Yannick Lansink, 900102137

Beroepsproduct

Software Developer, Hogeschool NOVI

18 Augustus, 2024

Afkortingenlijst

Afkortingen	**Definitie**
IV	Informatievoorziening
AVG	Algemene Verordening Gegevensbescherming
LLM	Large Language Model
IBS	Integratie Business Services
NLP	Natural Language Processing
RAG	Retrieval Augmented Generation
AI	Artificial Intelligence
ASP.NET	Active Server Pages .NET (een Microsoft framework)
Scaled Agile Framework	SAFe

Disclaimer

“Wegens de vertrouwelijke aard van de informatie in het werkstuk/ de afstudeerscriptie en de bijbehorende bijlagen, mag de inhoud noch geheel noch gedeeltelijk op enigerlei wijze worden aangepast, gewijzigd of verveelvoudigd zonder schriftelijke toestemming van de auteur en het betrokken bedrijf. Zowel de docent/ scriptiebegeleider, de examinator alsmede de medewerkers van NOVI Hogeschool worden gehouden de inhoud van het werkstuk/ de afstudeerscriptie als strikt vertrouwelijk te behandelen. NOVI Hogeschool bewaart het document in een afgesloten databank. Gedurende de bewaarperiode kunnen studentdossiers worden ingezien door medewerkers van NOVI, de Examencommissie van NOVI alsmede de Inspectie van het Onderwijs en een visitatiecommissie van de Nederlands Vlaams Accreditatieorganisatie (NVAO, bij (her)accreditaties).“

Inhoud

Inleiding

In een dynamische en complexe organisatie zoals de Belastingdienst is de toegang tot accurate en actuele informatie essentieel voor het effectief uitvoeren van taken. Integratie Business Services (IBS) Toeslagen, onderdeel van de Belastingdienst, heeft te maken met versnipperde informatiebronnen zoals repositories, wiki's en chatkanalen. Dit belemmert niet alleen de efficiëntie van medewerkers, maar ook de kwaliteit van kennisdeling binnen en tussen teams. In reactie hierop is een Retrieval-Augmented Generation (RAG) systeem ontwikkeld dat als geavanceerde chatapplicatie fungeert.

Deze applicatie biedt een innovatieve oplossing door gebruik te maken van semantische zoektechnologieën, vector-gebaseerde opslag en automatische documentanalyse. De inzet van geavanceerde AI-technologieën, zoals een Large Language Model (LLM), stelt medewerkers in staat om snel relevante informatie te vinden en directe antwoorden te verkrijgen op specifieke vragen. Dit bevordert niet alleen de productiviteit, maar ook de tevredenheid van gebruikers.

De technische architectuur van de applicatie omvat een combinatie van moderne technologieën, waaronder FastAPI, Pinecone, Azure Blob Storage en Blazor WebAssembly. De nadruk ligt op schaalbaarheid, beveiliging en gebruikersgemak. Daarnaast biedt het systeem een robuuste infrastructuur met real-time queryverwerking en automatische documentanalyse, wat resulteert in een efficiëntere en intuïtieve werkomgeving.

Dit rapport beschrijft het functionele en technische ontwerp van de applicatie, de implementatie van kernfunctionaliteiten en de voordelen die de applicatie biedt voor de organisatie. Hiermee draagt het bij aan de optimalisatie van werknemerstevredenheid en efficiëntie binnen IBS Toeslagen.

Voor de actuele software raadpleeg source control: https://github.com/yannicklansink/central-search-engine-toeslagen

Voor een beeld van de chat app heeft de auteur een video gemaakt waarbij hij door de technische implicaties loopt van de app: https://www.youtube.com/watch?v=Gk66Wk0aq-c

1. Functioneel ontwerp

De backend van de RAG-service is ontworpen om de chatapplicatie te ondersteunen die medewerkers van Integratie Business Services (IBS) Toeslagen helpt bij het vinden van informatie. De backend verwerkt verzoeken, voert zoekopdrachten uit in een vector database, en genereert antwoorden met behulp van een Large Language Model (LLM).

1.1 Functionele Requirements

Document Upload Functionaliteit

- **FR1.1:** Gebruikers moeten in staat zijn om documenten te uploaden via de frontend interface. Dit stelt gebruikers in staat om relevante informatie beschikbaar te maken voor verdere verwerking en analyse.
- **FR1.2:** De applicatie moet verschillende bestandsformaten ondersteunen, zoals .txt, .pdf en .html. Dit zorgt voor flexibiliteit en gebruiksgemak voor de gebruikers.
- **FR1.3:** Na het uploaden moet de gebruiker een bevestiging ontvangen dat het document succesvol is geüpload. Dit biedt duidelijkheid en zekerheid aan de gebruiker.

Realtime Vraag-Antwoord API

- **FR2.1:** Gebruikers moeten vragen kunnen stellen via de frontend interface. Dit maakt interactie met de applicatie mogelijk en verbetert de gebruikerservaring.
- **FR2.2:** De applicatie moet real-time antwoorden kunnen geven op basis van de geüploade documenten. Dit zorgt voor snelle en relevante informatieverstrekking.
- **FR2.3:** De antwoorden moeten relevant en nauwkeurig zijn. Dit verhoogt de betrouwbaarheid en bruikbaarheid van de applicatie.

Slimme Document Zoekfunctie

- **FR3.1:** De zoekfunctionaliteit moet semantische overeenkomsten kunnen vinden, zelfs als trefwoorden niet exact overeenkomen. Dit verbetert de nauwkeurigheid en relevantie van de zoekresultaten.
- **FR3.2:** De zoekresultaten moeten relevant en nauwkeurig zijn. Dit verhoogt de tevredenheid en efficiëntie van de gebruikers.

Automatische Documentanalyse met Embeddings

- **FR4.1:** Bij het uploaden moeten documenten automatisch worden geanalyseerd. Dit zorgt voor een efficiënte en effectieve verwerking van documenten.
- **FR4.2:** De applicatie moet embeddings genereren met behulp van AI (OpenAI). Dit verbetert de mogelijkheden voor verdere analyse en zoekfunctionaliteit.
- **FR4.3:** De gegenereerde embeddings moeten effectief worden opgeslagen in chunks. Dit zorgt voor een efficiënte opslag en verwerking van embeddings.

Persoonlijke Chatervaring met Geschiedenis

- **FR5.1:** Gebruikers moeten chatgesprekken kunnen voeren via de frontend interface. Dit verbetert de interactie en gebruikerservaring.

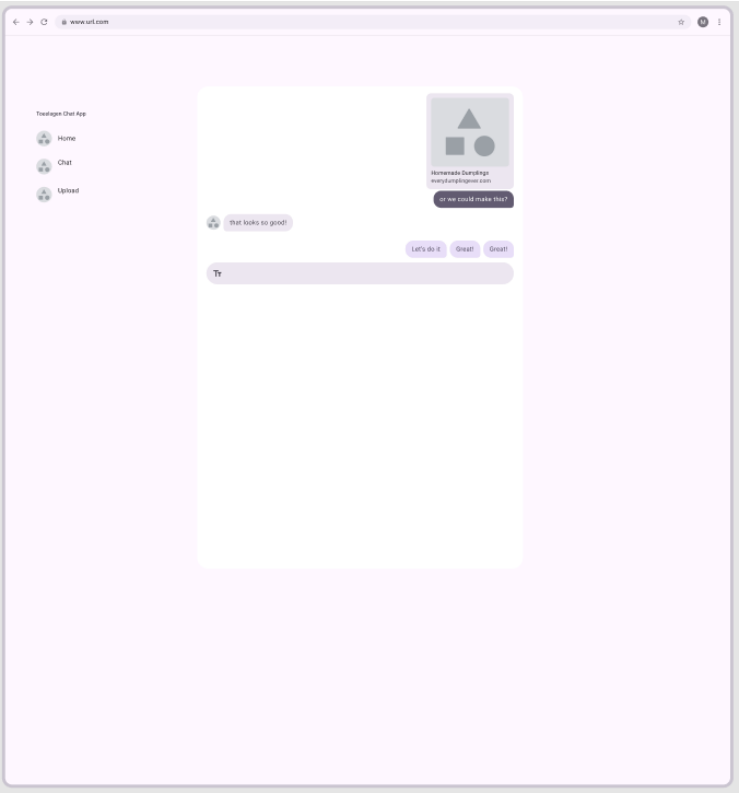
- **FR5.2:** De chatgeschiedenis moet lokaal worden opgeslagen zodat gebruikers deze kunnen terugkijken. Dit verhoogt de bruikbaarheid en functionaliteit van de applicatie.
- **FR5.3:** De chatgeschiedenis moet veilig en privé worden opgeslagen. Dit zorgt voor de privacy en veiligheid van de gebruikersgegevens.

1.2 Wireframe

Overzicht Pagina's

- **Home:** Welkomstpagina van de applicatie.
- **Chat:** Chatinterface waar gebruikers vragen kunnen stellen en antwoorden ontvangen.
- **Upload:** Interface voor het uploaden van documenten naar de blob storage.

