padcode die overeenkomt met de type-code, een beschrijving die de correlationId van de aanvraag bevat, en de bestandseigenaar in de vorm van het UPN-adres van de gebruiker die de ingest gestart heeft.

Na een succesvolle upload geeft de document storage API een bestandsidentifier terug. Deze identifier wordt samen met de bestandsnaam, het content-type, de bestandsgrootte en een SHA-256 checksum opgeslagen als StorageResult. De document-ID's uit deze resultaten worden later opgenomen in het Kafka-event zodat het CSBO-systeem de opgeslagen bestanden kan terugvinden en koppelen aan een taak.

3.5.4. Type-codes

De document storage API vereist dat elk geüpload bestand voorzien wordt van een type-code die aangeeft wat voor soort document het betreft. Na afstemming met de verantwoordelijke collega werden twee type-codes vastgelegd. E-mails in MSG-formaat worden opgeslagen met de juiste type code, wat overeenkomt met het type e-mailbericht binnen het document management systeem van H.Essers. Bijlagen worden opgeslagen met wat overeenkomt met het type CS Backoffice Document.

3.5.5. EML naar MSG-conversie

Zoals eerder vermeld in de analysebespreking ondersteunt de document storage API het EML-formaat niet. Elke e-mail die via Microsoft Graph opgehaald wordt in EML-formaat, wordt daarom eerst omgezet naar MSG via de EmlToMsgConverter voordat de upload plaatsvindt.

De conversie leest de MIME-structuur van de EML-inhoud in via de angus-mail bibliotheek. Vervolgens worden de relevante MAPI-properties zoals afzender, ontvangers, onderwerp en berichttekst weggeschreven in een OLE2 Compound Document structuur via de Apache POI bibliotheek. Dit is het native containerformaat van MSG-bestanden dat door Outlook herkend wordt.

Tijdens de implementatie werd vastgesteld dat Outlook de geconverteerde MSG-bestanden opende als concept in plaats van als ontvangen bericht. De oorzaak was een ontbrekende berichtstatus in de MAPI-properties. Door de MSGFLAG_READ property expliciet in te stellen bij de conversie wordt het bericht bij openen correct herkend als een gelezen ontvangen bericht.

3.5.6. Checksumberekening

Na elke upload berekent de ChecksumCalculator een SHA-256 hash van de bestandsbytes die geüpload werden. Deze checksum wordt opgenomen in het Kafka-event als onderdeel van de document-referentie. Downstream systemen zoals CSBO kunnen de checksum gebruiken om te verifiëren dat het opgeslagen bestand identiek is aan wat de Ingest API verstuurd heeft, zonder het bestand zelf opnieuw op te halen.

3.5.7. Foutafhandeling

Wanneer de document storage API een fout teruggeeft, wordt een onderscheid gemaakt tussen het falen van de e-mail zelf en het falen van individuele bijlagen. Als de upload van de e-mail mislukt, wordt de volledige verwerking stopgezet en krijgt de gebruiker een foutmelding. De e-mail is immers het kernbestand waar de volledige verdere verwerking op steunt.

Als de upload van een individuele bijlage mislukt, wordt de fout gelogd maar wordt de verwerking niet onderbroken. De overige bijlagen en de e-mail worden gewoon verder verwerkt en de beschikbare document-ID's worden opgenomen in het Kafka-event. Het CSBO-systeem maakt op basis daarvan een taak aan met de documenten die wel succesvol opgeslagen werden. Dit maakt het systeem robuust tegen gedeeltelijke storingen in de document storage zonder dat de volledige ingest-flow daarvoor afgebroken hoeft te worden.

3.5.8. Toegepaste principes en patronen

Interface-gebaseerd ontwerp. De document storage integratie is opgezet via een interface DocumentStorageService met een concrete implementatie DocumentStorageServiceImpl. De IngestService kent enkel de interface en heeft geen weet van de concrete implementatie. Dit maakt het mogelijk om in tests een alternatieve implementatie te injecteren die geen echte HTTP-calls uitvoert en vereenvoudigt een eventuele toekomstige vervanging van de document storage API.

Centrale bean-configuratie. De WebClient voor document storage wordt geconfigureerd als een centrale Spring bean. De basis-URL en de authenticatiegegevens worden via omgevingsvariabelen meegegeven en zijn nergens hardcoded in de codebase aanwezig. Het basisauthenticatiefilter wordt eenmalig toegevoegd aan de WebClient, zodat elke request automatisch voorzien wordt van de juiste credentials zonder dat dit in de uploadlogica herhaald hoeft te worden.

Partial failure handling. Bij het opslaan van bijlagen wordt bewust gekozen voor een strategie waarbij individuele fouten de volledige verwerkingsflow niet blokkeren. Een mislukte bijlage-upload wordt gelogd en overgeslagen, terwijl de overige bijlagen en de e-mail gewoon verder verwerkt worden. Enkel wanneer de upload van de e-mail zelf mislukt, wordt de volledige verwerking stopgezet. Dit maakt het systeem tolerant voor gedeeltelijke storingen zonder dat de gebruiker de volledige aanvraag opnieuw hoeft in te dienen.

Verantwoordelijkheidsscheiding bij conversie. De conversie van EML naar MSG is ondergebracht in een aparte EmlToMsgConverter klasse in het util-pakket. De DocumentStorageServiceImpl roept deze converter aan maar bevat zelf geen conversielogica. De converter is bovendien als een pure functie geïmplementeerd: gegeven dezelfde EML-bytes produceert hij altijd hetzelfde MSG-resultaat, zonder afhankelijkheden op externe toestand of services.

