



**NOVA**  
NOVA SCHOOL OF  
SCIENCE & TECHNOLOGY

DEPARTMENT OF  
COMPUTER SCIENCE

**RODRIGO DANIEL SAPATEIRO ALEGRIA**

Bsc in Computer Science

# AI CONVERSATIONAL AGENT TO SOLVE MULTILINGUAL ADMINISTRATIVE QUESTIONS

MASTER IN COMPUTER SCIENCE AND ENGINEERING

NOVA University Lisbon  
September, 2024



# AI CONVERSATIONAL AGENT TO SOLVE MULTILINGUAL ADMINISTRATIVE QUESTIONS

**RODRIGO DANIEL SAPATEIRO ALEGRIA**

Bsc in Computer Science

**Adviser:** Felix Aweh

*Senior Software Engineer, LinkThings*

**Co-adviser:** Artur Miguel Dias

*Assistant Professor, NOVA University Lisbon*

## Examination Committee

**Chair:** Dr. João Ricardo Viegas da Costa Seco

*Associate Professor, FCT-NOVA*

**Rapporteur:** Dr. Jörg Matthias Knorr

*Associate Professor, FCT-NOVA*

**Adviser:** Eng. Felix Aweh

*Software Engineer, LinkThings*

## **AI Conversational Agent to solve multilingual administrative questions**

Copyright © Rodrigo Daniel Sapateiro Alegria, NOVA School of Science and Technology, NOVA University Lisbon.

The NOVA School of Science and Technology and the NOVA University Lisbon have the right, perpetual and without geographical boundaries, to file and publish this dissertation through printed copies reproduced on paper or on digital form, or by any other means known or that may be invented, and to disseminate through scientific repositories and admit its copying and distribution for non-commercial, educational or research purposes, as long as credit is given to the author and editor.

To my parents. Thanks for keeping the interest rates low on  
everything I owe you.

## ACKNOWLEDGEMENTS

First and foremost, I would like to express my sincere gratitude to my advisor, Felix Aweh, for his support, valuable feedback, and guidance throughout this thesis. Despite the physical distance between us, your readiness and trust have been invaluable. I also want to thank Yuri and Diogo for welcoming me to the Portuguese part of this incredible company.

I would also like to extend my thanks to Maurits and Jin for providing me with this project at LinkThings. The environment you fostered was incredibly supportive, with a forward-thinking vision that greatly contributed to my work. Hope we can continue our partnership for many more projects.

I am deeply grateful to Liang Wu for your boundless support, patience, and the wealth of knowledge you have shared with me. Your never-ending availability and the freedom you gave me to explore and grow have been a cornerstone of my journey, and I am truly appreciative.

My heartfelt thanks go to my co-advisor, Professor Artur Miguel Dias. Your readiness to assist me in writing this document and offering critical advice has been indispensable. Without your guidance, I would have felt somewhat lost in this process.

I am also thankful to all my friends from the university, Alex, Boião, Braga, Caldeirão, Daniel, Xico, Gonçalo, Guilherme, João, Márcia, Pedro, Tiago, for the unforgettable moments we shared like the countless hours spent on projects, studying, in the nights out, and the celebrations. These experiences have made this journey enjoyable.

On a more personal note, I would like to thank my friends for all the time we spent together when I wasn't thinking about this project. A special thank you goes to Ana Sofia, Rita, Beatriz, Bento, Catarina, Coelho, Gordicho, Gomes, Joana, Marcos, Ramalho, for their understanding and support during the times I had to prioritize my work over socializing.

I would like to extend my deepest gratitude to my parents, Fidélío and Deolinda. Your unwavering support, love, and encouragement have been the foundation upon which this thesis was built. From a young age, you instilled in me the importance of perseverance. Your sacrifices, belief in my abilities, and constant motivation have been my guiding light

throughout this academic journey. I am forever indebted to you for everything you have done to help me reach this milestone.

Finally, to J  ssica, words cannot express how grateful I am for your endless patience, understanding, and support during this challenging time. Your encouragement and faith in me have been a constant source of motivation, and I am incredibly lucky to have you by my side. Thank you for always being there, for celebrating my successes, and for lifting me up during the tougher moments.

”

*“When you’re backed against the wall, break the  
goddamn thing down”*

— **Harvey Specter**, *Suits*

## ABSTRACT

This dissertation presents an AI solution designed to address the challenges faced by temporary employment agencies managing a diverse, multilingual workforce. The study focuses on leveraging an LLM model to enable accurate answers to frequently asked questions in multiple languages.

The project objectives include developing and implementing the LLM-Based Conversational Agent, improving the prompts, and evaluating its performance in providing correct answers to three queries. The results should demonstrate a 95% correctness rate, showcasing the efficacy of streamlining administrative processes.

This project contributes to the field of LLM-Based Conversational Agents by offering a practical solution to the language barriers in temporary employment agencies, emphasizing the potential for improved efficiency and reduced back-office support overhead. The findings underscore the significance of AI-driven solutions in addressing complex administrative challenges in multilingual environments.

**Keywords:** Artificial Intelligence, Administrative Questions, Multilingual, Chat bot, Large Language Model

## RESUMO

Esta dissertação apresenta uma solução de Inteligência Artificial desenhada para enfrentar os desafios enfrentados por agências de trabalho temporário ao gerir uma força de trabalho diversificada e multilíngue. O estudo foca-se em aproveitar um modelo de LLM para possibilitar o fornecimento de respostas para perguntas frequentes em várias línguas.

Os objetivos do projeto incluem desenvolver e implementar o Agente Conversacional, melhorar os *prompts* fornecidos ao modelo e avaliar seu desempenho ao fornecer respostas corretas para três perguntas. Os resultados devem demonstrar uma taxa de precisão de 95%, evidenciando a eficácia na automatização de processos administrativos.

Este projeto contribui para a área de Agentes Conversacionais baseados em modelos de linguagem de grande escala ao oferecer uma solução prática para as barreiras linguísticas em agências de trabalho temporário, realçando o potencial que existe para melhorar a eficiência e reduzir o apoio administrativo por membros dos Recursos Humanos das empresas. Esta solução sublinha a importância de soluções impulsionadas por IA para enfrentar desafios administrativos complexos em ambientes multilíngues.

**Palavras-chave:** Inteligência Artificial, Questões Administrativas, Multilíngue, Agente Conversacional, Modelo de linguagem de grande escala

# CONTENTS

<b>List of Figures</b>	<b>xii</b>
<b>Acronyms</b>	<b>xv</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Background and Motivation . . . . .	1
1.2 Problem Statement . . . . .	2
1.2.1 Communication Barriers . . . . .	2
1.2.2 Task Delegation Challenges . . . . .	2
1.2.3 Operational Inefficiencies . . . . .	2
1.2.4 Employee Engagement and Satisfaction . . . . .	2
1.2.5 Compliance and Documentation Issues . . . . .	3
1.3 Solution . . . . .	3
1.3.1 LLM-Based Conversational Agent . . . . .	3
1.3.2 Information Retrieval Module . . . . .	4
1.3.3 Back-Office Portal . . . . .	4
1.3.4 Data Privacy and Security Framework . . . . .	4
1.3.5 WhatsApp WebHook and Messages Processors . . . . .	5
1.4 Contributions . . . . .	5
1.5 Document Structure . . . . .	5
1.6 LinkThings . . . . .	7
<b>2 Background</b>	<b>8</b>
2.1 Introduction to Artificial Intelligence . . . . .	8
2.2 Ethical Considerations and Challenges of AI Systems . . . . .	9
2.3 Generative Artificial Intelligence . . . . .	10
2.4 Natural Language Processing . . . . .	10
2.5 Large Language Models . . . . .	12
2.5.1 Best LLM for the use case . . . . .	13
2.6 Word Embeddings . . . . .	16

2.7	Prompt Engineering	16
2.8	Conversational Agent	17
2.9	Machine Translation	18
2.9.1	Neural Machine Translation	18
2.10	RAG-Based Information Retrieval	19
2.11	Azure AI Studio	20
<b>3</b>	<b>Concrete Systems and Technological Solutions</b>	<b>22</b>
3.1	Microsoft Azure	23
3.1.1	Azure SQL Database	25
3.1.2	Azure Machine Learning Prompt Flow	25
3.1.3	Function App	26
3.2	Semantic Kernel	26
3.3	Django	27
3.4	GraphQL	28
3.5	Vue.js	28
3.6	WhatsApp Business API	29
<b>4</b>	<b>Related Work</b>	<b>30</b>
4.1	Conversational Health Agents	30
4.2	AI-Based Chatbot to Solve Modern-Day Enterprise Business Problems	31
4.3	Automating Customer Service using LangChain	32
<b>5</b>	<b>Solution</b>	<b>33</b>
5.1	Architecture and Flow	33
5.1.1	Database	33
5.1.2	Information Retrieval Module	36
5.1.3	WhatsApp WebHook	36
5.1.4	Incoming Messages Processor	36
5.1.5	AI Engine	37
5.1.6	Back-Office Portal	38
5.1.7	Outgoing Messages Processor	38
5.1.8	Design Choices	38
5.2	WhatsApp Interface	40
5.3	Back-Office Portal	41
5.3.1	Conversations	41
5.3.2	User and Company Management	41
5.3.3	Settings and Profile Management	42
5.4	Available Questions	44
5.5	Alternatives	44
5.5.1	Langchain	45
5.5.2	Ollama	45

5.5.3	Web Apps instead of Function Apps for the Message Processors .	46
<b>6</b>	<b>Implementation</b>	<b>47</b>
6.1	Environment Setup . . . . .	47
6.1.1	Back-end Setup . . . . .	47
6.1.2	Frontend Setup . . . . .	48
6.1.3	Local Environment for PromptFlow and Semantic Kernel . . . . .	48
6.1.4	Code Repositories . . . . .	48
6.1.5	Azure DevOps Board . . . . .	48
6.1.6	Azure Functions Apps and Event Hubs . . . . .	49
6.2	Database . . . . .	49
6.3	Information Retrieval Module . . . . .	51
6.4	WhatsApp WebHook . . . . .	52
6.5	Incoming Messages Processor . . . . .	54
6.5.1	Text Messages . . . . .	54
6.5.2	Interactive Messages . . . . .	55
6.5.3	Final Steps and Logging . . . . .	56
6.6	AI Engine . . . . .	58
6.6.1	API_KEY Validation . . . . .	59
6.6.2	Loading Deployment Name . . . . .	59
6.6.3	Formulating the Planner Prompt . . . . .	59
6.6.4	Finding and Executing the plan . . . . .	59
6.6.5	Final Prompt and Response Generation . . . . .	62
6.6.6	PromptFlow Tracing . . . . .	62
6.7	Outgoing Messages Processor . . . . .	64
6.8	Back-Office Portal . . . . .	66
6.8.1	Requirements . . . . .	66
6.8.2	Backend . . . . .	66
6.8.3	Frontend . . . . .	69
6.9	Deployment . . . . .	73
6.9.1	Deployment Location . . . . .	73
6.9.2	CI/CD Pipeline . . . . .	73
6.9.3	Docker Compose for the Back-Office Portal . . . . .	74
6.9.4	Function Apps . . . . .	74
6.9.5	WhatsApp Business API . . . . .	75
6.10	Maintenance and Support . . . . .	75
6.10.1	Error Alerts via Email . . . . .	75
6.10.2	Automated Health Checks . . . . .	76
6.10.3	Logging and Monitoring . . . . .	76
6.10.4	Performance Metrics . . . . .	77
6.10.5	Sequential Releases . . . . .	77

6.10.6 Automated Backups . . . . .	78
6.10.7 Communication and Coordination . . . . .	78
6.11 Challenges . . . . .	78
<b>7 Evaluation</b>	<b>81</b>
7.1 Criteria and Metrics . . . . .	81
7.2 System Specifications . . . . .	81
7.3 System Limitations . . . . .	82
7.3.1 Supported Query Types . . . . .	82
7.3.2 Response Time and Latency . . . . .	82
7.4 Response Time . . . . .	83
7.5 System Load and Resources Usage . . . . .	84
7.6 Accuracy . . . . .	86
7.6.1 Detailed Analysis . . . . .	86
7.7 User Testing . . . . .	88
<b>8 Future Work</b>	<b>90</b>
8.1 Handling Audio Messages . . . . .	90
8.2 Handling and Returning Files . . . . .	90
8.3 More actions . . . . .	91
8.4 Flow for Sensitive Questions . . . . .	91
8.5 Better Support Platform . . . . .	91
<b>Bibliography</b>	<b>92</b>

## LIST OF FIGURES

2.1	Named Entity Recognition Tagging . . . . .	11
2.2	LMSYS Chatbot Arena Leaderboard . . . . .	14
2.3	AlpacaEval LeaderBoard . . . . .	15
2.4	RAG process applied to question answering . . . . .	20
2.5	Azure AI Studio . . . . .	21
3.1	Greenie . . . . .	22
3.2	ColdCha . . . . .	23
4.1	LLM-powered framework leveraging a service-based architecture . . . . .	31
5.1	System Flow . . . . .	34
5.2	Database Schema . . . . .	35
5.3	WhatsApp Flow for user verification . . . . .	37
5.4	Prompt Flow Design . . . . .	38
5.5	WhatsApp Flow . . . . .	40
5.6	Conversations Table . . . . .	42
5.7	Conversation Detail . . . . .	42
5.8	Application Users Table . . . . .	43
5.9	Companies Table . . . . .	43
5.10	Settings and Profile Management . . . . .	44
6.1	Azure DevOps Repository . . . . .	48
6.2	Azure DevOps Planning Board . . . . .	49
6.3	Meta Developers portal with Verification Flow . . . . .	56
6.4	Prompt Flow Tracing . . . . .	64
6.5	Approve/Reject Answers . . . . .	71
6.6	Review AI Answers . . . . .	72
6.7	Email Alert . . . . .	76
6.8	Azure Email Alert . . . . .	77

