THOMAS MORE

STAGE: RETRIEVAL-AUGMENTED GENERATION

Realisatie Document

Student: Seppe Stroobants
Stage mentor: Yano Stulens
Stage begeleider: Kristine Mangelschots

2024 - 2025

Inhoud

1. Introductie stagebedrijf

1.1. Stageplaats: Wisemen

De stage wordt uitgevoerd bij **Wisemen**, een Belgisch digitaal agency gespecialiseerd in strategie, ontwikkeling en marketing. Sinds de oprichting in 2013 (voorheen bekend als Appwise) is Wisemen uitgegroeid tot een full-service digital agency.

1.1.1. Diensten en Expertise

- **Digitale Transformatie**: Wisemen ontwikkelt strategieën die bedrijfsdoelstellingen afstemmen op de behoeften van klanten en gebruikers, met oog voor technische mogelijkheden.
- **Design**: Het bureau ontwerpt gebruiksvriendelijke ervaringen en interfaces volgens een eigen design-driven aanpak, waardoor producten en diensten aantrekkelijk en effectief zijn.
- **Softwareontwikkeling**: Wisemen biedt maatwerksoftware, innovatieve websites en marketingstrategieën die efficiëntie bevorderen, klanten betrekken en groei stimuleren.
- **Marketing**: Met geïntegreerde marketingcampagnes helpt Wisemen merken om zich te onderscheiden en hun doelgroep effectief te bereiken.

1.1.2. Begeleiders en Teamleden

- Yano Stulens – Functioneel Analist & Stagebegeleider
- Gerrit Schalenbourg – Head of Operations
- Maarten Sijmkens – Data Engineer
- Kristine Mangelschots – Docent & Stagebegeleider (Thomas More)
- Seppe Stroobants – Stagiair Bachelor Toegepaste Informatica

Website: https://wisemen.digital/

1.2. Insteek van de projecten

Tijdens de stageperiode heb ik gewerkt aan twee concrete AI-gerelateerde projecten:

1.2.1. Retrieval-Augmented Generation (RAG)

Dit project richtte zich op de ontwikkeling van een RAG-pipeline die interne bedrijfsinformatie semantisch doorzoekbaar maakt met behulp van vector search en generatieve AI.

1.2.2. E-mailclassificatie en automatisatie

in dit project werd onderzocht hoe inkomende e-mails automatisch geclassificeerd en opgevolgd kunnen worden via Large Language Models (LLM's), met als doel de verwerkingstijd en werklast te verlagen.
Deze projecten tonen hoe AI-toepassingen binnen een bedrijfscontext kunnen bijdragen aan meer efficiënte informatieverwerking, betere klantenservice en verhoogde automatisatie van repetitieve processen.

2. Introductie RAG

De groeiende integratie van kunstmatige intelligentie binnen bedrijfsprocessen brengt aanzienlijke mogelijkheden met zich mee, maar staat ook voor uitdagingen op het gebied van gegevensprivacy. Tijdens deze stageopdracht ligt de focus op deze uitdagingen door zich te richten op Retrieval-Augmented Generation (RAG), een veelbelovende technologie die de kracht van retrieval-mechanismen combineert met generatieve AI-modellen. Het centrale doel van dit project was de ontwikkeling van een concrete proof-of-technology (PoT) die demonstreert hoe RAG ingezet kan worden om de kwaliteit en relevantie van AI-gegenereerde content significant te verbeteren, met name door AI-systemen toegang te geven tot actuele en bedrijfsspecifieke informatie.

Dit realisatiedocument beschrijft het gedetailleerde traject van de ontwikkeling en implementatie van de RAG proof-of-technology. Het belicht de technische keuzes die zijn gemaakt op basis van een gedegen onderzoeksfase, de concrete stappen in de prototyping en ontwikkeling, de behaalde resultaten met betrekking tot de functionaliteit en prestaties van het systeem, en een kritische reflectie op het gehele proces en de persoonlijke leerervaringen van de stagiair. Ten slotte worden aanbevelingen geformuleerd voor toekomstige stappen en potentiële toepassingen van RAG binnen de bedrijfscontext van Wisemen.

2.1. Stage/opdracht beschrijving

Tijdens deze stage wordt onderzoek gedaan naar Retrieval-Augmented Generation (RAG) in een bedrijfscontext. Het doel is om een proof-of-technology uit te werken waarin de werking en toepasbaarheid van RAG wordt geanalyseerd. De student zal experimenteren met RAG-modellen en evalueren hoe deze technologie kan bijdragen aan zowel efficiëntere als nauwkeurigere informatieverwerking binnen AI-toepassingen.

De focus ligt op het onderzoeken van een RAG-systeem, waarbij zowel data storage en retrieval als generatieve AI wordt gecombineerd. De student zal verschillende methodologieën en tools vergelijken.

Naast het technische luik is het ook belangrijk om de impact en mogelijke toepassingen van RAG in reële bedrijfscontexten te onderzoeken, met een nadruk op efficiëntie en praktische inzetbaarheid.

2.2. Onderzoekthema

“Hoe kan Retrieval-Augmented Generation RAG worden toegepast om de kwaliteit en efficiëntie van AI-gegenereerde content te verbeteren?”

3. Uitvoering van de werkzaamheden RAG

3.1. Onboardingsfase

De eerste twee weken van de stage stonden in het teken van onboarding. Tijdens deze periode maakte ik me vertrouwd met de ontwikkelmethodologie en projectstructuur die binnen Wisemen gehanteerd worden, welke afwijkt van de structuur die tijdens mijn opleiding werd aangeleerd.

De onboarding gebeurde via de bestaande front-end en back-end repositories, waarbij ik met behulp van een interne onboardingtool mijn eerste API-endpoints ontwikkelde volgens de gehanteerde logica.

Mijn focus lag voornamelijk op de backend, aangezien mijn stageopdracht volledig backendgericht was.

Enkele voorbeelden van hoe projecten op backend niveau worden opgesteld:

Module naam (documents)
- entities
  - document.entity.ts
- use-cases
  - delete-documents
    - delete-documents.controller.ts → API endpoint
    - delete-documents.module.ts → declareert de controller, de use-case, …
    - delete-documents.response.ts → definitie van de response van het endpoint
    - delete-documents.use-case.ts → core logica voor de endpoints
  - embed-documents
    - dtos
      - embed-documents.request.dto.ts
    - services
      - embedding.service.ts
      - file-fetch.service.ts
    - embed-documents.controller.ts
    - embed-documents.module.ts
    - embed-documents.response.ts
    - embed-documents.use-case.ts
  - get-documents
    - …
  - search-documents
    - ….

3.2. Onderzoeksfase

Na afronding van de onboardingfase startte het inhoudelijk onderzoek. Omdat mijn specialisatie binnen de opleiding Application Development ligt, had ik aanvankelijk weinig voorkennis over AI en RAG. Ik begon daarom met het bestuderen van fundamentele concepten rond RAG en de onderliggende architectuur.

3.2.1. Wat is Retrieval-Augmented Generation (RAG)?

3.2.1.1. Een combinatie van Retrieval en Generatie modellen

- **Retrieval-modellen:** halen relevante informatie op te halen uit externe databron (een database met documenten)
- **Generatieve modellen:** zoals Large Language Models (LLMs), genereren op basis van ingevoerde context nieuwe tekst

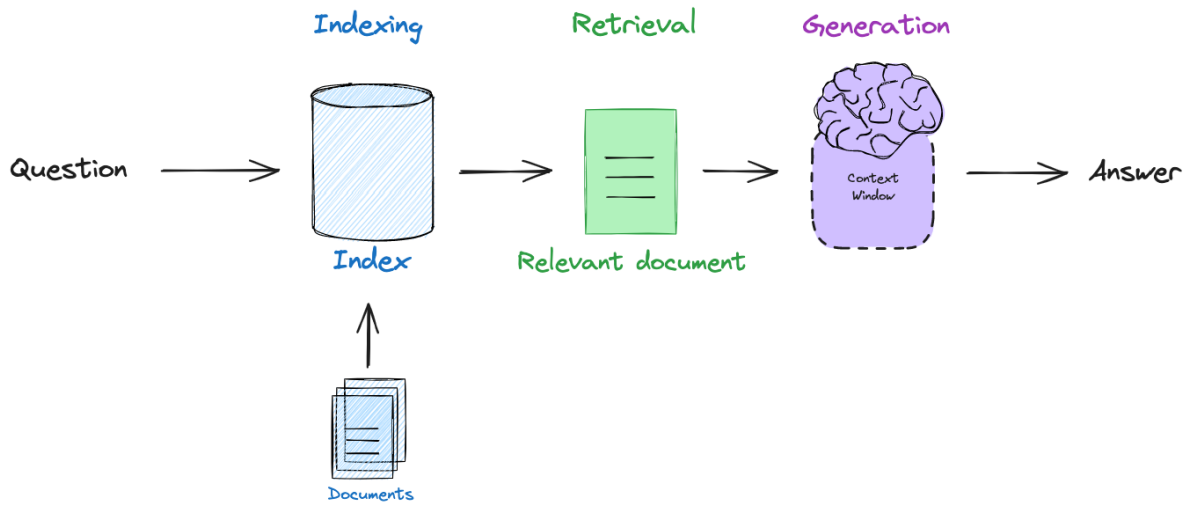
3.2.1.2. Waarom zouden we RAG willen gebruiken?

- **Verbeterde Kwaliteit en Relevantie:** Door contextuee en actuele data toe te voegen, verbetert RAG de relevantie van gegenereerde antwoorden.

3.2.1.3. Beperkingen van LLM's en de rol van RAG

- **Gebrek aan actuele kennis:** LLMs zijn getraind op een statische dataset en hebben een bepaalde *cutoff data* waardoor ze geen toegang hebben tot nieuwe informatie
- **Hallucinaties:** verzonnen of onjuiste informatie genereren.
- **Beperkte kennis van bedrijfsspecifieke data:** LLMs hebben mogelijk geen toegang tot interne data van organisaties.

3.2.2. Hoe werkt Retrieval-Augmented Generation (RAG)?



1. **Gebruikersvraag (Question)** ➔ De gebruiker stelt een vraag aan het RAG-systeem.
2. **Retrieval (ophalen van relevante informatie):**
    a. De gebruikersvraag wordt geprocessed en ge-embed (**indexing**).
    b. De vector wordt vergeleken met documenten in een vector database.
    c. De meest relevante context wordt geslecteerd. (**reterieval)**
3. **Augmentatie (Verrijking van de prompt):**
    a. De relevante context wordt gecombineerd met de originele vraag.
4. **Generatie (Genereren van het antwoord):**
    a. De augmented prompt wordt doorgegeven naar een generatief model.
    b. Het model genereert een contextueel onderbouwd antwoord.

3.2.3. Samenvatting

RAG combineert dynamische informatieopvraging met generatieve AI en biedt zo betrouwbaardere en actuelere antwoorden. Het stelt bedrijven in staat om AI-systemen te ontwikkelen die gevoed worden met eigen documentatie en actuele kennis.

3.3. Ontwikkelingsfase

De ontwikkelingsfase — veruit het grootste onderdeel van deze stage — draaide om het opzetten van een werkende RAG-pipeline. De applicatie werd opgebouwd uit twee hoofdcomponenten. Deze sectie focust op de eerste: de Python-backend. Deze backend werd ingezet voor specifieke taken zoals het embedden van documenten, het voorbereiden van tekstdata en het uitvoeren van zoekopdrachten binnen een vector database.

3.3.1. Python backend

Voor diverse AI-gerelateerde taken was een aparte Python-backend noodzakelijk. Denk hierbij aan het genereren van embeddings, het uitvoeren van zoekopdrachten en het voorbereiden van documenten voor opslag en retrieval. De backend werd opgebouwd met schaalbaarheid, snelheid en modulariteit in het achterhoofd.

3.3.1.1. Tekst preprocessing

WAAROM IS TEKST PREPROCESSING BELANGRIJK?

Tekstvoorbewerking is een fundamentele stap binnen een RAG-pipeline. Zowel het retrieval- als het generatieproces profiteren van schone, consistente data. Hieronder de drie belangrijkste voordelen:

1. **Verbeterde Kwaliteit van embeddings:**
    a. ***Consistentie**:* Door tekst te normaliseren (bijvoorbeeld hoofdlettergebruik of accenten verwijderen), worden semantisch gelijke termen op dezelfde manier geïnterpreteerd. Zonder normalisatie kunnen woorden als “Applicatie” en “applicatie” verschillende vectoren opleveren, waardoor relevante matches verloren gaan.
    b. ***Focus op kernbetekenis*:** Het weglaten van overbodige tekens zoals interpunctie helpt het model om zich te richten op de inhoud van de tekst.
    c. ***Vermindering van Ruis*:** Onnodige witruimte en andere inconsistenties kunnen ruis introduceren die de kwaliteit van de embeddings kan verminderen.
2. **Efficiëntere Retriaval**
    a. ***Verbeterde matching*:** Wanneer zowel documenten als zoekopdrachten op dezelfde manier zijn gepreprocessed, vergroot dit de kans op een correcte semantische match.
    b. ***Beter indexbeheer*:** Een gestroomlijnde input maakt de indexing in de vector database efficiënter, wat bijdraagt aan snellere zoekresultaten.
3. **Relevantere output voor het Generatieve Model (Generatie)**
    a. ***Schonere Context*:** De opgehaalde tekstfragmenten (context) zijn overzichtelijker, wat de kans vergroot dat het taalmodel correcte en relevante antwoorden genereert.
    b. ***Minder afleiding*:** Door onnodige tekens of inconsistenties in de context te verwijderen, wordt de kans verkleind dat het model zich laat afleiden

3.3.1.2. Embedden van documenten

WAT IS EMBEDDEN?

Embeddings zijn vectorrepresentaties van tekst. Een embeddingmodel zet tekst om in een getallenreeks die de betekenis van de tekst in een meer-dimensionale ruimte weergeeft.

- **Semantische representatie**: In deze vectorruimte liggen teksten met een vergelijkbare betekenis dicht bij elkaar. Bijvoorbeeld: een document over “hondenrassen” ligt in de vectorruimte dichter bij “huisdieren” dan bij “ruimtevaart”.

WAAROM IS EMBEDDING ESSENTIEEL VOOR RAG?

- **Semantisch zoeken:** In plaats van te zoeken op exacte trefwoorden, wordt gezocht op betekenis. Dit maakt het systeem veel flexibeler en effectiever.
- **Omgaan met synoniemen**: Zelfs als de gebruiker andere woorden gebruikt dan in het document staan, kan het systeem toch een match vinden.
- **Schaalbaarheid**: Omdat alle documenten vooraf worden omgezet naar vectoren, verloopt het zoeken in grote datasets razendsnel via optimalisaties in de vectordatabank.

HOE WERKT HET TECHNISCH?

- ***Embeddingmodellen:*** Deze AI-modellen zijn getraind op miljoenen tekstvoorbeelden om betekenisvolle relaties tussen woorden en zinnen in vectorvorm vast te leggen.
- ***Vector databases:*** De gegenereerde documentvectoren worden opgeslagen in gespecialiseerde databases die geoptimaliseerd zijn voor snelle ‘nearest neighbor’-zoekopdrachten.

UITLEG VAN DE CODE: EMBED_DOCUMENT_FROM_S3

De functie `embed_document_from_s3` verwerkt PDF-documenten die zijn opgeslagen in een S3-compatibele opslag. De inhoud van deze documenten wordt geëxtraheerd, opgesplitst in tekstfragmenten (chunks), omgezet naar vectorrepresentaties (embeddings), en vervolgens opgeslagen in een PostgreSQL-database met pgvector-extensie. Dit is essentieel voor de zoekfunctionaliteit of semantische analyse op basis van vectorvergelijkingen.

CODE:

```python
def embed_document_from_s3(s3_key: str, user_id: str, document_name: str):
    local_file_path = f"/tmp/{s3_key.replace('/', '_')}"
    s3_key_with_prefix = f"local/{s3_key}"

    print(f"Python: Attempting to download from S3 with key: {s3_key_with_prefix}, bucket: {S3_BUCKET_NAME}")

    try:
        logger.info(f"📥 Downloading document '{document_name}' (S3 key: {s3_key_with_prefix}) from S3...")
        s3_client.download_file(S3_BUCKET_NAME, s3_key_with_prefix, local_file_path)

        extracted_text = extract_text_from_pdf(local_file_path)
        os.remove(local_file_path)

        if not extracted_text:
            logger.warning(f"No text extracted from {document_name} (S3 key: {s3_key})")
            return

        with get_pg_connection() as conn:
            with conn.cursor() as cur:
                cur.execute("SELECT uuid FROM document WHERE name = %s AND user_id = %s", (document_name, user_id))
                result = cur.fetchone()
                if not result:
                    logger.warning(f"⚠️ File {document_name} (user_id: {user_id}) not found in document table. Skipping.")
                    return

                document_uuid = result[0]
                logger.info(f"📄 Processing {document_name} (UUID: {document_uuid}, S3 Key: {s3_key})")

                chunks = text_splitter.split_text(extracted_text)
                chunk_hashes = [generate_hash(chunk) for chunk in chunks]
                chunk_uuids = [uuid.UUID(hash_id) for hash_id in chunk_hashes]

                cur.execute(
                    "SELECT chunk_uuid FROM embedding WHERE chunk_uuid = ANY(%s::uuid[])",
                    (chunk_uuids,)
                )
                existing_hashes = {row[0] for row in cur.fetchall()}

                new_chunks = []
                new_embeddings = []
                chunks_to_embed = []

                for chunk, hash_id, chunk_uuid in zip(chunks, chunk_hashes, chunk_uuids):
                    if chunk_uuid in existing_hashes:
                        logger.debug(f"🔹 Chunk {chunk_uuid} already exists. Skipping.")
                        continue

                    new_chunks.append((chunk_uuid, chunk, document_uuid))
                    chunks_to_embed.append(chunk)

                if not new_chunks:
                    logger.info(f"✅ All chunks for {document_name} already exist.")
                    return

                vectors = embeddings_model.embed_documents(chunks_to_embed)
                for (chunk_uuid_for_chunks_table, _, _), vector in zip(new_chunks, vectors):
                    embedding_uuid = uuid.uuid4()
                    new_embeddings.append((str(embedding_uuid), str(document_uuid), str(chunk_uuid_for_chunks_table), vector))

                try:
                    execute_values(cur, """
                        INSERT INTO chunk (uuid, text, document_uuid)
                        VALUES %s
                        ON CONFLICT (uuid) DO NOTHING
                    """, new_chunks)

                    execute_values(cur, """
                        INSERT INTO embedding (uuid, document_uuid, chunk_uuid, embedding)
                        VALUES %s
                        ON CONFLICT (chunk_uuid) DO NOTHING
                    """, new_embeddings)

                    logger.info(f"✅ Inserted {len(new_chunks)} chunks for {document_name}")
                except Exception as e:
                    logger.error(f"⚠️ Error inserting chunks for {document_name}: {str(e)}")
                    traceback.print_exc()

            conn.commit()
            logger.info(f"✅ Finished processing {document_name}.")
```

Functiestappen:

- **Download document vanuit S3**
  Het PDF-bestand wordt opgehaald uit de S3-bucket met behulp van de opgegeven sleutel (`s3_key`).
- **Extractie van tekst**
  Met `extract_text_from_pdf()` wordt de tekstinhoud uit het PDF-bestand gehaald. Het tijdelijke lokale bestand wordt vervolgens verwijderd om opslag te besparen.
- **Validatie tegen de database**
  Er wordt gecontroleerd of het document al in de document-tabel van de database staat. Als dit niet het geval is, wordt de verwerking gestopt.

- **Splitsen en hashen**
  De geëxtraheerde tekst wordt opgesplitst in kleinere tekstfragmenten (chunks) met behulp van text_splitter.
  Elke chunk krijgt een unieke hash-ID via generate_hash, die vervolgens wordt omgezet naar een UUID.
- **Deduplicatie van chunks**
  Er wordt gecontroleerd of de chunks al eerder in de database zijn opgeslagen op basis van hun UUID.
  Alleen nieuwe chunks worden geselecteerd voor embedding en opslag.
- **Genereren van embeddings**
  De nieuwe tekstfragmenten worden via het embeddingmodel omgezet in vectorrepresentaties.
- **Opslaan in de database**
  De chunks worden opgeslagen in de chunk-tabel.
  De bijbehorende embeddings worden opgeslagen in de embedding-tabel. Dubbele invoer wordt hierbij vermeden.