3.6. Apache Kafka

3.6.1. Inleiding

Apache Kafka fungeert als de communicatielaag tussen de Ingest API en het downstream CSBO-systeem. Nadat de e-mail en bijlagen succesvol opgeslagen zijn in document storage, publiceert de Ingest API een event naar Kafka. Dit event bevat geen binaire bestanden maar enkel metadata en de document-ID's die verwijzen naar de opgeslagen bestanden. Het CSBO-systeem luistert op het betrokken topic en gebruikt de informatie uit het event om automatisch een taak aan te maken. Daarnaast is er een terugkoppeling voorzien waarbij het CSBO-systeem na het aanmaken van een taak een statusevent publiceert, dat de Ingest API opvangt om de Outlook add-in te informeren over de voltooide verwerking.

3.6.2. Configuratie

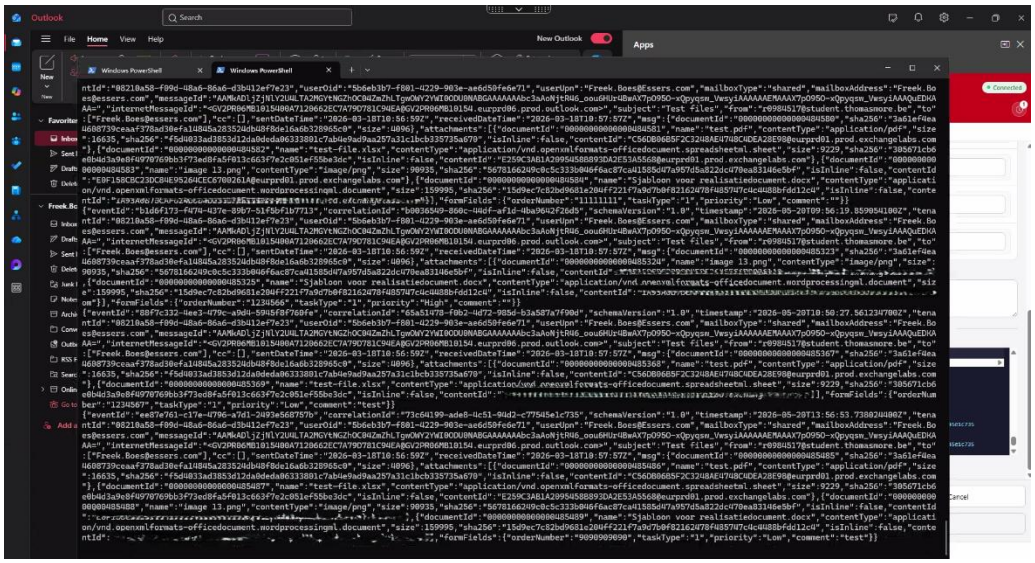
De producer-configuratie is centraal ondergebracht in de KafkaConfig klasse. Hierin worden de bootstrap-servers, de serialisatie en de betrouwbaarheidsinstellingen geconfigureerd als een centrale Spring bean. Events worden geserialiseerd als JSON via de Jackson-serializer van Spring Kafka. De type-informatie wordt niet als header meegestuurd, zodat het eventformaat clean blijft en downstream consumers geen afhankelijkheid hebben op de interne Java-klassenstructuur van de Ingest API.

De producer is geconfigureerd met acks=all, wat betekent dat de producer wacht op bevestiging van alle in-sync replica's voordat een publicatie als geslaagd beschouwd wordt. In combinatie met idempotente publicatie wordt gegarandeerd dat bij retries geen dubbele berichten in het topic terechtkomen.

3.6.3. Het ingest event

Na een succesvolle ophaling van de e-mail en opslag van alle bestanden in document storage publiceert de KafkaPublisherService een IngestEvent naar het geconfigureerde ingest-topic. Het event bevat uitsluitend metadata en document-referenties en nooit binaire bestandsinhoud. Dit houdt de berichten in het topic compact en maakt het mogelijk om bestanden onafhankelijk van Kafka te beheren in document storage.

Het event is opgebouwd uit een aantal vaste onderdelen. De event-identifiers omvatten een unieke eventId, de correlationId van de aanvraag, een schemaVersion en een tijdstempel. De gebruikerscontext bevat de tenant-ID, de object-ID en het UPN-adres van de gebruiker die de ingest gestart heeft. De mailboxcontext geeft aan of het om een primaire of gedeelde mailbox gaat en bevat het bijhorende mailboxadres. De e-mailmetadata bevat het onderwerp, de afzender, de ontvangers, de verzenddatum en de ontvangstdatum. De documentreferenties bestaan uit een MsgReference met het document-ID, de SHA-256 checksum en de bestandsgrootte van de opgeslagen e-mail, en een lijst van AttachmentReference objecten met dezelfde gegevens per bijlage aangevuld met het content-type, de bestandsnaam en informatie over inline afbeeldingen. Tot slot bevat het event de formFields die de gebruiker in de Outlook add-in ingevuld heeft, zodat het CSBO-systeem over alle nodige businesscontext beschikt zonder extra opzoekingen te moeten doen.



3.6.4. Message key en partitionering

De Kafka message key wordt samengesteld uit de tenant-ID, het mailboxadres en het internetMessageId van de e-mail. Als het internetMessageId niet beschikbaar is, wordt teruggevallen op het Graph-specifieke messageId. Door de tenant-ID en het mailboxadres op te nemen in de key wordt gegarandeerd dat berichten van dezelfde mailbox binnen dezelfde tenant altijd in dezelfde partition terechtkomen. Dit garandeert de volgorde van verwerking per mailbox en ondersteunt deduplicatie aan de kant van downstream consumers.

3.6.5. Statusopvolging via polling

Na het publiceren van een ingest-event wacht de Outlook add-in niet passief op een resultaat maar pollt ze periodiek het statusendpoint van de Ingest API met de ontvangen correlationId. Het CSBO-systeem publiceert na het aanmaken van een taak een IngestStatusEvent terug naar een apart statustopic. De IngestStatusListener in de Ingest API luistert op dit topic en slaat de ontvangen status op in de IngestStatusStore. Wanneer de add-in vervolgens het statusendpoint bevraagt, leest de controller de status op uit de store en geeft deze terug. Bij een voltooide verwerking bevat de response ook de URL van de

aangemaakte taak in het CSBO-systeem, zodat de gebruiker er direct naartoe kan navigeren vanuit de add-in.