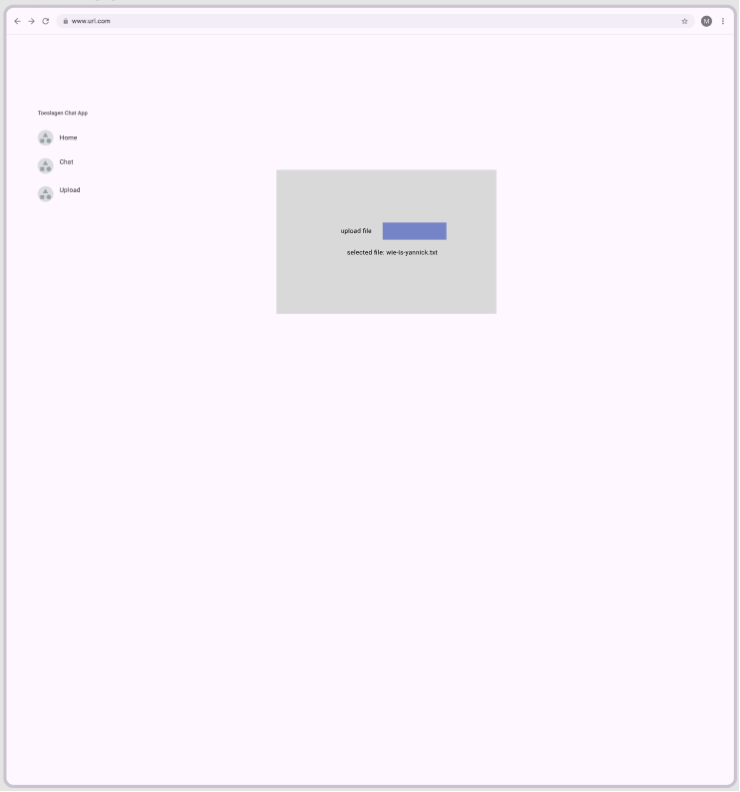
Schets van Chatinterface



Afbeelding 1.2.1 schets chat mockup

Beschrijving: Een intuïtieve en minimalistische interface waarmee gebruikers vragen kunnen stellen en antwoorden kunnen ontvangen.

Schets van Uploadinterface



Afbeelding 1.2.2 schets upload mockup

Beschrijving: Een eenvoudige uploadpagina waar gebruikers bestanden kunnen toevoegen en uploaden naar het systeem.

1.3 Technologiestack

- **Programmeertaal:** Python
- **Web Framework:** FastAPI
- **Vector Database:** Pinecone
- **Embeddings:** OpenAI
- **Cloud Storage:** Azure Blob Storage
- **Dependency Management:** Poetry en requirements.txt files

1.4 Architectuur

Belangrijke Componenten

- **FastAPI:** Voor het opzetten van de API-server en het verwerken van verzoeken.
- **Pinecone:** Voor het opslaan en doorzoeken van vectorrepresentaties van documenten.

- **OpenAI:** Voor het genereren van embeddings en het beantwoorden van vragen.
- **Azure Blob Storage:** Voor het opslaan van geüploade documenten.
- **LangChain:** Voor het opzetten van de keten van bewerkingen die nodig zijn om een antwoord te genereren.

1.5 Third-Party Integraties

- **Azure:** Voor blob storage.
- **Pinecone:** Voor vector database functionaliteit.
- **OpenAI:** Voor het genereren van embeddings en het beantwoorden van vragen.

1.6 API Specificaties

Endpoints

- **GET /:** Redirect naar de API-documentatie.
- **POST /RAG/stream:** Verwerkt een vraag en streamt het antwoord terug naar de client.

1.7 Database Structuur

Vector Database

- **Pinecone:** Slaat vectorrepresentaties van documenten op. Deze vectoren worden gebruikt om snel relevante informatie te vinden op basis van de semantische inhoud van de vraag.

Wat Wordt Opgeslagen

- **Documenten:** Geüploade documenten worden opgeslagen in Azure Blob Storage voor een tijdelijke duur.
- **Embeddings:** Vectorrepresentaties van documenten worden opgeslagen in Pinecone.
- **Chatgeschiedenis:** Opslaan van chatgeschiedenis in de browser's localStorage.

1.8 Azure Function

Azure Functions worden gebruikt om serverless computing mogelijk te maken, wat schaalbaarheid en kostenbesparing biedt. Ze worden ingezet voor specifieke taken, zoals het verwerken van geüploade bestanden.

Azure Functions worden geactiveerd door triggers, zoals het uploaden van een bestand naar een specifieke container in Azure Blob Storage. Wanneer een bestand wordt geüpload, wordt de functie geactiveerd om het bestand te verwerken en op te slaan.

1.9 Monitoring en Logging

LangSmith is een tool die wordt gebruikt voor het loggen en monitoren van API-aanroepen naar een Large Language Model (LLM). Het helpt bij het bijhouden van de keten van bewerkingen die worden uitgevoerd om een antwoord te genereren. Hierdoor kunnen ontwikkelaars beter inzicht krijgen in de prestaties en betrouwbaarheid van hun LLM-integraties.

Voorbeeld: Het loggen van verzoeken, antwoorden, en fouten tijdens de verwerking wordt gevisualiseerd en geanalyseerd.

1.10 Samenvatting

Dit functioneel ontwerp document biedt een volledig overzicht van de doelstellingen, technologieën, architectuur, API-specificaties, database-structuur, Azure Functions, en monitoring en logging van de RAG-service. De beschreven componenten en processen zorgen voor een robuuste en schaalbare oplossing die voldoet aan de behoeften van IBS Toeslagen.

2. Technisch ontwerp

De gekozen tech stack is ontworpen om een efficiënte, schaalbare en veilige chatapplicatie te ondersteunen die medewerkers van Integratie Business Services (IBS) Toeslagen helpt bij het vinden van informatie. De tech stack omvat zowel frontend- als backend-technologieën, hostingoplossingen en beveiligingsmaatregelen om een robuuste en gebruiksvriendelijke applicatie te leveren.

2.1 Technische Requirements

Documentbeheer en Slimme Opslag

- **TR1.1:** Gebruik van het InputFile component in Blazor voor het selecteren en uploaden van bestanden, waarbij de component robuust moet omgaan met verschillende bestandsformaten en foutafhandeling.
- **TR1.2:** Implementatie van een BlobService in C# om bestanden efficiënt en veilig naar Azure Blob Storage te uploaden.
- **TR1.3:** Configuratie van FormOptions in Program.cs om de maximale bestandsgrootte te beperken tot 10 MB, met duidelijke foutmeldingen voor gebruikers bij overschrijding.
- **TR1.4:** Gebruik van Azure Blob Storage voor het opslaan van geüploade documenten, waarbij gebruik wordt gemaakt van versleuteling en toegangsbeheer om de veiligheid van de data te waarborgen.
- **TR1.5:** Bestanden worden tijdelijk opgeslagen in Azure Blob Storage. Ze worden tijdelijk opgeslagen met een levensduur van 1 uur voordat ze automatisch worden verwijderd. Tijdens deze periode worden de bestanden versleuteld met industriestandaard encryptieprotocollen zoals TLS voor transport.

Realtime Vraag-Antwoord API

- **TR2.1:** Implementatie van API-endpoints in FastAPI voor het verwerken van vragen, met ondersteuning voor schaalbaarheid en foutafhandeling.
- **TR2.2:** Gebruik van OpenAI voor het genereren van antwoorden op basis van de documenten, waarbij de API efficiënt moet omgaan met grote hoeveelheden data en complexe queries.
- **TR2.3:** Integratie met Pinecone voor het doorzoeken van vectorrepresentaties van documenten, waarbij de zoekalgoritmen geoptimaliseerd zijn voor snelheid en nauwkeurigheid.
- **TR2.4:** Gebruik van streaming API's om berichten terug te sturen naar de frontend, zodat de gebruiker constant wordt bijgewerkt met real-time informatie, met een focus op lage latentie en betrouwbaarheid.

Slimme Document Zoekfunctie

- **TR3.1:** Implementatie van een zoekfunctionaliteit in de frontend om documenten op te halen uit de vector database, met een intuïtieve gebruikersinterface en snelle responstijden.
- **TR3.2:** Gebruik van Pinecone voor het doorzoeken van vectorrepresentaties van documenten, waarbij de zoekresultaten semantisch relevant en nauwkeurig moeten zijn.

- **TR3.3:** Integratie met OpenAI voor het genereren van embeddings en het uitvoeren van semantische zoekopdrachten, met een focus op hoge precisie en recall.