Belang van deze functionaliteit
Deze workflow maakt het mogelijk om PDF-documenten semantisch doorzoekbaar te maken. Door de tekstinhoud om te zetten naar vectorrepresentaties kunnen gebruikers zoeken op basis van betekenis, in plaats van op exacte termen. De opslag in een pgvector-gebaseerde database maakt snelle en schaalbare vectorvergelijkingen mogelijk — essentieel voor moderne zoekoplossingen die gebruikmaken van machine learning.

RESULTAAT IN DE DATABASE

`document-tabel`

uuid	name	user_id	visibility
9bcb38b3-08f3-40c1-a226-c80d0d088718	sprinter_service_manual	931c3f95-74d8-4eb8-b954-9edfe674c018	private

`embedding-tabel`

uuid	document_uuid	chunk_uuid	embedding
000ffb99-09c4-43fc-ad92-3aa1cc18ed59	9bcb38b3-08f3-40c1-a226-c80d0d088718	6cd594a9-b792-9886-2c73-616daafb23e8	[-0.06994248,0.025307273,0.014642057,-0.0320773,…
00127b11-eca6-401c-bccb-05bcd1cb3eb1	9bcb38b3-08f3-40c1-a226-c80d0d088718	94344f47-2074-43bb-fcda-52aa417453d8	[-0.020379283,0.025643773,-0.08655287,0.05877578…
00143455-6d4e-4773-a435-c34a8ff52d87	9bcb38b3-08f3-40c1-a226-c80d0d088718	6c2e6a0a-6e0a-a931-44d9-a589ee3e786a	[-0.05478052,-0.028697252,0.013482651,-0.0102984…
0019a6fc-5b44-4b05-ad11-e68d2fb6c137	9bcb38b3-08f3-40c1-a226-c80d0d088718	98b25959-7153-6040-2db0-249cfa427951	[0.00012191964,-0.06922024,0.04892852,0.00138334…
001c099d-87dd-4784-8d3a-fb640ffd9915	9bcb38b3-08f3-40c1-a226-c80d0d088718	25b3d582-248d-24ad-aed1-d36caed1883a	[-0.044662554,-0.0053902757,-0.057528585,0.03274…
00261f3b-85a8-48db-a98d-4d2e2ee0165a	9bcb38b3-08f3-40c1-a226-c80d0d088718	7e07c337-1917-659e-5137-7598b1d89690	[0.0063377423,-0.02458836,-0.016686,-0.010085255…
0027ef1e-034f-4834-82f4-11fd336fdcd1	9bcb38b3-08f3-40c1-a226-c80d0d088718	d97d6f76-7096-bbf9-efac-cff6c8e4bf29	[-0.084275626,0.03993596,0.054625697,0.002408833…
003ca533-4fb3-459f-9510-d440a2c4d5c8	9bcb38b3-08f3-40c1-a226-c80d0d088718	18960bef-94ab-6e3d-b4fb-feaf8a8f7569	[0.015553659,0.13044298,-0.038513985,0.075026155…

`vector data van uuid: 000ffb99-09c4-43fc-ad92-3aa1cc18ed59`

uuid	uuid
000ffb99-09c4-43fc-ad92-3aa1cc18ed59	
document_uuid	uuid
9bcb38b3-08f3-40c1-a226-c80d0d088718	
chunk_uuid	uuid
6cd594a9-b792-9886-2c73-616daafb23e8	
embedding	unsupported

[-0.06994248,0.025307273,0.014642057,-0.0320773,-0.026682818,0.022854643,-0.0037797021,-0.0038723033,-0.03907786,-0.021918152,0.002
5014065,0.0113435825,-0.034935802,-0.048687335,-0.05752353,0.02615789,-0.042760514,-0.12099478,-0.009524205,-0.023850834,0.122526266,0
.015329014,0.004128326,-0.0006895162,-0.036155205,0.048984595,-0.018953405,0.11746937,-0.042221848,-0.059425768,-0.046654273,-0.002599
5083,0.013777186,0.014020832,0.056371145,0.020536695,0.0024401273,-0.046134606,6.469608e-05,0.008677873,-0.026497848,-0.17504393,-0.02
1915823,0.010937642,0.016892795,0.017970499,0.022009067,-0.08039362,0.04119357,-0.040637348,-0.06107804,0.037779145,0.0146806,0.042569
347,0.048099857,-0.023724841,-0.02933438,0.0036769353,0.044736095,-0.04188461,0.039545044,-0.00092936284,-0.052493077,0.05659017,-0.00
63406946,0.049885336,-0.029378662,-0.07106029,0.10263919,-0.054777484,-0.08599481,0.036390465,-0.018997602,-0.013518038,0.011970455,-0
.004145232,0.035722382,-0.032439943,-0.058652118,-0.06500422,-0.010968052,-0.04382157,-0.023292212,0.07015054,-0.03435122,0.032490764,
-0.00897147,0.022253439,0.04051741,-0.025525613,-0.044180978,-0.024037128,0.01289884,0.033387363,0.07960356,0.0029426368,-0.026748821,
-0.11842299,-0.044844,0.0797343,0.09105297,-0.077646755,-0.035652805,0.056351516,-0.03981382,0.03496562,0.03626511,0.091561764,-0.0686
0493,-0.037247717,0.044936687,-0.021176873,-0.015493136,-0.07071556,-0.03070524,-0.026947945,-0.03079726,-0.012839096,-0.027679415,0.0
4432291,-0.0016969092,0.010003881,0.008801975,0.029952696,0.052161675,0.07973468,-0.029954009,1.3900525e-32,0.00903976,0.0023639232,0.
021915827,0.022714175,-0.07395305,0.07262814,-0.02086061,0.06331303,0.010323316,0.047040086,0.053783372,0.02060499,-0.048022665,-0.000
10156535,-0.034779448,-0.04820139,0.11813332,0.023356296,-0.067508794,-0.061531425,-0.07616193,0.017028533,-0.0016344503,-0.021927526,
0.015752116,0.020421356,-0.01971376,0.0070430916,-0.023388568,0.010393668,0.04024123,-0.05688201,0.020373872,0.04052469,-0.009000051,0
.016511926,-0.024651032,-0.071560286,-0.035756946,-0.022680338,-0.048513632,0.048545342,-0.003568395,0.040110167,0.021366697,0.0493361
13,-0.0925239,0.009427203,0.085813634,0.013618502,-0.05062334,0.04704234,-0.018043602,0.07637143,-0.008005669,0.04094779,-0.026691198,
-0.0049507557,-0.028364448,0.013511877,-0.0638914,0.024878878,-0.0378304,-0.06967285,-0.019251758,-0.077072814,0.000791264,-0.05423224
7,-0.00040237818,-0.0054315333,-0.06974272,-0.06394326,-0.03211528,0.1110488,-0.016635817,0.033180322,-0.08295148,-0.0066749323,-0.011
709105,0.053607356,0.026553433,0.01264367,0.060706317,-0.016859025,0.05795189,-0.04506724,0.018117366,-0.05558212,-0.06229577,-0.04851
9973,0.00986057,0.051792204,0.017027287,-0.023058858,0.018049004,-1.3650985e-32,-0.04090283,0.04524809,0.00631781,-0.010097445,0.06566
8486,0.061143126,-0.030039074,-0.0926637,0.026597373,0.043550957,0.07681078,0.0032533936,-0.0002518204,-0.031535476,-0.06283578,0.0339
2377,-0.07858252,-0.07444315,0.011249257,0.07423339,-0.016131507,0.08049552,-0.061744336,0.018459184,-0.12327955,-0.06588307,-0.035546
225,-0.020176744,-0.05130894,0.05048116,-0.0007647571,-0.0052093435,-0.057010073,0.19337761,-0.10792075,-0.0033876512,0.0513269,0.0389
83844,-0.046208583,0.014438699,0.07061541,0.017573189,0.07153066,-0.074718066,-0.120440535,0.004912608,0.08227137,-0.10946961,-0.04755
513,0.054709814,-0.053310588,0.040004585,-0.04695443,0.04252973,-0.017425835,-0.038703162,-0.031233368,0.044619985,-0.011146378,-0.002
7225728,0.115829475,-0.010356797,-0.015454769,0.017770693,0.027471505,-0.055054124,0.028815243,-0.01713387,-0.043287292,-0.03811519,-0
.06919797,0.012641337,0.012845367,0.00071878714,0.053884733,-0.0076206014,-0.017801933,0.008428583,0.058177292,-0.019194147,-0.0130308
39,-0.027335986,0.00927172,0.14183645,-0.025365299,0.012385359,0.045900013,-0.028920228,0.09853494,0.0028688135,0.050427105,0.09151776
,-0.045788776,0.0077552204,0.02826886,-4.030628e-08,0.010512567,-0.05559231,-0.031694274,-0.04658551,0.07180958,0.010494123,0.03258156
8,0.13345563,-0.050316717,-0.047381364,0.0614566,0.057271358,0.024414403,-0.000322078,-0.012339469,-0.03937013,-0.020554576,0.12714916
,-0.009887632,-0.018648135,0.008776816,-0.02941226,-0.034382947,0.016157288,0.0895577,-0.045839936,-0.04665453,0.078325495,-0.05119743
6,0.033448637,0.057832245,-0.015844142,-0.043028135,-0.11278583,0.08493246,0.10574116,-0.08101922,0.06533838,0.0025556295,0.02985383,-
0.007452511,0.06366886,0.012973521,0.010372,0.046904843,-0.018228844,-0.061490826,0.0147439875,-0.05079066,-0.069869615,0.013250594,0.
004484927,0.057216316,-0.03356484,-0.013517588,-0.058841825,0.025760459,0.109484434,-0.01485976,0.010347122,0.12713799,0.08020539,-0.0
9541631,-0.08307292]

`chunk-tabel`

uuid	text	document_uuid
0014bcaf-38b8-0a4f-a708-ddcc956fa8ac	pres sure psm is used in regulated amounts to supply the transmission lubrication system in…	9bcb38b3-08f3-40c1-a226-c80d0d088718 →
00233020-a7c7-1832-ac04-4aec7633058d	fig 74 produce a nonposi tive locking connection between two elements of a planetary gear s…	9bcb38b3-08f3-40c1-a226-c80d0d088718 →
0028de5e-5b1f-526f-4141-7f56b170060c	the vehicle battery cables the battery cables connect the battery terminal posts to the v…	9bcb38b3-08f3-40c1-a226-c80d0d088718 →
00410572-5db4-eb09-c060-006220bfe40c	mode door 2419 camshaft position standard procedure checking 927 ca…	9bcb38b3-08f3-40c1-a226-c80d0d088718 →
004a2c6d-5f09-5a58-3636-1c75bc8f819f	tone generator will generate a single extended chime tone for a duration of about six sec o…	9bcb38b3-08f3-40c1-a226-c80d0d088718 →
004b7c9d-2922-295c-d9d1-f92cd0cb5408	9 33 valve springs continued engine block standard procedure standard procedure replacing…	9bcb38b3-08f3-40c1-a226-c80d0d088718 →
00573437-7d2f-8cf4-47c0-cf313e9e77c1	side control b 16blyl rest system auxiliary heater control cabin heater assembly c2 cav cir…	9bcb38b3-08f3-40c1-a226-c80d0d088718 →
00586bd5-1040-b8d8-754b-eafb5fa7521a	latch 1 door lock knob 2 electric 3 cable 4 lock rod 5 latch assembly 6 electrical co…	9bcb38b3-08f3-40c1-a226-c80d0d088718 →
006adb18-ec51-d7a5-317c-a459d3000ca1	2346 glass installation side view mirror 2349 glass removal do…	9bcb38b3-08f3-40c1-a226-c80d0d088718 →
00730baf-6267-d868-aae4-ea627f07f9de	to operate when the instru ment cluster detects that the ignition switch is in the on posit…	9bcb38b3-08f3-40c1-a226-c80d0d088718 →
008c646d-d190-901c-c8fe-b8a1ac4e7c40	8 remove the seat belt turning loop height adjuster from the inner bpillar installation war…	9bcb38b3-08f3-40c1-a226-c80d0d088718 →
008fc59b-2caa-6974-1d69-0068e5c88b77	that the brake reservoir fluid level will decrease in proportion to normal lining wear also…	9bcb38b3-08f3-40c1-a226-c80d0d088718 →

3.3.1.3. Uitvoeren van een zoekopdrachten (search query)

Een “search query” vormt de brug tussen de gebruikersvraag en de relevante documenten. Het is een cruciale stap binnen elke RAG-architectuur. Het proces bestaat uit meerdere stappen:

1. **Gebruikersinvoer:** De gebruiker stelt een vraag of instructie in natuurlijke taal. Dit is de initiële “search query”
2. **Query Preprocessing:** Net als de documenten wordt ook de gebruikersquery gepreprocessed (normalisatie, verwijdering van ruis, etc.).
3. **Embedding van de query:** De vraag wordt met hetzelfde embeddingmodel omgezet naar een vector, zodat het systeem op betekenis kan zoeken.
4. **Vector Search (similarity search):** De queryvector wordt vergeleken met alle documentvectoren in de database, gebruikmakend van wiskundige afstandsmetingen (zoals cosine similarity).

5. **Retrieval van documenten (chunks):** De vector database retourneert de meest relevante documentfragmenten (ook wel chunks genoemd), gesorteerd op gelijkenisscore.
6. **Augmented prompt:** De opgehaalde documenten worden samen met de oorspronkelijke gebruikersvraag gevoegd tot een uitgebreide prompt. Deze prompt gaat naar het generatieve taalmodel, dat hiermee het definitieve antwoord genereert.

UITLEG VAN DE CODE: SEARCH_PGVECTOR & FIND_SIMILAR_DOCUMENTS_PGVECTOR

1. FIND_SIMILAR_DOCUMENTS_PGVECTOR

Deze functie is verantwoordelijk voor het uitvoeren van een vectorvergelijking op de `embedding`-tabel in de PostgreSQL pgvector-database.

```python
import numpy as np
from langchain_huggingface import HuggingFaceEmbeddings

from app.pgvector.pg_connection import get_pg_connection

embeddings = HuggingFaceEmbeddings(
    model_name="sentence-transformers/all-MiniLM-L6-v2",
    model_kwargs={'device': 'cpu'}
)

def find_similar_documents_pgvector(query: str, user_id: int, k: int = 5):
    embedding = np.array(embeddings.embed_query(query)).tolist()

    conn = get_pg_connection()
    cur = conn.cursor()

    cur.execute("""
        SELECT e.chunk_uuid, d.name, e.embedding <-> %s::vector AS distance, d.visibility
        FROM embedding e
        JOIN document d ON e.document_uuid = d.uuid
        WHERE (d.user_id = %s OR d.visibility = 'public')
        ORDER BY distance ASC
        LIMIT %s;
    """, (embedding, user_id, k))

    results = cur.fetchall()
    cur.close()
    conn.close()

    return results
```

Functiestappen

- **Embedding van de query:** de gebruikersinvoer wordt via het HuggingFace embeddingmodel omgezet naar een vector (hetzelfde model als dat gebruikt wordt bij het embedden van de documenten)
- **SQL-query met vectorvergelijking:** in de database wordt gezocht naar de dichtstbijzijnde `chunk_uuid` op basis van `embedding <-> %s::vector`. Dit maakt gebruik van cosine similarity
- **Beperking op zichtbaarheid:** Alleen documenten die van de gebruiker zelf zijn of openbaar zijn (`visibility = 'public'`) worden meegenomen
- **Beperking op aantal:** Alleen de top `k` worden teruggegeven.

2. SEARCH_PGVECTOR (FASTAPI-ENDPOINT)

Deze endpoint voer de volledige RAG-query uit: van gebruikersvraag tot eindantwoord.

```python
from fastapi import APIRouter, HTTPException, Query
import time
import requests
import traceback

from app.pgvector.pg_connection import get_pg_connection
from app.services.pgvector.search_query import find_similar_documents_pgvector
from app.services.input_guardrails.restrict_substrings import is_input_safe
from app.services.input_guardrails.restrict_topics import default_moderator

router = APIRouter()

@router.get("/search-pgvector/")
async def search_pgvector(
    query: str,
    user_id: str = Query(..., description="User performing the search"),
    k: int = 3
):
    if not is_input_safe(query):
        raise HTTPException(status_code=400, detail="Query contains prohibited substrings or patterns.")

    if default_moderator.is_content_toxic(query):
        raise HTTPException(status_code=400, detail="Query contains toxic content.")

    try:
        context_start = time.time()

        results = find_similar_documents_pgvector(query, user_id, k)
        chunk_ids = tuple(row[0] for row in results)

        if not chunk_ids:
            raise HTTPException(status_code=404, detail="No similar documents found.")

        conn = get_pg_connection()
        cur = conn.cursor()

        cur.execute(f"""
            SELECT c.uuid, c.text
            FROM chunk c
            JOIN document d ON c.document_uuid = d.uuid
            WHERE c.uuid IN %s
              AND (d.user_id = %s OR d.visibility = 'public');
        """, (chunk_ids, user_id))

        context_chunks_with_uuid = cur.fetchall()
        safe_context_chunks = []
        for uuid, text in context_chunks_with_uuid:
            if not default_moderator.is_content_toxic(text):
                safe_context_chunks.append((uuid, text))
            else:
                print(f"Filtered out toxic chunk (UUID: {uuid}): {text[:50]}...")

        context = "\n\n".join([f"[Chunk {uuid}]: {text}" for uuid, text in safe_context_chunks])
        context_time = time.time() - context_start

        cur.close()
        conn.close()

        if not context.strip():
            return {"answer": "I cannot answer based on the provided documents due to content restrictions.", "retrieved_chunks": [], "context_retrieval_time": round(context_time, 3)}

        prompt = f"""
        Answer the following question based on the provided context.
        If no relevant info is found, say "I cannot answer based on the provided documents."

        Question: {query}

        Context:
        {context}

        Answer:
        """
        answer_start = time.time()
        response = requests.post(
            "http://host.docker.internal:11434/api/generate",
            json={"model": "llama3.2:latest", "prompt": prompt, "stream": False}
        )
        response.raise_for_status()
        answer = response.json().get("response", "No response")

        return {
            "answer": answer,
            "retrieved_chunks": [uuid for uuid, _ in safe_context_chunks],
            "context_retrieval_time": round(context_time, 3),
            "answer_generation_time": round(time.time() - answer_start, 3),
            "total_time": round(time.time() - context_start, 3),
        }

    except Exception as e:
        traceback.print_exc()
        raise HTTPException(status_code=500, detail=str(e))
```

Functiestappen:

- **Inputbeveiliging:** de query wordt gecontoleerd op verboden patronen (`is_input_safe()`).
  De query wordt gescreend op toxische inhoud (`default_moderator.is_content_toxic()`).
- **Vector search:** `find_similar_documents_pgvector()` haalt de relevante chunks op die het meest lijken op de query.
  Chunk UUID's worden verzameld en gebruikt om de bijhorende tekst op te halen uit de database.
- **Toxicity filter (chunkniveeau):** Elke opgehaalde chunk wordt opnieuw gecontroleerd op ongepaste inhoud. Alleen veilige chunks worden gebruikt voor de prompt.
- **Samenstellen van augmented prompt:** De chunks en de oorspronkelijke vraag worden gecombineerd in een generatieve prompt.
  Deze prompt wordt verstuurd naar het lokaal draaiend LLM-model van een API-call

- **Antwoord genereren:** Het gegenereerde antwoord wordt teruggestuurd, samen met de identifiers van de gebruikte chunks en prestatiemetingen (zoals tijdsduur van context en antwoordgeneratie)

3.3.1.4. Input guardrails

Om de veiligheid en betrouwbaarheid van de RAG-pipeline te waarborgen, heb ik input guardrails toegevoegd die gebruikersinvoer controleren ip mogelijk schadelijke of ongewenste content. Deze maatregelen zijn essentieel voor het voorkomen van beveiligingslekken zoals SQL-injecties en Cross-Site Scripting (XSS), mmar ook voor het detecteren van toxische of ongepaste taal in gebruikersvragen.