7.1	Chart relating response time (duration) to number of tokens . . . . .	83
7.2	Graph with CPU Usage . . . . .	85
7.3	Graph with Memory Usage . . . . .	85

## LIST OF LISTINGS

1	Company Model . . . . .	50
2	Overridden Delete Function . . . . .	51
3	Function get_all_employee_current_contracts . . . . .	53
4	Webhook handler . . . . .	54
5	Send Message Request Body . . . . .	57
6	Outgoing Messages Processor Input Object . . . . .	57
7	Planner Prompt . . . . .	59
8	P4F Day Work Planning Output . . . . .	60
9	Wage Year Week Salary Output . . . . .	61
10	Final Prompt to generate answer . . . . .	63
11	WhatsApp Text Messages Sender . . . . .	65
12	WhatsApp Flow Messages Sender . . . . .	65
13	AppUser Node . . . . .	67
14	AppUser Filter . . . . .	68
15	Update App User Block Status Mutation . . . . .	69

## ACRONYMS

<b>AI</b>	Artificial Intelligence ( <i>p. 1</i> )
<b>API</b>	Application Programming Interface ( <i>pp. 5, 6, 23, 28, 29, 37</i> )
<b>BERT</b>	Bidirectional Encoder Representations from Transformers ( <i>p. 16</i> )
<b>CD</b>	Continuous Deployment ( <i>p. 73</i> )
<b>CI</b>	Continuous Integration ( <i>p. 73</i> )
<b>CLI</b>	Command Line Interface ( <i>p. 48</i> )
<b>CPU</b>	Central Processing Unit ( <i>pp. 82, 84–86</i> )
<b>GDP</b>	Gross Domestic Product ( <i>p. 10</i> )
<b>GDPR</b>	General Data Protection Regulation ( <i>p. 73</i> )
<b>HR</b>	Human Resources ( <i>pp. 31, 39, 41, 43, 66</i> )
<b>HTTP</b>	Hypertext Transfer Protocol ( <i>pp. 26, 51</i> )
<b>HTTPS</b>	Hypertext Transfer Protocol Secure ( <i>p. 75</i> )
<b>IaaS</b>	Infrastructure as a Service ( <i>p. 24</i> )
<b>IRM</b>	Information Retrieval Module ( <i>p. 51</i> )
<b>LLM</b>	Large Language Model ( <i>pp. 3, 5, 10, 14, 16, 21, 30, 37</i> )
<b>NER</b>	Named Entity Recognition ( <i>p. 11</i> )
<b>NLP</b>	Natural Language Processing ( <i>pp. 10–12, 16, 17</i> )
<b>NMT</b>	Neural Machine Translation ( <i>p. 18</i> )
<b>P4F</b>	Plan4Flex ( <i>p. 52</i> )
<b>PaaS</b>	Platform as a Service ( <i>p. 24</i> )

<b>RAG</b>	Retrieval Augmented Generation ( <i>pp. 6, 17, 19, 20, 25, 26</i> )
<b>RLHF</b>	Reinforcement Learning from Human Feedback ( <i>p. 13</i> )
<b>SaaS</b>	Software as a Service ( <i>p. 24</i> )
<b>SLA</b>	Service Level Agreement ( <i>p. 82</i> )
<b>SQL</b>	Structured Query Language ( <i>pp. 24, 25</i> )
<b>VM</b>	Virtual Machine ( <i>p. 24</i> )
<b>VS</b>	Visual Studio ( <i>p. 74</i> )

# INTRODUCTION

## 1.1 Background and Motivation

The global workforce is undergoing significant transformations, and one of the key contributors to this change is the rise of temporary agency employment. In the Netherlands, as illustrated by the data from Statista [56], approximately 5.2 percent of the workers were engaged in temporary agency employment in 2022. This statistic places the Netherlands above the European Union average in terms of the proportion of temporary workers in the overall workforce of the country. In Portugal, the utilization of temporary employment agencies is also a growing trend, significantly influenced by migration flows. The National Statistics Institute (INE) reported that in 2022 [29], these agencies facilitated 1,629,883 services, showcasing the critical role they play in the labor market. This number is expected to rise as migrations to Portugal increase, highlighting the country's appeal as a destination for workers and the demand for flexible employment solutions.

The dynamic nature of temporary agency employment brings forth a unique set of challenges, particularly when it comes to managing a workforce with diverse linguistic backgrounds. Temporary employment agencies often grapple with the complexities of communication and coordination when dealing with employees who speak multiple languages. This poses operational challenges, ranging from effective task delegation to ensuring a cohesive work environment.

The most common solution to this issue is employing translators who can deal with all the languages spoken by the workers but this remedy becomes very expensive. Companies do not want to be hostages in this situation and are searching for an adaptive solution that can end this issue.

Against this backdrop, the exploration of an AI-driven solution becomes imperative. Artificial Intelligence has emerged as a transformative force across various industries, and its potential to address the challenges faced by temporary employment agencies in managing multilingual workforces is both promising and timely.

By leveraging advanced language-processing capabilities, AI-driven solutions can enhance communication, streamline task allocation, and foster a more inclusive and

efficient work environment.

## **1.2 Problem Statement**

In the contemporary landscape of employment, temporary employment agencies play a crucial role in facilitating flexible labor solutions for organizations. However, a pervasive challenge faced by these agencies is the effective management of a workforce characterized by language diversity. This problem is particularly pronounced in back-office activities, where seamless communication and efficient task execution are paramount for operational success.

The impact of language diversity on administrative activities is multifaceted and poses significant challenges to the overall effectiveness of temporary employment agency operations. The primary issues include [3, 46]:

### **1.2.1 Communication Barriers**

The diverse linguistic backgrounds of temporarily employed workers can lead to communication barriers in the back-office. Misinterpretations, misunderstandings, and miscommunications may occur, affecting the smooth flow of critical information to daily operations.

### **1.2.2 Task Delegation Challenges**

One of the biggest issues appears when delegating tasks. It involves activities that require clear and precise instructions. Language diversity can impede the effective delegation of tasks, leading to confusion among workers and potentially compromising the quality and punctuality of deliverables.

### **1.2.3 Operational Inefficiencies**

The presence of multiple languages within the workforce can result in operational inefficiencies, as additional time and resources may be required to navigate language-specific challenges. This can lead to delays in processing, reduced productivity, and increased probability of errors.

### **1.2.4 Employee Engagement and Satisfaction**

Language barriers can contribute to a sense of isolation among workers, affecting their engagement and satisfaction levels. A cohesive work environment is essential for employee well-being and can directly impact retention rates.

### 1.2.5 Compliance and Documentation Issues

Back-office activities often involve handling sensitive information and ensuring compliance with regulations. Language diversity may complicate documentation processes, increasing the risk of errors and potential compliance issues.

The identified problem of language diversity in administrative activities is not only a localized challenge but also has broader implications for the overall efficiency and competitiveness of temporary employment agencies. As the global workforce continues to diversify, addressing this issue becomes imperative for agencies aiming to remain agile and responsive to the evolving needs of the labor market.

## 1.3 Solution

The project outlines specific objectives to ensure the successful development, implementation, and evaluation of the AI-driven solution designed to address language diversity challenges in temporary agency employment. These objectives serve as a road map, guiding the research toward achieving its overarching goals.

By orchestrating the integration of these components, this solution provides a comprehensive strategy. This not only conquers language barriers but also significantly elevates the efficiency and effectiveness of administrative processes within temporary employment agencies.

At its core, the solution leverages a Large Language Model **LLM**-based Conversational Agent, capable of handling multilingual queries. This ensures smooth communication between employees and back-office staff, regardless of language barriers.

To further enhance efficiency, the system includes real-time information retrieval, ensuring accurate and up-to-date responses. A Back-Office Portal allows administrators to oversee the interactions and make necessary adjustments to improve conversations and operations.

Data privacy and security are prioritized, ensuring that all sensitive information is handled securely and in compliance with regulations. Additionally, by integrating with WhatsApp, the solution provides a familiar communication platform for workers, making it easy for them to access information and engage with the system.

### 1.3.1 LLM-Based Conversational Agent

At the heart of this solution is an **LLM**-based Conversational Agent. This is done using Prompt Flow and Semantic Kernel and is meticulously configured to comprehend and respond to multi-language queries, exhibiting the capability to provide precise and contextually relevant answers. By harnessing the latest advancements in natural language processing, the Conversational Agent aims to dismantle language barriers, facilitating seamless communication across the diverse workforce within temporary employment agencies.

### 1.3.2 Information Retrieval Module

In tandem with the Conversational Agent, an Information Retrieval Module was implemented. This module serves as an intelligent mechanism to augment the agent's ability to access and use pertinent, up-to-date information. The Information Retrieval Module ensures that the agent remains adaptive and well-informed, providing accurate responses in real-time, through direct connection to the database.

### 1.3.3 Back-Office Portal

A pivotal element of this solution was the development of a user-friendly Back-Office Portal. Constructed using the most popular frameworks for web development, this portal provides an intuitive platform for seamless interaction with the AI agent. This component is used by back-office users to manage the application and evaluate its performance.

### 1.3.4 Data Privacy and Security Framework

Recognizing the critical importance of data privacy and security, this solution incorporates a robust framework to safeguard sensitive information. It ensures that the chatbot operates with the utmost integrity and compliance with global privacy regulations, such as GDPR. This framework integrates several key components:

- **Azure Security Services**

Azure provides a range of security services, including Azure Security Center, Azure Active Directory, and Azure Key Vault, which are leveraged to secure applications, data, and identities. These services support various compliance standards.

- **Encryption**

All data handled by the system is encrypted both in transit and at rest. HTTPS protocols are employed to ensure secure communication, and Azure's encryption standards further protect data while stored in Azure SQL Databases and other services.

- **Authentication and Authorization**

The application strictly enforces authentication for all users, meaning that no user can access the application or its features without proper credentials. This guarantees that only authorized users can interact with the system, enhancing security and privacy.

- **User Data Protection**

A fundamental principle of the application is ensuring that no user's data is visible to others. Data isolation mechanisms are in place so that each user's information remains entirely separate, preventing accidental or malicious access to another user's data.

This ensures that sensitive user data remains secure, compliant, and protected, reflecting the solution's commitment to high privacy and security standards.

### 1.3.5 WhatsApp WebHook and Messages Processors

To enhance communication capabilities and make it more convenient to the end user, this solution is integrated with the WhatsApp Business [API](#). This feature enables temporary workers to send and receive messages directly through WhatsApp, allowing for a more direct and accessible line of communication. The webhook acts as a bridge between the AI system and the popular messaging platform, ensuring timely updates and improving engagement and response times.

## 1.4 Contributions

This project explores how AI conversational agents, specifically those based on Large Language Models (LLMs), can be tailored to solve multilingual administrative questions. This approach is not just about bridging language barriers, it is about creating a more inclusive and accessible way for people to interact with technology, no matter what language they speak, and also to innovate the way human resources operate. This is done with the help of a friendly user interface which I believe is crucial for wider adoption.

Data security and privacy are also relevant in this project. In an era where data breaches are all too common, I am focused on ensuring that sensitive information is handled with the utmost care.

The practical application of this technology in administrative settings is a big part of the dissertation. I am showing how these AI agents can streamline processes, reduce workloads, and provide accurate information efficiently. This is not just a theoretical exercise, it has real-world implications that could transform how administrative tasks are managed.

Overall, this dissertation is not just contributing to the field of [LLM-Based Conversational Agents](#) by orchestrating innovative tools together, it is using the newest standards in ethical practices and in the use of Artificial Intelligence. I ensured that this agent operates fairly across different languages and cultures.

## 1.5 Document Structure

This section provides an overview of the structure and layout of the dissertation, outlining each chapter and its contents.

- **Chapter 1: Introduction**

This chapter sets the stage for the project, providing the background, motivation, problem statement, and proposed solutions. It also outlines the contributions of the dissertation to the field.

- **Chapter 2: Background**

This chapter delves into the theoretical underpinnings of the project, covering areas such as Artificial Intelligence, ethical considerations, generative AI, natural language processing, large language models, Semantic Kernel, word embeddings, conversational agents, machine translation, RAG-based information retrieval, and Prompt Engineering.

- **Chapter 3: Concrete Systems and Technological Solutions**

Here, it explores various technological solutions, systems, and tools relevant to the project, such as Microsoft Azure, Django, GraphQL, Vue.js, WhatsApp Business API, and others. The chapter focuses on how these technologies are utilized in the context of the dissertation.

- **Chapter 4: Related Work**

This chapter reviews existing studies related to conversational agents, highlighting how this work is positioned within the broader field. It includes discussions on conversational health agents, AI-based chatbots for business problems, and customer service automation using LangChain.

- **Chapter 5: Solution**

This chapter presents the proposed solution in detail. It contains an overview of the solution architecture and design principles, followed by a step-by-step explanation of the system components and their interactions. Also, some discussion is made to justify the decisions made.

- **Chapter 6: Implementation**

This chapter focuses on the practical aspects of bringing the proposed solution to life. The chapter provides a detailed walkthrough of the implementation stages, from initial setup and configuration to the integration of various components. Challenges encountered during the development process and the strategies used to overcome them are also discussed, offering insights into the practical considerations of implementing the solution.