De IngestStatusStore bewaart de verwerkingsstatus per correlationId in een in-memory structuur. Dit is een tijdelijke oplossing die voldoende is voor de huidige fase van het project maar bij een herstart van de applicatie alle statussen verliest. Voor een productieomgeving dient deze implementatie vervangen te worden door een persistente oplossing zoals Redis, zodat de status ook na een herstart beschikbaar blijft en correct werkt wanneer meerdere instanties van de Ingest API actief zijn.

3.6.6. Testomgeving

Voor het testen van de Kafka-integratie werd door een collega een dedicated Kafka pod opgezet binnen het Kubernetes-cluster van H.Essers. Deze pod draait in het interne netwerk van H.Essers en is uitsluitend beschikbaar als testomgeving voor dit project. Dit maakte het mogelijk om de volledige producer- en consumerflow te testen in een omgeving die de productieomgeving zo dicht mogelijk benadert, zonder impact op andere systemen binnen het bedrijf. De bootstrap-servers en topic-namen worden via omgevingsvariabelen aan de applicatie meegegeven, zodat er zonder codewijzigingen gewisseld kan worden tussen de testomgeving en een eventuele productieomgeving.

3.6.7. Toegepaste principes en patronen

Scheiding van producer en consumer. De publicatie van ingest-events en de consumptie van statusevents zijn ondergebracht in twee afzonderlijke klassen met elk hun eigen verantwoordelijkheid. De KafkaPublisherService is uitsluitend verantwoordelijk voor het samenstellen en publiceren van events, terwijl de IngestStatusListener enkel verantwoordelijk is voor het ontvangen en opslaan van statusevents. Dit maakt beide klassen onafhankelijk testbaar en aanpasbaar.

Geen binaire data in Kafka. Het IngestEvent bevat uitsluitend metadata en document-referenties. Alle binaire bestandsinhoud wordt opgeslagen in document storage en enkel via document-ID's gerefereerd vanuit het event. Dit houdt de berichten in het topic compact, vermijdt performantieproblemen bij grote bijlagen en maakt het mogelijk om bestanden onafhankelijk te beheren zonder Kafka als opslagmedium te misbruiken.

Betrouwbare publicatie. De producer is geconfigureerd met bevestiging van alle replica's en idempotente publicatie. Dit garandeert dat events niet verloren gaan bij tijdelijke brokerfouten en dat retries nooit leiden tot dubbele berichten in het topic.

Asynchrone statusopvolging. De keuze voor een polling-mechanisme in combinatie met een apart statustopic zorgt voor een volledige ontkoppeling tussen de Ingest API en het CSBO-systeem. De Ingest API hoeft niet te wachten op een reactie van CSBO en CSBO hoeft de Ingest API niet rechtstreeks aan te roepen. Beide systemen communiceren uitsluitend via Kafka-topics, wat de architectuur schaalbaar en onderhoudbaar houdt.

Optionele Kafka-afhankelijkheid. De KafkaPublisherService wordt als Optional geïnjecteerd in de IngestService. Dit zorgt ervoor dat de volledige verwerkingsflow operationeel blijft in omgevingen waar Kafka niet beschikbaar is, wat het lokaal ontwikkelen en testen aanzienlijk vereenvoudigt zonder dat de applicatie zelf aangepast hoeft te worden.

3.7. Versiebeheer met Github

3.7.1. Inleiding

De broncode van de Ingest API wordt beheerd via GitHub in de interne GitHub-omgeving van H.Essers. Hoewel het project individueel uitgevoerd werd zonder een team van medewerkers, werd er bewust voor gekozen om toch een professionele werkwijze te hanteren rond versiebeheer. Dit sluit aan bij de standaarden die binnen H.Essers gehanteerd worden en zorgt ervoor dat de codebase overzichtelijk en traceerbaar blijft.

3.7.2. Conventional commits

Voor alle commits werd de conventional commits conventie gehanteerd. Dit betekent dat elke commit een gestructureerd bericht heeft met een type, een optionele scope en een beschrijving. Veelgebruikte types zijn feat voor nieuwe functionaliteit, fix voor bugfixes, refactor voor codeverbeteringen zonder functionele wijziging en test voor het toevoegen of aanpassen van testen. Door deze conventie consequent toe te passen is de geschiedenis van het project leesbaar en geeft elke commit onmiddellijk duidelijkheid over wat er gewijzigd werd en waarom.

3.7.3. Branching en pull request

Voor elke nieuwe functionaliteit of wijziging werd een aparte feature branch aangemaakt vanuit de hoofdbranch. Na het afronden van een feature werd een pull request aangemaakt om de wijzigingen samen te voegen. In de beschrijving van elk pull request werd toegelicht wat er gewijzigd werd, waarom de wijziging nodig was en welke onderdelen van het systeem beïnvloed werden. Ook al werd het pull request door niemand gereviewd, deze werkwijze dwong tot het nadenken over de impact van elke wijziging en zorgde voor een gestructureerde en traceerbare ontwikkelgeschiedenis die aansluit bij hoe er in een team gewerkt zou worden.

4. Besluit

Tijdens de dertien weken durende stage bij H.Essers werd een breed uitgewerkte proof of concept gerealiseerd: een event-driven integratie tussen Microsoft Outlook en het Customer Service Back Office platform. Het uitgangspunt was een manueel en tijdrovend proces waarbij medewerkers e-mails en bijlagen handmatig moesten verwerken in het interne systeem. De ontwikkelde oplossing automatiseert deze volledige flow.