DETECTIE VAN GEVAARLIJKE PATRONEN (SQL/XSS-PREVENTIE)

Een eerste beveiligingslaag controleert of de ingevoerde tekst bepaalde verdachte patronen of substrings bevat die wijzen op een poging tot code-injectie. Hiervoor is de lijst gedefineerd met:

- Verboden substrings:
  - **Drop table**, **select * from**, **<script>**, **javascript;**, enzovoort.
- Reguliere expressies (regex) om compolextere partronen te detecteren
  - Bijvoorbeeld het gebruik van **ALTER TABLE**, **XP_CMDSHELL**,
  - HTML-tags met inline event handlers zoals **onload=** of **onclick=**

Alle gebruikersinvoer wordt systematisch vergeleken met deze lijsten. Wanneer een match wordt gevonden, wordt de invoer als onveilig gemarkeerd en afgewezen. Deze aanpak voorkomt dat kwaadwillende gebruikers ongewenste code of databasecommando's kunnen uitvoeren.

```python
import re

PROHIBITED_SUBSTRINGS = [
   "drop table",
    "select * from",
    "insert into",
    "delete from",
    ";--",
    "/*",
    "*/",
    "<script>",
    "</script>",
    "javascript:",
    "vbscript:",
    "<iframe>",
    "<frame>",
    "union select",
    "information_schema",
    "--",
]

PROHIBITED_PATTERNS = [
    re.compile(r"\b(ALTER|CREATE|TRUNCATE)\s+TABLE\b", re.IGNORECASE),
    re.compile(r"\b(EXEC|XP_CMDSHELL)\b", re.IGNORECASE),
    re.compile(r"<[^>]*on\w+\s*=[^>]*>", re.IGNORECASE),
]

def is_input_safe(query: str) -> bool:
    """Checks if the input query contains any prohibited substrings or patterns."""
    lower_query = query.lower()
    for sub in PROHIBITED_SUBSTRINGS:
        if sub in lower_query:
            print(f"Input contains prohibited substring: {sub}")
            return False
    for pattern in PROHIBITED_PATTERNS:
        if pattern.search(query):
            print(f"Input contains prohibited pattern: {pattern.pattern}")
            return False
    return True
```

MODERATIE VAN INHOUD (TOXISCHE TEKSTDETECTIE)

Naast technische veiligheid is ook inhoudelijke veiligheid belangrijk. Via een toxiciteitspipeline wordt gebruikersinvoer gecontroleerd op schadelijke of beledigende taal. Hiervoor wordt het *unitary/toxic-bert* model ingezet, een getraind model dat tekst classificeert op basis van waarschijnlijk van toxiciteit.

Het systeem werkt op basis van drempelwaardes: als een invoer boven een bepaalde waarschijnlijkheidsscore als “toxic” wordt ingeschat, dat wordt deze inhoud geblookkeerd of genegeerd.

Voordelen van deze aanpak:

- Beveiliging: voorkomt veelvoorkomende aanvalsvormen zoals SQL-injectie en XSS
- Robuustheid: voorlomt dat de pipeline wordt verstoord door onbedoelde of kwaadaardige input.
- Sociale verantwoordelijkheid: waarborgt dat het systeem niet wordt misbruikt voor het genereren of verwerken van haatdragende, kwetsende of ongepaste taal.

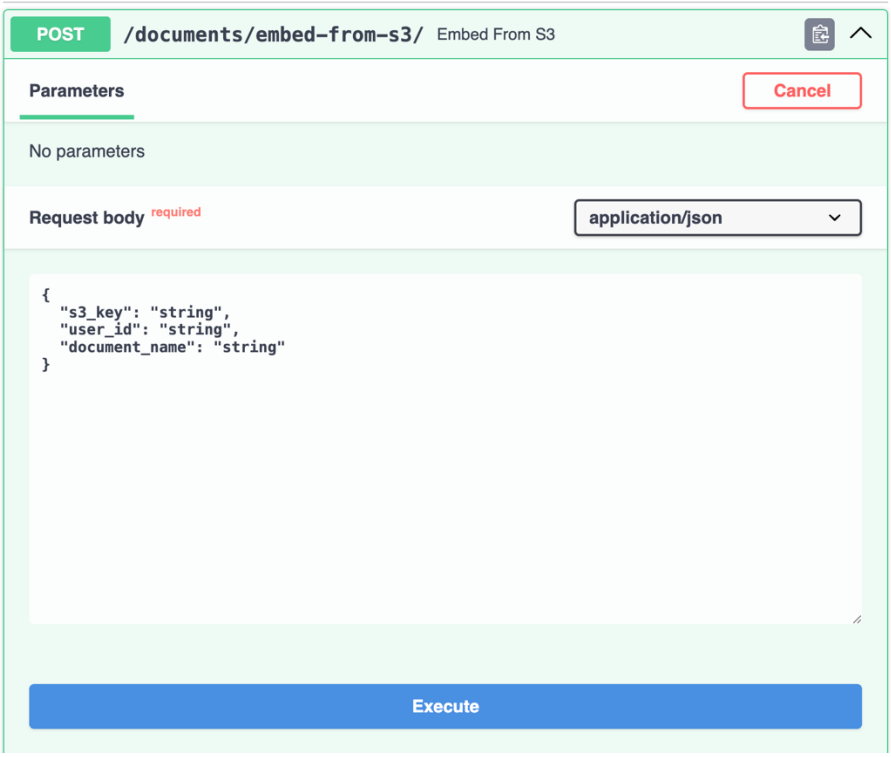
Deze combinatie van input filtering en content moderation vormt een belangrijke laag in de algehele betrouwbaarheid en ethische werking van de RAG-pipeline.

```python
from transformers import pipeline

class ContentModerator:
    def __init__(self, model_name="unitary/toxic-bert", threshold=0.5):
        self.toxicity_pipeline = pipeline("text-classification", model=model_name)
        self.toxicity_threshold = threshold
        self.toxic_labels = ["toxic"]

    def is_content_toxic(self, text: str) -> bool:
        results = self.toxicity_pipeline(text)
        for result in results:
            if result['label'] in self.toxic_labels and result['score'] > self.toxicity_threshold:
                return True
        return False

default_moderator = ContentModerator()
```

3.3.1.5. Swagger-UI API

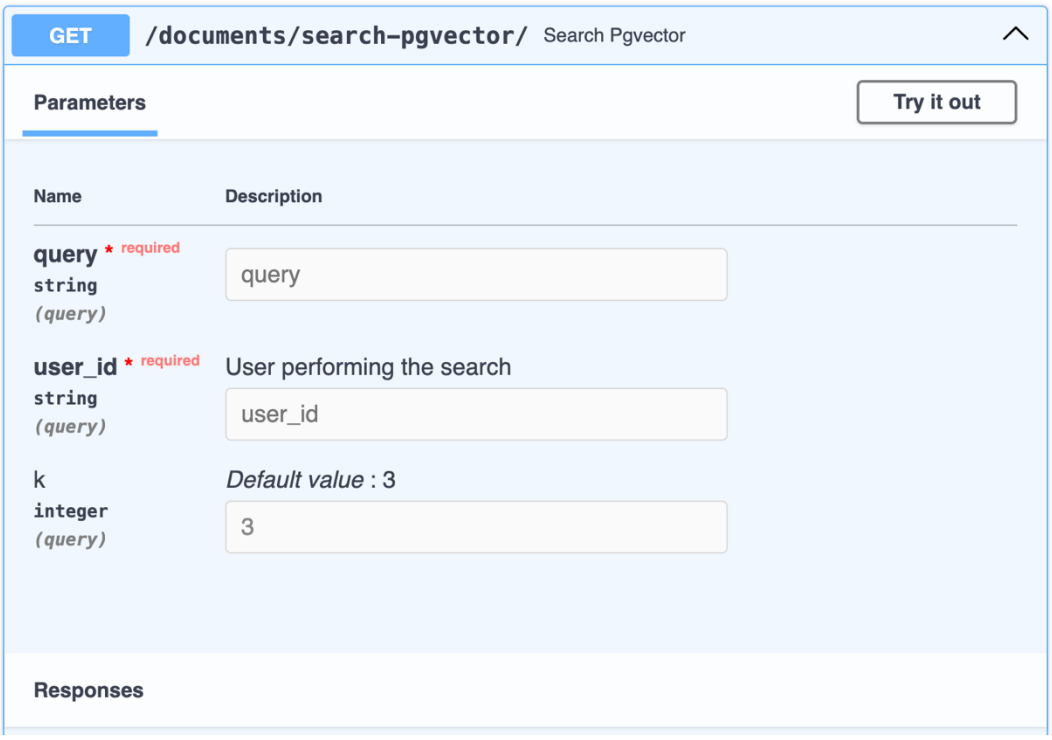
POST /DOCUMENTS/EMBED-FROM-S3

- s3_key: wordt doorgegeven om een tijdelijke download url te genereren.
- user_id: wordt doorgegeven door de NestJS backend (zie embed-document module)
- document_name: wordt doorgegeven door NestJS backend (zie embed-document module)



GET DOCUMENTS/SEARCH-PGVECTOR

query: is de vraag die de gebruiker wilt stellen aan de RAG pipeline
user_id: de gebruiker die de vraag stelt deze komt uit de NestJS backend (zie search-query module)
k: dit is het aantal documenten dat het LLM mag gebruiken om zijn antwoord te geven



3.3.1.6. Conclusie van Python backend

De Python-backend vormt de ruggengraat van de RAG-pipeline. Dankzij robuuste tekstvoorbewerking, nauwkeurige embeddinggeneratie en snelle vectorzoekopdrachten, wordt de kwaliteit van het gegenereerde antwoord aanzienlijk verhoogd. Deze backend is schaalbaar, uitbreidbaar en vormt een solide basis voor verdere integratie in bedrijfsspecifieke toepassingen van RAG.

3.3.2. NestJS backend

3.3.2.1. Introductie: Wat is NestJS?

NestJS is een progressief Node.js framework dat is ontworpen voor het bouwen van schaalbare en onderhoudbare backendtoepassingen. Het combineert elementen uit objectgeoriënteerd programmeren, functioneel programmeren en het is gebouwd bovenop Express.js. Dankzij de modulaire architectuur en het gebruik van TypeScript sluit NestJS perfect aan bij enterprice-toepassingen en microservices zoals deze RAG-pipeline.

3.3.2.2. Gebruik van NestJS in dit project

In dit project speelt de NestJS backend een centrale rol in het verbinden van de business logica van Wisemen met onze Python backend, beheren van documenten en het uitvoegen van authenticatie doormiddel van Zitadel. De NestJS is opgedeeld in meerdere modules die op hun beurt elk verantwoordelijk zijn voor specicifieke functionaliteiten.

INTERACTIE MET PYTHON API ENDPOINTS

NestJS fungeert als tussenlaag die om request op te stellen naar onze Python API. Hiervoor is er gebruikt gemaakt van HttpService uit @nestjs/axios, waarmee de backend externe endpoints kan aanspreken en data kan uitwisselen.

Het is fijn om meerdere API endpoints uit verschillende projecten te laten samen werken want zo werkt het in het dagelijks leven ook denk maar aan een bestelling bij Bol.com bijvoorbeeld als we een bestelling plaatsen wordt er een API call naar de bank gedaan, naar de bezorg dienst om ons te voorzien van een track and trace code en ga zo maar door.

3.3.2.3. FileModule voor file uploads naar S3 bucket

De NestJS backend is modulair opgezet, wat inhoudt dat elke functionaliteit netjes is ingekapseld in een eigen module. Dit biedt voordelen op het gebied van onderhoudbaarheid, testbaarheid én herbruikbaarheid. In dit project kunnen met name de volgende modules en services hergebruikt worden in toekomstige toepassingen of microservices:

FILEMODULE

De `FileModule` bevat alle logica rondom bestandsbeheer. Deze module is opgebouwd uit meerdere lagen:

- **Controllers** zoals `FileController`, die HTTP-verzoeken afhandelen.
- **DTOs** zoals `CreateFileDto`, die input valideren en transformeren.
- **Services** zoals `FileFlowService` en `S3Service`, die de kernlogica uitvoeren (zoals het aanmaken, downloaden en verwijderen van bestanden).
- **Repositories** zoals `FileRepository`, voor directe interactie met de database (middels TypeORM).
- **Entities** zoals `File`, die de database structuur beschrijven.
- **Enums** zoals `MimeType`, die consistentie brengen in bestandsindelingen.

Door de heldere scheiding van verantwoordelijkheden is deze module bijzonder goed herbruikbaar in andere applicaties waar bestandsbeheer via S3 nodig is. Denk hierbij aan interne tooling, klantportalen of documentmanagementsystemen.

S3SERVICE

De `S3Service` is verantwoordelijk voor communicatie met S3. Deze service bevat methoden voor

- Het genereren van **tijdelijke upload- en download-URLs**.
- Het **uploaden van bestanden** (ook via stream).
- Het **verwijderen van bestanden** uit de S3 bucket.

Omdat deze service enkel afhankelijk is van configuratie via de `ConfigService` en geen hardgekoppelde businesslogica bevat, kan deze service zeer eenvoudig worden gekopieerd of als aparte bibliotheek worden aangeboden binnen het bedrijf.

3.3.2.4. Authenticatie via Zitadel

De backend is volledig beschermd via **Zitadel authenticatie** en **permissie-gebaseerde autorisatie**. Elk endpoint vereist een geldige JWT token uitgegeven door Zitadel en alleen gebruikers met de juiste permissies mogen bepaalde acties uitvoeren.

JWT-VALIDATIE VIA MIDDLEWARE

Elke inkomende HTTP-request wordt onderschept door de `AuthMiddleware`. Deze middleware:

- Controleert op de aanwezigheid van een `Bearer` token in de `Authorization` header;
- Verifieert het token met behulp van het public JWKS endpoint van Zitadel;
- Zoekt bij succesvolle validatie de interne gebruiker op aan de hand van de `sub` (Zitadel user ID);
- Bewaart de authenticatiecontext (`AuthContent`) in `AsyncLocalStorage` voor gebruik in de rest van de request lifecycle.

ENDPOINTBESCHERMING VIA GUARDS

De applicatie maakt gebruik van NestJS guards op globaal niveau:

- `AuthGuard`: vereist dat er een geldige authenticatiecontext bestaat.
- `PermissionsGuard`: controleert of de gebruiker de juiste rechten heeft op basis van de **@Permissions()** decorator op methoden.

VOORBEELD VBAN EEN BESCHEMENDE ENDPOINT

Onderstaand voorbeeld toon een endpoint die verantwoordelijk is voor het verwijderen van documenten. Het is beveiligd op twee niveaus:

1. Je moet **ingelogd** zijn via Zitadel (authenticatie vereist).
2. Je moet beschikken over de permissie **document.delete** (Dit is een persmission die beschreven staat in een Enum en wordt toegewezen aan een rol)

```
@Delete('delete-documents')
@Permissions(Permission.DOCUMENT_DELETE)
@ApiOkResponse({ description: 'Documents removed successfully!' })
async removeFromPgVector(
  @Body() fileUuids: string[]
): Promise<DeleteDocumentsResponse> {
  return await this.deleteDocumentsUseCase.execute(fileUuids)
}
```

DECLAREREN VAN TOEGANSRECHTEN

```ts
export enum Permission {
  DOCUMENT_DELETE = 'document.delete',
  DOCUMENT_READ = 'document.read',
  ...
}
```

AUTHRISATIEFLOW

1. **Request** komt binnen met **JWT**.
2. **AuthMiddelware** valideert de token en zet **AuthContent** klaar.
3. **AuthGuard** checkt of een gebruiker aanwezig is.
4. **PermissionsGuard** checkt of de gebruiker de juiste **permissies** heeft
5. **Controller** voert de **use case** uit als alle checks slagen.

3.3.2.5. DocumentsModule

De `documents` module is verantwoordelijk voor het beheren van documenten, waaronder het uploaden, embedden, zoeken, ophalen en verwijderen van documenten. Deze module is modulair opgebouwd en volgt de architectuur van Wisemen, waarbij elke use-case zijn eigen subfolder en bijhorende controller, use case class, response type en eventuele services of DTO's bevat.

```
documents/
  ├── entities/                  ← Domeinmodellen van documenten
  ├── use-cases/                 ← Elke use-case in een aparte map
  │     ├── delete-documents/
  │     ├── embed-documents/
  │     ├── get-documents/
  │     └── search-query/
  └── document.module.ts         ← Centrale moduledefinitie
```

DOCUMENT.MODULE.TS

De `DocumentModule` fungeert als de hoofdingang van deze featuremodule, het registreet alle submodules van de verschillende use-cases die samen de documentfunctionaliteit vormen.

```ts
import { Module } from '@nestjs/common'
import { PermissionModule } from '../permission/permission.module.js'
import { GetDocumentsModule } from './use-cases/get-documents/get-documents.module.js'
import { DeleteDocumentsModule } from './use-cases/delete-documents/delete-documents.module.js'
import { EmbedDocumentsModule } from './use-cases/embed-documents/embed-documents.module.js'
import { SearchDocumentsModule } from './use-cases/search-query/search-query.module.js'

@Module({
  imports: [
    GetDocumentsModule,
    DeleteDocumentsModule,
    EmbedDocumentsModule,
    SearchDocumentsModule,
    PermissionModule
  ]
})
export class DocumentModule {}
```

DOEL VAN DEZE MODULE

- **Centralisatie**: alle documentgerelateerde functies zijn logisch gegroepeerd in één overkoepelende module
- **Losse koppeling**: Elke submodule is zelfstandig en kan gemakkelijk worden aangepast of uitgebreid zonder de rest te beïnvloeden
- **Herbruikbaarheid & schaalbaarheid**: Nieuwe document-use-cases kunnen eenvoudig toegevoegd worden via een import
- **Security**: Elke endpoint is standaard beschermd door authenticatie en autorisatie op basis van Zitadel & persissions

3.3.2.6. Get Document Module

De `GetDocumentModule` bevat de functionaliteti voor het ophalen van documenten die behoren tot een geauthenticeerde gebruiker. Dit is een van de basisfunctionaliteiten binnen de documentverwerking van de pipeline

Doel
Het ophalen van documentnamen op basis van de gebruiker die is aangemeld. De module zorgt ervoor dat alleen documenten van de ingelogde gebruiker zichtbaar zijn, dankzij `AuthStorage` die gekoppeld is aan Zitadel-authenticatie

GET-DOCUMENT.CONTROLLER.TS

Bevat de route-definitie en koppelt die aan de use-case.

```ts
import { Controller, Get } from '@nestjs/common'
import { ApiTags, ApiOkResponse, ApiOAuth2, ApiOperation } from '@nestjs/swagger'
import { Permission } from '../../../../modules/permission/permission.enum.js'
import { Permissions } from '../../../../modules/permission/permission.decorator.js'
import { GetDocumentsUseCase } from './get-documents.use-case.js'
import { GetDocumentsResponse } from './get-documents.response.js'

@ApiTags('Documents')
@ApiOAuth2([])
@Controller('documents')
export class GetDocumentsController {
  constructor (private readonly useCase: GetDocumentsUseCase) {}

  @Get('/get-documents')
  @Permissions(Permission.DOCUMENT_READ)
  @ApiOperation({ summary: 'Get documents' })
  @ApiOkResponse({
    description: 'Successfully retrieved document names',
    type: GetDocumentsResponse
  })
  async getFilenames (): Promise<GetDocumentsResponse> {
    return await this.useCase.execute()
  }
}

```

GET-DOCUMENT.MODULE

Registreert de controller en de use-case provider. Via TypeOrmModule.ForFeature wordt de Document entitiy toegankelijk gemaakt voor dependency injection.

```ts
import { Module } from '@nestjs/common'
import { TypeOrmModule } from '@wisemen/nestjs-typeorm'
import { Document } from '../../entities/document.entity.js'
import { GetDocumentsUseCase } from './get-documents.use-case.js'
import { GetDocumentsController } from './get-documents.contoller.js'

@Module({
  imports: [TypeOrmModule.forFeature([Document])],
  controllers: [GetDocumentsController],
  providers: [GetDocumentsUseCase]
})
export class GetDocumentsModule {}

```

GET DOCUMENTS.USE-CASE.TS

Deze use-case bevat de businesslogica

- Haalt de userUuid op via AuthStorage
- Voert een query uit op de document-tabel gefilterd op userId
- Filtert en retourneert de documentnamen

```
GetDocumentsUseCase {
  constructor (
    @InjectRepository(Document)
    private documentRepository: Repository<Document>,
    private authStorage: AuthStorage
  ) {}

  async execute (): Promise<GetDocumentsResponse> {
    const userUuid = this.authStorage.getUserUuid()
    const documents = await this.documentRepository.find({
      where: { userId: userUuid },
      select: ['name']
    })

    const filenames = documents
      .map(doc => doc.name)
      .filter(name => name !== null)

    return new GetDocumentsResponse(filenames)
  }
}
```

GET-DOCUMENT.RESPONSE.TS

Een eenvoudige DTO die de lijst van documentnamen bevat en beschrijft via Swagger-Decorators

```
GetDocumentsResponse {
  @ApiProperty({ type: [String], description: 'List of document names' })
  documents: string[]

  constructor (documents: string[]) {
    this.documents = documents
  }
}

```

3.3.2.7. Delete Documents Module

De DeleteDocumentsModulev verzorgt de logica voor de het verwijderen van documenten en alle bjhoordende entiteiten, zoals chunks en vector-embeddings. Deze operatie is permanent en kan ekel worden uitgevoerd door gebruikers met de juiste premissie.