- **Chapter 7: Evaluation**

In this chapter, the effectiveness and performance of the implemented solution are assessed. It outlines the evaluation criteria and metrics used to measure success. The chapter describes the testing environment and methodology, including both quantitative and qualitative approaches. Results from experiments and user studies are presented, accompanied by analysis and interpretation.

- **Chapter 8: Future Work**

The final chapter explores potential directions for future development based on the findings and limitations identified in the realization of the project.

This structure is designed to guide the reader logically through the dissertation, from introducing the topic and its significance, through the theoretical background, tools that were used, a review of related work, a broad view of the solution, and then detailed implementation, followed by evaluation and finally ideas for future work.

## 1.6 LinkThings

This project is done in collaboration with [LinkThings](#), a Dutch company involved in software development and business analytics, particularly in building tools for industries that require complex data analysis, management, and automation solutions. LinkThings has developed applications like Greenie and ColdCha, which offer sophisticated solutions for managing agricultural operations and logistics, respectively. These applications leverage technology to streamline and improve efficiency in their respective domains by providing real-time data tracking and optimization tools.

In the context of this project, it fits within LinkThings' broader goal of creating efficient, technology-driven solutions. This project utilizes cutting-edge technologies, specifically Large Language Models and Semantic Kernel, integrated into a backend management system supported by frameworks like Django, GraphQL, and Vue.js. This agent streamlines administrative tasks, reduces operational inefficiencies, and improves communication in temporary employment agencies, complementing the company's other data-driven tools.

## BACKGROUND

### 2.1 Introduction to Artificial Intelligence

Artificial Intelligence, mostly known as AI, has emerged as a transformative technology that seeks to replicate or simulate human intelligence in machines. This field encompasses various techniques and applications, from rule-based systems to advanced machine-learning algorithms. As technology continues to evolve, AI plays a crucial role in reshaping industries, enhancing efficiency, and addressing complex challenges across various domains as the one focused on this project.

Defining AI can be nuanced, as it encompasses a multitude of approaches and methodologies but in its essence, it “is the science and engineering of making intelligent machines, especially intelligent computer programs. It is related to the similar task of using computers to understand human intelligence” [40]. The uses of Artificial Intelligence are vast ranging from translation to problem-solving [27], research, reasoning, understanding natural language, etc.

The roots of AI can be traced back to ancient times but the formal inception of AI as a scientific discipline occurred in the mid-20th century with the work of pioneers like Alan Turing and John McCarthy from Stanford University. These laid the foundation for AI research and development, which has since witnessed periods of rapid progress (despite some setbacks) and has now achieved a more highlighted role in tech. The global Artificial Intelligence market size was valued at 136.55 billion dollars in 2022 and is expected to reach 1,811.8 billion dollars by the year 2030, according to Forbes [38] and GrandViewResearch [51].

AI systems comprise several interconnected components, each contributing to the overall functionality. Machine Learning empowers systems to learn from data and improve performance over time. Natural Language Processing enables machines to understand and interact with human language (later we will take a deeper look at this theme), while computer vision facilitates the interpretation of visual information.

## 2.2 Ethical Considerations and Challenges of AI Systems

This rapid evolution of Artificial Intelligence has raised lots of concerns around society. I decided to have a section just focused on this subject to highlight its importance and that every engineer should always consider it in every small development he produces.

As these systems consume a lot of data, one of the major challenges is how to conserve people's privacy. Because of that, several measures are implemented in our day-to-day intelligence systems:

- **Data anonymization and hygiene**

Only the data types necessary to create the AI should be collected, and the data should be kept secure and only maintained for as long as necessary to accomplish the purpose. All personal identifiers are removed or replaced by synonyms.

- **Data Encryption**

Encrypting data during the transmission and while it is stored helps safeguard it from unauthorized access. Encryption ensures that even if data is intercepted, it remains unreadable without the proper decryption keys [15].

- **Transparent Algorithms**

Making AI algorithms more interpretable by users helps them understand how decisions are made without revealing sensitive information. This transparency is crucial for maintaining trust and ensuring accountability.

Biased AI is also one of the biggest concerns [59]. In the huge amount of data used to train the models, some data can contain embedded societal biases and because of that, there can be a presence of unfair and prejudiced outcomes in Artificial Intelligence systems. Bias in AI may result in discriminatory or inequitable treatment of certain individuals or groups. To fight this issue, engineers should always:

- **Use good datasets**

Developers should build AI using "accurate, fair, and representative" [41] data sets. To ensure this, they can build AI algorithms just to validate the quality of the datasets and of the algorithms so we are using AI to help improve AI.

- **Human Oversight**

As it can be hard for an Artificial Intelligence System to understand if it is being biased or not, human vigilance is one of the main ways to ensure fairness in the entire interaction. Although being a costly way to guarantee this, it is an important action to take.

Some other issues are related to job replacement [10, 60]. According to a report by the highly regarded investment bank Goldman Sachs [53], Artificial Intelligence could be responsible for replacing 300 million full-time jobs. Although these numbers are awesome

for the world economy, increasing the [GDP](#) by 16% by 2030, it is always a worry for people to lose their jobs. In contrast to this, various sources report that there will be a high demand for data scientists, machine learning specialists, and cybersecurity professionals so it is not putting 300 million people into unemployment but just a case to create more specialized workers [50].

Having said all this, during the implementation of this project, most of these measures were used to ensure the ethical validity of the [LLM](#)-based Conversational Agent.

### 2.3 Generative Artificial Intelligence

Generative AI is a subset of Artificial Intelligence that leverages machine learning techniques to generate human-like content. It's capable of creating new content that is similar in structure and theme to the input data it was trained on. Unlike traditional AI models that perform specific tasks based on predefined rules, generative models exhibit creativity by producing content that is not explicitly present in the training data.

Generative Artificial Intelligence models use neural networks to identify the patterns and structures within existing data to generate new and original content.

One of the breakthroughs with these models is the ability to leverage different learning approaches, including unsupervised or semi-supervised learning for training. This has given organizations the ability to more easily and quickly leverage a large amount of unlabeled data to create foundation models [66].

Examples of foundation models include [GPT-4](#), [GPT-4o](#), and [Meta's Llama](#) models, which allow users to leverage the power of language. For example, popular chatbots like [ChatGPT](#) and [Copilot Chat](#) give the power to users to generate text and images but other more specialized apps can produce sounds, animations, 3D models, and other types of data [69].

### 2.4 Natural Language Processing

Generative AI stands as a cornerstone in the broader realm of Artificial Intelligence, showcasing its powers in creating novel content. A notable intersection within this landscape is with Natural Language Processing ([NLP](#)), where generative models take center stage. The fusion of Generative AI and [NLP](#) has given rise to models capable of generating coherent and contextually rich text, revolutionizing how machines understand and produce human-like language.

[NLP](#) combines rule-based modeling of human language with statistical, machine learning, and deep learning models [28]. All these technologies, when together, enable machines to compute human languages like English or Portuguese and understand their full meaning.

NLP is used for a wide variety of language-related tasks, including answering questions, classifying text in a variety of ways, and chatting with users. Some of the most relevant tasks carried out by Natural Language Processing are [18]:

- **Sentiment analysis**

Sentiment analysis involves categorizing the emotional tone of the text, typically classifying it as positive, negative, or neutral. The input is a piece of text, and the output is the probability of the expressed sentiment.

- **Machine translation**

Machine translation automates the process of translating text between different languages, taking input in a source language and generating output in a target language. Notable applications include Google Translate. Successful machine translation models distinguish between words with similar meanings and may additionally perform language identification, classifying text into specific languages.

- **Named Entity Recognition**

Also known as **NER**, involves identifying and categorizing entities in text, such as personal names, organizations, locations, and quantities. The input is typically text, and the output consists of identified entities with their respective start and end positions.

Andrew Yan-Tak Ng **PERSON** ( **Chinese NORP** : **吳恩達 NORP**; born **1976 DATE** ) is a **British NORP** -born **American NORP** computer scientist and technology entrepreneur focusing on machine learning and **AI GPE** .  
 Ng was a co-founder and head of **Google Brain ORG** and was the former chief scientist at **Baidu ORG** ,  
 building the company's **Artificial Intelligence Group ORG** into a team of **several thousand CARDINAL** people.

Figure 2.1: “Named Entity Recognition [NER] Tagging”. From DeepLearning.AI. <https://www.deeplearning.ai/resources/natural-language-processing/> (accessed January 23, 2024).

- **Question answering**

Question answering is the process of finding the right solution to a certain problem. The input is a question typically in text and the result is an answer also in text. It is able to deal with different types of questions, for example, multiple choice and open answer.

- **Information retrieval**

Information retrieval addresses the challenge of finding the most relevant documents for a given query, a common task in search and recommendation systems. The objective is to retrieve a subset of documents, often from large collections, that best match the query.

- **Text Generation**

Also known as Natural Language Generation (NLG), produces human-like text and can be tailored for different genres like tweets, blogs, and even code. This is one of the most relevant features of [NLP](#) as it is the base of some of the most known AI applications like Chatbots and tools to auto-complete text.

## 2.5 Large Language Models

Large Language Models (LLMs) represent a breakthrough in natural language processing, empowered by extensive training on massive datasets. They excel in understanding and generating content. LLMs have gained prominence in both public and enterprise domains, reshaping how people adopt Artificial Intelligence across various functions. LLMs operate on a foundation model premise, eliminating the need to build and train domain-specific models for each use case individually, thus providing a cost-effective and synergistic approach [67].

LLMs function using a transformer architecture, introduced in the paper “Attention Is All You Need” by Vaswani et al. [64], a type of neural network particularly suited for processing sequential data. The core principle of transformers is the use of self-attention mechanisms, which allow the model to weigh the importance of different parts of the input data relative to each other. This is crucial for understanding the context and dependencies in language.

Neural Networks are computational models inspired by the human brain’s structure, designed to recognize patterns and solve complex problems. A neural network consists of layers of interconnected nodes (or neurons), each performing simple computations. Data input into the network passes through these layers, with each node processing and transforming the input [55]. Neural Network performance is severely impacted by the scale of the size of the model, the size of the training dataset, and the computational power used for training. The bigger these are, typically, the better the model performance. Through training, the network learns to make accurate predictions or decisions based on its input.

The process starts with the conversion of input text into numerical form, typically using word embeddings. Each word or token in the text is represented as a high-dimensional vector. These vectors capture semantic and syntactic information about the words, enabling the model to process and understand language in a mathematical framework.

During training, the transformer model [68] processes these vectors through multiple layers of neural networks. Each layer consists of self-attention mechanisms and fully connected neural networks. The self-attention mechanism assigns three vectors to each word in a sentence: a query, a key, and a value. These vectors are derived from the word’s embedding by multiplying it with different weight matrices, which the model learns during training. For each word, the self-attention mechanism calculates attention scores by taking the dot product of its query vector with the key vector of every other word in

the sequence. These scores determine how much attention or focus the model should give to other parts of the sentence when processing this particular word, allowing the model to weigh the importance of different parts of the input data relative to each other. This mechanism outputs a series of context-enriched word representations that are fed into subsequent layers of the neural network for further processing.

Training involves two key phases: pre-training and fine-tuning. Pre-training is foundational, that's where the model learns from a broad spectrum of data, more specifically text data. The goal of this training stage is for the model to understand the general patterns, structures, and details of the language. Tasks like Masked Language Modeling, where the model is presented with sentences with hidden words and learns to predict these missing pieces of the phrase, are central to this phase, where the model needs to understand context, grammar, syntax, and semantics on a broader and larger scale.

On the other hand, Fine-tuning is useful to tailor the model for specific tasks or domains. Here the model is presented with a dataset more focused on the theme and is trained with that data. This helps to adapt the broad knowledge gained in the first training phase.

To better improve the model, some advanced training strategies are also employed to enhance the model's performance and reliability. Reinforcement Learning from Human Feedback (RLHF) [33] is one of the most effective techniques where the model's responses are evaluated by human trainers.

Continual Learning is also a strategy that is crucial for keeping the model updated and relevant over time. This method ensures that the model keeps learning from new data and experiences by being trained with an up-to-date dataset and not reaching "catastrophic forgetting", where learning new information causes the model to forget previously learned information.

In essence, the combination of transformer architecture, vector-based word representations, sophisticated neural networks, and strategic training methodologies underpin the remarkable capabilities of LLMs. These models continuously evolve, reshaping the way we interact with technology and access information through their advanced language-processing abilities.

### 2.5.1 Best LLM for the use case

In the rapidly evolving landscape of Large Language Models, selecting the right one for this project is a crucial decision. The wide range of options available makes this task both exciting and challenging. To be sure that I am choosing the right one, I compared data from tests and evaluations from different sources.

One of the most used tools to address this is Chatbot Arena [57] [14], an open-source project developed in its majority by students from Large Model Systems Organization (LMSYS Org), an open research organization. The main goal of these students is to make large language models accessible to everyone by providing the public with tools to better use and understand the subject [6]. The platform is an evaluation tool that offers a couple

of ways for users to compare and rate LLMs, including chatting with two anonymous models side-by-side and voting for which model is better [74]. Their leaderboard is the result of:

- **Chatbot Arena** more than 200K user votes from battles done in the arena between models [1].
- **MT-Bench** a multi-turn question set also developed by them, that consists of 80 high-quality questions designed to test multi-turn conversation and instruction-following ability. These are capable of covering common use cases and focusing on challenging questions to differentiate models [74].
- **MMMLU**. Measuring Massive Multitask Language Understanding is a new test, proposed by students of various American Universities, to measure a text model’s multitask accuracy. This is done by running 57 tasks of general culture and then more specialized information like elementary mathematics, US history, computer science, law, and others [26].