De kern van het systeem bestaat uit drie samenhangende componenten. De Outlook add-in biedt gebruikers een eenvoudige interface om een e-mail te selecteren, bijlagen aan te duiden en een aanvraag in te dienen. De Ingest API neemt die aanvraag over, haalt via Microsoft Graph de volledige e-mail en geselecteerde bijlagen op, slaat alles op in document storage en publiceert een event naar Apache Kafka. Downstream pikt het CSBO-systeem dat event op en maakt automatisch een taak aan met de bijhorende documenten gekoppeld.

Terugkijkend op de initieel geformuleerde doelstellingen kan het volgende worden vastgesteld. De eerste doelstelling, het ontwikkelen van een veilige en gebruiksvriendelijke Outlook add-in, is gerealiseerd. De add-in beschikt over een volledig werkend formulier, bijlagenselectie, offline detectie, statusopvolging via polling en een inzendingsgeschiedenis. De tweede doelstelling, het bouwen van een schaalbare ingest-API met Kafka-integratie, is eveneens gerealiseerd. De Ingest API verwerkt e-mails via de On-Behalf-Of flow, slaat bestanden op in document storage, past deduplicatie toe op basis van het internetMessageId en publiceert events naar Kafka met de juiste producerconfiguratie. De derde doelstelling, de integratie van de Kafka-events met een downstream consumer, is gedeeltelijk gerealiseerd. De events worden correct gepubliceerd. De definitieve integratie aan de CSBO-kant moet door het interne team worden voltooid, aangezien de toegang tot die repository tijdens de stage werd ingetrokken. De vierde doelstelling, kwaliteit en documentatie, is gerealiseerd via unit- en integratietesten voor zowel de add-in als de Ingest API, uitgebreide technische documentatie, een GitHub wiki, setup- en deployment guides en een geautomatiseerde Jenkins pipeline.

De belangrijkste technische uitdaging was de beveiligde infrastructuuromgeving. De firewallconfiguratie van het Kubernetes-cluster verhinderde dat Gravitee de JWKS-endpoint van Azure Entra ID kon bereiken, waardoor de aanbevolen JWT-validatie via JWKS_URL niet kon worden ingezet. Als pragmatische oplossing werd gewerkt met API key-validatie, zodat de ontwikkeling van de kern van de proof of concept niet geblokkeerd werd. Voor een productie-uitrol is het een vereiste dat de outbound netwerkpermissies worden opengesteld en de volledige JWT-validatieflow geconfigureerd wordt.

Voor de verdere uitwerking worden drie aanbevelingen meegegeven. Ten eerste dient de Gravitee JWT-validatieflow volledig geconfigureerd te worden voordat het systeem in productie gaat. Ten tweede moet het interne team de Kafka consumer aan de CSBO-kant implementeren op basis van de aangeleverde technische documentatie en het ontwerp dat in dit document beschreven staat. Zodra beide onderdelen gerealiseerd zijn, is de volledige end-to-end flow operationeel. Ten derde biedt de huidige architectuur een goede basis voor verdere uitbreidingen, zoals ondersteuning voor meerdere downstream consumers op hetzelfde Kafka-topic of het toevoegen van monitoring dashboards op basis van de reeds voorziene structurele logging met correlationId.

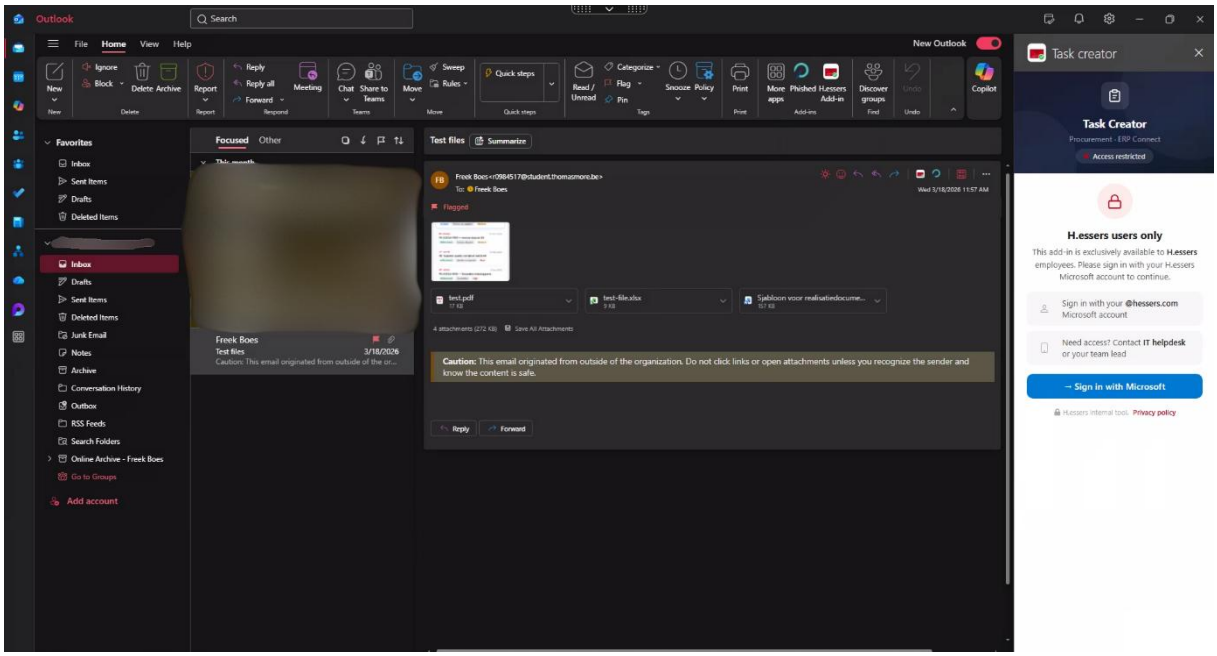
De proof of concept toont aan dat het manuele en foutgevoelige proces vervangen kan worden door een geautomatiseerde, event-driven flow. De architecturele keuzes, zoals het gebruik van document storage als tussenlaag voor binaire bestanden, idempotente verwerking via internetMessageId en de ontkoppeling tussen ingest en downstream verwerking via Kafka, zorgen ervoor dat het systeem schaalbaar, traceerbaar en onderhoudbaar is. Het project is klaar om verder uitgerold te worden.