DOEL

Deze module laat gebruikers toe om hun eigen documenten en alle gerelateerde data volledig te verwijderen uit de database. Dit is vooral belangrijk voor gegevensbeheerd en het opruimen van vectorgegeven die niet meer nodig zijn.

DELETE-DOCUMENTS.CONTROLLER.TS

Beheert de API endpoint en koppelt het aan de use-case:

```ts
@ApiTags('Documents')
@ApiOAuth2([])
@Controller('documents')
export class GetDocumentsController {
  constructor (private readonly useCase: GetDocumentsUseCase) {}

  @Get('/get-documents')
  @Permissions(Permission.DOCUMENT_READ)
  @ApiOperation({ summary: 'Get documents' })
  @ApiOkResponse({
    description: 'Successfully retrieved document names',
    type: GetDocumentsResponse
  })
  async getFilenames (): Promise<GetDocumentsResponse> {
    return await this.useCase.execute()
  }
}
```

DELETE-DOCUMENTS.MODULE.TS

Verbindt de controller, use-case en entiteiten

```ts
@Module({
  imports: [TypeOrmModule.forFeature([Document])],
  controllers: [GetDocumentsController],
  providers: [GetDocumentsUseCase]
})
export class GetDocumentsModule {}

```

DELETE-DOCUMENTS.USE-CASE.TS

Deze use-case bevat de businesslogica:

- Valideert dat documenten zijn meegegeven
- Verifeert of de gebruiker eigenaar is van de documenten
- Verwijdert eerst gerelateerde data (embedding, chunk), daarna het document zelf.
- Werkt met een QueryBuilder voor performance en controle
- Gooi een fout als er een mismatch is in eigenaarschap

BELANGRIJKE SECURITY CHECK

```
if (documentsToDelete.length !== documentUuids.length) {
  throw new Error('Not all documents could be deleted. Check ownership.')
}
```

VERWIJDEREN VAN GERELATEERDE DATA

```
await this.documentRepository
        .createQueryBuilder()
        .delete()
        .from('embedding')
        .where('document_uuid IN (:...documentUuids)', { documentUuids: documentUuidsToDelete })
        .execute()
```

3.3.2.8. Embed Documents Module

De `EmbedDocumentsModule` verzorgt de integratie tussen onze NestJS-backend en een Python-gebaseerde service die verantwoordelijk is voor het analyseren en embedden (vectoriseren) van documenten. Deze module wordt gebruikt wanneer een gebruiker een bestaand document (bijvoorbeeld PDF) wil laten verwerken tot doorzoekbare vectoren, zonder het opnieuw te hoeven uploaden.

DOEL

Deze module stelt gebruikers in staat om bestaande documenten uit S3 te embedden, wat betekent dat ze automatisch worden omgezet naar vectorrepresentaties. Dit is essentieel voor zoekfunctionaliteit, semantische analyse, en AI-gedreven toepassingen.
De daadwerkelijke embedding gebeurt in een Python FastAPI-service — de NestJS-module is verantwoordelijk voor:

- Validatie
- Documentregistratie
- Verzoek naar de Python API versturen

EMBED-DOCUMENT.CONTROLLER.TS

Beheert het HTTP-endpoitn en verbindt het met de use-case

```
@Post('embed-existing-document')
@Permissions(Permission.DOCUMENT_UPLOAD)
@ApiCreatedResponse({
  description: 'Successfully initiated embedding for an existing document'
})
async embedExistingDocument (
  @Body() body: EmbedExistingDocumentRequest
)
```

EMBED-DOCUMENTS.MODULE.TS

Verbindt de controller, use-case, repositories en services

```
@Module({
  imports: [TypeOrmModule.forFeature([Document]), HttpModule, FileModule],
  controllers: [EmbedDocumentController],
  providers: [EmbedDocumentUseCase, FileFetcherService, EmbeddingService],
  exports: [EmbedDocumentUseCase]
})
export class EmbedDocumentsModule {}
```

- Registreert de Document entitiy voor database-interactie.
- Maakt HttpModule beschikbaar om HTTP-verzoeken naar de Python-backend te sturen
- Importeert FileModule voor het ophalen van bestanden uit S3

EMBED-DOCUMENT.USE-CASE.TS

Bevat de volgende businesslogica

```
const fileBuffer = await this.fileFetcherService.fetchFileBuffer(s3Key)

```

- Haalt via een tijdelijke S3-link de inhoud van het document op
- We doen dit om de bestandsnaam te kunenn loggen en valideren

```
const newDocument = this.documentRepository.create({
    uuid: randomUUID(),
    name: fileBuffer.fileName,
    userId: userUuid,
    visibility: request.visibility
  })

  const savedDocument = await this.documentRepository.save(newDocument)
```

- Hier registeren we het document ion onze database met een nieuw UUID, gekoppeld aan de gebruiker
- De beschikbaarheid (public/private) wordt vastgelegd via *visiibility*
- Dit garandeert traceerbaarheid en eigenaarschap

```
const embedRequest: EmbedDocumentToApiServiceRequest = {
      s3_key: s3Key,
      user_id: userUuid,
      document_name: fileBuffer.fileName
    }

    try {
      this.logger.log(`Sending embedding request for existing document: ${fileBuffer.fileName} with S3 key: ${s3Key}`)
      await this.embeddingService.triggerEmbedding(embedRequest)
      this.logger.log(`Embedding request sent successfully for document: ${fileBuffer.fileName}`)
    } catch (error) {
      this.logger.error(`Error sending embedding request for ${fileBuffer.fileName}: ${error}`)
    }
```

- Stel een embed-verzoek samen en stuurt het via HTTP naar de Python-service
- Deze sercice zal het bestand ophalen uit S3, splitsen en vectoriseren.

EMBEDDING.SERVICE.TS

De `EmbedDocumentsModule` in de NestJS backend vormt een cruciale schakel in de RAG-architectuur. Het is verantwoordelijk voor het initiëren van het document-embeddingproces, waarbij documenten worden omgezet in een formaat dat geschikt is voor semantische zoekopdrachten.

DOEL

De EmbeddingService verstuurt een HTTP-verzoek met documentgegevens naar een Python-gebaseerde FastAPI-service. Deze service haalt vervolgens het document op vanuit Amazon S3, verwerkt de inhoud (zoals tekstsplitsing en vectorisatie), en slaat de gegenereerde vectoren op in een vector database.

Het verplaatsen van deze zware NLP-logica naar een aparte Python-service maakt het systeem schaalbaar, modulair en beter geschikt voor AI-taken.

```typescript
export interface EmbedDocumentToApiServiceRequest {
  s3_key: string
  user_id: string
  document_name: string
}

@Injectable()
export class EmbeddingService {
  private readonly fastApiEmbedBaseUrl: string = process.env.PYTHON_API_ENDPOINT!
  private readonly fastApiEmbedPath: string = '/documents/embed-from-s3/'
  private readonly logger = new Logger(EmbeddingService.name)

  constructor (private readonly httpService: HttpService) {}

  async triggerEmbedding (embedRequest: EmbedDocumentToApiServiceRequest): Promise<void> {
    const fullUrl = `${this.fastApiEmbedBaseUrl}${this.fastApiEmbedPath}`

    try {
      this.logger.log(`Sending embedding request to: ${fullUrl}`, embedRequest)
      await firstValueFrom(this.httpService.post(fullUrl, embedRequest))
      this.logger.log('Embedding request sent successfully')
    } catch (error) {
      this.logger.error('Error sending embedding request:', error)

      throw error
    }
  }
}

```

- **EmbedDocumentToApiServiceRequest Interface:** definieert de structuur van de data die naar de Python API wordt gestuurd:
    - **s3_key:** de key van het S3 document
    - **user_id:** ID van de gebruiker
    - **document_name:** Naam van het document.
- **EmbeddingService Class:**
    - **fastApiEmbedBaseUrl:** basis-URL van Python
- **TriggerEmbedding Method:** Asynchrone methode om het embedding-proces the starten.
    - **fullUrl:** Samenstelling van de volledige URL voor Python API

FILE-FETCH.SERVICE.TS

De FileFetcherService in de NestJS backend is verantwoordelijk voor het ophalen van de inhoud van bestanden op basis van hun unieke ID (fileUuid). Het maakt gebruik van een andere service (FileFlowService) om een tijdelijke URL te genereren waarmee het bestand kan worden gedownload.

DOEL

De FileFetcherService haalt de binaire inhoud (buffer) van een bestand op. Het ontvangt een fileUuid, vraagt een tijdelijke downloadlink aan bij de FileFlowService, downloadt het bestand via die link, controleert op fouten tijdens het downloaden, en retourneert de inhoud als een Buffer samen met de originele bestandsnaam.

```typescript
interface FetchedFileBuffer {
  buffer: Buffer
  fileName: string
}

@Injectable()
export class FileFetcherService {
  private readonly logger = new Logger(FileFetcherService.name) // Create an instance of Logger

  constructor (private readonly fileFlowService: FileFlowService) {}

  async fetchFileBuffer (fileUuid: string): Promise<FetchedFileBuffer> {
    const { file, temporaryUrl } = await this.fileFlowService.getTemporaryUrl(fileUuid)

    this.logger.debug(`Generated Temporary URL for file UUID ${fileUuid}: ${temporaryUrl}`)

    const response = await fetch(temporaryUrl)

    if (!response.ok) {
      this.logger.error(`Failed to fetch file from temporary URL: ${response.status} ${response.statusText} - URL: ${temporaryUrl}`)

      throw new Error(`Failed to fetch file from temporary URL: ${response.status} ${response.statusText}`)
    }
    const buffer = await response.arrayBuffer()

    return {
      buffer: Buffer.from(buffer),
      fileName: file.name
    }
```

- **FetchedFileBuffer Interface:** defineert de structuur van het object dat door de fetchFileBuffer-methode wordt geretourneerd:
    - **Buffer:** node.js buffer-object dat de binaire inhoud van het bestand bevat.
    - **Filename:** originele naam van het bestand
- **FileFetchService Class:**
    - **Injectable():** Markeert de klasse als service die kan worden geinjecteerd in andere componenten
    - **constructor(private readonly fileFlowService: FileFlowService):** De constructor ontvangt een instantie van de FileFlowService via dependency injection. Dit stelt de FileFetcherService in staat om de methoden van de FileFlowService te gebruiken
- **fetchFileBuffer(fileUuid: string): Promise<FetchedFileBuffer> Method:** Een asynchrone methode die het ophalen van het bestand initieert.

3.3.2.9. Search Query Module

De **Search Documents Module** in de NestJS backend is specifiek ontworpen om gebruikers de mogelijkheid te bieden om te zoeken binnen de geïndexeerde documenten. Het fungeert als het toegangspunt en de orkestrator voor het zoekproces.

DOEL

Het primaire **doel** van de Search Documents Module is om een zoekfunctionaliteit aan te bieden waarmee gebruikers relevante informatie kunnen vinden binnen de documenten die eerder via het EmbedDocumentsModule zijn verwerkt en gevectoriseerd. Dit omvat:

- **Het ontvangen van zoekopdrachten van gebruikers:** Via een HTTP-eindpunt.
- **Het valideren van de zoekopdracht:** Zorgen dat de input aan de vereisten voldoet.
- **Het delegeren van de zoekactie:** Het doorsturen van de zoekopdracht naar een andere service (SearchQueryUseCase) die de daadwerkelijke logica voor het uitvoeren van de zoekopdracht bevat.
- **Het retourneren van de zoekresultaten:** Het formatteren en presenteren van de resultaten aan de gebruiker.

Kortom, deze module maakt het **zoeken naar informatie binnen de ge-embedde documenten** mogelijk, waardoor gebruikers snel de antwoorden en context kunnen vinden die ze nodig hebben. Het is een cruciaal onderdeel van een Retrieval-Augmented Generation (RAG) systeem, waarbij het ophalen van relevante context de basis vormt voor het genereren van antwoorden.

SEARCH-QUERY.CONTROLLER.TS

Deze fungeert als een verkeersleider voor de zoekopdrachten. Wanneer een gebruikter een zoekopdracht invult via de /documents/search endpoint, ontvangt de controller dat verzoek. Hij zorgt ervoor dat de zoekopdracht en de eventueel gewenste aantal resultaten correct zijn ontvangen. Vervolgens delegeert de controller de daadwerkelijke zoekachtie naar de SearchQueryUseCase

```
@ApiTags('Documents')
@ApiOAuth2([])
@Controller('documents')
export class SearchQueryController {
  constructor (
    private readonly searchQueryUseCase: SearchQueryUseCase
  ) {}

  @Get('/search')
  @Permissions(Permission.DOCUMENT_SEARCH)
  @ApiOperation({ summary: 'Search documents' })
  @ApiQuery({
    name: 'query',
    type: String,
    required: true
  })
  @ApiQuery({
    name: 'k',
    type: Number,
    required: false
  })
  @ApiResponse({
    status: 200,
    description: 'Search results',
    type: SearchQueryResponse
  })
  async search (@Query() searchQueryDto: SearchQueryDto): Promise<SearchQueryResponse> {
    return this.searchQueryUseCase.search(searchQueryDto)
  }
}

```

SEARCH-QUERY.USE-CASE.TS

Dit zijn de “hersenen” van de zoekfunctionaliteit. Deze service ontvangt de zoekopdracht van de controller. Eerst controleren we of de gebruiker is ingelogd (via AuthStorage). Vervolgens stuurt hij een verzoek (via HttpService) naar de Python backend met de zoekterm en de gebruikers-ID. De service in Python voert de echte zoekactie uit. Zodra onze Python backend een antwoord terug geeft verwerkt de use-case de data en stuurt deze teurg naar de controller, die het vervolgens naar de gebruiker stuurt.

```typescript
@Injectable()
export class SearchQueryUseCase {
  constructor (
    private readonly httpService: HttpService,
    private readonly authStorage: AuthStorage
  ) {}

  async search (dto: SearchQueryDto): Promise<SearchQueryResponse> {
    const userUuid = this.authStorage.getUserUuid()

    if (!userUuid) {
      throw new InternalServerErrorException('User not authenticated or missing.')
    }

    try {
      const response = await this.httpService.axiosRef.get(
        'http://localhost:8000/documents/search-pgvector/',
        {
          params: {
            query: dto.query,
            user_id: userUuid,
            k: dto.k ?? 3
          }
        }
      )

      const data = response.data as SearchQueryResponse

      return {
        answer: data.answer,
        contextRetrievalTime: data.contextRetrievalTime,
        answerGenerationTime: data.answerGenerationTime,
        totalTime: data.totalTime,
        retrievedChunks: data.retrievedChunks
      }
    } catch (error: unknown) {
      throw new InternalServerErrorException(`Failed to search query: ${error instanceof Error ? error.message : String(error)}`)
    }
  }
}

```

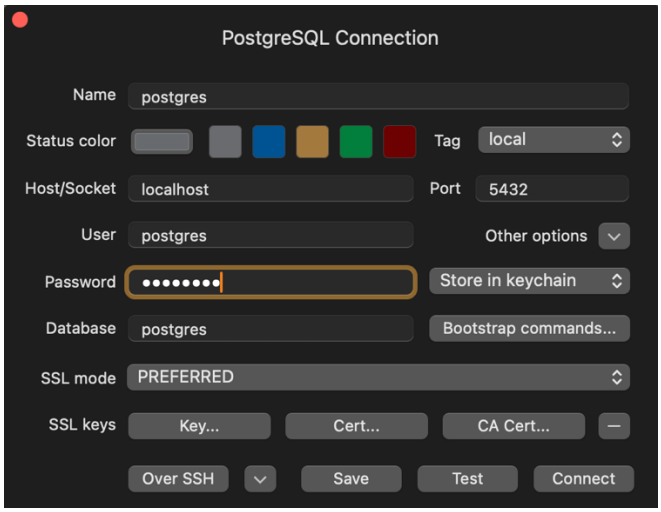
3.3.3. Hoe de applicatie te gebruiken

Om onze applicatie te laten draaien in development (applicatie is nog niet in productie hier kom ik later op terug in het deel conclusie RAG) werken moeten we een paar stappen uitvoeren zodat de werkomgeving klaar is om al onze request af te handelen.

3.3.3.1. PostgreSQL setup handleiding

Tijdens mijn project maken we gebruik van een PostgreSQL database ik gebruikte TablePlus voor het aanmaken van mijn databases.

- Open TablePlus → rechter muisklik new connection
- Kies dan voor PostgreSQL
- Vul alle gegevens in zoals op onderstaande foto (password = password)



3.3.3.2. NestJS setup handleiding

Eerst en vooral moeten we onze github repository clone dit doen we als volgt.