That being said, as of the 25th of August 2024, OpenAI GPT-4o (latest) ranks first overall but also in 1 (Arena Elo) of the 3 benchmarking methods above (no data is mentioned about MMMLU score but GPT-4o-2024-05-13, the initial released version, is the leader in that. Also no data is mentioned about GPT-4o MT-Bench but the leader is the predecessor GPT-4). This new LLM released on the 13th of May 2024 is the most recent model developed by the global leader OpenAI [2].

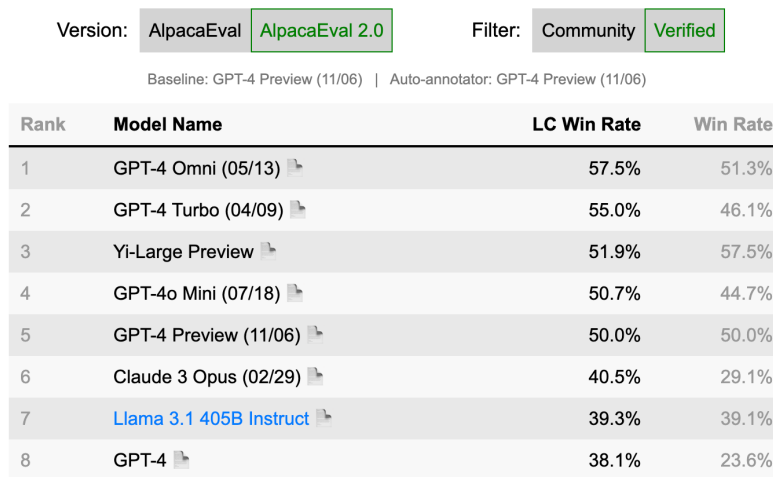
Model	Arena Elo	Arena-Hard-Auto	MT-bench	MMLU	Organization	License
ChatGPT-4o-latest (2024-08-08)	1317				OpenAI	Proprietary
Gemini-1.5-Pro-Exp-0801	1298				Google	Proprietary
Grok-2-08-13	1293				xAI	Proprietary
GPT-4o-2024-05-13	1286	79.21		88.7	OpenAI	Proprietary
GPT-4o-mini-2024-07-18	1275	74.94		82	OpenAI	Proprietary
Claude-3.5-Sonnet	1271	79.35		88.7	Anthropic	Proprietary
Grok-2-Mini-08-13	1268				xAI	Proprietary
Gemini Advanced App (2024-05-14)	1267				Google	Proprietary
Meta-Llama-3.1-405b-Instruct	1266	64.09		88.6	Meta	Llama 3.1 Community
GPT-4o-2024-08-06	1262				OpenAI	Proprietary
Gemini-1.5-Pro-0801	1260	71.96		85.9	Google	Proprietary
GPT-4-Turbo-2024-04-09	1257	82.63			OpenAI	Proprietary
Gemini-1.5-Pro-Preview-0409	1257	53.37		81.9	Google	Proprietary
GPT-4-1106-preview	1251		9.32		OpenAI	Proprietary

Figure 2.2: LMSYS Chatbot Arena Leaderboard. From Hugging Face Spaces. <https://huggingface.co/spaces/lmsys/chatbot-arena-leaderboard> (accessed August 25, 2024).

Another very useful tool is the AlpacaEval Leaderboard. AlpacaEval is the name of an Automatic Evaluator for Instruction-following Language Models [35]. This is an open-source project on GitHub created to reduce human interaction in evaluating LLMs,

as this is expensive, takes a lot of time, and is hard to replicate. So, AlpacaEval is cheap, fast, and easy to replicate while keeping accuracy in the judgment process thanks to being validated against 20K human annotations. They use models to verify models and although it is more intended for verifying in development models, it can also be used to verify released models.

As of the 25th of August 2024, GPT-4o ranks first, in AlpacaEval [7], with a 57.5% win rate, winning by 2.5% to OpenAI's previously released model, GPT-4 Turbo. According to the developers, the win rate consists of the measures of the fraction of time the model's output is preferred over the reference's outputs (GPT-4 Preview (11/06)). In other words, they get the output from the testing model and compare it with the reference output with the help of the auto-annotator (GPT-4 Preview (11/06)).



Rank	Model Name	LC Win Rate	Win Rate
1	GPT-4 Omni (05/13)	57.5%	51.3%
2	GPT-4 Turbo (04/09)	55.0%	46.1%
3	Yi-Large Preview	51.9%	57.5%
4	GPT-4o Mini (07/18)	50.7%	44.7%
5	GPT-4 Preview (11/06)	50.0%	50.0%
6	Claude 3 Opus (02/29)	40.5%	29.1%
7	Llama 3.1 405B Instruct	39.3%	39.1%
8	GPT-4	38.1%	23.6%

Figure 2.3: AlpacaEval LeaderBoard. From Tatsu-Lab Github Page. [https://tatsu-lab.github.io/alpaca\\_eval/](https://tatsu-lab.github.io/alpaca_eval/) (accessed August 25, 2024).

Having analyzed all this, I also needed to take into consideration the accessibility to implement the Conversational Agent with those models. As this project was developed using Microsoft Azure services, one highlight point is the partnership between Microsoft and OpenAI. Since 2016, Microsoft has committed to building Azure into an AI supercomputer for the world, serving as the foundation of its vision to democratize AI as a platform. The deal with OpenAI helped accelerate the process and now Azure has a lot of features connected to Artificial Intelligence that make it stand out from other competitors [42].

Thanks to Azure AI Studio which is available to all Microsoft Azure clients, it is intuitive and much simpler than before to use all the licensed models and implement AI applications. Later on, we will take a deeper look at the tool but for now, I wanted to display that the accessibility to use GPT models in Azure and all the support available was really a crucial point towards my decision.

Summing up, GPT models, especially GPT-4o, are the best Large Language Models for this use case, being that of performance, accuracy, accessibility, support, and all important factors.

## 2.6 Word Embeddings

Word embeddings offer a way to represent words and sentences as numerical vectors in a high-dimensional space [9]. Unlike earlier methods such as one-hot encoding, where words were treated as isolated entities, word embeddings allow for the representation of words in a manner that captures their semantic and syntactic similarities. The creation of word embeddings typically involves learning from a large amount of text that uses neural network-based models to understand word associations from their context [72].

A remarkable feature of word embeddings is their ability to capture complex relationships between words. In a well-trained model, simple vector arithmetic can uncover semantic and syntactic relationships, such as the ability to equate "doctor" - "medical" + "education" = "teacher": Here, the profession "doctor" is associated with the field "medical". By replacing "medical" with "education", the vector shifts to a profession in the education field, like "teacher".

In modern NLP tasks, the role of word embeddings cannot be overstated. They enable models to process text data by understanding the meanings and relationships of words, which is crucial for a wide range of applications including text classification, sentiment analysis, machine translation, and question-answering systems.

However, word embeddings are not without limitations. They can inadvertently reflect and perpetuate biases present in the training data. Additionally, traditional word embeddings assign a single, context-independent representation to each word, which can be limiting for words with multiple meanings.

To address this, the field has evolved to contextual embeddings with advanced models like BERT and GPT. These models generate word embeddings that are sensitive to the surrounding text, offering dynamic, context-aware representations. This evolution marks a significant stride in enhancing the capability of machines to understand and process human language with greater accuracy and sophistication.

## 2.7 Prompt Engineering

Prompt engineering has emerged as a critical aspect of interacting with large language models, especially relevant for the design and optimization of conversational agents. The process involves designing and refining input prompts to achieve desired outputs from these models [36]. Effective prompt engineering enhances the utility of language models across various applications, from natural language understanding and generation to specialized tasks in research and industry.

Prompt engineering is essential for several reasons:

- **Enhancing Accuracy** Well-designed prompts help reduce ambiguities and guide the LLM toward generating more accurate and contextually relevant responses.

- **Improving Efficiency** By refining prompts, the interaction between the user and the AI system becomes more efficient, minimizing the need for follow-up questions and clarifications.
- **Addressing Multilingual Challenges** In multilingual settings, prompt engineering helps ensure that the AI can handle diverse linguistic inputs and provide coherent responses in the desired language.

The process of systematic design of input prompts to elicit specific responses from language models can be broken down into several key stages. Understanding model behavior is essential to comprehend how language models interpret and respond to different input types. This includes recognizing the impact of prompt phrasing, context length, and the presence of specific keywords or instructions.

Developing effective prompts often requires an iterative approach. Initial prompts are tested, and their outputs are analyzed to identify areas for improvement. This iterative cycle continues until the desired output quality is achieved. Embedding relevant context within the prompt can significantly influence the model's response. This involves providing background information, defining the scope of the task, and specifying any constraints or expected formats.

Evaluating the effectiveness of prompts requires robust metrics. These can include precision, recall, and the overall relevance and coherence of the generated outputs. Automated tools and human evaluators are often used in tandem to assess prompt quality. In my case, I used human evaluators.

In conclusion, Prompt Engineering is pivotal in leveraging the full potential of LLMs. By systematically designing and refining prompts, it is possible to enhance the accuracy, efficiency, and multilingual capabilities of AI systems. This not only improves the performance of conversational agents but also broadens the scope of their applications in various domains, driving innovation and efficiency across multiple fields [49]. As the field of AI continues to evolve, the importance of prompt engineering will only grow, making it a fundamental skill for developing advanced and effective AI solutions.

## 2.8 Conversational Agent

A conversational agent, also known as a chatbot, is a software application designed to interact with humans in their natural language. These interactions can occur in written or spoken form, typically over the Internet.

Conversational agents leverage advanced technologies like [NLP](#) and Machine Learning ([RAG](#)) to understand, learn, and respond. They can handle complex queries, learn from past interactions, and even generate human-like text. They are capable of understanding context, and sentiment, and can provide more personalized experiences.

These agents are commonly integrated into messaging platforms, websites, mobile applications, and other digital interfaces to enhance user engagement and provide efficient

and personalized assistance. As technology continues to evolve, conversational agents are likely to play an increasingly significant role in shaping the way humans interact with digital systems and this project is designed to also help this process.

## 2.9 Machine Translation

One of the most prominent applications of AI and one of the biggest issues I am trying to solve with this project is translation. Machine Translation is very successful thanks to intelligent algorithms that are deployed to bridge language gaps and facilitate effective global communication.

Machine Translation works by leveraging advanced algorithms that analyze patterns, context, and semantic nuances within language. Traditional methods often relied on rule-based systems or statistical models, but the evolution of AI has brought a new era with the development of Neural Machine Translation (NMT) [44]. This new method can produce high-quality translations with 80%+ accuracy in almost every language.

### 2.9.1 Neural Machine Translation

A Neural Machine Translation model translates a sentence  $x$  in the source language to a sentence  $y$  in the target language [65]. NMT models are usually auto-regressive and generate the new sentence from left to right.

The foundational architecture of NMT is the encoder-decoder framework [65]. The encoder part of the model takes the input sentence in the source language and transforms it into a series of high-dimensional context vectors. These vectors are designed to encapsulate the semantic and syntactic properties of the input text. The decoder then comes into play, generating the target language sentence word by word. It relies not just on the context vectors provided by the encoder but also on the words it has already generated, using them to predict the next word in the sequence.

A critical advancement in NMT has been the introduction of the attention mechanism [54, 32] which allows the decoder to focus on the important parts of the sentence, significantly enhancing the quality of the output.

NMT models are often framed as sequence-to-sequence learning problems. In this setup, both the input (the source sentence) and the output (the target sentence) are sequences that the model learns to map. This approach is more streamlined compared to earlier translation methods, as it eliminates the need for intermediate steps like word alignment, which were common in statistical models.

The superiority of NMT comes from its contextual understanding and its ability to continuously improve with more data. Unlike previous methods, NMT is better equipped to handle idioms, colloquialisms, and the subtleties of human language. As a result, NMT has become the preferred method for machine translation, known for producing translations that are not just literal but also contextually and linguistically coherent.

## 2.10 RAG-Based Information Retrieval

Large Language Models are very important but they also have issues like hallucination, outdated knowledge, and nontransparent, untraceable reasoning processes. Retrieval-augmented generation (RAG) [22] has emerged as a promising solution by incorporating knowledge from external databases. It is a novel approach that combines the strengths of retrieval and generation techniques to improve the performance of large language models. RAG models are capable of retrieving relevant information from a knowledge source and incorporating it into generated text.

The traditional (Naive) RAG consists of three steps: indexing, retrieval, and generation [22]. Because of this composition, it is also known as the “Retrieve-Read” framework. Each step has an important role in the process and it is relevant to understand each:

- **Indexing**

The indexing process, a pivotal initial step in data preparation, takes place offline and involves multiple stages. It commences with the cleansing and extraction of data, converting diverse file formats such as PDF, HTML, Word, and Markdown into standardized plain text. To conform to language model context limitations, the text undergoes segmentation into more manageable chunks through a process known as chunking. Subsequently, these chunks are transformed into vector representations. This facilitates similarity comparisons during retrieval, culminating in the creation of an index to store text chunks and their vector embeddings as key-value pairs, ensuring efficient and scalable search capabilities.

- **Retrieval**

Upon receiving a user query, the system employs the same encoding model used during indexing to transcode the input into a vector representation. It then calculates similarity scores between the query vector and vectorized chunks within the indexed corpus. Prioritizing and retrieving the top K chunks demonstrating the greatest similarity to the query, these chunks serve as the expanded contextual basis for addressing the user’s request.