LITERATUURLIJST

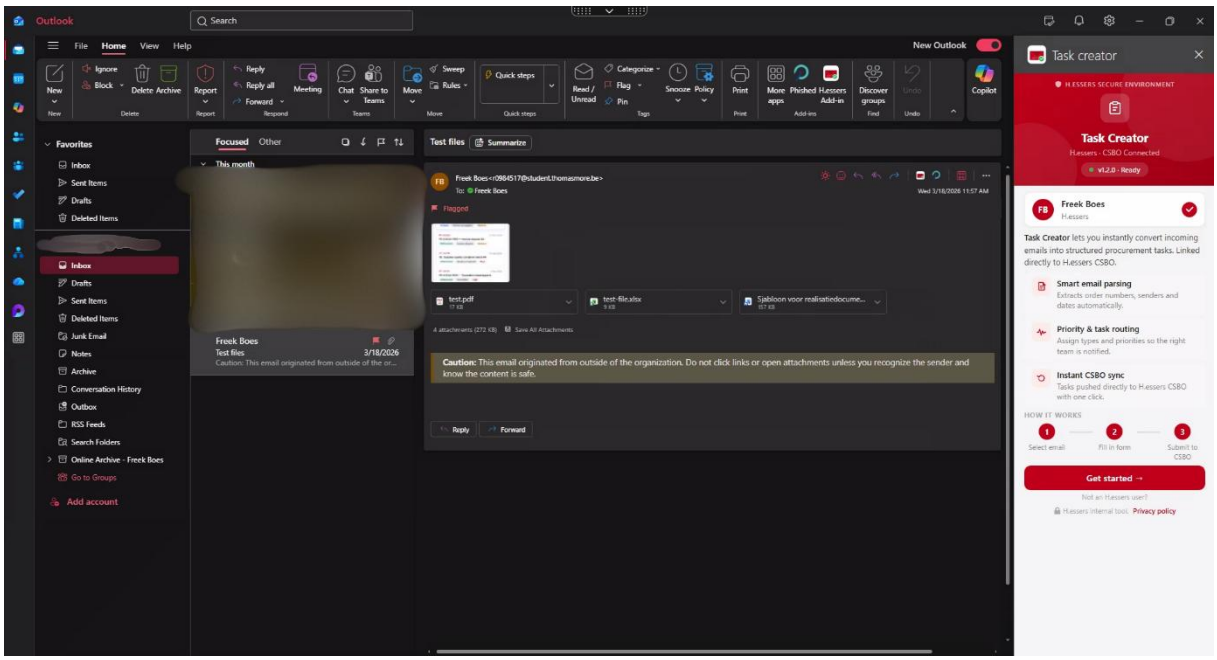
Apache Kafka. *Apache Kafka documentation*. https://kafka.apache.org/documentation/
Apache POI. *Apache POI documentation.* https://poi.apache.org/
Azure. Azure *Kubernetes Service documentation.* https://learn.microsoft.com/en-us/azure/aks/
Docker. *Docker documentation*. https://docs.docker.com/
Eclipse Angus. *Angus Mail documentation*. https://eclipse-ee4j.github.io/angus-mail/
Gravitee. Gravitee *documentation*. https://documentation.gravitee.io/
Jenkins. *Jenkins documentation*. https://www.jenkins.io/doc/
Jest. *Jest documentation.* https://jestjs.io/docs/getting-started
k6. *k6 documentation*. https://grafana.com/docs/k6/latest/
Microsoft. *Microsoft Graph documentation.* https://learn.microsoft.com/en-us/graph/overview
Microsoft. *Microsoft identity platform and OAuth 2.0 On-Behalf-Of flow*. https://learn.microsoft.com/en-us/entra/identity-platform/v2-oauth2-on-behalf-of-flow
Microsoft. *Microsoft identity platform documentation (Azure Entra ID).* https://learn.microsoft.com/en-us/entra/identity-platform/
Microsoft. *Nested App Authentication in Outlook add-ins.* https://learn.microsoft.com/en-us/office/dev/add-ins/outlook/nested-app-auth-outlook-legacy-tokens
Microsoft. *Office Add-ins documentation (Office.js).* https://learn.microsoft.com/en-us/office/dev/add-ins/overview/office-add-ins
Microsoft. *Office.js API reference.* https://learn.microsoft.com/en-us/javascript/api/overview
MSAL4J. *Microsoft Authentication Library for Java (MSAL4J).* https://learn.microsoft.com/en-us/entra/msal/java/
SonarQube. *SonarQube documentation.* https://docs.sonarsource.com/sonarqube/latest/
Spring. *Spring Boot documentation.* https://docs.spring.io/spring-boot/index.html
Spring. *Spring for Apache Kafka documentation.* https://docs.spring.io/spring-kafka/reference/
Webpack. *Webpack documentation.* https://webpack.js.org/concepts/
H.Essers. (2026). *Development environment configuration guide. Interne documentatie, niet publiek beschikbaar.*
Lardenoije, A. (2026). *Feedbackmomenten, alignmentgesprekken en technische overlegmomenten tijdens de stage. Niet publiek beschikbaar.*