1. Open je terminal op de gewenste locatie
2. Voer het volgende commando uit: git clone https://github.com/wisemen-digital/wisemen-rag-nestjs-api.git && cd wisemen-rag-nestjs-api && code .

```
❯ git clone https://github.com/wisemen-digital/wisemen-rag-nestjs-api.git
Cloning into 'wisemen-rag-nestjs-api'...
remote: Enumerating objects: 955, done.
remote: Counting objects: 100% (955/955), done.
remote: Compressing objects: 100% (613/613), done.
remote: Total 955 (delta 335), reused 890 (delta 278), pack-reused 0 (from 0)
Receiving objects: 100% (955/955), 1.07 MiB | 6.95 MiB/s, done.
Resolving deltas: 100% (335/335), done.
❯ cd wisemen-rag-nestjs-api
❯ code .
```

3. Open de terminal in onze applicatie
4. Voer het volgende commando uit: pnpm install

```
❯ pnpm install
Lockfile is up to date, resolution step is skipped
Packages: +1002
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

                Update available! 9.15.6 → 10.11.0.
Changelog: https://github.com/pnpm/pnpm/releases/tag/v10.11.0
                Run "pnpm self-update" to update.

Progress: resolved 1002, reused 990, downloaded 11, added 1002, done
node_modules/.pnpm/@nestjs+core@11.0.12_@nestjs+common@11.0.16_class-transformer@0.5.1_class-val
idator@0.14.1_fi_u5icn6gsifweexue65bgaml4fa/node_modules/@nestjs/core: Running postinstall scrip
node_modules/.pnpm/@nestjs+core@11.0.12_@nestjs+common@11.0.16_class-transformer@0.5.1_class-val
idator@0.14.1_fi_u5icn6gsifweexue65bgaml4fa/node_modules/@nestjs/core: Running postinstall scrip
t, done in 380ms
```

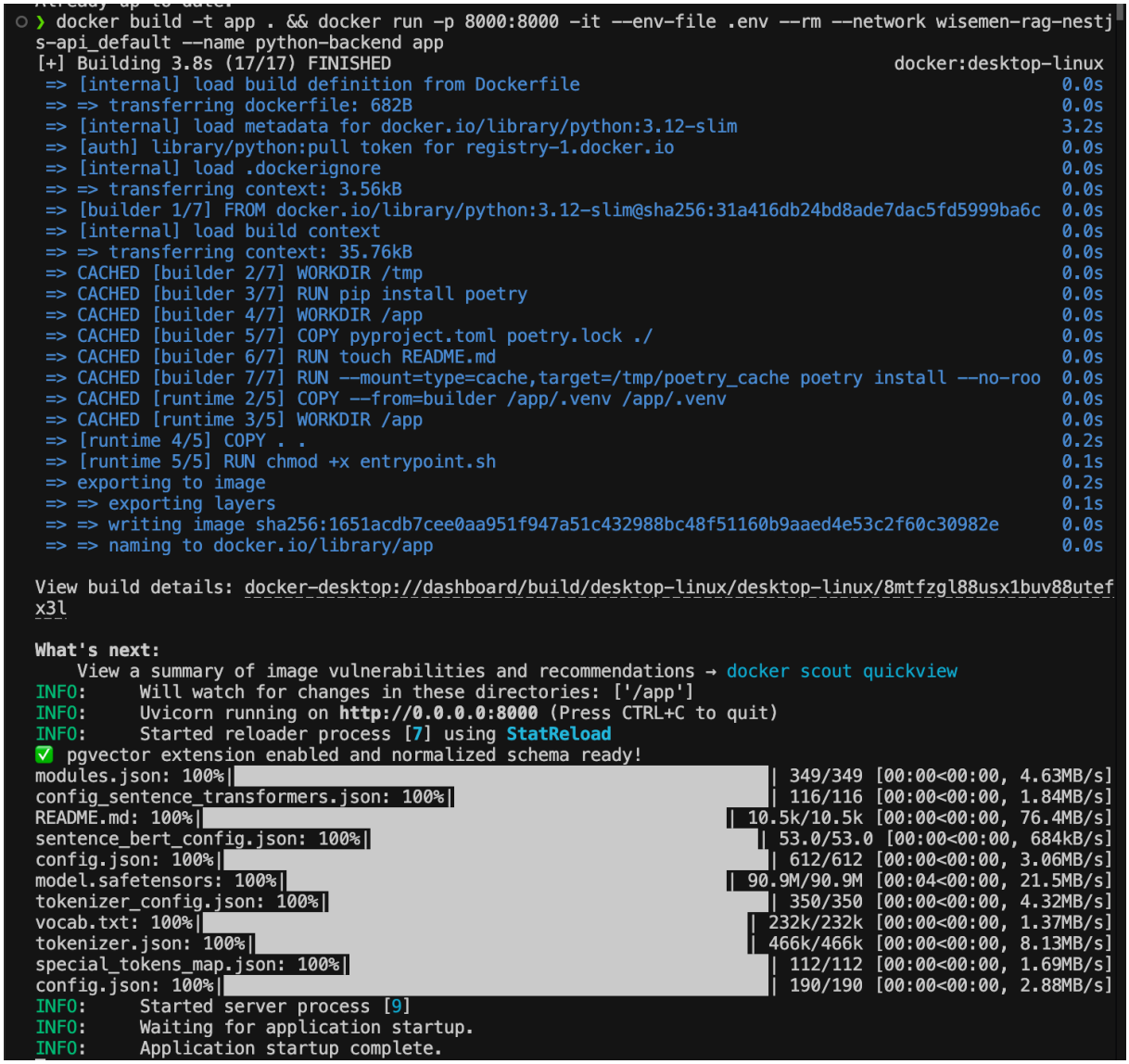
5. Maak een kopie van de .env.test file en noem deze .env
6. Start alle nodige containers op doormiddel van dit commando: docker compose up -d
7. Start de applicatie doormiddel van: pnpm run start:dev
8. Ga naar localhost:3000/docs/api

3.3.3.3. Python setup handleiding

We hebben een tweede backend repository gemaakt in Python voor API endpoints die specifieke LLM packages nodig hebben die gespeciaal gemaakt zijn voor Python

1. Open je terminal op de gewenste locatie

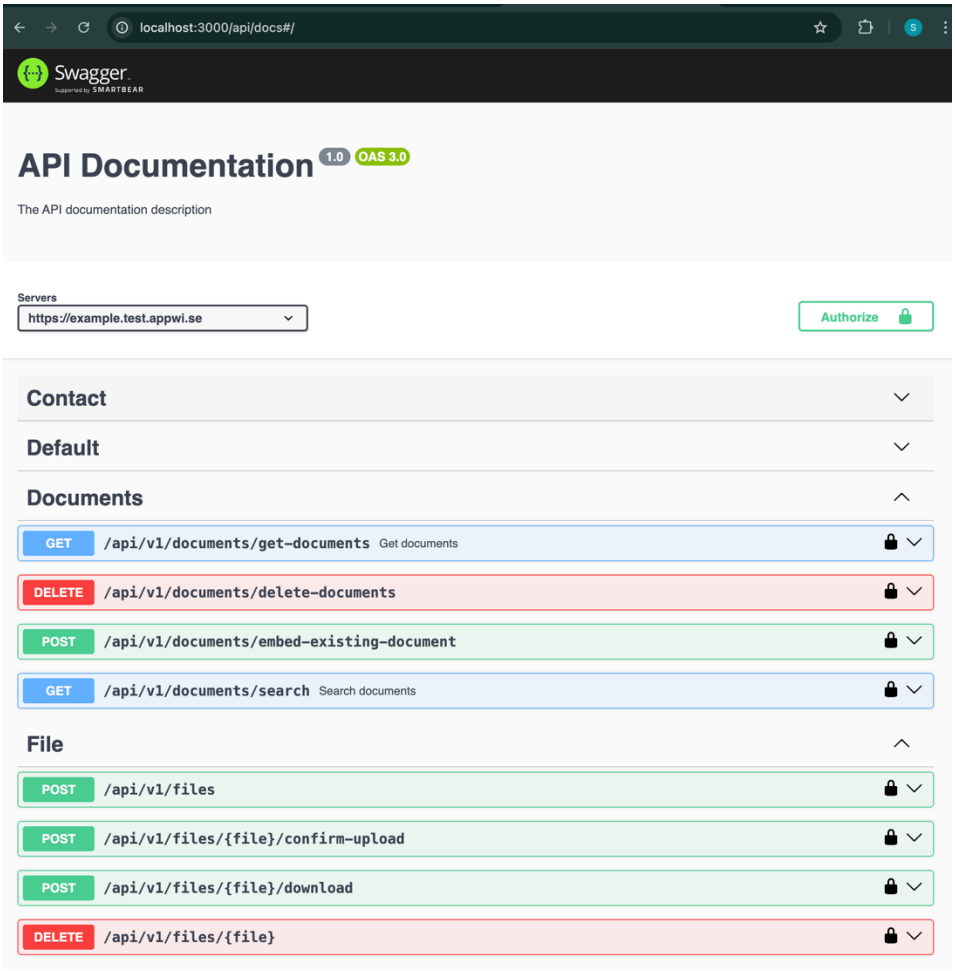
2. Voer het volgende commando uit: git clone https://github.com/wisemen-digital/wisemen-rag-python-api.git && cd wisemen-rag-nestjs-api && code .
3. Open de terminal in je project en voer het volgende commando uit: docker build -t app . && docker run -p 8000:8000 -it --env-file .env --rm --network wisemen-rag-nestjs-api_default --name python-backend app

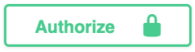


3.3.3.4. RAG flow showcase

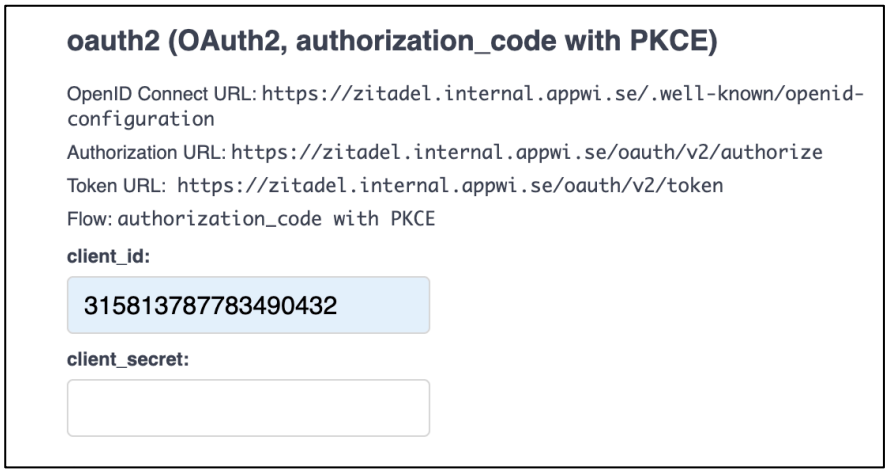
Wanneer alle twee de backend repositories, docker containers en database klaar staan kunnen we onze applicatie pipeline gaan gebruiken

We gaan naar localhost:3000/api/docs

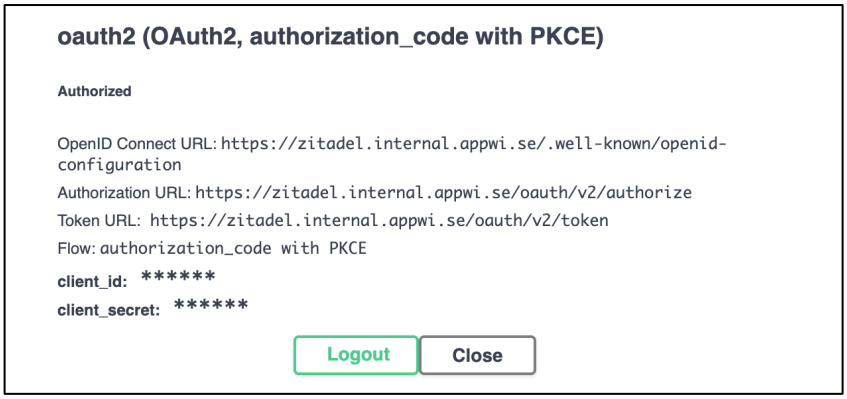


Vervolgens klikken we op  om een correcte bearer token te ontvangen zodat we de applicatie kunnen uitvoeren

Kies voor de volgende methode wanneer we ons gaan authoriseren

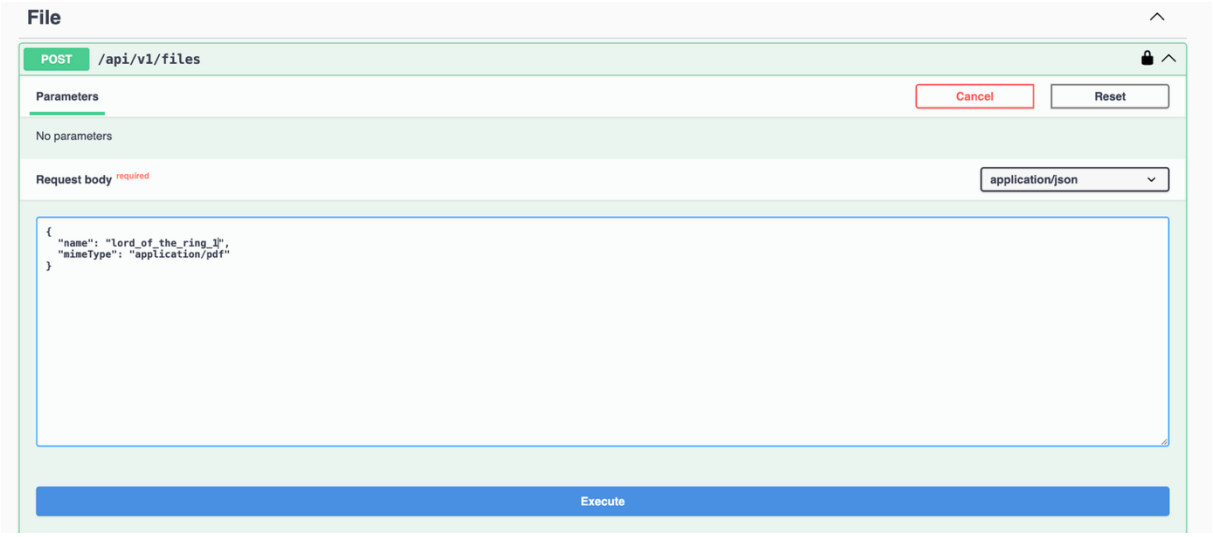


Vervolgens krijgen we een redirect naar zitadel en daar selecteren we het account waarmee we ons willen aanmelden. Als dit de eerste keer is dat je jezelf aanmeld in de applicatie worden je gegevens ook opgeslagen in de database.

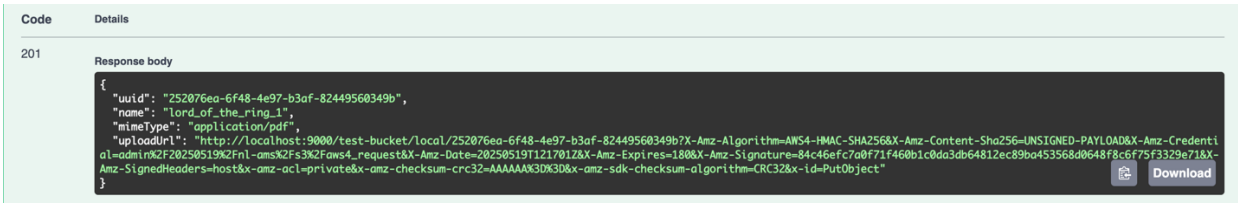


Eerst en vooral willen we dat er documenten worden geupload zodat we deze later kunnen embedden in onze database.

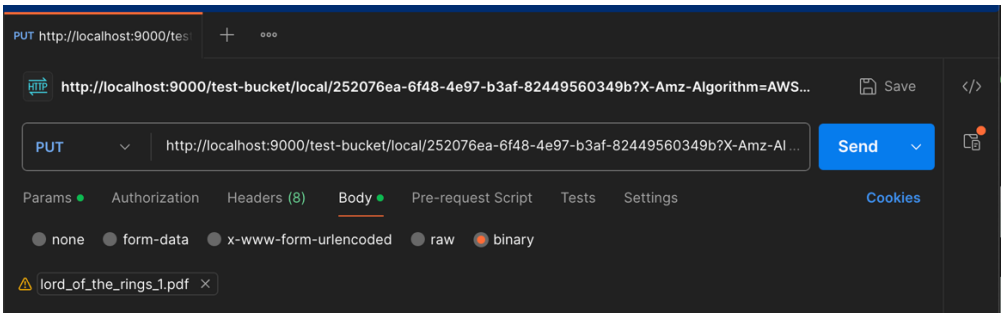
We gaan nu de boeken van Lord Of The Ring uploaden.
We geven de naam in van het document en geven het geweste mineType in daarna klikken we op Execute



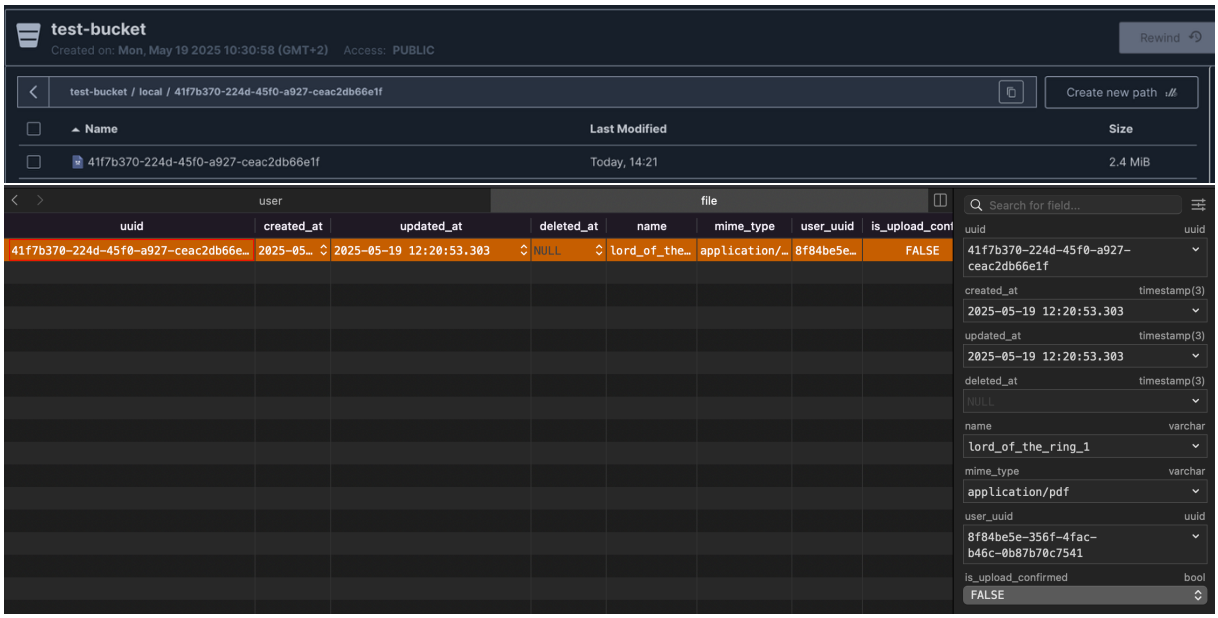
Dit is onze response body we zien dat we een uploadUrl krijgen dit is een API endpoint die we kunnen gebruiken in Postman van MinIO, waarom laten we MinIO dit afhandelen? Omdat we er zo voor zorgen dat er minder traffic is op onze applicatie.



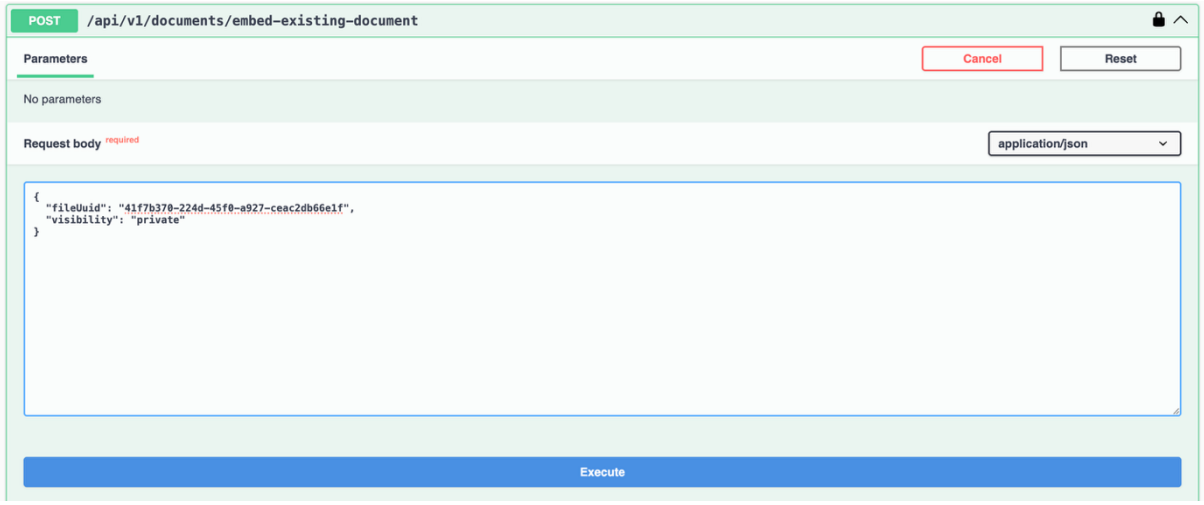
In Postman kiezen we nu voor een put methode en plakken we de uploadUrl hierin.
Vervolgens selecteren we de body tab en selecteren we de binary radio button om zo de gewenste file te selecteren en klikken we op Send



File is nu opgeslagen in onze S3 bucket en in onze database



Volgende stap in onze RAG pipeline is het embeden van de geuploade documenten. Dit doen we met de api/v1/documents/embed-existing-documents endpoint. Hier geven we de fileUuid mee uit de file tabel in onze database of S3 bucket en selecteren we de geweste visibility.