Automatische Documentanalyse met Embeddings

- **TR4.1:** Gebruik van Azure Functions om een trigger te activeren bij het uploaden van een document, met een focus op betrouwbaarheid en schaalbaarheid.
- **TR4.2:** Implementatie van een Python script (function_app.py) om de documenten te analyseren en embeddings te genereren, waarbij gebruik wordt gemaakt van efficiënte en nauwkeurige AI-modellen.
- **TR4.3:** Integratie met Pinecone voor het opslaan van de gegenereerde embeddings, met een focus op snelle opslag en retrieval.
- **TR4.4:** Gebruik van OpenAI voor het genereren van embeddings, waarbij de modellen geoptimaliseerd zijn voor de specifieke documenttypes en zoekopdrachten.
- **TR4.5:** Opsplitsing van documenten in chunks voor efficiënte opslag en verwerking van embeddings, met aandacht voor gegevensintegriteit en schaalbaarheid.
- **TR4.6:** Implementatie van een mechanisme voor de controle en normalisatie van bestandsnamen om ervoor te zorgen dat bestanden met identieke namen worden overschreven. Dit voorkomt veroudering van data en waarborgt de consistentie en actualiteit van de opgeslagen informatie.

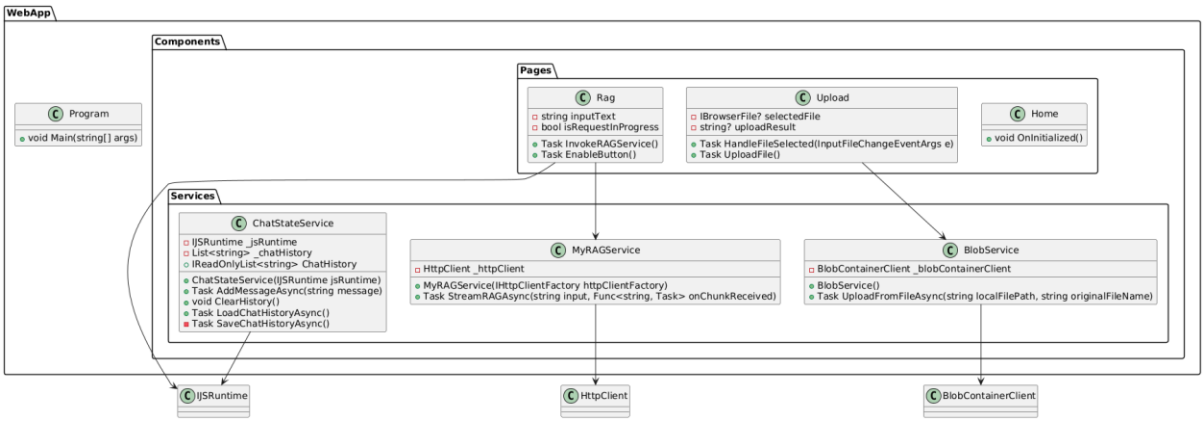
Persoonlijke Chatervaring met Geschiedenis

- **TR5.1:** Implementatie van een chatinterface in Blazor voor het voeren van gesprekken, met een focus op gebruiksvriendelijkheid en responsiviteit.
- **TR5.2:** Gebruik van localStorage in de browser voor het opslaan van de chatgeschiedenis, met aandacht voor gegevensintegriteit en beveiliging.
- **TR5.3:** Implementatie van een ChatStateService in C# om de chatgeschiedenis te beheren, met ondersteuning voor synchronisatie en herstel bij fouten.
- **TR5.4:** Gebruik van JavaScript interop (IJSRuntime) om de chatgeschiedenis op te slaan en op te halen uit localStorage, met een focus op prestaties en betrouwbaarheid.

2.2 Technische Infrastructuur

2.2.1 Frontend

UML Class Diagram van de Frontend



Afbeelding 2.2 class diagram frontend

2.2.1.1 Uitleg van klassen

ChatStateService: Beheert de chatgeschiedenis.

- IJSRuntime _jsRuntime: Interface voor het uitvoeren van JavaScript-functies.
- List<string> _chatHistory: Lijst met chatgeschiedenis.
- ChatStateService(IJSRuntime jsRuntime): Constructor.
- IReadOnlyList<string> ChatHistory: Eigenschap voor het ophalen van de chatgeschiedenis.
- Task AddMessageAsync(string message): Voegt een bericht toe aan de chatgeschiedenis.
- void ClearHistory(): Leegt de chatgeschiedenis.
- Task LoadChatHistoryAsync(): Laadt de chatgeschiedenis.
- Task SaveChatHistoryAsync(): Slaat de chatgeschiedenis op.

BlobService: Beheert bestanden in Azure Blob Storage.

- BlobContainerClient _blobContainerClient: Client voor het beheren van blob containers.
- BlobService(): Constructor.
- Task UploadFromFileAsync(string localFilePath, string originalFileName): Uploadt een bestand naar Azure Blob Storage.

MyRAGService: Verwerkt vragen en haalt antwoorden op.

- HttpClient _httpClient: Client voor het uitvoeren van HTTP-aanvragen.
- MyRAGService(IHttpClientFactory httpClientFactory): Constructor.
- Task StreamRAGAsync(string input, Func<string, Task> onChunkReceived): Verwerkt een vraag en haalt een antwoord op.

Home: Beheert de homepagina.

- void OnInitialized(): Methode die wordt aangeroepen bij initialisatie.

Rag: Beheert de RAG-invoegpagina.

- string inputText: Invoertekst voor de vraag.
- bool isRequestInProgress: Geeft aan of er een verzoek in behandeling is.
- Task InvokeRAGService(): Roept de RAG-service aan.
- Task EnableButton(): Schakelt de knop in.

Upload: Beheert de uploadpagina.

- IBrowserFile? selectedFile: Geselecteerd bestand.
- string? uploadResult: Resultaat van de upload.
- Task HandleFileSelected(InputFileChangeEventArgs e): Handelt de bestandsselectie af.
- Task UploadFile(): Uploadt het bestand.

Program: Beheert de hoofdprogramma-initialisatie.

- void Main(string[] args): Hoofdprogramma-initialisatie.

2.2.1.2 Technologische basis

Programeertaal

- C#

Frameworks:

- Blazor WebAssembly: Voor het bouwen van interactieve webapplicaties met behulp van C# en .NET.

Styling:

- CSS: Voor het stylen van de gebruikersinterface.

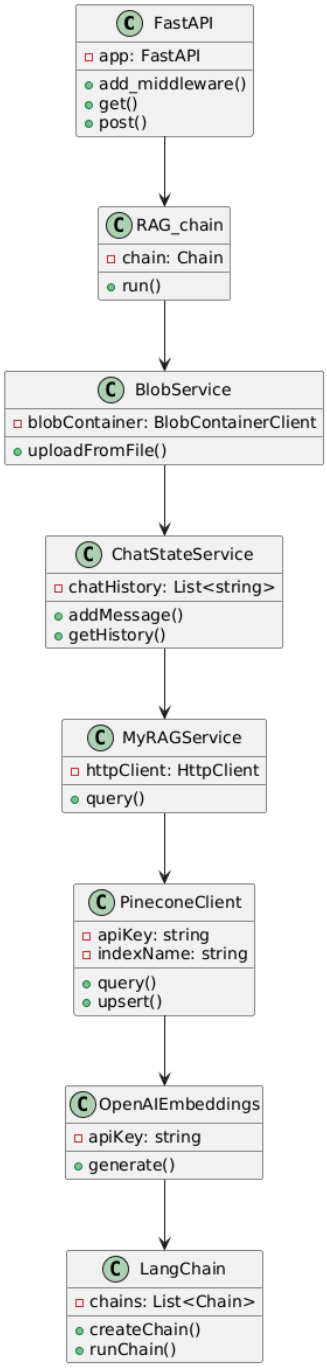
- Bootstrap: Voor het gebruik van kant-en-klare stijlen en componenten.

Build Tools:

- .NET SDK: Voor het bouwen en publiceren van de Blazor WebAssembly applicatie.

2.2.2 Backend

UML Class Diagram van de Backend Service



Afbeelding 2.2.2 class diagram backend service (rag-service)

2.2.2.1 Uitleg Klassen

FastAPI

De hoofdklasse voor het opzetten van de API-server en het verwerken van verzoeken.

- **add_middleware()**: Voegt middleware toe aan de applicatie.
- **get()**: Definieert een GET-endpoint.

- **post()**: Definieert een POST-endpoint.

RAG_chain
De klasse die de keten van bewerkingen definieert voor het verwerken van vragen en het genereren van antwoorden.

- **run()**: Voert de keten van bewerkingen uit.

BlobService
De klasse voor het beheren van bestanden in Azure Blob Storage.

- **uploadFromFile()**: Uploadt een bestand naar Azure Blob Storage.

ChatStateService
De klasse voor het beheren van de chatgeschiedenis.

- **addMessage()**: Voegt een bericht toe aan de chatgeschiedenis.
- **getHistory()**: Haalt de chatgeschiedenis op.

MyRAGService
De klasse voor het verwerken van vragen en het ophalen van antwoorden.

- **query()**: Verwerkt een vraag en haalt een antwoord op.

PineconeClient
De klasse voor het beheren van de Pinecone vector database.