- **Generation**

The posed query and selected documents are synthesized into a prompt, challenging the large language model to craft a response. In the case of ongoing dialogues, existing conversational history can be seamlessly integrated into the prompt, empowering the model to engage effectively in multi-turn dialogue interactions.

To make use of this process, Azure AI Studio and Azure Services return to the debate. Azure AI Search, formerly known as Azure Cognitive Search [31], is a tool that can index (with data chunking and vectorization) and query user-owned data in a secure and scalable way [62]. Thanks to being in the Azure environment, it has integration with the Azure AI Studio, automates data ingestion/retrieval from Azure data sources, and is connected

with all other related services that can be necessary for the development of RAG and/or Generative AI apps.

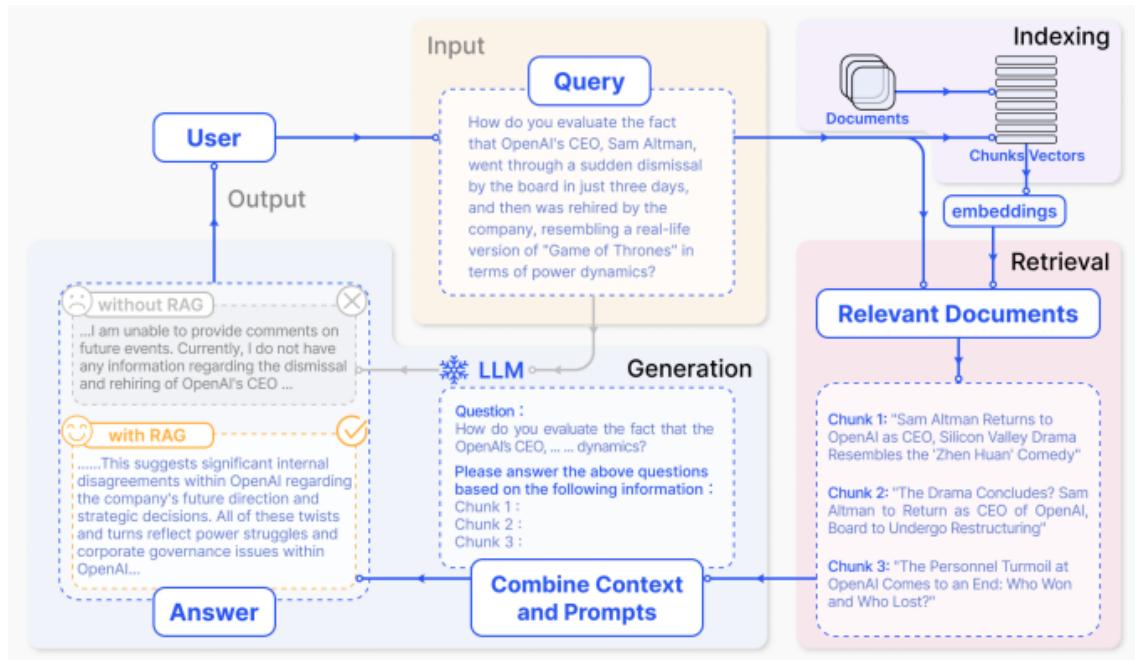


Figure 2.4: "A representative instance of the RAG process applied to question answering". Figure 2 from "Retrieval-Augmented Generation for Large Language Models: A Survey" by Gao, Y., Xiong, Y., Gao, X., Jia, K., Pan, J., Bi, Y., Dai, Y., Sun, J. and Wang, H. (n.d.).

## 2.11 Azure AI Studio

As already mentioned in the previous chapters, Azure AI Studio is a crucial part of the project and it is important to understand this powerful tool's capabilities. Launched for public preview on the 15th of November 2023, it is Microsoft's cloud-based "state-of-the-art platform designed to simplify generative AI application development, leveraging OpenAI models such as GPT4, alongside a wide array of other cutting-edge models and services." [61].

As a centralized platform for all AI development components, this tool was developed to make developers' efforts easier including loading data from a variety of sources, preparing it for training, training a model, and deploying the model to production. The main function of this platform is for professional software developers to create generative AI applications and custom copilots using built-in security, compliance, and integration with Azure resources. With a simple yet powerful user interface, with highlighted tabs for exploration, creation, and management of the AI resources.

The explore page should be your initial page, where you can find a model, service, or solution to start developing the product. As it is a stateless page, it is always available and the continuation of the development is on the Build page [63].

This page is where developers can build or customize AI solutions and models and follow the Azure Low-code approach of having a studio version (with less code and more intuitive functions like drag and drop) and also a code version. Some of the most highlighted tools on this page are related to the simplified development of large language model (LLM) solutions and copilots with end-to-end app templates and prompt samples for common use cases, handle the complex mapping of functions and code between LLMs, tools, custom code, prompts, data, search indexes, and more and evaluate, deploy and monitor the AI app [63].

Lastly, we have the Manage page, more destined to the admin of the project. It is possible to adjust settings like connections, create teams, check usage and billing, and more administrative stuff [63].

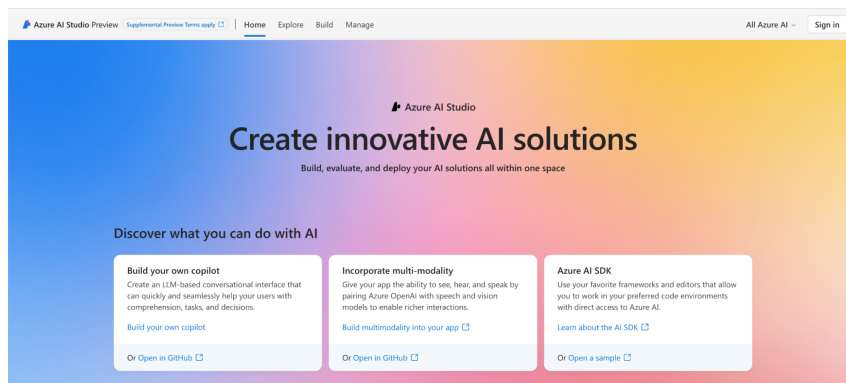


Figure 2.5: Azure AI Studio Homepage. From Azure. <https://ai.azure.com/> (accessed January 23, 2024).

## CONCRETE SYSTEMS AND TECHNOLOGICAL SOLUTIONS

Besides the AI component of this project, there is also a WhatsApp component for the users to interact with the conversational agent and a full-stack application for back-office management. To do the last, I used LinkThings default stacks for web development. These tools have had success in various projects, like Greenie and ColdCha.

Greenie is a planning tool for growers and greenhouse operators and offers a range of features that make managing agricultural operations more efficient and streamlined. Its functionalities include the ability for growers to schedule and track various farming tasks, such as planting and harvesting, ensuring all activities are well-organized and timely. The tool also might provide an inventory management system to monitor stock levels of materials like seeds and fertilizers, track usage, and even predict future needs based on historical data. Additionally, Greenie helps optimize greenhouse space, which involves layout planning, crop rotation schedules, and optimizing planting density for better yields. It also offers tools for tracking profits from crops, forecasting future earnings based on current schedules and market trends, and analyzing financial performance over time.

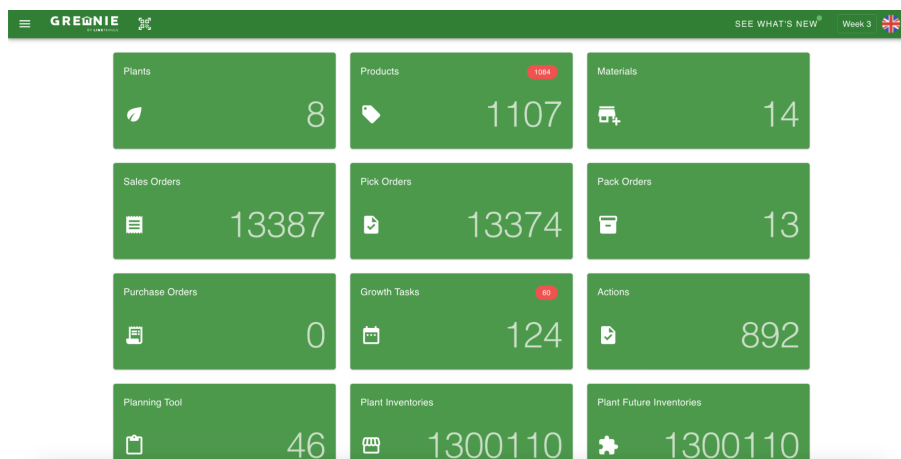
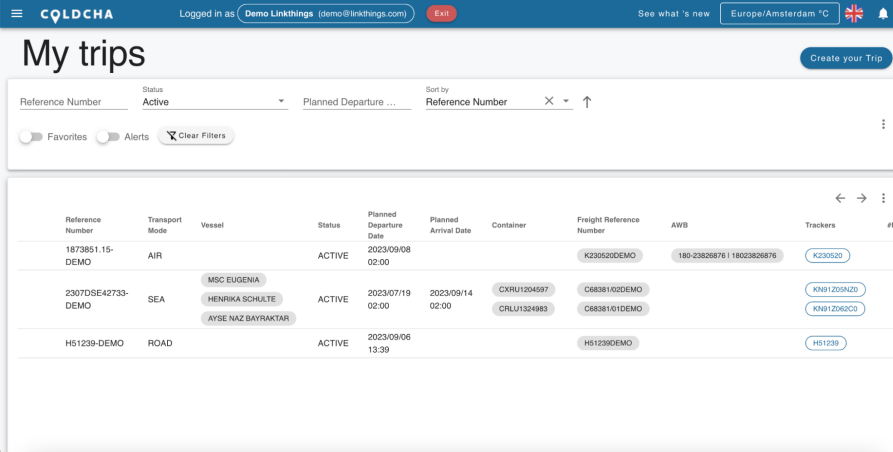


Figure 3.1: Greenie Homepage. From Greenie.

Coldcha is a logistics management system that specializes in analyzing extensive information and addressing a multitude of inquiries pertinent to the transportation phase of the supply chain. Its capabilities are particularly relevant in the area of real-time transport tracking. This is achieved through the use of trackers placed within shipping containers, which continuously monitor and report on key environmental parameters such as temperature, humidity, and light exposure. One of the standout features of Coldcha is its ability to issue alerts if any of these tracked values deviate from predefined norms. This ensures that any potential issues can be addressed promptly, thereby safeguarding the integrity of the goods being transported. Moreover, Coldcha enhances its effectiveness by integrating several APIs, which collectively work to provide the most accurate and up-to-date information to its customers.



Reference Number	Transport Mode	Vessel	Status	Planned Departure Date	Planned Arrival Date	Container	Freight Reference Number	AWB	Trackers	#
1873851.15-DEMO	AIR	MSC EUGENIA	ACTIVE	2023/09/08 02:00			K230520DEMO	180-23828876   18023828876	K230520	
2307DSE42733-DEMO	SEA	HENRIKA SCHULTE AYSE NAZ BAYRAKTAR	ACTIVE	2023/07/19 02:00	2023/09/14 02:00	CKRU1204597 CRLU1324983	C6838102DEMO C6838101DEMO		KN912094ZD KN912096CQ	
HS1239-DEMO	ROAD		ACTIVE	2023/09/06 13:39			HS1239DEMO		HS1239	

Figure 3.2: ColdCha Homepage. From ColdCha.

The majority of the following tools have been employed in both projects and have demonstrated their reliability, user-friendliness, and scalability. By using these tools I could also get better support from the company as the developers have more experience with them. Additionally, I have 2 years of experience with these technologies due to being a member of the development team for these projects. I use in my daily work life Django, Vue.js, GraphQL, Microsoft Azure, and Vuetify. The new tools are PromptFlow, Semantic Kernel, and WhatsApp Business API which were found and used for the first time in this project.

### 3.1 Microsoft Azure

Microsoft Azure is a cloud computing platform provided by Microsoft that offers a wide range of services and solutions to help organizations build, deploy, and manage applications and services through a global network of data centers. Azure is known for its global presence with data centers strategically located around the world, ensuring reliable and scalable cloud solutions. As a key player in the cloud industry, Azure caters

to the diverse needs of businesses, ranging from startups to large enterprises. According to estimates from Synergy Research Group available in Statista [52], Microsoft Azure's market share in the worldwide cloud infrastructure market as of Q2 2023 is 22%, only being surpassed by Amazon AWS with 32%.

The platform is rich in functionalities, with more than 200 available products but some of the key features are:

- **IaaS (Infrastructure as a Service)**

Azure provides a range of virtual machines (VMs) that can be used to run applications, host websites, and store data [43]. Azure VMs offer a variety of operating systems, including Windows and Linux, and can be customized to meet specific needs.

- **PaaS (Platform as a Service)**

Azure offers a range of platform services [11], including Azure App Service, Azure Functions (that we will take a deeper look next), and Azure Logic Apps, that allow developers to build, deploy, and manage applications without having to worry about the underlying infrastructure. Azure PaaS services support a variety of programming languages, including .NET, Java, Node.js, and Python.

- **SaaS (Software as a Service)**

Azure provides a range of software services, including Microsoft 365, Dynamics 365, and Power Platform, that can be used to manage business operations, automate workflows, and analyze data.

- **Data Services**

Azure provides a range of data services [39], including Azure SQL Database, Azure Cosmos DB, and Azure Data Lake Storage, that can be used to store, manage, and analyze data. Azure data services support a variety of data types, including structured, semi-structured, and unstructured data.

- **Security**

Azure provides a range of security services [17], including Azure Security Center, Azure Active Directory, and Azure Key Vault, that can be used to secure applications, data, and identities. Azure security services support a variety of compliance standards, including HIPAA, ISO 27001, and SOC 2.