Code | Details

201

Response body

```
{
  "documentUuid": "35d8040b-97d5-47ed-baaf-9e077fe992d1"
}
```

Download

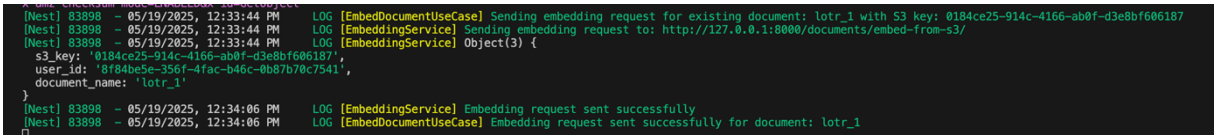
Response headers

```
access-control-expose-headers: Content-Disposition
connection: keep-alive
content-length: 55
content-type: application/json; charset=utf-8
date: Mon,19 May 2025 12:34:06 GMT
etag: W/"37-pE/SdtlPcKSH1jqpvrmP0nj/lsA"
keep-alive: timeout=5
vary: Origin
x-powered-by: Express
```

Responses

Code	Description	Links
201	Successfully initiated embedding for an existing document	*No links*

NestJS backend



```
[Nest] 83898  - 05/19/2025, 12:33:44 PM     LOG [EmbedDocumentUseCase] Sending embedding request for existing document: lotr_1 with S3 key: 0184ce25-914c-4166-ab0f-d3e8bf606187
[Nest] 83898  - 05/19/2025, 12:33:44 PM     LOG [EmbeddingService] Sending embedding request to: http://127.0.0.1:8000/documents/embed-from-s3/
[Nest] 83898  - 05/19/2025, 12:33:44 PM     LOG [EmbeddingService] Object(3) {
  s3_key: '0184ce25-914c-4166-ab0f-d3e8bf606187',
  user_id: '8f84be5e-356f-4fac-b46c-0b87b70c7541',
  document_name: 'lotr_1'
}
[Nest] 83898  - 05/19/2025, 12:34:06 PM     LOG [EmbeddingService] Embedding request sent successfully
[Nest] 83898  - 05/19/2025, 12:34:06 PM     LOG [EmbedDocumentUseCase] Embedding request sent successfully for document: lotr_1
```

Python backend

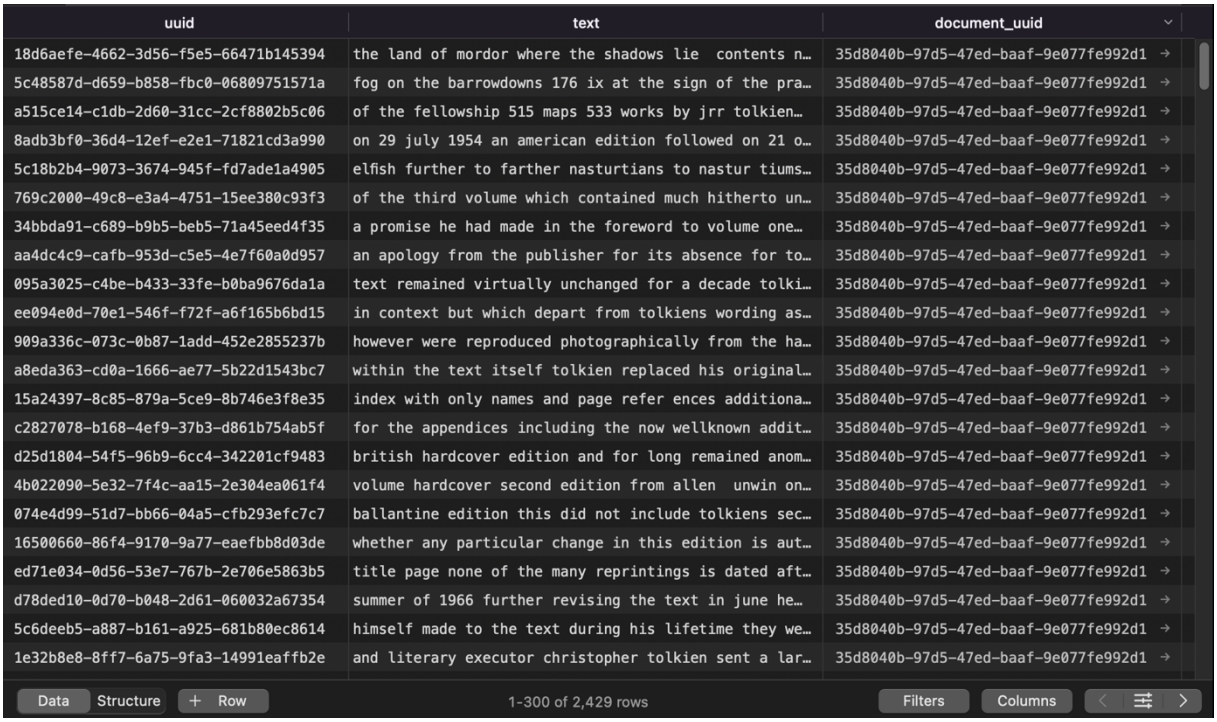
```
Python: Attempting to download from S3 with key: local/0184ce25-914c-4166-ab0f-d3e8bf606187, bucket: te
st-bucket
INFO:     192.168.65.1:19572 - "POST /documents/embed-from-s3/ HTTP/1.1" 200 OK
```

Database

- Document-tabel

uuid	name	user_id	visibility
35d8040b-97d5-47ed-baaf-9e077fe992d1	lotr_1	8f84be5e-356f-4fac-b46c-0b87b70c7541	private

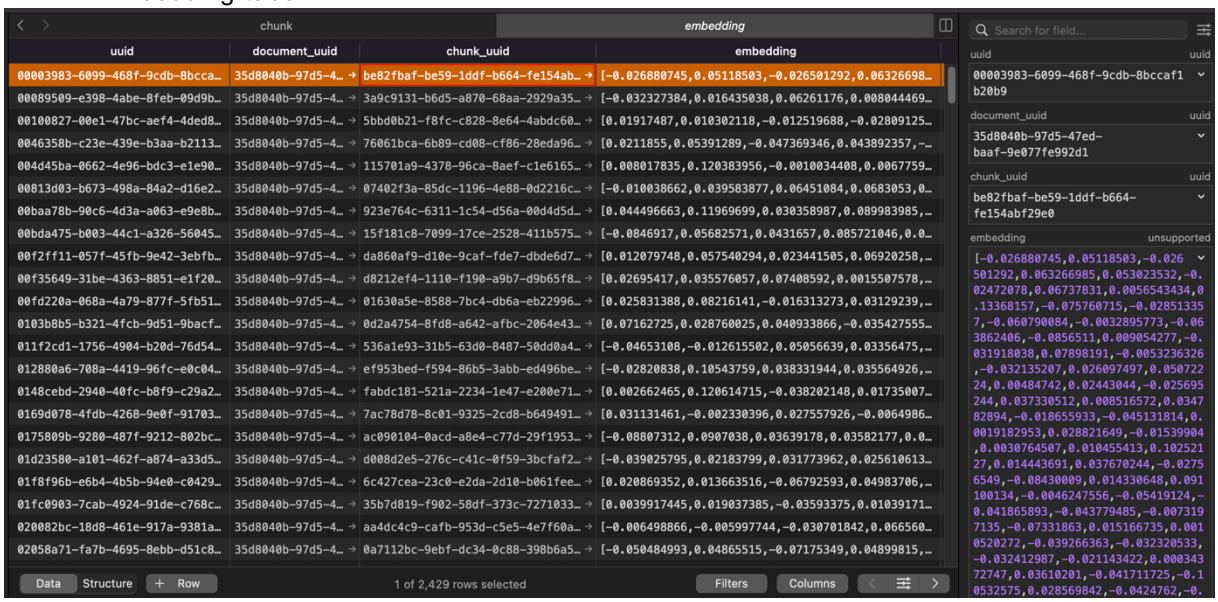
- Chunk-tabel



uuid	text	document_uuid
18d6aefe-4662-3d56-f5e5-66471b145394	the land of mordor where the shadows lie contents n…	35d8040b-97d5-47ed-baaf-9e077fe992d1
5c48587d-d659-b858-fbc0-06809751571a	fog on the barrowdowns 176 ix at the sign of the pra…	35d8040b-97d5-47ed-baaf-9e077fe992d1
a515ce14-c1db-2d60-31cc-2cf8802b5c06	of the fellowship 515 maps 533 works by jrr tolkien…	35d8040b-97d5-47ed-baaf-9e077fe992d1
8adb3bf0-36d4-12ef-e2e1-71821cd3a990	on 29 july 1954 an american edition followed on 21 o…	35d8040b-97d5-47ed-baaf-9e077fe992d1
5c18b2b4-9073-3674-945f-fd7ade1a4905	elfish further to farther nasturtians to nastur tiums…	35d8040b-97d5-47ed-baaf-9e077fe992d1
769c2000-49c8-e3a4-4751-15ee380c93f3	of the third volume which contained much hitherto un…	35d8040b-97d5-47ed-baaf-9e077fe992d1
34bbda91-c689-b9b5-beb5-71a45eed4f35	a promise he had made in the foreword to volume one…	35d8040b-97d5-47ed-baaf-9e077fe992d1
aa4dc4c9-cafb-953d-c5e5-4e7f60a0d957	an apology from the publisher for its absence for to…	35d8040b-97d5-47ed-baaf-9e077fe992d1
095a3025-c4be-b433-33fe-b0ba9676da1a	text remained virtually unchanged for a decade tolki…	35d8040b-97d5-47ed-baaf-9e077fe992d1
ee094e0d-70e1-546f-f72f-a6f165b6bd15	in context but which depart from tolkiens wording as…	35d8040b-97d5-47ed-baaf-9e077fe992d1
909a336c-073c-0b87-1add-452e2855237b	however were reproduced photographically from the ha…	35d8040b-97d5-47ed-baaf-9e077fe992d1
a8eda363-cd0a-1666-ae77-5b22d1543bc7	within the text itself tolkien replaced his original…	35d8040b-97d5-47ed-baaf-9e077fe992d1
15a24397-8c85-879a-5ce9-8b746e3f8e35	index with only names and page refer ences additiona…	35d8040b-97d5-47ed-baaf-9e077fe992d1
c2827078-b168-4ef9-37b3-d861b754ab5f	for the appendices including the now wellknown addit…	35d8040b-97d5-47ed-baaf-9e077fe992d1
d25d1804-54f5-96b9-6cc4-342201cf9483	british hardcover edition and for long remained anom…	35d8040b-97d5-47ed-baaf-9e077fe992d1
4b022090-5e32-7f4c-aa15-2e304ea061f4	volume hardcover second edition from allen unwin on…	35d8040b-97d5-47ed-baaf-9e077fe992d1
074e4d99-51d7-bb66-04a5-cfb293efc7c7	ballantine edition this did not include tolkiens sec…	35d8040b-97d5-47ed-baaf-9e077fe992d1
16500660-86f4-9170-9a77-eaefbb8d03de	whether any particular change in this edition is aut…	35d8040b-97d5-47ed-baaf-9e077fe992d1
ed71e034-0d56-53e7-767b-2e706e5863b5	title page none of the many reprintings is dated aft…	35d8040b-97d5-47ed-baaf-9e077fe992d1
d78ded10-0d70-b048-2d61-060032a67354	summer of 1966 further revising the text in june he…	35d8040b-97d5-47ed-baaf-9e077fe992d1
5c6deeb5-a887-b161-a925-681b80ec8614	himself made to the text during his lifetime they we…	35d8040b-97d5-47ed-baaf-9e077fe992d1
1e32b8e8-8ff7-6a75-9fa3-14991eaffb2e	and literary executor christopher tolkien sent a lar…	35d8040b-97d5-47ed-baaf-9e077fe992d1

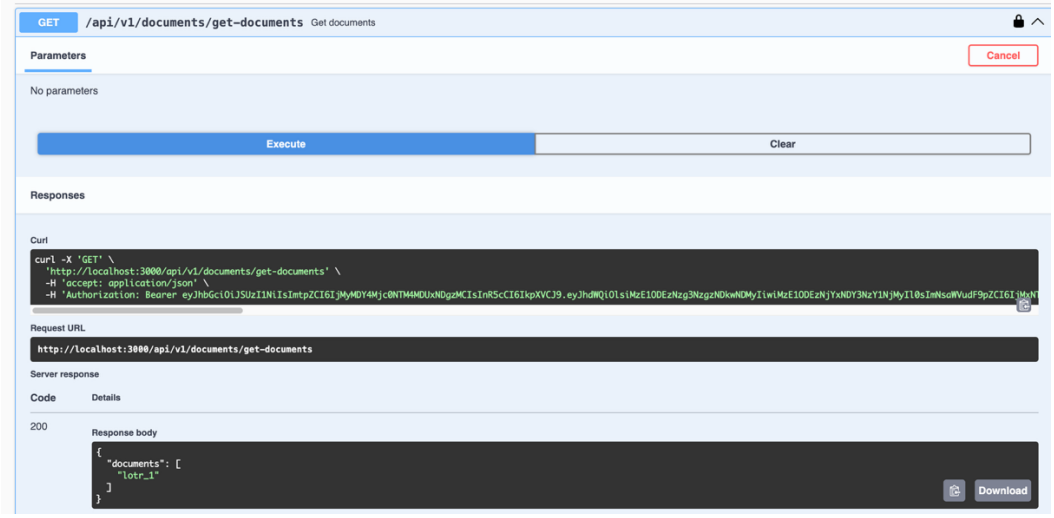
Data | Structure | + Row | 1-300 of 2,429 rows | Filters | Columns

- Embedding-tabel



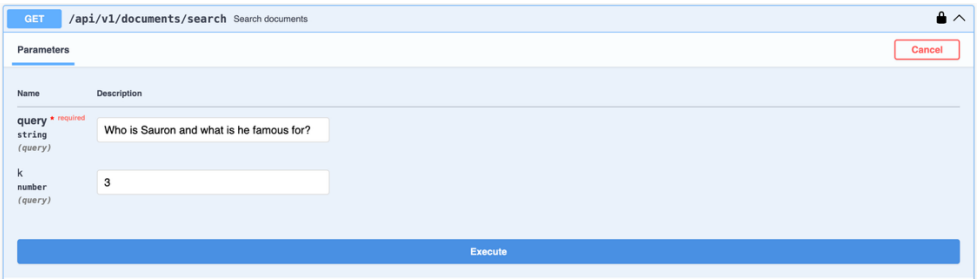
We hebben nu 1 boek omgezet naar 2429 vector afmetingen. De gebruiker kan kiezen om meer bestanden toe te voegen

Nu kan de gebruiker opvragen welke bestanden al zijn geupload dit is vooral naar de toekomst toe interessant om dit te verwerken in een frontend

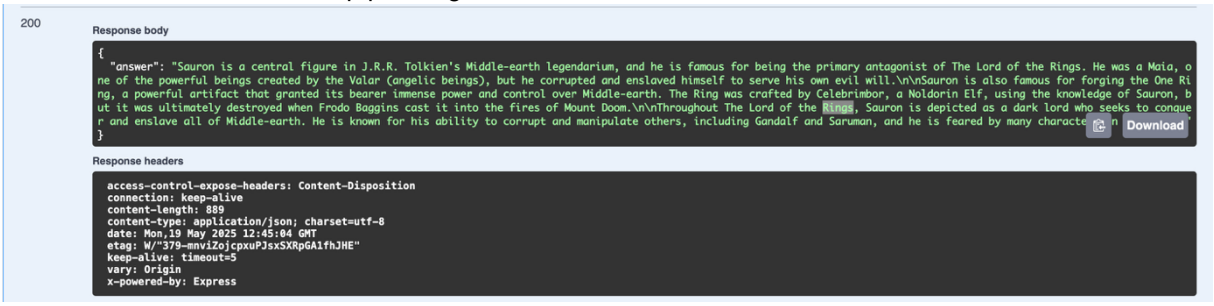


Nu komen we aan de magie van de RAG pipeline het stellen van vragen dit doen we door de api/v1/documents/search-query uit te voeren

In deze endpoint kunnen we de vraag stellen en kiezen hoeveel context documenten ons model mag gebruiken



Dit is het antwoord dat onze pipeline genereerde



Stel dat we vragen stellen met grof taalgebruik of ongepaste content zal onze pipeline hier nietn op antwoorden maar als we nu een vraag stellen over een topic dat niet voorkomt in de de opgeslage context zal het model antwoorden dat hij hier geen info over heeft en dus daarom niet kan antwoorden

3.4. Conclusie over de RAG pipeline en hoe deze waarde kan brengen in een business context.

In dit project is een fundamentele RAG-pipeline ontwikkeld die de potentie van deze technologie aantoont om de toegang tot informatie binnen een bedrijfscontext te transformeren. Door relevante contextuele data op te halen en te integreren met generatieve AI, biedt RAG een mechanisme om de beperkingen van Large Language Models (LLMs) te overwinnen, zoals hun gebrek aan actuele kennis en de neiging tot het produceren van onjuiste informatie. De proof-of-concept demonstreert hoe bedrijfsspecifieke documenten effectief kunnen worden doorzocht en gebruikt om de antwoorden van een AI-systeem te onderbouwen, waardoor de betrouwbaarheid en relevantie van de output aanzienlijk toenemen.
Echter, zoals in de realisatie is gebleken, is de geïmplementeerde RAG-pipeline nog niet zonder uitdagingen. Er zijn gevallen waarin de pipeline suboptimale antwoorden genereert, wat de noodzaak van verdere verfijning en robuustheid benadrukt. Om de volledige potentie van RAG te benutten en een succesvolle implementatie in een bedrijfsomgeving te waarborgen, is het cruciaal om deze uitdagingen aan te pakken en de architectuur van de pipeline verder te ontwikkelen.

3.4.1. Waarde van RAG in een business context

Desondanks de huidige beperkingen, biedt RAG aanzienlijke voordelen voor bedrijven. Het stelt organisaties in staat om hun enorme hoeveelheden interne data toegankelijk te maken voor AI-systemen, waardoor de volgende use-cases mogelijk worden:

- **Verbeterde klantenservice:** RAG kan worden gebruikt om chatbots en virtuele assistenten te voorzien van de meest recente productinformatie, handleidingen en FAQ's, wat resulteert in snellere en accuratere antwoorden op klantvragen.
- **Efficiënter kennismanagement:** Door medewerkers in staat te stellen om snel relevante informatie te vinden in interne documenten, zoals beleidslijnen, rapporten en trainingsmateriaal, kan RAG de productiviteit verhogen en de besluitvorming verbeteren.
- **Automatisering van complexe taken:** RAG kan worden ingezet om AI-systemen te voeden met de nodige context om complexe taken uit te voeren, zoals het samenvatten van lange documenten, het beantwoorden van juridische vragen of het genereren van gepersonaliseerde rapporten.

3.4.2. Naar een enterprise-ready RAG-architectuur

Om deze voordelen te realiseren, is het essentieel om een RAG-architectuur te implementeren die voldoet aan de eisen van een bedrijfsomgeving. Dit omvat, zoals beschreven in "Mastering RAG: How to Architect an Enterprise RAG System" (Galileo), de volgende aspecten:

- **Robuuste data-integratie:** Het vermogen om data uit verschillende bronnen te halen, te transformeren en te laden in een formaat dat geschikt is voor RAG is cruciaal.
- **Geavanceerde retrieval-strategieën:** Het implementeren van geavanceerde technieken, zoals chunking, metadata-filtering en query-optimalisatie, is noodzakelijk om de relevantie van de opgehaalde context te verbeteren.
- **Schaalbaarheid en prestaties:** De RAG-pipeline moet in staat zijn om grote hoeveelheden data en een toenemend aantal gebruikers te verwerken zonder prestatieverlies.
- **Beveiliging en toegangscontrole:** Er moeten mechanismen worden geïmplementeerd om gevoelige informatie te beschermen en ervoor te zorgen dat alleen geautoriseerde gebruikers toegang hebben tot specifieke data.
- **Monitoring en logging:** Het is essentieel om de prestaties van de RAG-pipeline continu te monitoren en gedetailleerde logs bij te houden om problemen te identificeren en de kwaliteit van de resultaten te verbeteren.