BIJLAGEN

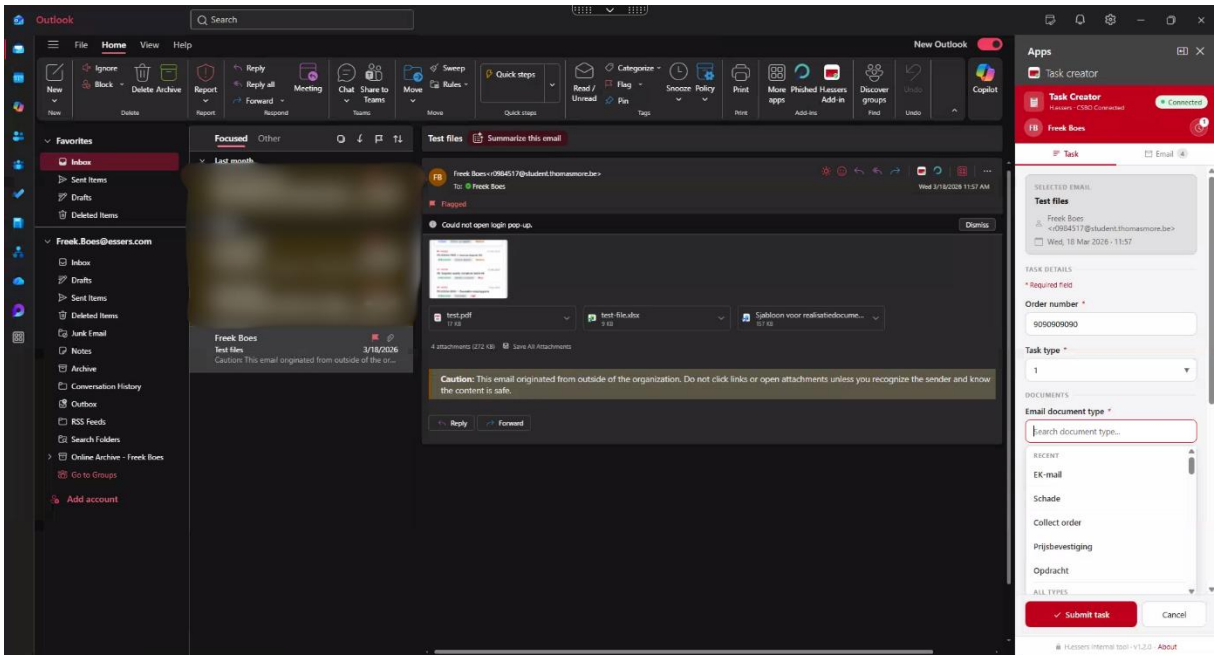
1. OUTLOOK ADD-IN SCREENSHOTS



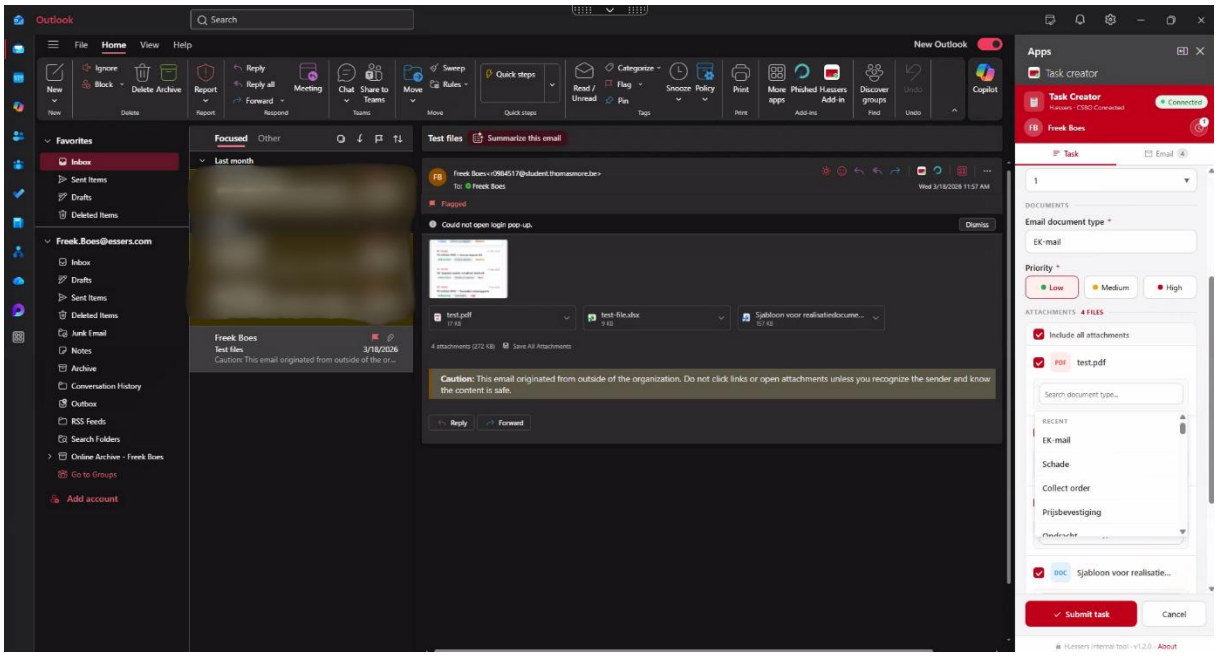
Figuur 1 - vergrendeld scherm



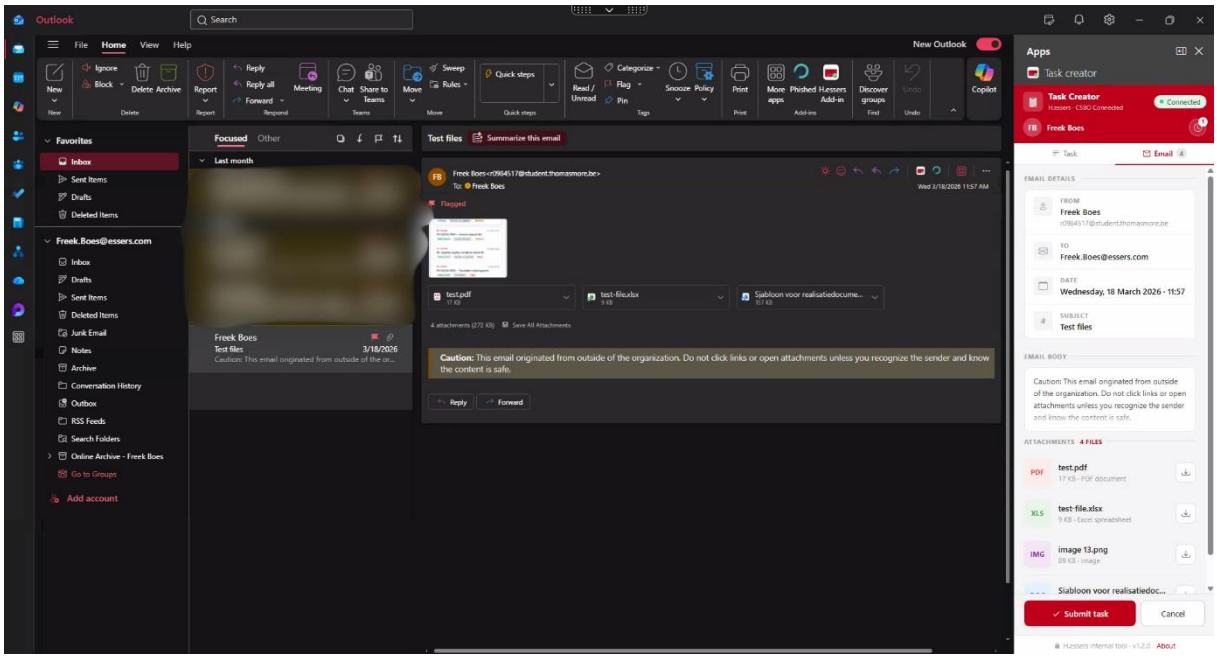
Figuur 2 - Samenvattingsscherm