- **query()**: Voert een query uit op de Pinecone database.
- **upsert()**: Voegt nieuwe vectoren toe aan de Pinecone database.

OpenAIEmbeddings
De klasse voor het genereren van embeddings met behulp van OpenAI.

- **generate()**: Genereert embeddings voor een gegeven tekst.

LangChain
De klasse voor het beheren van ketens van bewerkingen.

- **createChain()**: Maakt een nieuwe keten van bewerkingen.
- **runChain()**: Voert een keten van bewerkingen uit.

2.2.2.2 Technologische basis

Programmeertaal

- **Python**

Frameworks

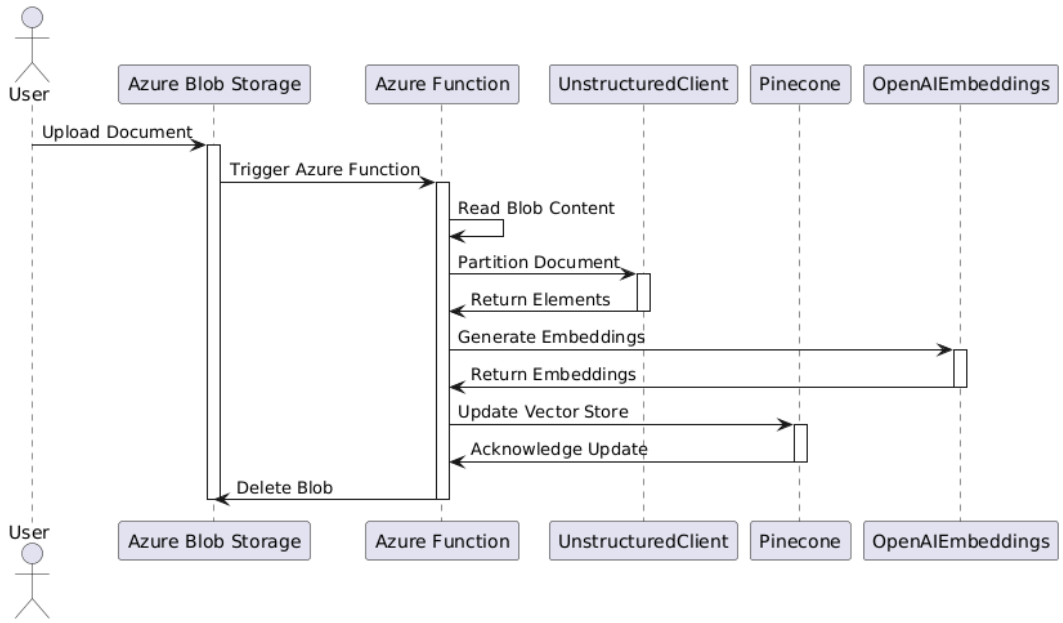
- **FastAPI**: Voor het opzetten van de API-server en het verwerken van verzoeken.
- **LangChain**: Voor het opzetten van de keten van bewerkingen die nodig zijn om een antwoord te genereren.
- **Uvicorn**: ASGI-server voor het draaien van de FastAPI applicatie.

Database

- **Pinecone**: Voor het opslaan en doorzoeken van vectorrepresentaties van documenten.

2.2.3 Azure Functie

Sequence diagram Azure function



Afbeelding 2.2.3 sequence diagram azure function

Uitleg van de interacties van de sequence diagram.

1. **User -> BlobStorage: Upload Document**
    - De gebruiker uploadt een document naar Azure Blob Storage.
2. **BlobStorage -> FunctionApp: Trigger Azure Function**

    - Azure Blob Storage triggert de Azure Function wanneer een nieuw document wordt geüpload.
3. **FunctionApp -> FunctionApp: Read Blob Content**
    - De Azure Function leest de inhoud van het geüploade document.
4. **FunctionApp -> UnstructuredClient: Partition Document**
    - De Azure Function roept de UnstructuredClient aan om het document te partitioneren.
5. **UnstructuredClient -> FunctionApp: Return Elements**
    - De UnstructuredClient retourneert de gepartitioneerde elementen van het document.
6. **FunctionApp -> OpenAIEmbeddings: Generate Embeddings**
    - De Azure Function roept OpenAIEmbeddings aan om embeddings te genereren voor de gepartitioneerde elementen.
7. **OpenAIEmbeddings -> FunctionApp: Return Embeddings**
    - OpenAIEmbeddings retourneert de gegenereerde embeddings.
8. **FunctionApp -> Pinecone: Update Vector Store**
    - De Azure Function roept Pinecone aan om de vector store bij te werken met de nieuwe embeddings.
9. **Pinecone -> FunctionApp: Acknowledge Update**
    - Pinecone bevestigt de update van de vector store.
10. **FunctionApp -> BlobStorage: Delete Blob**
    - De Azure Function verwijdert het geüploade document uit Azure Blob Storage.

- **Azure Functions:** Voor het uitvoeren van serverless functies die specifieke taken uitvoeren, zoals het verwerken van geüploade bestanden.

Vector Database

- **Pinecone:** Voor het hosten van de vector database die wordt gebruikt voor het opslaan en doorzoeken van vectorrepresentaties van documenten.

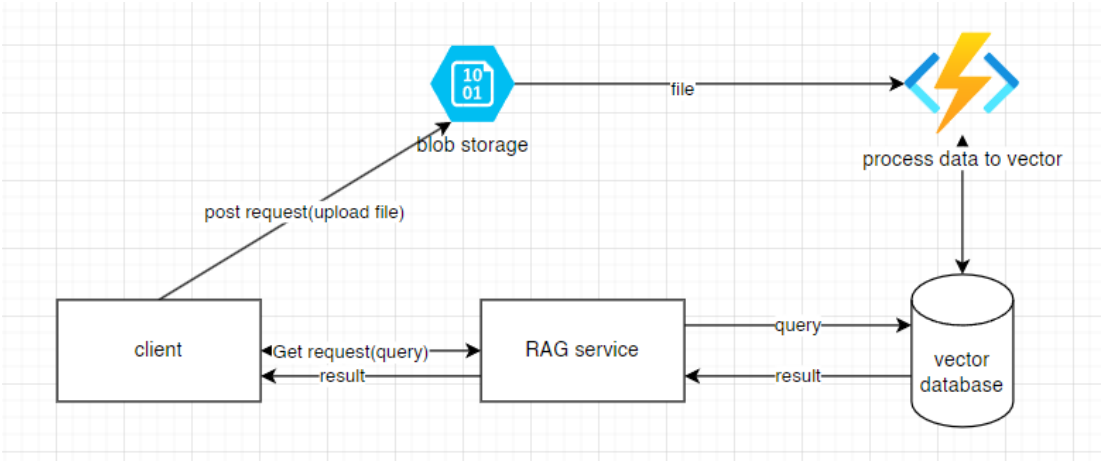
Source Code

- **GitHub:** Voor het beheren van de broncode en versiebeheer.

Azure Blob Storage

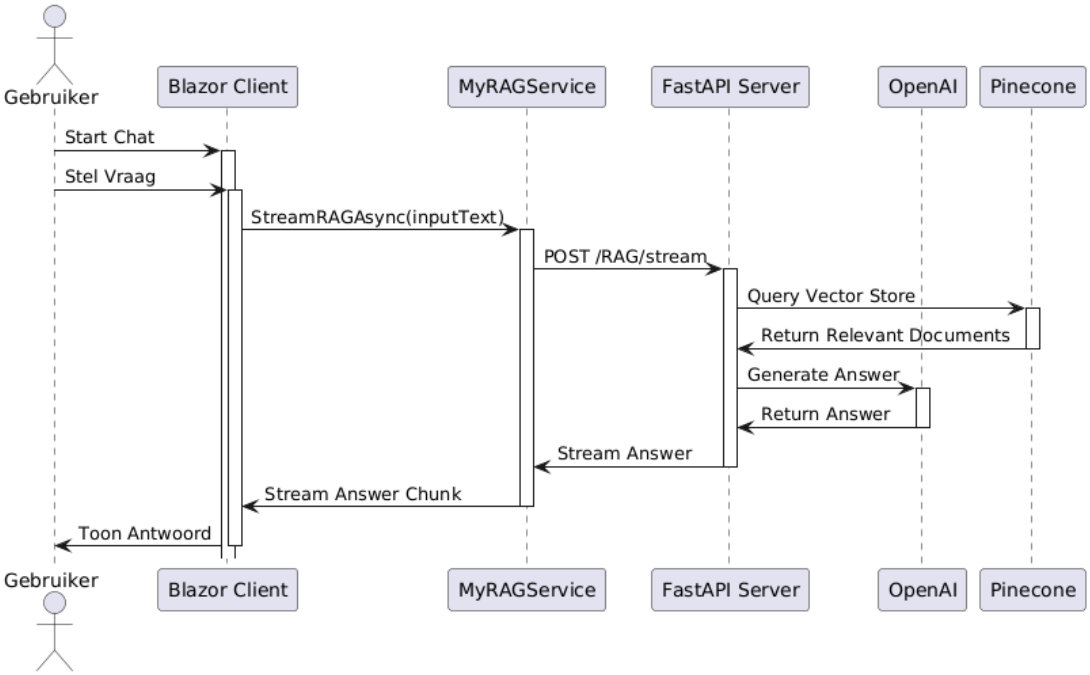
- **Azure Blob Storage:** Voor het opslaan van geüploade bestanden.