Door deze aspecten aan te pakken, kan de RAG-technologie worden getransformeerd van een veelbelovende proof-of-concept naar een betrouwbare en waardevolle tool die een aanzienlijke impact kan hebben op de bedrijfsresultaten.

4. Introductie e-mail classificatie & automatisatie

4.1. Stage opdracht 2.0 beschrijving/probleem

In de huidige digitale wereld worden organisaties dagelijks overspoeld met een enorme hoeveelheid e-mailberichten. Voor veel bedrijven is het beheren en verwerken van deze e-mailstroom een aanzienlijke uitdaging, die vaak resulteert in een hoge werkdruk voor werknemers en vertragingen in de communicatie. Dit project richt zich op de ontwikkeling van een e-mailclassificatie en automatisatiepipeline om deze uitdaging aan te gaan.

De context voor dit project is een klant die gemiddeld 1500 e-mails per dag ontvangt en een aanzienlijk aantal werknemers inzet om deze stroom handmatig te verwerken. Dit handmatige proces is tijdrovend, foutgevoelig en schaalt niet goed mee met de groei van het bedrijf. Er is dus een duidelijke behoefte aan een efficiëntere en geautomatiseerde oplossing.

Dit project onderzoekt de mogelijkheden van het toepassen van AI-technieken, in het bijzonder Natural Language Processing (NLP), om e-mails automatisch te classificeren en bepaalde acties te automatiseren. Het primaire doel is om de werklast van de werknemers te verminderen, de responstijd te verbeteren en de algehele efficiëntie van de e-mailafhandeling te verhogen.

Echter, de visie van dit project reikt verder dan de directe automatisering van e-mailafhandeling. In de toekomst kan de geclassificeerde e-maildata worden gebruikt om waardevolle informatie te extraheren. Deze geëxtraheerde data kan vervolgens dienen als input voor de automatisering van bredere bedrijfsprocessen. Denk bijvoorbeeld aan het automatisch verwerken van bestellingen uit e-mails, het genereren van rapporten op basis van klantvragen, of het initiëren van supporttickets op basis van probleemmeldingen. Deze vorm van data-extractie en procesautomatisering heeft het potentieel om de operationele efficiëntie van de klant verder te optimaliseren en nieuwe mogelijkheden te creëren.

4.2. Onderzoeksthema

"Hoe kan een e-mailclassificatie en automatisatiepipeline worden ontwikkeld en geïmplementeerd om de efficiëntie van de verwerking van een hoge e-mailstroom te verbeteren, de tijdsbesteding van werknemers te verminderen, en de basis te leggen voor toekomstige data-extractie en automatisering van bedrijfsprocessen?"

5. Uitvoering van werkzaamheden e-mail classificatie & automatisatie

5.1. Onderzoeksfase

In de onderzoeksfase van dit project lag de focus op het identificeren van effectieve methoden voor e-mailclassificatie, een cruciale stap in het automatiseren van de verwerking van grote volumes aan inkomende berichten. Aanvankelijk werd de mogelijkheid onderzocht om een specifiek model te trainen voor deze taak, voortbouwend op eerdere ervaringen met machine learning en het idee dat een fijn afgestemd model superieure prestaties zou leveren. Deze aanpak stuitte echter al snel op een aantal aanzienlijke uitdagingen die de haalbaarheid ervan in de gegeven context in twijfel trokken.

Een primair probleem was het **gebrek aan voldoende trainingsdata**. Voor de ontwikkeling van een robuust en nauwkeurig model is een grote hoeveelheid gelabelde data vereist. De klant was echter niet in staat om tijdig de benodigde hoeveelheid e-mails aan te leveren. Dit is een veelvoorkomend probleem bij de implementatie van AI-oplossingen in bedrijven, waar data vaak verspreid is over verschillende systemen en de beschikbaarheid ervan beperkt kan zijn. Om de ontwikkeling niet te vertragen en toch een bruikbare dataset te hebben, werd overgegaan op het handmatig opstellen van testmails. Dit bleek echter een zeer tijdrovende bezigheid, die qua arbeidsintensiteit vergelijkbaar was met de handmatige verwerking van e-mails die de automatisering juist zou moeten vervangen. Bovendien was de representativiteit van deze handmatig opgestelde e-mails voor de diversiteit van echte klantcommunicatie twijfelachtig.

Bovendien bleek dat het model, hoewel het tijdens de trainingsfase veelbelovende resultaten liet zien op de beperkte, zelf gegenereerde dataset, **slecht presteerde in de testfase**. Vooral bij *edge cases*, zoals ongebruikelijke formuleringen, afwijkende verzoeken of berichten die meerdere intenties bevatten, en met een toenemend aantal classificatiecategorieën nam de nauwkeurigheid van het model sterk af. Dit fenomeen, bekend als *overfitting*, duidde op een aanzienlijk risico op onbetrouwbare classificaties in een real-world scenario, waar de variatie in e-mails veel groter is dan in een gecontroleerde trainingsomgeving. De noodzaak om het model voortdurend te her trainen en aan te passen aan nieuwe data werd duidelijk, wat de onderhoudskosten en complexiteit van de oplossing aanzienlijk zou verhogen.

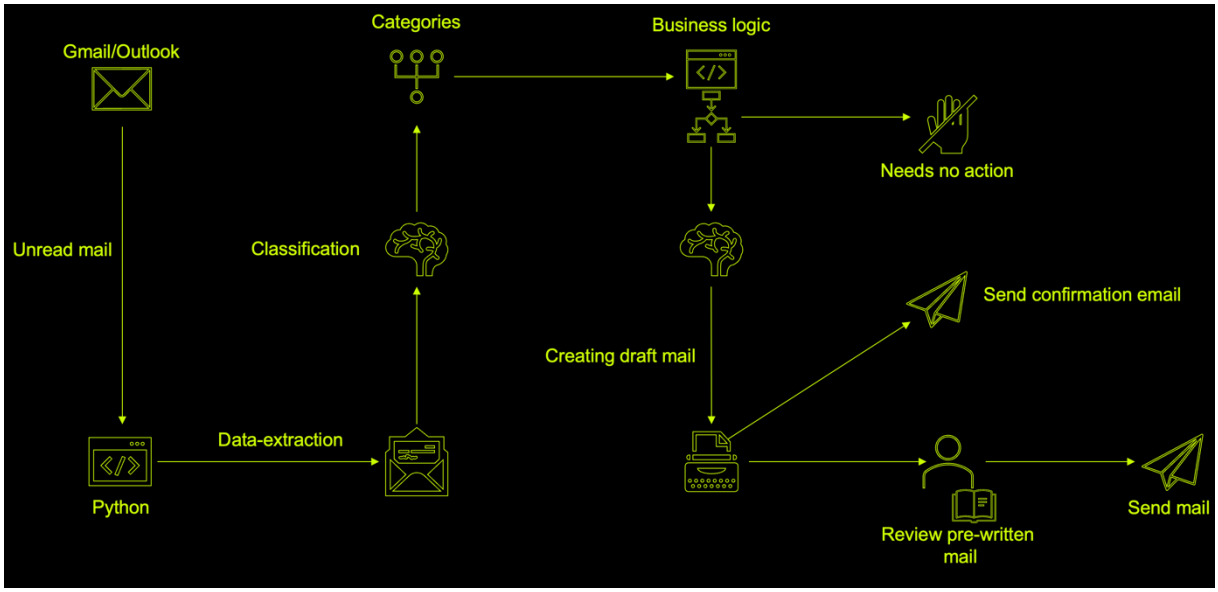
Deze ervaringen hebben geleid tot een heroverweging van de gekozen aanpak. Er werd gezocht naar een meer flexibele en robuuste oplossing die minder afhankelijk is van grote hoeveelheden klantspecifieke trainingsdata. In plaats van een specifiek LLM-model te trainen, werd besloten om een generiek LLM-model in te zetten voor de classificatie van e-mails. Deze aanpak maakt gebruik van de reeds uitgebreide kennis en het generalisatievermogen van vooraf getrainde modellen, die zijn getraind op enorme hoeveelheden tekstdata. Het inzetten van een generiek LLM voor e-mailclassificatie werd tevens ingegeven door de observatie dat een LLM, zoals Gemini, in staat is om de belangrijkste context uit een e-mail te halen, er een passende categorie aan toe te kennen en zelfs antwoorden te formuleren. Deze veelzijdigheid van LLM's maakt het mogelijk om met één model zowel de classificatie als de antwoordgeneratie te verzorgen, wat resulteert in een efficiëntere en potentieel krachtigere oplossing. Deze aanpak leverde **onmiddellijk betere resultaten** op, wat de potentie van LLM's voor e-mailclassificatie in complexe, real-world scenario's onderstreepte. Het elimineerde de noodzaak voor uitgebreide dataverzameling en training, en bleek beter in staat om te gaan met de diversiteit en complexiteit van echte e-mailcommunicatie.

5.2. Ontwikkelingsfase

De ontwikkelingsfase kende verschillende stappen, ondanks de relatief korte tijd die ervoor beschikbaar was. We zijn gestart met het schrijven van ondersteunende functies die nodig zijn voor de classificatiemethoden. Vervolgens hebben we twee services opgezet, zodat een gebruiker zowel e-mails uit een **Outlook-inbox** als uit een **Gmail-inbox** kan ophalen, laten classificeren en verwerken.

Een belangrijk onderdeel van deze fase was het experimenteren met verschillende classificatiebenaderingen. Hieronder bespreken we drie methoden die elk gebruikmaken van een ander type Large Language Model (LLM): **Ollama**, **Gemini** en **OpenAI**.

De flow van het project gaat er zo uitzien



5.2.1. Ondersteundende functie voor classificatie

Vooraleer we de effectieve classificatiefuncties (Ollama, Gemini, OpenAI) konden ontwikkelen, hebben we eerst een reeks **ondersteunende datamodellen en structuren** opgezet. Deze vormen de fundering van het classificatiesysteem en zorgen voor een gestandaardiseerde manier om AI-uitvoer te interpreteren en verwerken.

- **Structuur en validatie** van de AI-output,
- **Uniformiteit** tussen verschillende modellen (Ollama, Gemini, OpenAI),
- **Foutafhandeling en betrouwbaarheid** bij onvolledige of onjuiste data.

5.2.1.1. Enum EmailCategory

De EmailCategory Enum definieert de mogelijke categorieën waarin een e-mail kan worden ingedeeld. Dit willen we doen om ervoor te zorgen dat een LLM model zelf geen categorieën gaat verzinnen.

- **Order –** bestellingen van diensten
- **Klacht –** melding van problemen of ontevredenheid
- **Vraag –** algemene of informatieve vraag
- **Reclame –** reclame of spam
- **Andere –** overige e-mails als model onzeker is
- **Onbekend –** fallback wanneer de classificatie mislukt

5.2.1.2. Hoofdmodel: EmailClassification

Het EmailClassification-model is de centrale datastructuur waarin het resultaat van de AI-classificatie wordt opgeslagen. Het bevat volgende velden:

- category: De toegekende e-mailcategorie (type: EmailCategory)
- confidence: Een score tussen 0 en 1 die aangeeft hoe zeker het model is van zijn classificatie
- key_information: Een lijst met kernpunten uit de e-mail (bijvoorbeeld: datum, klantnaam, adres)
- suggested_action: Een korte beschrijving van de aanbevolen vervolgstap (bijvoorbeeld: “doorgeven aan planning”)
- order_details: Een optioneel object dat bijkomende structuur bevat als het om een bestelling (order) gaat

5.2.1.3. SubModellen voor orders

Orders zijn complexer en vereisen bijkomende informatie. Daarom zijn er drie gescheiden submodellen gemaakt.

- **OrderDetails**
  Bevat algemene informatie over de bestelling, zoals afzender, werfadres, klantnaam en type order. Afhankelijk van het ordertype wordt een van de drie onderstaande modellen gebruikt:
- **LeveringDetails**
  Beschrijft details van een levering, zoals het type afval, containergrootte en eventuele extra opties (zoals hijsogen of afsluitdeksel).
- **OphalingDetails**
  Bevat enkel het containernummer dat opgehaald moet worden.
- **WisselDetails**
  Combineert aspecten van ophaling en levering: bevat het containernummer van de huidige container, en de eigenschappen van de nieuwe container (afvaltype, grootte, extra opties).

Alle submodellen zijn optioneel, zodat e-mails met onvolledige informatie toch verwerkt kunnen worden. De validatie gebeurt met behulp van **Pydantic**, wat garandeert dat alleen correcte types en structuren worden aanvaard. Dit beperkt het risico op fouten wanneer de AI-output naar Python-objecten wordt omgezet.

5.2.2. Classificatie functies

In deze fase ontwikkelden we drie verschillende methodes voor het classificeren van e-mails en het genereren van conceptantwoorden. Elke methode maakt gebruik van een ander Large Language Model (LLM), elk met eigen sterktes en beperkingen. We vergelijken hieronder de werking en de voor- en nadelen van de gebruikte modellen.

5.2.2.1. Ollama

Ollama is een platform waarmee LLM's lokaal op een computer kunnen worden gedraaid. Modellen worden als een soort image gepulled van het Ollama-platform en vervolgens lokaal uitgevoerd. In onze toepassing gebruikten we het **Gemma 4B model**, een relatief lichtgewicht model geschikt voor inferentie op ee gewone pc of laptop.

Voordelen

- **Offline beschikbaar**: omdat het model lokaal draait, is er geen internetverbinding nodig na de initiële installatie.
- **Controle en privacy**: Alle data blijft lokaal, wat voordelig is in het kader van GDPR en privacygevoeligheid.
- **Flexibiliteit**: Ollama ondersteunt meerdere modellen, zoals Gemma, LLaMA, Mistral en Vicuna.
  Nadelen
- **Langzamer dan cloud-gebaseerde modellen**: De inferentie duurt gemiddeld enkele seconden langer per e-mail.
- **Hoge geheugendruk**: De modellen nemen veel RAM en GPU-resources in beslag. Niet de snelste toepassing
- **Minder accuraat bij complexe prompts**: Door de beperkte schaal van lokale modellen zijn ze gevoeliger voor hallucinerende of inconsistente antwoorden.

5.2.2.2. Geminin

Gemini is de AI-service van Google, waarmee krachtige LLM-modellen via een cloudgebaseerde API kunnen worden aangesproken. Tijdens het project maakten we gebruik van het model *gemini-1.5-flash-001* via de *google.generativeai*-bibliotheek. Dit model werd ingezet voor zowel classificatie als het genereren van conceptantwoorden op inkomende e-mails.
De functie *classify_email_gemini(email_text: str)* verstuurt een prompt naar het Gemini-model waarin wordt gevraagd om een e-mail te classificeren op basis van vijf mogelijke categorieën: **order**, **klacht**, **vraag**, **reclame** of **anderen**. Daarnaast moet het model bijkomende informatie geven zoals:

- een **vertrouwensscore** (confidence),
- een lijst van **key_information** (belangrijke kernpunten),
- een **suggested_action** (voorstel voor automatische afhandeling),
- en indien van toepassing: **order_details** met nested informatie over leveringen, ophalingen of wissels.

De gegenereerde JSON-output wordt via regex uit de ruwe response gehaald, geparsed en gevalideerd met behulp van Pydantic. Als de JSON niet correct is opgebouwd of als er een validatiefout optreedt, wordt dit opgevangen en krijgt de gebruiker een fallbackresultaat met een lage confidence-score en een instructie voor manuele controle.

Voordelen van Gemini:

- **Snel** in inferentie, zeker vergeleken met lokale modellen.
- **Accuraat** in het detecteren van semantische patronen in e-mails.
- **Flexibel** in de promptstructuur en foutafhandeling.

Nadelen:

- **Afhankelijk van internetverbinding en Google API's**.
- **Beperkte dataprivacy** in vergelijking met lokale oplossingen.

5.2.2.3. OpenAI

De derde methode die werd onderzocht is gebaseerd op **OpenAI's GPT-4.1-nano** model. Dit model werd aangesproken via de openai-bibliotheek en is bekend om zijn taalvaardigheid, betrouwbaarheid en nauwkeurigheid in real-world NLP-taken.

In de functie classify_email_openai(email_text: str) wordt, net als bij Gemini, een prompt samengesteld waarin het model gevraagd wordt om een e-mail te analyseren en exact dezelfde gestructureerde JSON-output te genereren. OpenAI ondersteunt ook instructies zoals response_format=json_object, waarmee de kans vergroot wordt dat het model correcte, parseerbare JSON terugstuurt.

Net als bij de andere methoden worden de resultaten gevalideerd via Pydantic. Indien het model foutieve JSON retourneert of een validatie-uitzondering wordt opgegooid, wordt de e-mail met lage confidence als "onbekend" geclassificeerd en wordt een fallbackactie voorgesteld.

Voordelen van OpenAI:

- **Zeer hoge classificatienauwkeurigheid**.
- **Sterk generalisatievermogen** in complexe en ongebruikelijke e-mails.
- **Gebruiksvriendelijke API** met veel documentatie.

Nadelen van OpenAI:

- **Hogere kosten** in vergelijking met Gemini of lokale alternatieven.
- **Afhankelijk van een stabiele internetverbinding**.
- **Privacyrisico's** bij gevoelige e-maildata.

5.2.2.4. Classificatie output

Voorbeeld e-mail die we gebruiken we zien dat het om een bestelling gaat in onderstaande mail ze willen een container bestellen van 12m3 voor bouwafval op het adres Modenstraat 15 in Hasselt en de contact persoon is Jan De Vries (dit is een AI gegenereerde e-mail)

```
example_email = """
Betreft: Bestelling container

Graag een 12m3 container voor bouwafval leveren op de werf te Molenstraat 15, 3500 Hasselt.
Gewenste leverdatum is 22-05-2025 tussen 7 en 10u.
De contactpersoon is Jan De Vries. Mijn e-mail is jan.devries@example.com.
"""

classification_result = classify_email_openai(example_email)
print(classification_result.model_dump_json(indent=2))
```

Output van de voorbeeld email in de backend: we zien dat het model voor 95% zeker is dat het gaat om een order en heeft een aantal key_information uit de mail gehaald en doet een suggested action. De info uit de order_details kan in de toekomst gebruikt worden om bepaalde processen nog verder te automatiseren zoals het vast leggen van de container bestelling in een ERP systeem, inplannen van levering, …

Preformance test:

- Ollama: 11.4s
- Gemini: 2.4s → MAX 15 LLM calls per minuut
- OpenAI: 3.9s → business account


```
{
  "category": "order",
  "confidence": 0.95,
  "key_information": [
    "12m3 container voor bouwafval",
    "werf te Molenstraat 15, 3500 Hasselt",
    "leverdatum 22-05-2025",
    "tussen 7 en 10u",
    "Jan De Vries"
  ],
  "suggested_action": "Bevestig de order en plan de levering",
  "order_details": {
    "afzender_mailadres": "jan.devries@example.com",
    "werf_adres": "Molenstraat 15, 3500 Hasselt",
    "klant_naam": "Jan De Vries",
    "order_datum": "22-05-2025",
    "gewenst_uur_levering": "07:00-10:00",
    "type_order": "Levering",
    "levering_details": {
      "afvaltype": "bouwafval",
      "containergrootte": "12m3",
      "extra_opties": []
    },
    "ophaling_details": {
      "containernummer": ""
    },
    "wissel_details": {
      "containernummer_ophaling": "",
      "afvaltype_nieuw": "",
      "containergrootte_nieuw": "",
      "extra_opties_nieuw": []
    }
  }
}
```

5.2.3. Acties na classificatie

Zodra een e-mail succesvol geclassifiseerd is door een van de LLM-modellen (Ollama, Gemini of OpenAI), worden er bijkomende acties automatisch uitgevoerd. Deze vervolgstappen zijn erop gericht om de efficiëntie van het e-mailverwerkingsproces te verhogen door:

- Het automatisch genereren van een conceptantwoord
- Het versturen van ontvangsbevestiging (indien van toepassing)

Deze automatisering ontlast werknemers van repetitieve taken zoals het opstellen van standaardantwoorden en het bevestigen van ontvangst, en laat hen toe zich te focussen op het controleren en aanpassen van antwoorden waar nodig.