- **DevOps**

Azure provides a range of DevOps services, including Azure DevOps, Azure Artifacts, and Azure Test Plans, that can be used to automate software development processes, manage source code, and test applications. Azure DevOps services support a variety of development methodologies, including Agile, Scrum, and Kanban.

### 3.1.1 Azure SQL Database

Azure SQL Database is a fully managed, relational database service provided by Microsoft, and is part of the Azure cloud platform [71]. Offering a robust and scalable platform for building and deploying modern applications, it is positioned as a Platform as a Service (PaaS), which abstracts the complexities of database management, allowing developers and organizations to focus on application development without the burden of maintaining the underlying infrastructure.

The service provides flexible scalability options to adapt to varying workloads. Users can easily scale up or down based on demand, ensuring optimal performance and cost-efficiency. It also enables developers to process both relational data and nonrelational structures, such as graphs, JSON, spatial, and XML [71]. The service automatically performs regular backups, and users can leverage point-in-time restore capabilities to recover databases to specific points in time, providing a safety net against accidental data loss or corruption. Geo-replication and easy integration with all other Microsoft Azure services help this powerful tool to be one of the best for the project. Azure SQL Database's architecture allows for advanced features like automated performance tuning, where the database continuously monitors queries and automatically adjusts indexes and query plans for optimal performance [71].

One of the core strengths of Azure SQL Database is its support for Transact-SQL (T-SQL), the primary means of programming and managing SQL Server databases. T-SQL, an extension of SQL (Structured Query Language), includes additional features for declaring variables, processing data, and error handling, making it more powerful for complex data operations. Azure SQL Database leverages this to offer sophisticated querying capabilities, efficient data manipulation, and robust transaction control, which are essential for reliable applications.

Stored procedures are also supported by Azure SQL Database. They are sets of T-SQL statements that are stored and executed on the server and play a critical role in executing complex business logic efficiently. Stored procedures help in improving performance as they are compiled once and can be executed repeatedly, reducing the amount of information sent over the network. This is particularly beneficial in cloud environments where network latency can impact application performance. Additionally, stored procedures ensure a higher degree of security as they can encapsulate the logic, restricting direct access to underlying data tables. These procedures are useful for running periodic functions to update data inside the databases.

### 3.1.2 Azure Machine Learning Prompt Flow

Azure Machine Learning (Azure RAG) Prompt Flow is an advanced feature within the Azure RAG ecosystem, designed to streamline and enhance the workflow for developing, deploying, and managing machine learning models [73]. This feature is pivotal to accelerate the iterative process of model development and operationalization.

Azure [RAG Prompt Flow](#) is built on a scalable, cloud-based architecture that integrates seamlessly with the broader Azure ecosystem. Key components include:

- **Azure ML Workspace**

The central hub for managing all machine learning resources, including datasets, experiments, pipelines, and models.

- **Compute Resources**

A diverse range of compute options, from local compute instances for development and testing to scalable clusters and virtual machines for training and inference tasks.

- **Experimentation**

Tools for running and tracking experiments, supporting version control, and collaboration among teams.

Azure ML Prompt Flow is designed for interoperability, supporting integration with various development environments, including Jupyter Notebooks, Visual Studio Code, and GitHub. Additionally, it supports interoperability with other Azure services such as Azure Databricks for big data processing, Azure Synapse for data analytics, and Power BI for visualization and reporting.

Adding all this to a low code approach, Prompt Flow is almost everything a developer needs to develop high-quality AI-based solutions. This is one of the core components of the project because it is what enabled me to manage the model and interactions with it.

### 3.1.3 Function App

Azure Function Apps is a serverless computing service provided by Microsoft Azure, designed to run small pieces of code, known as functions, in response to various events [23]. This service abstracts the underlying infrastructure, allowing developers to focus solely on the functionality of their code without the need to manage servers, patching, or scaling.

At the core of the Function Apps is the Azure Functions runtime, which provides the necessary environment for executing code. Each function app contains one or more individual functions, which are the basic units of deployment, execution, and scaling. These functions can be written in multiple programming languages, including C#, JavaScript, Python, Java, and PowerShell, providing flexibility.

These functions can be triggered manually, by a queue of messages, through an [HTTP](#) endpoint, by changes in databases, by timer and so many more useful ways [24]. Thanks to this, Function Apps are well-suited for various applications, from simple event-driven tasks to complex, stateful workflows.

## 3.2 Semantic Kernel

Microsoft Semantic Kernel (SK) is an innovative open-source project designed to facilitate the integration of AI capabilities into applications by providing a robust framework for

working with language models [16]. Developed by Microsoft and the first production version released in May 2024 (in the middle of the development but prior to this launch I used a pre-release version), Semantic Kernel aims to simplify the process of embedding AI functionalities into various software systems. This framework offers a structured approach to leverage AI models, enabling developers to create more intelligent and context-aware applications.

The architecture of the Semantic Kernel is modular, allowing developers to integrate specific components based on their application needs. The key components of the Semantic Kernel that I use are:

- **Kernel**

The core component that manages the lifecycle of AI models and coordinates their interactions. The Kernel provides APIs for loading models, executing inferences, and handling input/output data transformations.

- **Plugins**

Extend the functionality of the Semantic Kernel by adding support for additional AI models, custom processing steps, or integration with third-party services. This extensibility ensures that the Semantic Kernel can adapt to the evolving needs of AI applications. Here we define functions to be used by the kernel to solve the issues it is presented with.

Microsoft Semantic Kernel represents a significant advancement in the integration of AI technologies into software systems and it is a core component of this project.

### 3.3 Django

Django is a free and open-source high-level Python web framework that encourages rapid development and clean, pragmatic design [19]. It follows the Model-View-Controller (MVC) architectural pattern, but in Django, it is referred to as the Model-View-Template (MVT) pattern. Django's market share is around 32.89% [20], being considered the number 1 framework for backend development.

Django provides a clean, elegant URL scheme which is an important detail in a high-quality web application. It also offers a built-in Admin interface that simplifies the management of users and permissions, a feature that is especially handy during the early stages of development. Although it has a single-tenant architecture, LinkThings has developed a strategy to enable the use of multi-tenant architectures in Django-based projects, broadening its applicability in various scenarios. This adaptability, combined with its robust and feature-rich environment, makes Django a preferred framework for this project.

## 3.4 GraphQL

GraphQL is a query language for APIs and a runtime environment that empowers clients to request the specific data they need [21]. Developed by Facebook in 2012 and later open-sourced in 2015 [34], GraphQL has gained widespread adoption in the web development community for its efficiency and flexibility being used by companies that range from Airbnb to Audi, GitHub, Netflix, Shopify, X (former Twitter), Facebook and The New York Times.

Unlike traditional REST APIs, where the server defines the structure and content of the response, GraphQL allows clients to request only the data they require, eliminating unnecessary data transfer and improving overall performance.

In GraphQL, clients can specify the shape and structure of the response, enabling a more interactive and responsive development process. This makes it particularly well-suited for applications with complex data requirements, such as single-page applications and mobile apps, where optimizing data transfer is crucial.

The core concepts of GraphQL include schema, queries, and mutations. GraphQL APIs are defined by a schema, which serves as a contract between the client and the server. The schema outlines the types of data that can be queried and the relationships between them. For data retrieval, clients use the GraphQL query language to request specific data from the server. Queries are hierarchical and mirror the shape of the desired response. While queries are used for reading data, mutations enable clients to modify data on the server. This includes creating, updating, or deleting records.

By combining GraphQL with Graphene-Django [25], I was able to create a powerful and efficient API with a Python backend. Graphene simplifies the implementation of a GraphQL server, making it accessible to developers working with Python-based technologies. For the client side of the web application, I used Apollo [8], a comprehensive platform for building GraphQL-powered applications. It provides a set of tools and libraries for both the client and server sides, facilitating the development, testing, and maintenance of GraphQL applications.

## 3.5 Vue.js

This project uses Vue.js, a progressive JavaScript framework for building user interfaces released in 2014. Vue.js builds on top of standard HTML, CSS, and JavaScript, providing a declarative and component-based programming model that helps efficiently develop user interfaces, whether they are simple or complex.

Vue.js has gained significant popularity among developers worldwide [4]. As of 2023, it is used by 16.38% of developers, making it one of the most popular web frameworks. Moreover, Vue.js is used by over 35,547 companies, with a market share of about 0.3% in the Software Frameworks category [45].

This framework is known for its minimal core size, enabling faster download and rendering times, and contributing to an enhanced user experience. Vue.js also employs a reactive data binding mechanism, allowing seamless synchronization between the model and the view. This ensures that any changes to the data are automatically reflected in the user interface, simplifying development [30, 12].

In addition to Vue.js, I also used Vuetify [70], an open-source UI Component Framework for Vue.js. Vuetify provides a collection of beautifully handcrafted Vue components, allowing us to create visually appealing and content-rich applications. It is a Material Design component framework.

By leveraging the power of Vue.js and Vuetify, I aimed to build a robust, efficient, and visually appealing project.

### 3.6 WhatsApp Business API

WhatsApp, developed by WhatsApp Inc. and acquired by Facebook (now Meta Platforms, Inc.) in 2014, is a cross-platform messaging application that has become one of the most widely used communication tools globally. Launched in 2009, WhatsApp allows users to send text messages, voice messages, images, and videos, make voice and video calls, and share documents. As of 2023, WhatsApp reported over 3.03 billion users, demonstrating its widespread adoption and significant impact on global communication [58].

WhatsApp Business [API](#) allows medium and large businesses to connect with customers at scale [13]. Unlike the standard WhatsApp application, the Business [API](#) does not have a graphical user interface; instead, it provides endpoints for sending and receiving messages programmatically, integrating with existing CRM (Customer Relationship Management) systems and customer support platforms

This tool is used to communicate with the end user, either by using a webhook to receive all the communication or with HTTP requests to send messages and interactive questions to them.

## RELATED WORK

As the project is centered around a very recent and emerging topic, the body of existing literature and scholarly contributions are still minimal. In light of these circumstances, the examination of related work becomes not only an exploration of established knowledge but also an endeavor to identify and comprehend the most recent developments in the field. We will navigate through the limited yet invaluable resources available, aiming to extract key insights and discern patterns that may contribute to the development and evaluation of the [LLM](#)-based conversational agent.

### 4.1 Conversational Health Agents

In the realm of conversational agents, the work titled “Conversational Health Agents: A Personalized [LLM](#)-Powered Agent Framework” [5] is particularly noteworthy. The authors propose a framework that leverages Large Language Models (LLMs) to empower Conversational Health Agents (CHAs) to generate personalized responses for users’ healthcare queries.

The proposed framework integrates healthcare data sources, enables multilingual conversations, and interacts with various user data analysis tools. The authors argue that while current CHAs, especially those utilizing LLMs, primarily focus on conversation, they often lack comprehensive agent capabilities. This includes the ability to access personal user health data, 24/7 data collection sources, and electronic health records, as well as integrating the latest published health insights.

This work is relevant to the development of the conversational agent in this dissertation as it highlights the potential of LLMs in conversational agents and the ongoing efforts to overcome their limitations. The integration of data sources and the ability to interact with various user data analysis tools can be adapted to administrative tasks, such as accessing and analyzing employee data, company policies, or other relevant administrative questions. Furthermore, the ability to generate personalized responses can be crucial in administrative tasks, where the context and the individual’s role within the organization can significantly impact the appropriate response. Also, the proposed [LLM](#)-powered

## 4.2. AI-BASED CHATBOT TO SOLVE MODERN-DAY ENTERPRISE BUSINESS PROBLEMS

framework leveraging a service-based architecture is very similar to the one in this project and can be used as a proof of concept.

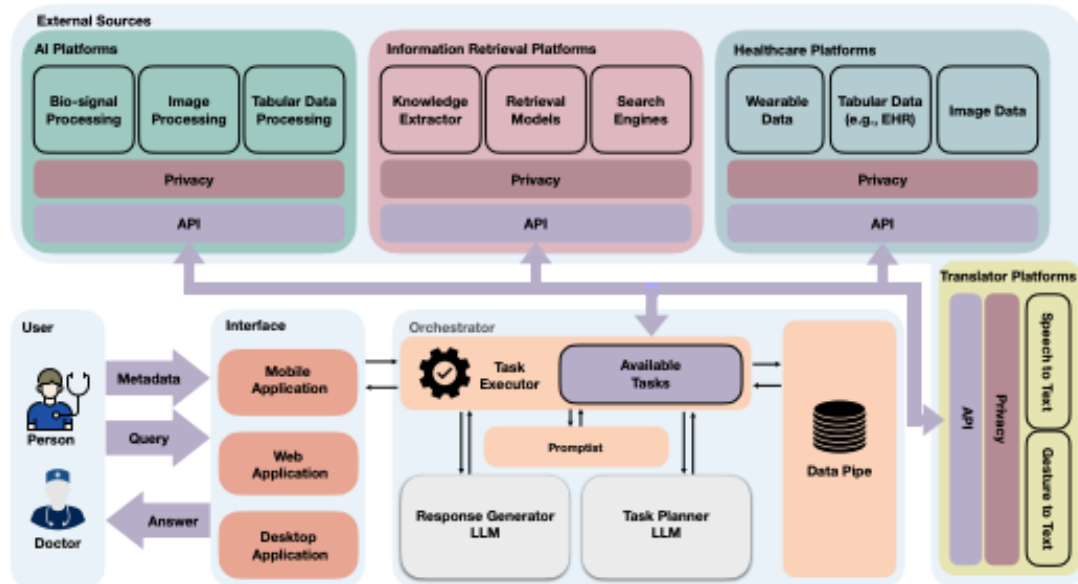


Figure 4.1: "An overview of the proposed LLM-powered framework leveraging a service-based architecture". Figure 2 from "Conversational Health Agents: A Personalized LLM-Powered Agent Framework" by Abbasian, M., Azimi, I., Rahmani, A. and Jain, R. (2023).