Hier een tekening van de componenten in het landschap en hoe ze met elkaar communiceren



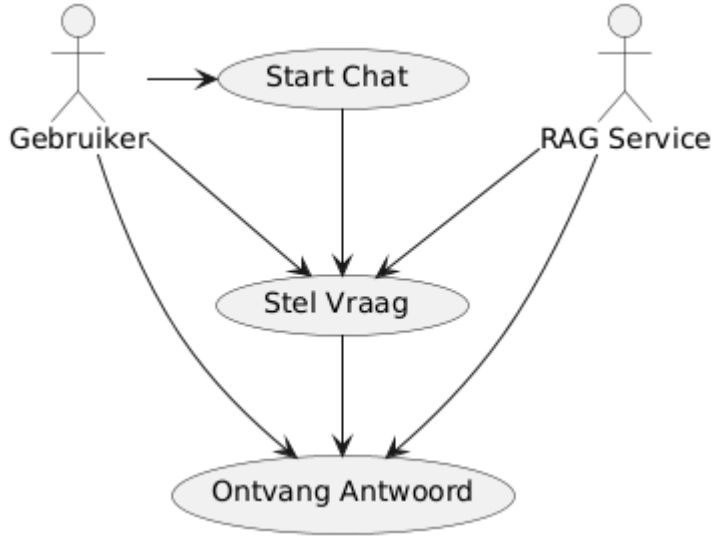
2.2.4 Requirements Sequences en Use Cases

Chat interactie sequence diagram



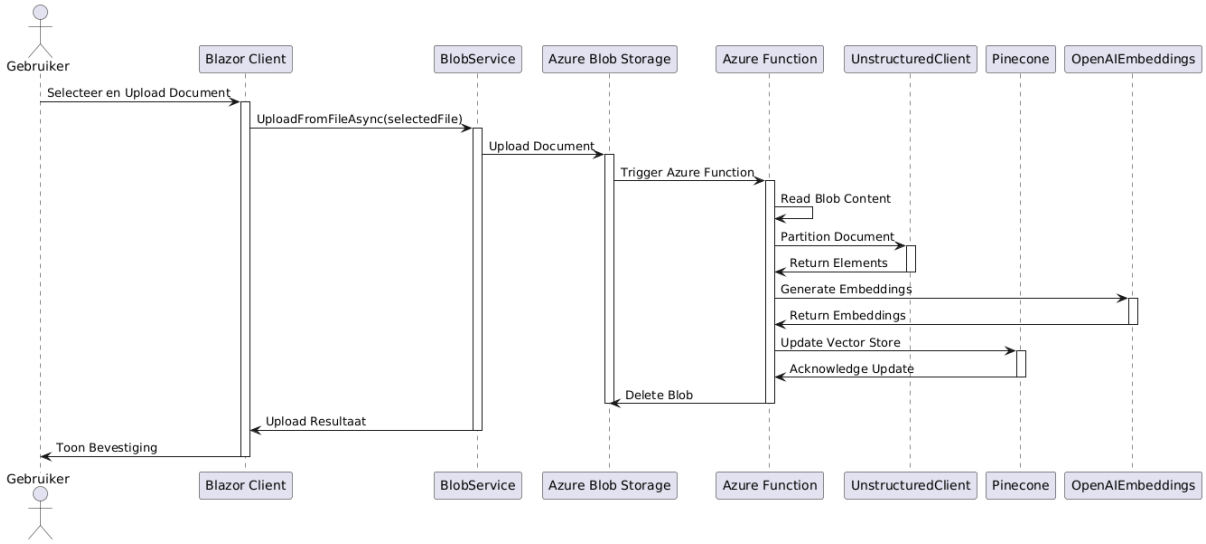
Afbeelding 2.2.4.1 sequence diagram chat interactie

Chat interactie use case diagram



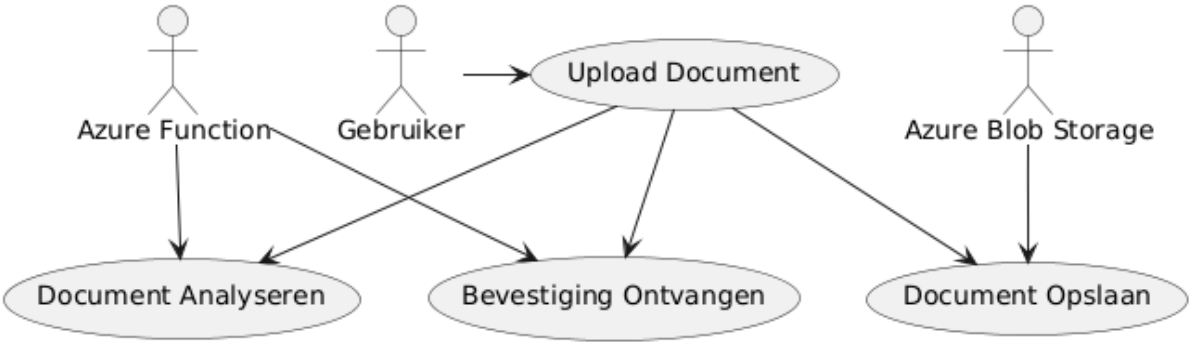
Afbeelding 2.2.4.2 use case diagram chat interactie

Upload interactie sequence diagram



Afbeelding 2.2.4.3 sequence diagram upload interactie

Upload interactie use case diagram



Afbeelding 2.2.4.4 Use case diagram upload functionaliteit

2.2.5 Beveiliging

Huidige status: Momenteel wordt nog weinig gedaan op het gebied van beveiliging omdat het om een prototype gaat.

Toekomstige maatregelen: In de toekomst worden de volgende maatregelen genomen om ervoor te zorgen dat onze data veilig blijft:

- **Data Encryptie:** Alle data die wordt opgeslagen in Azure Blob Storage en Pinecone wordt versleuteld om ongeautoriseerde toegang te voorkomen.
- **Veilige API-aanroepen:** Gebruik van HTTPS voor alle API-aanroepen om ervoor te zorgen dat de data tijdens het transport wordt versleuteld.
- **Beveiligingsprotocollen:** Implementatie van strikte beveiligingsprotocollen en best practices om ervoor te zorgen dat gevoelige informatie niet toegankelijk is voor onbevoegde partijen.
- **Regelmatige Beveiligingsaudits:** Voeren van regelmatige beveiligingsaudits en penetratietests om potentiële kwetsbaarheden te identificeren en te verhelpen.
- **Gebruik van Secure Credentials:** Gebruik van veilige methoden voor het opslaan en beheren van API-sleutels en andere gevoelige informatie, zoals Azure Key Vault.
- **LLM op eigen servers draaien:** Het draaien van de Language Learning Models (LLM) op eigen servers om data in huis te houden.

2.3 Risicoanalyse

Een risicoanalyse is essentieel om potentiële problemen te identificeren die de voortgang en het succes van het project kunnen beïnvloeden. Hieronder volgt een gedetailleerde risicoanalyse voor de ontwikkeling en implementatie van de RAG-service.

Tabel 2.3 Risicoanalyse

Risico	**Impact**	**Kans van optreden**	**Beheersmaatregel**
Integratieproblemen met Pinecone en OpenAI	Hoog	Middel	Monitor en log de prestaties van de integraties met behulp van Langsmith om problemen vroegtijdig te detecteren en op te lossen.
Schaalbaarheidsproblemen	Hoog	Middel	Optimaliseer de code en gebruik schaalbare cloud-oplossingen zoals Azure Functions en Blob Storage.
Beveiligingslekken	Zeer hoog	Middel	Gebruik voor het prototype geen gevoelige data en implementeer basisbeveiligingsmaatregelen om de impact van eventuele lekken te minimaliseren.
Onvoldoende training van gebruikers	Middel	Hoog	Organiseer een demo bij IBS Toeslagen, waarbij gebruikers Toeslagen breed een voorproefje krijgen van de chatapplicatie. Dit stelt hen in staat om de functionaliteiten te verkennen en vragen te stellen.
Systeem downtime	Hoog	Laag	Redundante systemen, disaster recovery planning
Data kwaliteit	Hoog	Middel	Implementatie van data validatie processen
Weerstand tegen verandering	Middel	Hoog	Change management strategie, stakeholder engagement
Afhankelijkheid van externe vendors	Hoog	Middel	Vendor diversificatie, fall-back scenarios
Budget overschrijding	Hoog	Middel	Strikte projectmonitoring, fase-gewijze implementatie

De risicoanalyse identificeert verschillende potentiële risico's die de ontwikkeling en implementatie van de RAG-service kunnen beïnvloeden. Door deze risico's te identificeren en passende mitigatiestrategieën te implementeren, kan ik de impact van deze risico's minimaliseren en de kans op een succesvol project vergroten. Het is belangrijk om regelmatig de risico's te evalueren en de mitigatiestrategieën bij te werken om in te spelen op nieuwe uitdagingen en veranderingen in de projectomgeving.