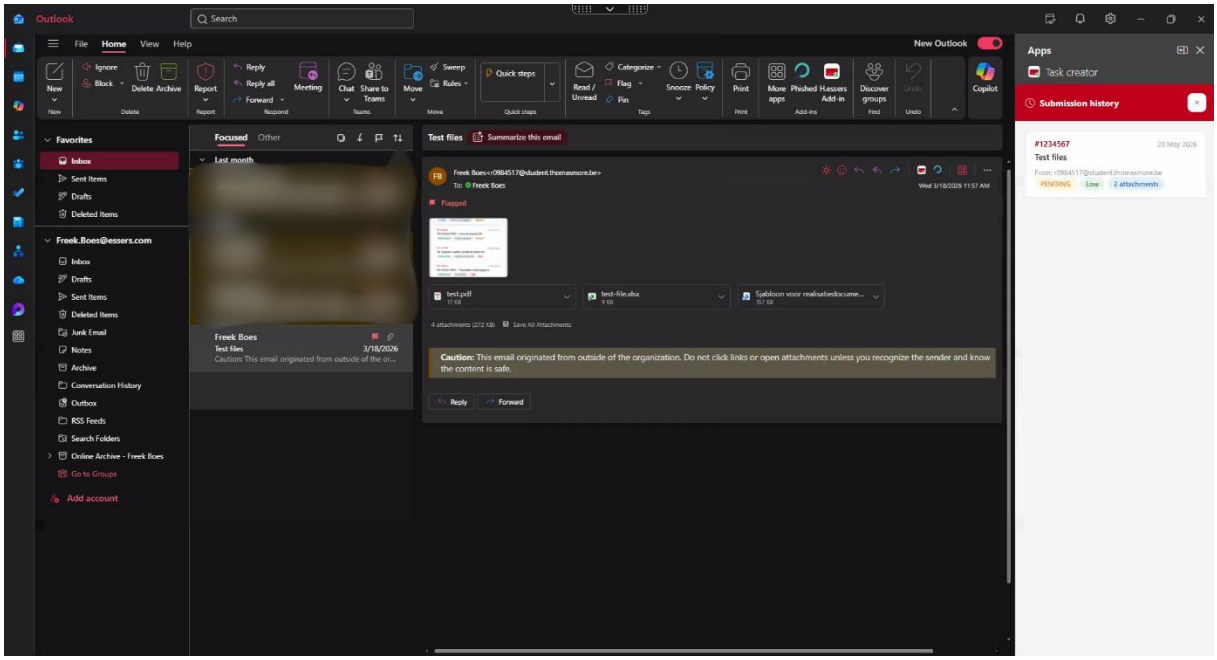
Figuur 3 - formulieroverzicht



Figuur 4 - gedeeltelijke bijlagenselectie

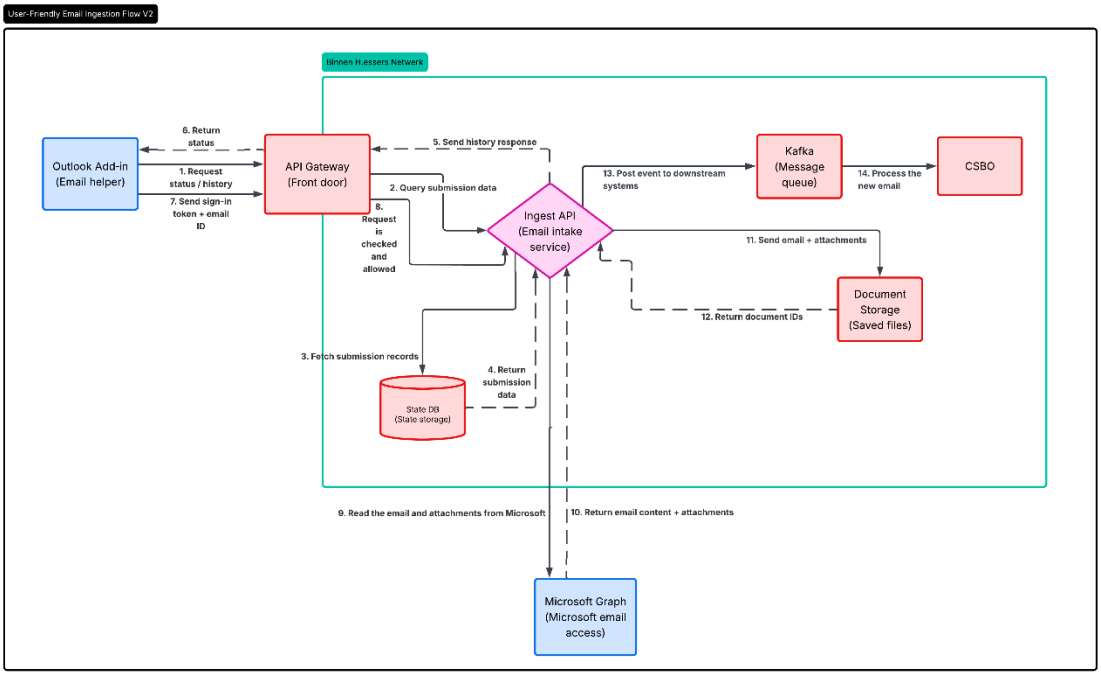


Figuur 5 - email details



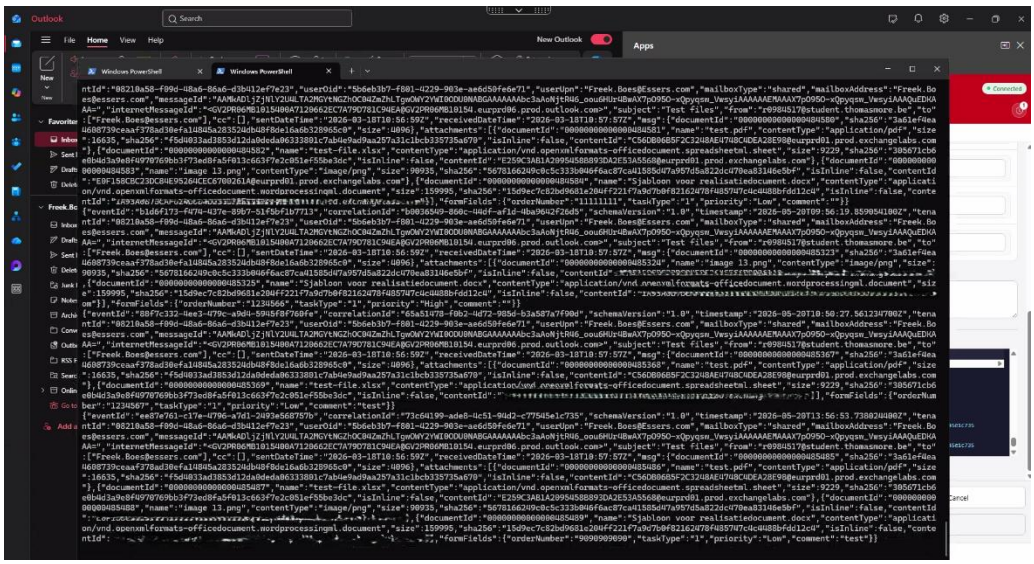
Figuur 6 – Inzendingsgeschiedenis