5.2.3.1. Automatisch genereren van conceptantwoorden

Met behulp van de functies create_draft_answer() (voor Gemini) en create_draft_answer_openai() (voor OpenAI) wordt er een gepersonaliseerd antwoord gegenereerd op basis van:

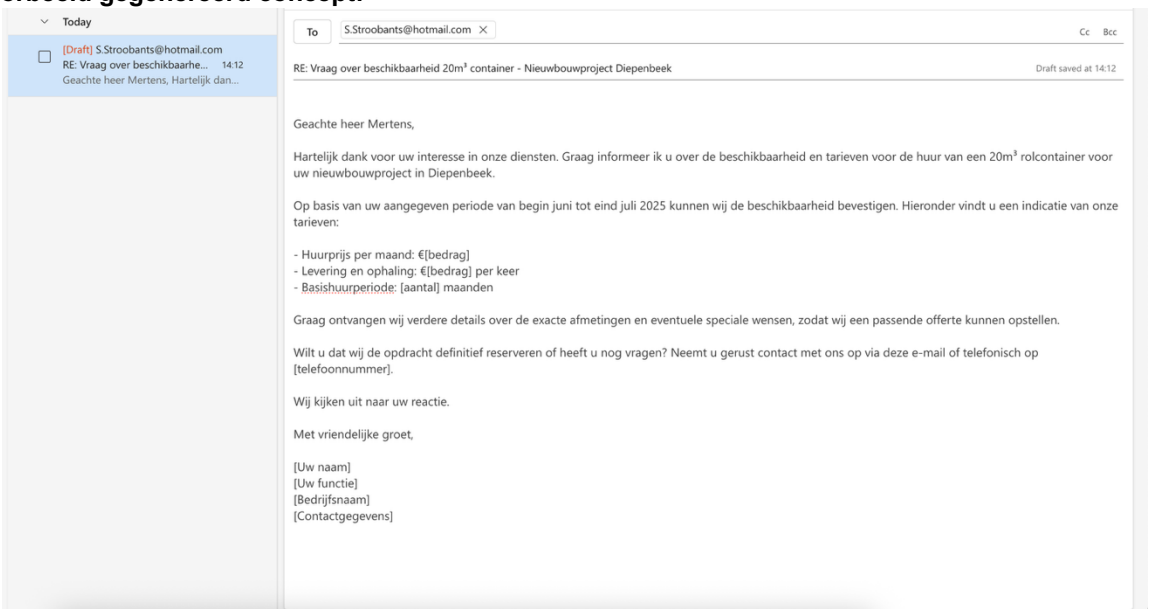
- de classificatiecategorie (order, klacht, vraag, …),
- de kerninformatie die uit de e-mail werd gehaald,
- de voorgestelde actie door het LLM-model,
- en – indien beschikbaar – de gestructureerde order_details.

De gegnereerde antwoorden zijn volledig, contextueel relevant en bevatten waar nodig ook de ordergegevens. Deze tekst wordt vervolgens als concept opgeslagen, zodat een werknemer enkel nog moet nalezen, licht aanpassen indien nodig, en vervolgens versturen. Dit versnelt de afhandeling van e-mails aanzienlijk.

Voorbeeld workflow:

1. E-mail wordt geclassificeerd als order.
2. Orderdetails worden herkend en uitgelicht (zoals containergrootte, leveringsdatum, werfadres).
3. Het model genereert een gepast antwoord met bevestiging en verwachte leverinfo.
4. Antwoord verschijnt als concept in de applicatie.

Voorbeeld gegenereerd concept:



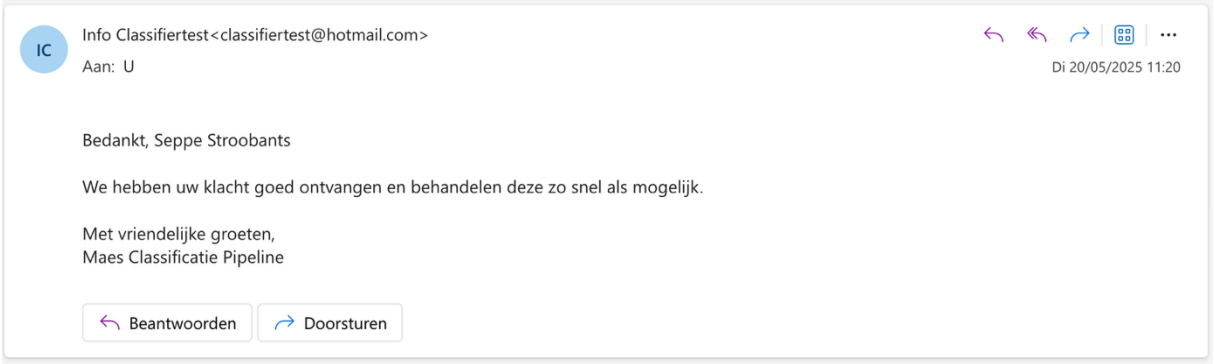
5.2.3.2. Ontvangsbevestiging versturen

Bij e-mails die geclassificeerd zijn als **order**, **klacht** of **vraag**, wordt automatisch een korte en professionele ontvangstbevestiging gegenereerd via de functie genereer_ontvangstbevestiging(). Deze bevestiging:

- stelt de afzender gerust dat zijn e-mail goed ontvangen is,
- bevestigt het type verzoek (vb. "We hebben uw klacht goed ontvangen"),
- en biedt een duidelijke, klantgerichte communicatie zonder manuele tussenkomst.

E-mails die onder de categorieën reclame of anderen vallen, worden bewust niet automatisch beantwoord, om ongewenste interactie of ruis te vermijden.

Voorbeeld gegenereerd bericht:



Deze aanpak zorgt ervoor dat inkomende e-mails niet enkel snel herkend worden, maar ook onmiddellijk gepaste opvolging krijgen. De combinatie van classificatie, automatische feedback en een voorgesteld antwoord maakt het hele verwerkingsproces sneller, consistenter en minder foutgevoelig.

5.2.4. Koppelen van inboxservices

Om inkomende e-mails automatisch te kunnen verwerken, werd een koppeling opgezet met de inbox van de gebruiker. In het project werden twee services ontwikkeld: één voor **Gmail** en één voor **Outlook**. Deze services zorgen ervoor dat ongelezen e-mails worden opgehaald, geanalyseerd, geclassificeerd en automatisch verwerkt.

5.2.4.1. Gmail

De Gmail-service is opgezet met behulp van de officiële **Gmail API**, via de bibliotheken google-api-python-client en google-auth.

WERKING VAN DE SERVICE

De belangrijkste functies van deze service zijn:

- ***get_gmail_service():*** Maakt verbinding met de Gmail API via OAuth 2.0 en bouwt een service-object op basis van lokale credentials (token.pickle).
- ***get_unread_emails_gmail():*** Vraagt alle ongelezen e-mails op uit de inbox.
- ***get_email_content_gmail():*** Haalt het onderwerp, afzender en de tekstuele inhoud van een specifieke e-mail op.
- ***classify_email_gekozenModule():*** Classificeert de e-mailinhoud
- create_draft_answer_openai(): Genereert een conceptantwoord op basis van de classificatie.
- ***save_email_as_draft_gmail():*** Slaat het gegenereerde antwoord op als concept in de Gmail-inbox.
- ***genereer_ontvangstbevestiging():*** Maakt een ontvangstbericht aan op basis van de e-mailcategorie.

- **send_confirmation_email_gmail():** Stuurt de ontvangstbevestiging terug naar de afzender.
- **add_label_gmail():** Labelt de e-mail in Gmail volgens de toegekende categorie (bv. "Order", "Klacht", ...).
- **mark_as_read_gmail():** Markeert de e-mails als gelezen.

De hoofdlogica gebeurt binnen de functie process_emails_gmail(), die alle bovenstaande stappen combineert. Dit betekent dat een werknemer met één druk op de knop zijn of haar inbox kan laten analyseren en structureren.

CONFIGURATIE IN GOOGLE CLOUD CONSOLE

Voor de werking van de Gmail API en integratie met Gemini (Google AI), werd een project opgezet in de **Google Cloud Console**. De configuratiestappen zijn als volgt uitgevoerd:

1. Ga naar console.cloud.google.com.
2. Maak een nieuw project aan.
3. Selecteer het project.
4. Navigeer naar **APIs & Services**.
    - Klik op **Enable APIs** en activeer:
        - Gmail API
        - Generative Language API (voor Gemini-functionaliteit)
5. Ga naar het tabblad **Credentials**:
    - Kies **Create credentials** > OAuth client ID
    - Download het bestand client_secret.json en plaats het lokaal in het project
6. Open het tabblad **OAuth consent screen**:
    - Voeg alle nodige testgebruikers toe onder de **Test users**-sectie

Bij de eerste keer uitvoeren wordt een lokaal OAuth-proces opgestart, waarin de gebruiker handmatig toegang geeft tot zijn Gmail-account. De gegenereerde token.pickle wordt vervolgens hergebruikt voor volgende sessies.

5.2.4.2. Outlook

Naast Gmail werd ook ondersteuning voorzien voor Microsoft Outlook. Via de **Microsoft Graph API** worden ongelezen e-mails uit de inbox opgehaald en verwerkt volgens dezelfde logica als bij Gmail: classificatie, conceptantwoord, ontvangstbevestiging, labelen en markeren als gelezen.

WERKING VAN DE SERVICE

De Outlook-service werd geïmplementeerd met behulp van de pakketten msal (Microsoft Authentication Library) en httpx voor HTTP-verzoeken naar de Graph API. De service omvat onder andere:

- ***get_access_token_outlook():*** Haalt een access token op via OAuth 2.0 met client credentials of refresh token. Dit token wordt lokaal opgeslagen in refresh_token.txt voor hergebruik.
- ***get_unread_emails_outlook():*** Haalt ongelezen e-mails op via Microsoft Graph. De e-mails worden gestructureerd als dictionaries met velden zoals onderwerp, afzender, inhoud, enz.
- ***classify_email_gekozenModule():*** Voert classificatie uit (zoals beschreven in hoofdstuk 4.2).
- ***create_draft_answer_openai():*** Genereert een gepersonaliseerd conceptantwoord op basis van de classificatie.
- ***save_email_as_draft_outlook():*** Slaat het conceptantwoord op in de Outlook-inbox.
- ***send_confirmation_email_outlook():*** Stuurt een ontvangstbevestiging naar de afzender.
- ***add_category_to_email():*** Labelt de e-mail met de juiste categorie.
- ***mark_as_read_outlook():*** Markeert de e-mail als gelezen.

Deze stappen worden gecombineerd in de functie process_emails_outlook(), waarmee het volledige verwerkingsproces voor inkomende Outlook-mails automatisch verloopt.

Voorbeeldflow:

1. Ongelezen e-mails worden opgehaald.
2. Elke e-mail wordt geanalyseerd en geclassificeerd (bv. als ‘klacht’ of ‘order’).
3. Er wordt een gepast conceptantwoord gegenereerd en opgeslagen.
4. Een ontvangstbevestiging wordt automatisch verstuurd.
5. De e-mail wordt gelabeld en gemarkeerd als gelezen.

CONFIGURATIE IN AZURE PORTAL

Om toegang te krijgen tot de Outlook-inbox via Microsoft Graph, werd een applicatie geregistreerd in de **Azure Active Directory**. Hieronder volgt een overzicht van de stappen om de setup correct uit te voeren:

1. Ga naar portal.azure.com.
2. Navigeer naar **App registrations**.
3. Kies **New registration**:
    - Geef de applicatie een naam.
    - Selecteer het geschikte accounttype (bv. "Accounts in any organizational directory and personal Microsoft accounts").
    - Voeg een **Redirect URI** toe indien je een interactieve login gebruikt.
4. Ga naar **Certificates & secrets**:
    - Maak een nieuwe **Client Secret** aan.
    - Kopieer de waarde en sla die op in het .env-bestand.
5. Maak een .env-bestand aan in je project met de volgende inhoud:

```dotenv
APPLICATION_ID='<< jouw Application (client) ID >>'
CLIENT_SECRET='<< jouw Client Secret Value >>'
```

6. Ga naar **API Permissions**:
    - Voeg de nodige permissies toe, zoals:
        - Mail.ReadWrite
        - Mail.Send
        - User.Read
    - Sta deze permissies toe voor je organisatie of tenant.

Na deze eenmalige configuratie kan de service via een lokaal script of applicatie ongelezen Outlook-mails ophalen en verwerken, volledig automatisch.

5.2.5. Toekomst van dit project

Hoewel deze applicatie momenteel als een externe service werkt op basis van e-mailinboxen (Gmail en Outlook), ligt de echte meerwaarde in een volledige integratie binnen het bredere bedrijfsproces. In de toekomst willen we deze oplossing verder ontwikkelen tot een volwaardig onderdeel van het bestaande **ERP-systeem** van het bedrijf.

5.2.5.1. ERP-integratie

De huidige implementatie werkt als een tussenlaag: e-mails worden opgehaald, geanalyseerd, geclassificeerd, en eventueel voorbereid voor opvolging. Maar idealiter verloopt dit proces volledig **binnen één centraal systeem** – het ERP-platform. Daardoor zouden we het gebruik van losse inboxinterfaces kunnen vermijden en rechtstreeks werken met data binnen de bedrijfsapplicatie.

In een geïntegreerd scenario zouden e-mails automatisch binnenkomen via een centraal communicatiemodule in het ERP-systeem. De classificatie- en extractielogica die nu is uitgewerkt, zou dan direct worden toegepast op deze berichten, zonder tussenkomst van externe e-mailproviders of manuele synchronisatie.

5.2.5.2. Volledige procesautomatisatie

Op termijn willen we verder evolueren naar een **volledig geautomatiseerde orderflow**:

- E-mails die alle vereiste informatie bevatten (zoals klantnaam, werfadres, containertype, leverdatum...) worden automatisch verwerkt.
- Zodra een e-mail als order wordt herkend én alle velden zijn correct ingevuld, wordt er automatisch een bestelling aangemaakt in het ERP-systeem.
- Voor klachten of vragen kunnen er tickets worden aangemaakt binnen een CRM- of supportmodule.
- Interne meldingen of taken kunnen via workflowautomatisering worden toegekend aan de juiste afdelingen (vb. planning of logistiek).

Door deze processen te automatiseren, verwachten we:

- **Tijdswinst** bij het verwerken van inkomende communicatie.
- **Betere opvolging** van klanten dankzij snellere antwoorden en consistente communicatie.
- **Betere gegevensbewaking**, doordat informatie direct van e-mail naar gestructureerde systemen vloeit zonder kopiëren/plakken of interpretatiefouten.

5.2.5.3. AI als kerncomponent

De AI-component zal centraal blijven staan in de toekomst van dit project. De classificatiemodellen en extractielogica kunnen verder verfijnd worden, bijvoorbeeld via:

- **Trainen op bedrijfsdata**, voor betere accuraatheid.
- **Feedbackloops** vanuit medewerkers om fouten te corrigeren.
- **Slimme controlemechanismen**, die enkel handmatige tussenkomst vragen bij twijfelgevallen.

Deze toekomstvisie toont aan dat het project meer is dan een e-mailtool. Het is een fundament voor **intelligente procesautomatisatie**, waarbij AI een brug vormt tussen ongestructureerde communicatie (zoals e-mails) en gestructureerde systemen (zoals ERP, CRM of planningstools).

Laat me weten of je dit ook visueel of puntsgewijs verwerkt wil zien in je verslag, of of je dit wilt inkorten/verwerken als samenvattend besluit.

5.3. Conclusie e-mail classificatie en automatisatie

Dit project vormde een belangrijke eerste stap in de toepassing van artificiële intelligentie op een klassiek maar tijdrovend bedrijfsproces: de verwerking van inkomende e-mails. Door gebruik te maken van bestaande Large Language Models (LLM's) zoals Gemini en OpenAI, werd een schaalbare en flexibele oplossing ontwikkeld die toelaat om automatisch e-mails te classificeren, te interpreteren en er actiegerichte conceptantwoorden voor te genereren.

Wat begon als een proof-of-concept rond e-mailclassificatie, groeide uit tot een volwaardig systeem dat in staat is om binnenkomende communicatie te structureren, te analyseren en gepaste opvolging te voorzien. De combinatie van datavalidatie, modelgestuurde extractie en directe koppelingen met Gmail en Outlook toont aan dat zelfs complexe processen stapsgewijs geautomatiseerd kunnen worden met behulp van AI. Bovendien is het systeem ontworpen met het oog op uitbreidbaarheid, zodat toekomstige integratie met ERP-systemen en andere interne tools vlot mogelijk is.

Wat dit project écht onderscheidt, is de **visie op intelligente automatisatie**. In plaats van louter manuele taken te versnellen, werd er bewust gekozen om AI in te zetten als **kenniscomponent**: het model begrijpt de context, herkent intenties, en weet welke informatie relevant is voor verdere verwerking. Door de koppeling met ontvangstbevestiging en conceptantwoorden verschuift het werk van werknemers van uitvoerend naar controlerend, wat leidt tot meer tijd voor kwalitatieve opvolging.

De toekomst van dit project ligt dan ook in de volledige integratie met de interne processen van het bedrijf. Denk aan een systeem waarin een e-mail met een duidelijke bestelling automatisch een order aanmaakt in het ERP-systeem, zonder tussenkomst van een medewerker. Of waarbij een klacht automatisch een supportticket aanmaakt, inclusief de belangrijkste details. Op die manier wordt AI een natuurlijke partner in het bedrijfsproces, die de brug slaat tussen **ongestructureerde input** (e-mails, tekst, taal) en **gestructureerde actie** (ERP, CRM, planning).

Kortom: dit project toont aan hoe AI vandaag al kan bijdragen aan echte, tastbare bedrijfsoptimalisatie. Het vormt een **fundamentele stap in de richting van intelligente, zelfsturende processen**, en legt een robuuste basis waarop toekomstige automatisaties gebouwd kunnen worden – met efficiëntie, schaalbaarheid en inzicht als kernwaarden.

6. Stage conclusie

Mijn stageperiode bij Wisemen was een bijzonder leerrijke en verrijkende ervaring. Als student Toegepaste Informatica met een specialisatie in Application Development, kreeg ik een unieke kans om me te verdiepen in de wereld van artificiële intelligentie en haar praktische toepassingen binnen een bedrijfscontext.

Wat begon met beperkte voorkennis rond AI, groeide uit tot een diepgaande exploratie van geavanceerde technologieën zoals Retrieval-Augmented Generation (RAG) en e-mailclassificatie via Large Language Models (LLM's). Door de ontwikkeling van een eigen RAG-pipeline en een e-mailautomatisatiesysteem, leerde ik niet alleen de technische aspecten van AI, maar ook hoe deze technologieën strategische meerwaarde kunnen bieden voor organisaties.

Tijdens deze stage heb ik mijn vaardigheden in backendontwikkeling sterk kunnen uitbreiden, zowel in Python als NestJS. Daarnaast leerde ik werken met moderne tools zoals vector databases, S3-integraties, OAuth-authenticatie, en API-architecturen. Ook het belang van databeveiliging, schaalbaarheid en contentmoderatie kwam uitgebreid aan bod.

Naast technische kennis, ontwikkelde ik ook kritisch denkvermogen en probleemoplossend inzicht, door uitdagingen zoals modelkeuze, performantie, en datakwaliteit te analyseren en aan te pakken. Het zelfstandig uitwerken van een proof-of-technology en het bijdragen aan een bredere visie op intelligente automatisatie, gaf me het vertrouwen dat ik klaar ben om binnen het werkveld innovatieve oplossingen te helpen realiseren.

Kortom: deze stage heeft mijn kijk op softwareontwikkeling verruimd en me laten zien hoe krachtig en relevant AI vandaag al is binnen uiteenlopende bedrijfsdomeinen. Ik kijk met trots terug op het traject dat ik heb afgelegd en neem deze ervaring mee als stevige basis voor mijn verdere carrière.

CONTACT

Seppe Stroobants | Student Toegepaste Informatica
R0955288@student.thomasmore.be

VOLG ONS

www.thomasmore.be
fb.com/ThomasMoreBE
#WeAreMore