3. Werkende software

3.1 Doelstelling en Functionaliteit

De ontwikkelde applicatie is een geavanceerd Retrieval-Augmented Generation (RAG) systeem dat als chatapplicatie functioneert, specifiek ontworpen voor Integratie Business Services (IBS) Toeslagen. Het primaire doel is om medewerkers te ondersteunen bij het snel en efficiënt vinden van relevante informatie, waardoor de algehele productiviteit en tevredenheid worden verhoogd.

3.2 Gebruikersgroepen en Toepassingsgebied

De primary gebruikers van de applicatie zijn:

- IBS Toeslagen medewerkers (±150 IT-professionals)
- Nieuwe medewerkers in inwerktrajecten

De applicatie wordt ingezet binnen een Scaled Agile Framework (SAFe) omgeving met 15 gedistribueerde teams, waar informatie momenteel verspreid is over diverse bronnen zoals repositories, wiki's en chatkanalen (Jansen, 2023).

3.3 Systeemvoordelen

Uiteindelijke operationele voordelen waar we op doelen met dit project:

1. **Reductie van Zoektijd voor Informatie**

Waarom dit een voordeel is: In een complexe organisatie zoals IBS Toeslagen, waar informatie verspreid is over verschillende bronnen zoals repositories, wiki's, en chatkanalen, kan het vinden van de juiste informatie tijdrovend zijn. Medewerkers besteden vaak veel tijd aan het zoeken naar antwoorden op vragen, wat de productiviteit vermindert.

Hoe de chatapplicatie dit oplost: De chatapplicatie maakt gebruik van een geavanceerde vector database en een Large Language Model (LLM) om snel en nauwkeurig antwoorden te genereren op basis van de context van de vraag. Door gebruik te maken van semantische zoekfunctionaliteit, kunnen medewerkers relevante informatie vinden zonder exacte trefwoorden te kennen. Dit vermindert de zoektijd aanzienlijk en verhoogt de efficiëntie.

2. **Verhoogde Medewerkerstevredenheid door Efficiëntere Opzoek Functionaliteit**

Waarom dit een voordeel is: Medewerkers ervaren vaak frustratie wanneer ze niet snel de benodigde informatie kunnen vinden. Dit kan leiden tot een negatieve werkervaring en verminderde tevredenheid.

Hoe de chatapplicatie dit oplost: De chatapplicatie biedt een intuïtieve en gebruiksvriendelijke interface waarmee medewerkers snel en eenvoudig vragen kunnen stellen en antwoorden kunnen ontvangen. De real-time vraag-antwoord API zorgt ervoor dat medewerkers direct relevante informatie krijgen, wat hun efficiëntie verbetert en hun tevredenheid verhoogt. 2 van de belangrijkste business doelstellingen van dit project (Lansink, 2023).

3. **Verbeterde Onboarding-Ervaring voor Nieuwe Medewerkers**

Waarom dit een voordeel is: Nieuwe medewerkers hebben vaak moeite om hun weg te vinden in een complexe informatieomgeving. Dit kan de onboarding-ervaring negatief beïnvloeden en de tijd die nodig is om productief te worden verlengen.

Hoe de chatapplicatie dit oplost: De chatapplicatie biedt nieuwe medewerkers een centrale plek waar ze snel antwoorden kunnen vinden op hun vragen. Door gebruik te maken van de semantische zoekfunctionaliteit en de contextuele antwoorden van de LLM, kunnen nieuwe medewerkers sneller de benodigde informatie vinden en zich sneller aanpassen aan hun nieuwe rol. Dit verbetert de onboarding-ervaring en verkort de tijd die nodig is om productief te worden.

4. **Geoptimaliseerde Kennisdeling tussen Teams**

Waarom dit een voordeel is: In een organisatie met meerdere teams kan het delen van kennis en informatie een uitdaging zijn. Informatie kan verloren gaan of moeilijk toegankelijk zijn voor andere teams, wat de samenwerking en efficiëntie vermindert.

Hoe de chatapplicatie dit oplost: De chatapplicatie fungeert als een centrale hub voor informatie, waar alle teams toegang hebben tot dezelfde bron van waarheid. Door gebruik te maken van de vector database en de LLM, kunnen teams snel en eenvoudig informatie delen en vinden. Dit bevordert de samenwerking en zorgt ervoor dat kennis efficiënt wordt gedeeld tussen teams, wat de algehele productiviteit en effectiviteit verhoogt.

3.4 Technische Implementatie

De technische implementatie van de RAG-service (Retrieval-Augmented Generation) is ontworpen om een robuuste, efficiënte en schaalbare oplossing te bieden voor het snel en nauwkeurig vinden van informatie binnen Integratie Business Services (IBS) Toeslagen. Hieronder volgt een uitgebreide uitleg van de belangrijkste componenten en hun rol in het systeem.

3.4.1 Vector-gebaseerde Zoekfunctionaliteit

Afbeelding 3.4.1 Code in RAG-service – Opzetten van retriever om vectoren op te halen

```
embeddings = OpenAIEmbeddings(model="text-embedding-ada-002")
vectorstore = Pinecone.from_existing_index(index_name=PINECONE_INDEX_NAME,
                                           embedding=embeddings)

retriever = vectorstore.as_retriever(
    search_type="similarity_score_threshold",
    search_kwargs={
        "score_threshold": 0.85,
        "k": 3,
        }
)
```

De vector-gebaseerde zoekfunctionaliteit maakt gebruik van cosine similarity voor semantische matching. Documenten worden gerepresenteerd als vectoren in een hoogdimensionale ruimte,

waarbij de afstand tussen vectoren de semantische overeenkomst tussen documenten weergeeft. De threshold van 0.85 waarborgt hoge precisie in zoekresultaten door alleen resultaten terug te geven die een hoge mate van overeenkomst hebben met de zoekvraag.

Traditionele zoekmethoden zijn vaak beperkt tot exacte trefwoordovereenkomsten, wat inefficiënt is bij het doorzoeken van grote hoeveelheden ongestructureerde data. Door gebruik te maken van vectorrepresentaties en cosine similarity, kan het systeem semantische relaties tussen documenten identificeren, zelfs als de exacte woorden niet overeenkomen. Dit verhoogt de nauwkeurigheid en relevantie van de zoekresultaten aanzienlijk.

Deze technologie stelt medewerkers van IBS Toeslagen in staat om snel en efficiënt relevante informatie te vinden, wat de productiviteit verhoogt en de tijd die besteed wordt aan het zoeken naar informatie vermindert. Dit is cruciaal voor het succes van het project, aangezien het de kernfunctionaliteit van de applicatie ondersteunt.

3.4.2 Real-time Query Processing

Afbeelding 3.4.2.1 Code in RAG-service – Onderdeel van het terug streamen van antwoorden aan de gebruiker

```python
prompt = ChatPromptTemplate.from_template(template)

# Chain is de runnable that will be executed
chain = (
    {
        "context": retriever,
        "question": RunnablePassthrough(),
    }
    | prompt
    | model
    | StrOutputParser()
)
```

Afbeelding 3.4.2.2 Code in Client (frontend) – Onderdeel van het terug streamen van antwoorden aan de gebruiker

```
1 reference
public async Task StreamRAGAsync(string input, Func<string, Task> onChunkReceived)
{
    var requestData = new { input = input };
    var json = JsonSerializer.Serialize(requestData);
    var content = new StringContent(json, Encoding.UTF8, "application/json");

    var request = new HttpRequestMessage(HttpMethod.Post, "/RAG/stream")
    {
        Content = content
    };

    var response = await _httpClient.SendAsync(request, HttpCompletionOption.ResponseHeadersRead);

    if (response.IsSuccessStatusCode)
    {
        var stream = await response.Content.ReadAsStreamAsync();
        using (var reader = new StreamReader(stream))
        {
            while (!reader.EndOfStream)
            {
                var chunk = await reader.ReadLineAsync();
                if (!string.IsNullOrEmpty(chunk) && chunk.StartsWith("data: "))
                {
                    var chunkData = chunk.Substring(6);
                    await onChunkReceived(chunkData);
                }
            }
        }
    }
    else
    {
        var errorContent = await response.Content.ReadAsStringAsync();
        throw new ApplicationException($"Error: {response.StatusCode}, Details: {errorContent}");
    }
}
```

Het systeem implementeert asynchrone streaming responses voor real-time interactie, gebruikmakend van FastAPI's streaming mogelijkheden. Dit betekent dat gebruikers direct feedback

krijgen terwijl hun vraag wordt verwerkt, wat de interactie met de applicatie vloeiender en efficiënter maakt.