## 4.2 AI-Based Chatbot to Solve Modern-Day Enterprise Business Problems

This article titled "AI-Based Chatbot to Solve Modern-Day Enterprise Business Problems" [48] describes how to build an AI-based chatbot for solving modern-day enterprise business problems, especially in the human resource domain. The article explains how to use Microsoft Cognitive Services (now named Microsoft AI Search), such as Q&A Maker, Power Virtual Agents, Power Automate, and Teams, to create a chatbot that can dynamically respond to employee queries.

The article also provides a flowchart of the chatbot architecture and some screenshots of the chatbot interface. The article claims that the chatbot can reduce the workload of the HR department, improve employee satisfaction, and save time and cost for the organization.

The work is relevant to this dissertation because it also develops the same idea, using some similar technologies but in a much simpler way. Instead of WhatsApp, they used Microsoft Teams. I also have a back-office portal capable of chatting like they intend to develop in future work. I think it is valid for the development made but in my scenario, the product would lose value in terms of customization and scalability.

### 4.3 Automating Customer Service using LangChain

In the context of modernizing customer service through AI, the paper "Automating Customer Service using LangChain: Building custom open-source GPT Chatbot for Organizations" [47] stands out. This work introduces a custom-built language model using LangChain that is designed to revolutionize customer service. The model combines advanced language processing capabilities with data collected through web scraping, utilizing Google's Flan T5 language models. It is tailored to provide responsive, context-aware, and personalized customer interactions. Its integration into customer service platforms showcases its potential in educational institutions, transforming customer support into a dynamic, user-centric service.

This approach underscores the evolution of customer service, leveraging LLMs to offer efficient, personalized support that aligns with this project focus. They also provide some performance comparisons paired with questions made to the model. Not directly related to this project but of similar difficulty to the questions I made the model answer in this first stage.

## SOLUTION

In this chapter, we delve into the solution developed to address the multifaceted challenges outlined in the previous sections. This solution integrates several advanced technologies to create a seamless and efficient system capable of overcoming the linguistic and operational barriers faced by temporary employment agencies.

### 5.1 Architecture and Flow

The architecture of this system has undergone multiple iterations of thought, discussion, and modification to ensure it provides the most effective solution. This system is designed to integrate seamlessly with external data sources. All users interacting with this agent are pre-registered in these external databases. In this project, the external databases used are related to employee management.

The flow [5.1](#) represents the system and the information flow at a high level. The main components of the system are:

#### 5.1.1 Database

This section provides an overview of the database, specially the schema, [5.2](#) designed to support a multi-tenant application architecture. The database was created using Azure SQL Database which ensures scalability, reliability, and integration with the other Azure services I have. Security is maintained through an IP whitelist, restricting access to approved IP addresses only. Daily backups are handled by Azure, ensuring data protection and availability.

The schema incorporates Django's default user and group models, multi-tenant structures, company-related data, and communication modules, highlighting the relationships between key entities for a scalable system.

Django has its own user authentication system and permissions system. This schema integrates this collection of tables as seen in the tables `user.User`, `auth.Group`, `user.UserGroup`, `auth.Permission`, and `auth.GroupPermissions`. These tables are the foundation of the database, ensuring secure and managed access to various parts of the application.

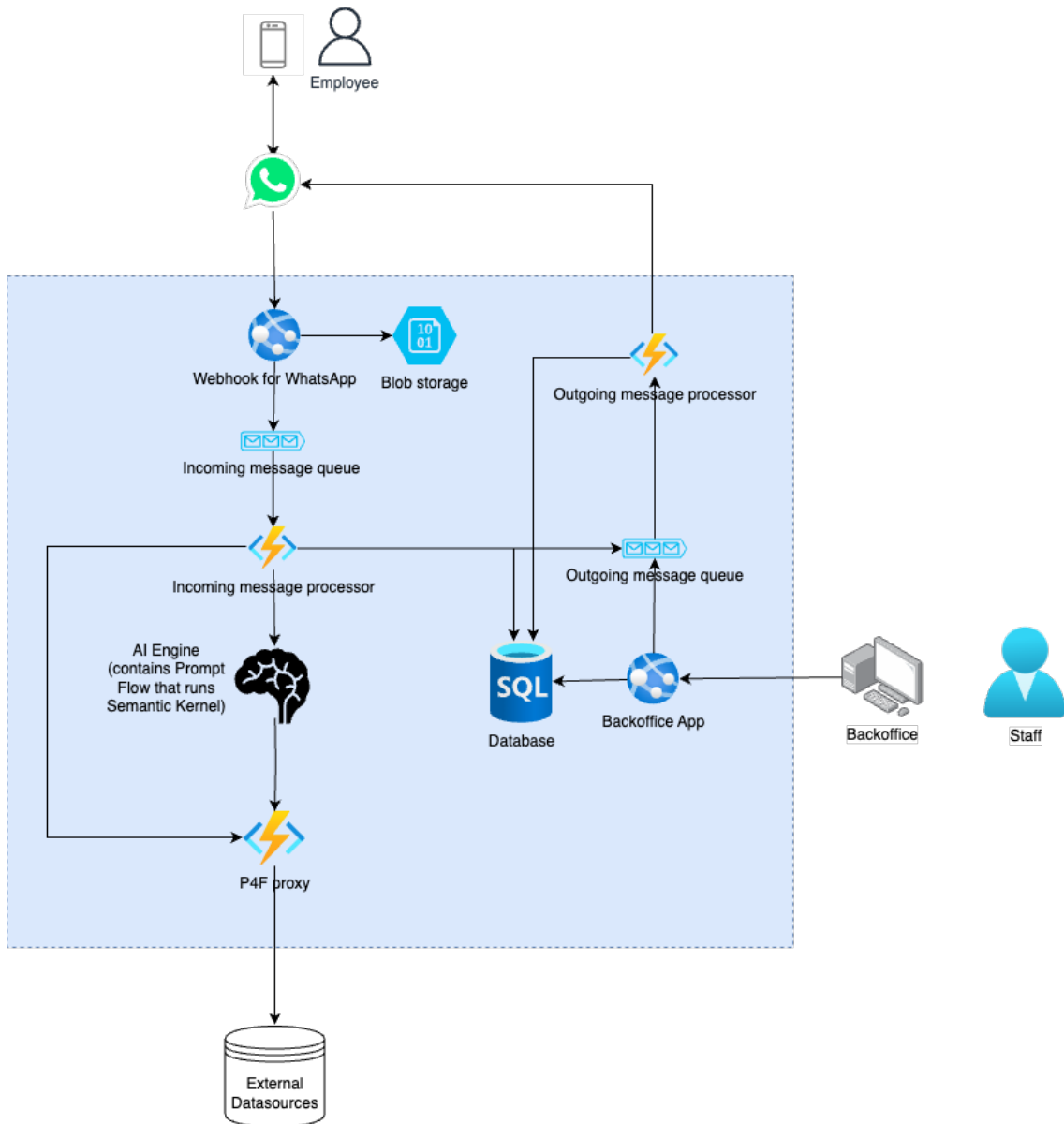


Figure 5.1: "System Components and Flow"

The multi-tenant architecture is a crucial aspect of this schema, ensuring that data is properly segregated and access is controlled across different tenants. This is not supported by default by "mssql-django", which is the third party library we use to connect Django to Microsoft SQL Database, but LinkThings developed a mechanism that allows multi-tenant. The main table of that mechanism is the Tenant table where is the declaration of the tenants, and then we have TenantStaff (that is related to user.User) and TenantStaffGroup (that is related to auth.Group) and inherit their properties. Then, in the normal use of the database, I always referred to these tables instead of the Django ones.

To facilitate external integrations and detailed user management, the schema includes entities representing companies and application users. Because a temporary working agency can operate with diverse companies, the table Company displays an important

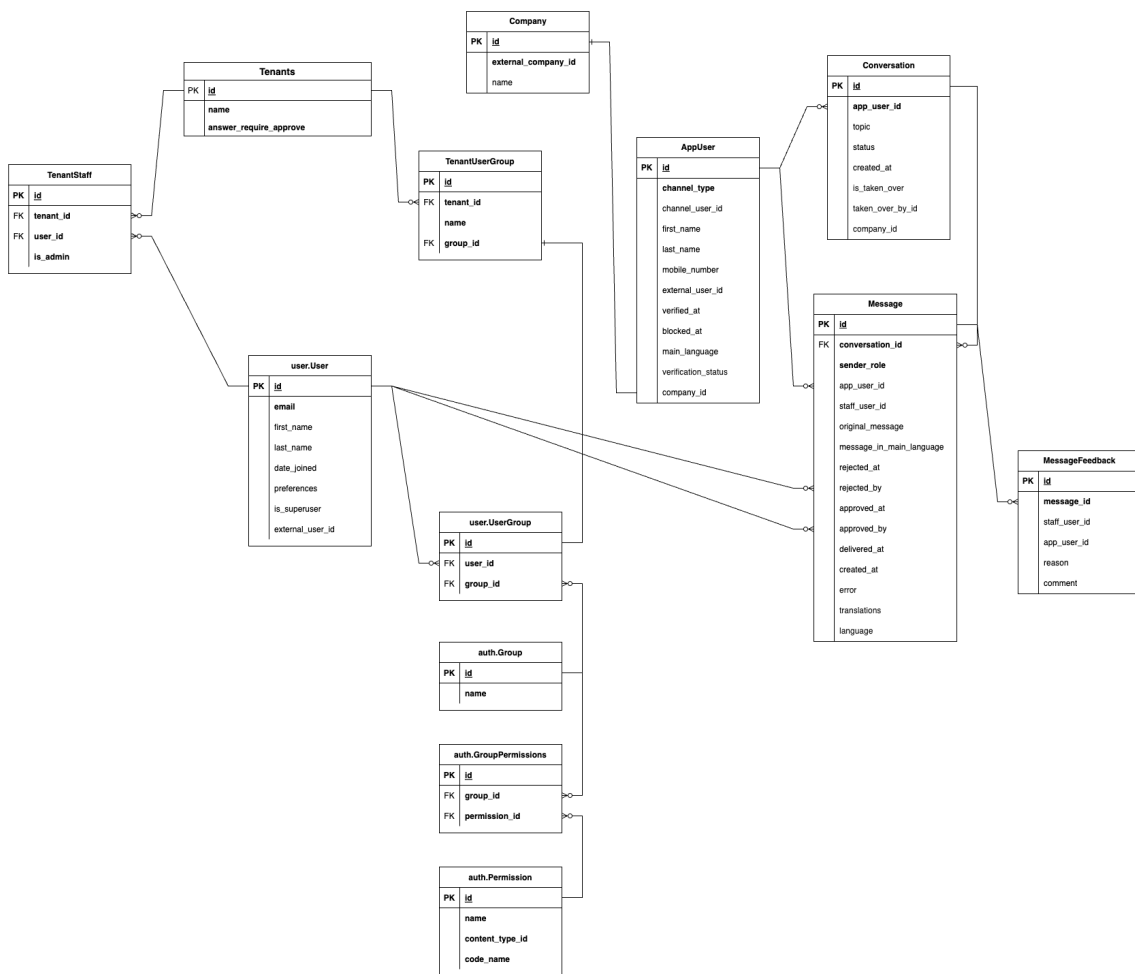


Figure 5.2: "Schema of the database"

role in differentiating them. Then each App User (that is a temporary worker) has all the important data about the person that is relevant to the app. Channel type field was thought so, in the future if there's the need, I can extend this app to other different channels and not just to WhatsApp.

Lastly, we have the communication modules that are essential for facilitating user interaction within the application. They are designed to manage and track conversations and messages between users, ensuring a seamless communication flow. These modules consist of three main tables: Conversation, Message, and MessageFeedback. Each table plays a distinct role in handling different aspects of communication.

The Conversation table acts as the cornerstone for managing the user interactions with the chatbot. It maintains the context and status of each conversation, allowing the application to keep track of ongoing and past interactions.

The Message table handles the individual messages exchanged within a conversation. It stores the content of each message, as well as metadata that helps manage the message lifecycle. Some other fields like **error**, **language**, **approved\_at**, **rejected\_at**, **delivered\_at**, and others related are used for status tracking and error management. The translation field

is used by the Back-Office portal to store the translation of the message in the language used by the back-office members.

The MessageFeedback table is designed to capture detailed feedback on messages. This feedback mechanism ensures that messages meet quality standards and allows for continuous improvement of communication within the application.

This database schema is designed to support a multi-tenant application by leveraging Django's robust authentication system and incorporating entities for tenants, companies, users, and communication modules. The relationships between these entities ensure scalable, secure, and efficient data management across multiple tenants and user groups.

### 5.1.2 Information Retrieval Module

The Information Retrieval Module is a critical component of the system, designed to interface with the external databases, which contain highly sensitive data. This module is implemented as a function app that leverages [SQLAlchemy](#) for database interaction, ensuring efficient and secure data querying and manipulation.

Each topic has an associated function that validates the input, queries the database to get the data, and returns such data in a JSON object, structured to be easily interpreted by the AI engine.