2. FLOW VAN HET HELE PROCESS



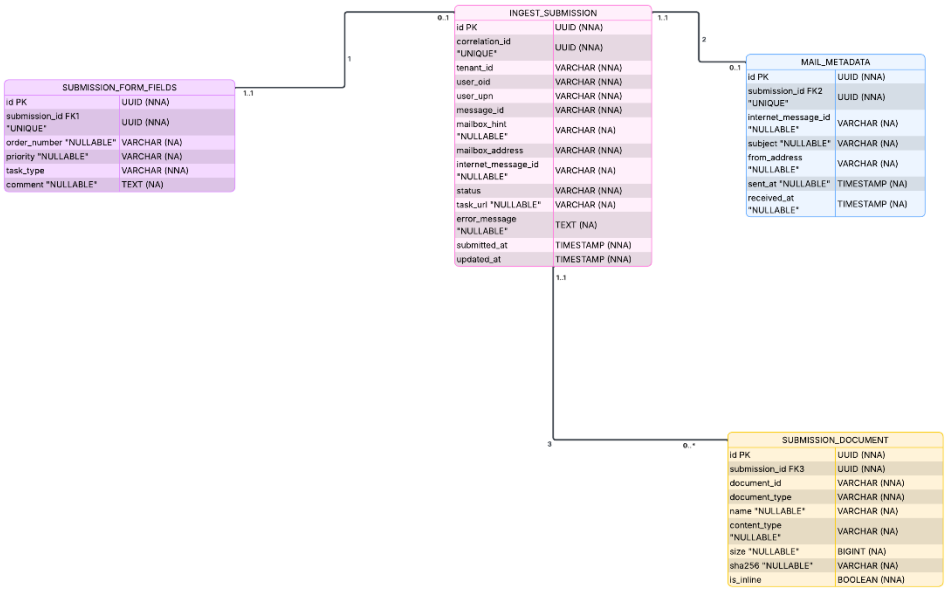
Figuur 7 – Flow van outlook tot in csbo

3. APACHE KAFKA



Figuur 8 – kafka reference voor ingestuurde emails

4. ERD DIAGRAM



Figuur 9 – State DB ERD diagram