Traditionele systemen voor vraag-antwoord verwerking kunnen traag en inefficiënt zijn, vooral bij complexe queries die veel data vereisen. Door gebruik te maken van asynchrone streaming responses, kan het systeem gebruikers constant bijwerken met real-time informatie, wat de gebruikerservaring aanzienlijk verbetert.

Real-time query processing is essentieel voor de gebruiksvriendelijkheid van de applicatie. Het zorgt ervoor dat medewerkers snel antwoorden krijgen op hun vragen, wat de efficiëntie verhoogt en de tevredenheid van de gebruikers verbetert. Dit draagt bij aan de algehele doelstellingen van het project om de informatiezoekprocessen binnen IBS Toeslagen te optimaliseren.

3.4.3 Automatische Document Analyse

Afbeelding 3.4.3 Code in Azure function – 1 methode, update document die content omzet voor de vector database

```python
def update_document(blob_content, blob_name, index_name, embeddings):

    if not is_valid_file_type(blob_name):
        logging.error(f"Unsupported file type for {blob_name}. Supported file type
        return

    prefix = blob_name  # Use the blob name as prefix
    logging.info(f'filename: {blob_name} en prefix: {prefix}')

    # Delete existing embeddings based on the prefix
    ids_to_delete = []
    for ids in index.list(prefix=f"{prefix}#", namespace=''):
        ids_to_delete.extend(ids)
    if ids_to_delete:
        index.delete(ids=ids_to_delete, namespace='')
        logging.info(f"Existing embeddings for {blob_name} removed: {ids_to_delete
    else:
        logging.info(f"No existing embeddings found for filename: {blob_name}")

    # Process the new blob content and add the new embeddings
    files = shared.Files(content=blob_content, file_name=blob_name)

    req = shared.PartitionParameters(
        files=files,
        strategy="hi_res",
    )
    try:
        resp = unstructured_client.general.partition(req)
        elements = dict_to_elements(resp.elements)
    except SDKError as e:
        logging.error(f"Failed to partition document: {e}")
        return
```

```python
documents = []
ids = []
for element in chunked_elements:
    metadata = element.metadata.to_dict()
    del metadata["languages"]
    doc_id = generate_id(metadata["filename"])
    ids.append(doc_id)
    documents.append(Document(page_content=element.text, metadata=metadata))

# Add the new documents to the Pinecone vector store
try:
    vectorstore = PineconeVectorStore.from_existing_index(
        index_name=index_name,
        embedding=embeddings
    )

    vectorstore.add_texts(
        texts=[doc.page_content for doc in documents],
        metadatas=[doc.metadata for doc in documents],
        ids=ids
    )
    logging.info(f"New embeddings for {blob_name} added.")
except Exception as e:
    logging.error(f"Failed to add new embeddings: {e}")
```

De Azure Function implementeert een event-driven architectuur voor documentverwerking, waarbij nieuwe uploads automatisch worden gedetecteerd en verwerkt. Dit betekent dat zodra een document wordt geüpload naar Azure Blob Storage, de Azure Function wordt geactiveerd om het document te analyseren en embeddings te genereren.

Embeddings zijn

Traditionele documentverwerkingssystemen vereisen vaak handmatige interventie om documenten te analyseren en te indexeren. Door gebruik te maken van een event-driven architectuur en AI-modellen voor het genereren van embeddings, kan het systeem automatisch en efficiënt documenten verwerken zonder menselijke tussenkomst. Dit verhoogt de snelheid en nauwkeurigheid van de documentanalyse.

Automatische documentanalyse is van cruciaal belang voor het succes van het project, omdat het de basis vormt voor de zoekfunctionaliteit en de vraag-antwoord verwerking. Door documenten automatisch te analyseren en in chunks op te splitsen, kan het systeem snel en nauwkeurig relevante informatie vinden en antwoorden genereren.

3.5 Systeemarchitectuur

De systeemarchitectuur van de chat applicatie is zorgvuldig ontworpen om een robuuste, schaalbare en efficiënte oplossing te bieden voor het snel en nauwkeurig vinden van informatie binnen IBS Toeslagen. De architectuur bestaat uit vijf hoofdcomponenten die naadloos samenwerken om de functionaliteit van de applicatie te waarborgen.

Componenten:

1. Frontend (Blazor WebAssembly)

De frontend is gebouwd met Blazor WebAssembly, een framework dat het mogelijk maakt om interactieve webapplicaties te ontwikkelen met behulp van C# en .NET. Dit biedt een consistente en responsieve gebruikerservaring.

2. Backend (FastAPI)

De backend is ontwikkeld met FastAPI, een modern web framework voor het bouwen van API's met Python. FastAPI staat bekend om zijn hoge prestaties en gebruiksgemak.

3. Vector Database (Pinecone)

Pinecone is een gespecialiseerde vector database die is ontworpen voor het opslaan en doorzoeken van vectorrepresentaties van documenten. Het maakt gebruik van geavanceerde zoekalgoritmen om semantische overeenkomsten te vinden.

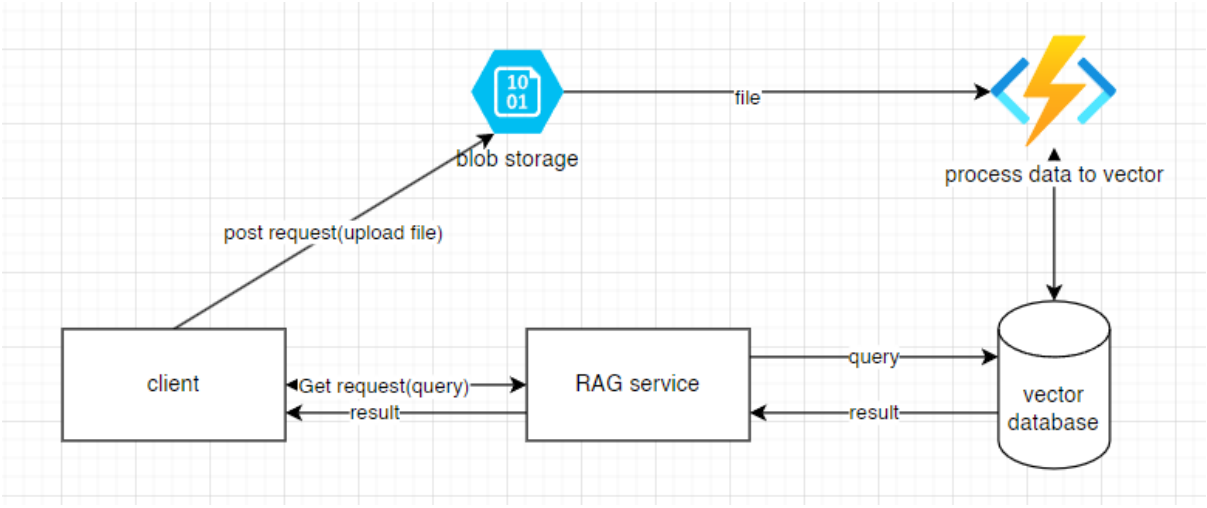
4. Document Processing (Azure Functions)

Azure Functions is een serverless compute service die het mogelijk maakt om kleine stukjes code uit te voeren in de cloud zonder de noodzaak van serverbeheer. Het biedt schaalbaarheid en kostenbesparing.

5. Blob storage (Azure)

Blob Storage is in het leven geroepen om tijdelijk geuploade bestanden van gebruikers op te slaan. Het heeft een ingebouwd mechanisme dat het na 1 uur de bestanden weer opruimt uit het systeem om kosten te besparen. Dit is 1 van de business doelstelling van IBS Toeslagen.

Interactiepatroon:



Afbeelding 3.5 interactiepatroon

3.6 Beveiligingsmaatregelen

Hieronder volgt een uitgebreide beschrijving van de beveiligingsmaatregelen die zijn geïmplementeerd om te voldoen aan de hoogste standaarden van software engineering, met specifieke aandacht voor beveiliging en gebruikersgemak binnen de context van IBS Toeslagen.

3.6.1 End-to-End Encryptie voor Datatransport

End-to-end encryptie zorgt ervoor dat gegevens tijdens het transport tussen de verschillende componenten van de applicatie versleuteld zijn. Dit betekent dat gegevens niet leesbaar zijn voor onbevoegde partijen die mogelijk toegang krijgen tot de gegevens tijdens het transport.

Technologische Implementatie:

- **TLS (Transport Layer Security):** Alle communicatie tussen de frontend, backend, en externe diensten zoals Pinecone en OpenAI wordt beveiligd met TLS. Dit protocol zorgt voor een veilige verbinding door gegevens te versleutelen voordat ze worden verzonden en te ontsleutelen zodra ze zijn ontvangen.
- **HTTPS:** Alle API-aanroepen maken gebruik van HTTPS om ervoor te zorgen dat de gegevens tijdens het transport versleuteld zijn.

End-to-end encryptie beschermt de gegevens van gebruikers tegen afluisteren en manipulatie tijdens het transport. Dit verhoogt de vertrouwelijkheid en integriteit van de gegevens.

3.6.2 Tijdelijke Documentopslag (1-uurs Retentie)

Documenten die door gebruikers worden geüpload, worden tijdelijk opgeslagen in Azure Blob Storage met een retentieperiode van één uur. Dit zorgt ervoor dat documenten alleen worden bewaard zolang als nodig is voor verwerking en daarna automatisch worden verwijderd. Dit geeft de Azure functie de tijd om de content op te pakken en verder te verwerken.

Technologische Implementatie:

- **Azure Blob Storage:** Documenten worden geüpload naar een beveiligde container in Azure Blob Storage. De opslag is geconfigureerd om documenten na één uur automatisch te verwijderen.
- **Versleuteling:** Tijdens de opslagperiode worden documenten versleuteld met industriestandaard encryptieprotocollen zoals TLS voor transport en AES-256 voor opslag.

De tijdelijke opslag van documenten minimaliseert het risico van langdurige blootstelling van gevoelige gegevens.

3.6.3 Lokale Chatgeschiedenis Opslag

De chatgeschiedenis van gebruikers wordt lokaal opgeslagen in de browser's localStorage. Dit zorgt ervoor dat gebruikers toegang hebben tot hun eerdere gesprekken zonder dat de gegevens naar een externe server worden verzonden.

Technologische Implementatie:

- **localStorage:** De chatgeschiedenis wordt opgeslagen in de localStorage van de browser, wat betekent dat de gegevens alleen toegankelijk zijn op het apparaat van de gebruiker.
- **JavaScript Interop:** De opslag en het ophalen van de chatgeschiedenis worden beheerd met behulp van JavaScript interop (IJSRuntime) in Blazor.

Lokale opslag van de chatgeschiedenis verhoogt het gebruikersgemak. Dit omdat gebruikers geen onnodige en overbodige calls moeten doen naar de server. Ook legt dit om die reden minder belasting op de servers, wat kostenreductie stimuleert.

3.6.4 Dummy data

Voor dit project is er gekozen, na het interview met de security specialist, om dummy data te gebruiken. Dit besluit is genomen om de veiligheid en privacy van echte gebruikersgegevens te waarborgen tijdens de ontwikkelings- en testfase van de applicatie.

Technologische Implementatie:

- **Dummy Data:** In plaats van echte gebruikersgegevens worden gesimuleerde gegevens gebruikt die geen gevoelige informatie bevatten. Deze dummy data wordt gebruikt voor het testen van de functionaliteiten van de applicatie zonder risico op datalekken.
- **Data Masking:** Waar nodig worden technieken voor data masking toegepast om ervoor te zorgen dat de dummy data realistisch genoeg is voor testdoeleinden, maar geen echte gevoelige informatie bevat.

Het gebruik van dummy data minimaliseert het risico op datalekken en zorgt ervoor dat de ontwikkelings- en testprocessen veilig kunnen worden uitgevoerd.

4. Installatiehandleiding

Deze handleiding biedt een uitgebreide uitleg over hoe je de "IBS Toeslagen Chat App" applicatie lokaal kunt opzetten en uitvoeren. De beschreven stappen zijn bedoeld om een gebruiksvriendelijke ervaring te bieden voor zowel beginners als ervaren ontwikkelaars. Het document is zorgvuldig samengesteld om ervoor te zorgen dat alle essentiële aspecten van de installatie worden behandeld. Voor een overzichtelijke versie van deze handleiding kun je ook terecht op de officiële GitHub-repository: https://github.com/yannicklansink/central-search-engine-toeslagen/blob/main/how-to-run-locally.md

Voorbereidende stap

Clone de repository **o**m toegang te krijgen tot de broncode en de projectbestanden:

```
git clone https://github.com/yannicklansink/central-search-engine-toeslagen
```

Installeer vereiste software:

- Python 3.11 of hoger
- Poetry: Voor dependency management en virtual environments.
- .NET SDK: Voor het draaien van de Blazor WebAssembly applicatie.

Download de nieuwste versies van de bovengenoemde software vanaf hun officiële websites en volg de installatie-instructies voor jouw besturingssysteem.

Stap 1: Configureer de Omgevingsvariabelen

Maak een .env bestand:

1. Kopieer de inhoud van env.example naar een nieuw bestand genaamd .env.

   cp env.example .env

2. Vul de juiste API-sleutels in voor Pinecone, OpenAI en LangChain Tracing.

   PINECONE_API_KEY=<your-pinecone-api-key>

   PINECONE_INDEX_NAME=langchain-unstructured-data

   OPENAI_API_KEY=<your-openai-api-key>

   LANGCHAIN_TRACING_V2=true

   LANGCHAIN_ENDPOINT=https://api.smith.langchain.com

   LANGCHAIN_API_KEY=<your-langchain-api-key>

Stap 2: Start de Backend (RAG-service)

1. Navigeer naar de RAG-service directory:

cd RAG-service

2. Installeer de afhankelijkheden met Poetry:

   poetry install

3. Start de RAG-service:

   poetry run langchain serve

Stap 3: Start de Frontend (Blazor WebAssembly)

1. Navigeer naar de client/WebApp directory:

   cd client/WebApp

2. Start de Blazor WebAssembly applicatie:

   dotnet run

Opmerkingen

Upload Functionaliteit: Om gebruik te maken van de upload functionaliteit, moet je verbinding maken met het Azure account van yannick.lansink@live.nl. Dit is momenteel niet mogelijk aangezien het een persoonlijk account is. De rest van de functionaliteit zijn wel mogelijk om na te bootsen.

Samenvatting

1. Clone de repository en installeer vereiste software.
2. Configureer de omgevingsvariabelen in een .env bestand.
3. Start de backend door naar de RAG-service directory te navigeren en poetry run langchain serve uit te voeren.
4. Start de frontend door naar de client/WebApp directory te navigeren en dotnet run uit te voeren.

Met deze stappen zou je de applicatie lokaal moeten kunnen draaien.

5. Belangrijke functionaliteiten

De applicatie is opgebouwd uit 5 fundamentele kernfunctionaliteiten die gezamenlijk een geavanceerd Retrieval-Augmented Generation (RAG) systeem vormen. Deze 5 worden in dit hoofdstuk besproken.

5.1 Gedetailleerde Analyse van Kernfunctionaliteiten

5.1.1 Document Management en Intelligente Opslag

Technische Implementatie:

- Azure Blob Storage integratie voor veilige documentretentie
- Automatische document analyse pipeline
- Tijdelijk opslagmechanisme met 1-uurs retentiebeleid
- Industriestandaard TLS-encryptie voor data in transit

Gebruikersbehoefte: "Hoe kan ik waarborgen dat mijn documenten veilig worden opgeslagen en efficiënt kunnen worden teruggevonden?"

5.1.2 Real-time Query Processing Systeem

Architectuur:

- FastAPI backend implementatie
- Asynchrone streaming responses
- OpenAI-integratie voor semantische verwerking
- Vector database query optimalisatie

Functionaliteit: Het systeem verwerkt gebruikersverzoeken via geoptimaliseerde API-endpoints die direct relevante informatie extraheren uit de vector database en deze verrijken met contextuele analyse.

5.1.3 Geavanceerde Semantische Zoekfunctionaliteit

Technische Componenten:

- Vector-gebaseerde zoekarchitectuur
- Semantische matching algoritmes
- Pinecone vector database integratie
- Cosine similarity berekeningen voor nauwkeurige resultaten

Implementatie Details: De zoekfunctionaliteit maakt gebruik van geavanceerde vector embeddings om semantische relaties tussen documenten te identificeren, waardoor gebruikers relevante informatie kunnen vinden zonder exacte trefwoorden te kennen.

5.1.4 Automatische Document Analyse met AI

Procesflow:

- Document upload trigger activatie
- Conversie naar embeddings via OpenAI

- Vector opslag in Pinecone
- Automatische metadata extractie
- Semantische indexering

Privacy en Security:

- End-to-end encryptie
- Lokale data retentie

5.1.5 Gepersonaliseerde Chat Interface

Technische Features:

- Browser-based localStorage implementatie
- Real-time synchronisatie
- Persistente chat historie
- Gebruikersspecifieke contextuele verwerking

Deze architectuur stelt IBS Toeslagen in staat om efficiënt informatie te ontsluiten en te analyseren, waarbij geavanceerde AI-technologieën worden gecombineerd met robuuste beveiligingsmaatregelen.

Bronvermelding

Yannicklansink. (z.d.). GitHub - yannicklansink/central-search-engine-toeslagen. GitHub. https://github.com/yannicklansink/central-search-engine-toeslagen/tree/main

Jansen, J. (2023). Lange termijn plannen IBS Toeslagen. IBS Toeslagen.

ICT bij de Belastingdienst - Werken bij de Belastingdienst | Werken bij de Belastingdienst. (z.d.). Werken Bij de Belastingdienst. https://werken.belastingdienst.nl/expertises/ict

Lansink, Y. J. (2023). Plan van Aanpak.