### 5.1.3 WhatsApp WebHook

The WhatsApp WebHook serves as the system's entry point. Users send messages via WhatsApp, which the webhook receives. This webhook is configured using the WhatsApp Business API to handle incoming messages. Upon receipt, the message is forwarded to the Incoming Messages Event Hub and stored in Blob Storage for security purposes.

### 5.1.4 Incoming Messages Processor

Messages in the Incoming Event Hub are processed by the function app "Incoming Messages Processor" which performs several key actions. When a user message is received, a verification process begins. This involves prompting the user to provide their birth date and verifying this information against the employee records in the external database.

This verification process leverages the WhatsApp Business API [3.6 Flow](#), which provides a user-friendly dialog within the WhatsApp conversation. Users can easily enter their birth date in a form, reducing the likelihood of errors such as incorrect date formats.

The above [image 5.3](#) is the main page of the flow but more details can be found in the [5.2 WhatsApp Interface](#) section. After the verification process is concluded, the following message will be sent to the "AI Engine" (which is the name given during the development of this project to the Prompt Flow with Semantic Kernel) through an HTTP request.

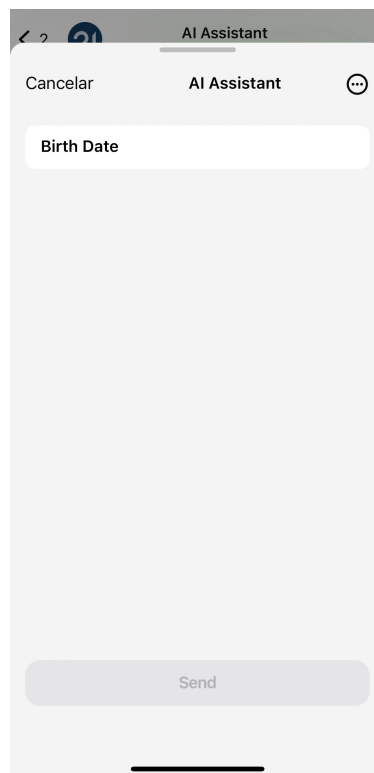


Figure 5.3: "WhatsApp Flow for user verification"

### 5.1.5 AI Engine

The AI Engine initializes its process in the Prompt Flow 5.4. The AI Engine is the core component of this project. It will receive as input the message/question, the employee ID, an API Key (for security reasons), and the chat history (if there's any). The first step of the flow is to verify the API Key. If it isn't successful, it will immediately raise an Exception and no more resources will be spent with that request. Following that the deployment name of the LLM is loaded through a special process (different from all other environment variables because it needs to be used in the flow settings).

After that, the question and the chat history are inserted in a single prompt and fed to the planner (the main component of Semantic Kernel). The planner will then check the plugins and access the best functions to solve the query and will use it and return the data gathered from it. The data will be obtained from the Information Retrieval Module. This data is then fed to a prompt and asks the model to answer the question based on the context (data that comes from the planner), chat history, question, employee name, and today's date.

After this process, there is an answer and this is returned to the Incoming Messages Processor as the response to the HTTP request. The processor will then check if it is to return directly to the user (and forward it to the Outgoing Messages Event Hub) or if it needs to be approved (and just save the answer in the database) based on the tenant setting "answer\_require\_approve" that is a field in the Tenant table.

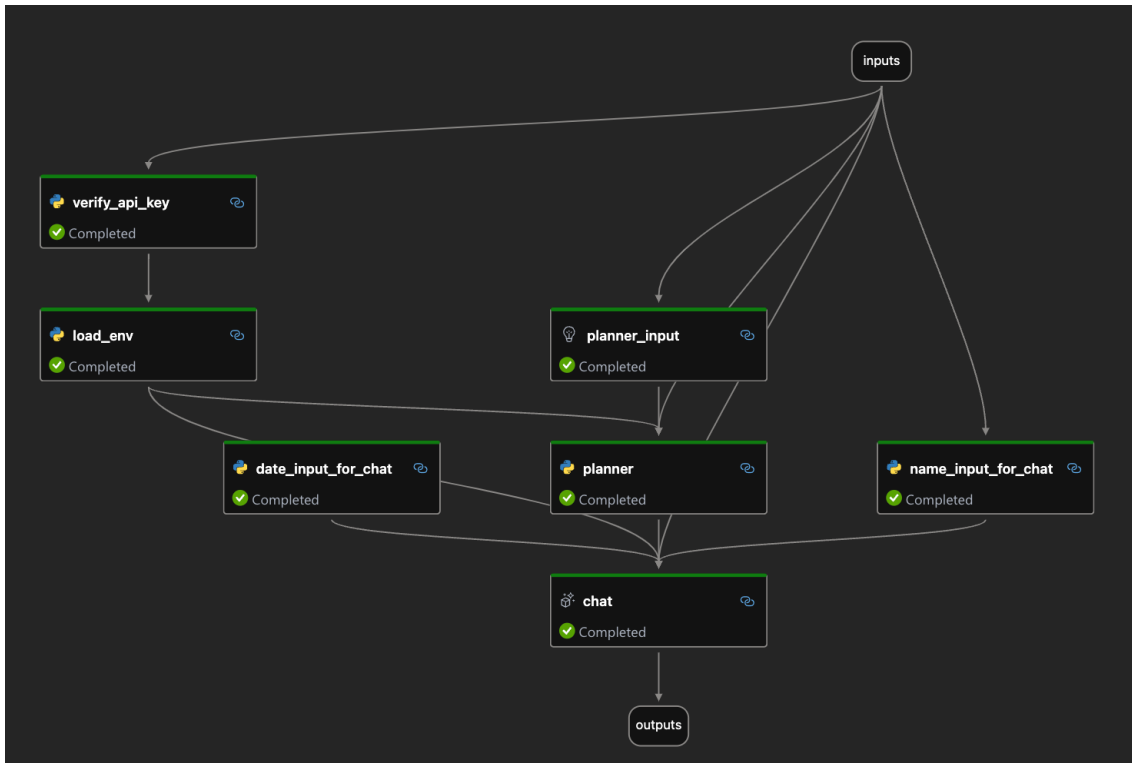


Figure 5.4: "Prompt Flow design"

### 5.1.6 Back-Office Portal

In the back-office portal, tenant members are responsible for checking the answer validation and assessing the response. If the answer is valid, it is approved and sent to the Outgoing Messages Event Hub. If incorrect or inaccurate, the response is rejected, and feedback is provided through the Messages Feedback system. This feedback helps improve the system. Back-office members can also take over the conversation and directly write to the user. Their messages can be written in their desired language and are then translated into the end-user's language.

### 5.1.7 Outgoing Messages Processor

The "Outgoing Messages Processor" function app processes messages in the Outgoing Messages Event Hub by forwarding the answer to the WhatsApp Business API and registering the delivery time in the database.

### 5.1.8 Design Choices

During the development of this system, numerous critical design decisions were made and rigorously tested to ensure optimal performance and reliability.

- **Addressing Duplicate Messages from WhatsApp Business API**

Initially, I encountered an issue with receiving duplicate messages from the WhatsApp Business API. The root cause was identified as a delay in returning the 200 "OK" status to the API, which led the API to assume the reception had failed and subsequently resend the message. This delay occurred because I was processing the message and retrieving the AI Engine's response before confirming successful reception. To resolve this, I implemented Event Hubs to queue incoming messages for later processing by Function Apps. This approach allowed me to promptly return the "OK" status to the API, thus preventing duplicates.

- **Manual Verification by Back-Office Users**

Another significant decision was the requirement for all messages to be verified by a back-office user initially. This measure was taken to mitigate the risk of the AI Engine disclosing sensitive information or providing incorrect responses. To ensure security and accuracy, every response from the chatbot must be reviewed and either approved or rejected by an HR member. This not only prevents potential mishaps but also facilitates better feedback and continuous improvement of the system.

- **Using Semantic Kernel for API Calls**

A pivotal change in the project involved deciding not to connect the AI Engine directly to the database or to index it. Instead, I opted to use the Semantic Kernel to determine the appropriate API call to fetch the necessary information. This decision was driven by concerns over information security and privacy. By avoiding direct database access, there's a significant reduction in the risk of compromising sensitive data, thereby enhancing the overall security of the system.

These carefully considered design choices have been instrumental in developing a robust, secure, and efficient system capable of meeting the stringent demands of the users.

## 5.2 WhatsApp Interface

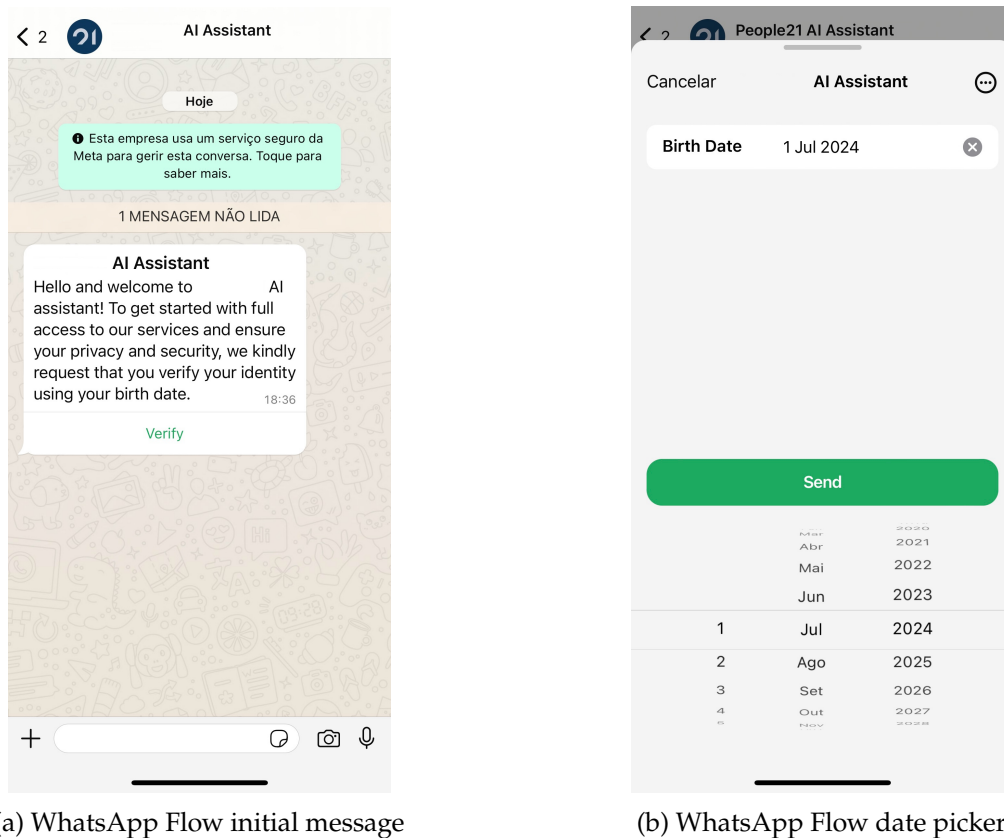


Figure 5.5: WhatsApp Flow

The WhatsApp Interface 5.5 is the unique interaction mechanism that the end-user has with this tool, and it is important that I make it as intuitive as possible. Upon sending a message for the first time, users are presented with a WhatsApp Flow that welcomes them to the AI Assistant and requests their date of birth to verify their identity. This flow is very simple to use and leaves low chances of error.

Once the user's identity is verified, they can proceed to ask questions, which will be answered by the chatbot. When a user initiates a chat, they are greeted by a welcome message. This message sets the tone and provides an overview of what they can expect from the AI Assistant.

The initial request for the user's date of birth ensures security and personalized interaction. This step is crucial for confirming the user's identity and tailoring responses appropriately. After verification, users can freely ask questions. The chatbot is equipped to understand and process these queries, providing accurate and relevant responses.

Answers from the chatbot are well-structured and easy to read. I use formatting techniques, such as bullet points and dots, to highlight important information and make the responses more digestible.

## 5.3 Back-Office Portal

The Back-Office Portal is the central hub where HR members and other relevant employees manage interactions between employees and the chatbot. I have designed it to be user-friendly and straightforward, as it won't be used frequently, minimizing the need for extensive training on this auxiliary tool. The portal can be used both on a mobile device and on a desktop, so I had a responsive approach to the entire website.

Always present on the screen is a top bar with the company logo and where you choose the language according to your preferences. A navigation bar is also present on the left side of the screen, in a side panel.

### 5.3.1 Conversations

The main component of the Back-Office portal is the Conversations section. Here, users can:

- **View All Conversations**  
Access a comprehensive list of all existing conversations, with filtering options to narrow down specific interactions (5.6).
- **Conversation Details**  
Click on any conversation to view detailed information. This includes key details at the top and the chat history in the center (5.7).
- **Approve/Reject Answers**  
Evaluate the responses provided by the AI Engine, approving or rejecting them as needed.
- **Takeover Conversations**  
Step in and manage the conversation directly, responding as a back-office member instead of the AI Engine. When releasing the takeover, the back-office member is asked if he wants the AI Engine to generate an answer if there are any questions to be answered. If he clicks to proceed, a request is made to the AI Engine, and the answer is saved in the database and forwarded to the Outgoing Messages Processor.
- **Review AI Answers**  
Analyze and review the answers given by the AI Engine to ensure accuracy and appropriateness.

### 5.3.2 User and Company Management

The portal also features sections for managing users and companies:

- **User List**  
View a list of all users registered in the external database (5.8).

ID	Topic	ID	Company	Last Message Time	Number of Messages	Status	Created At	Actions
80	Overstapt, What Now?	21550	Stipt Employment Services B.V.	01-07-2024 17:00	42	OPEN	01-07-2024 14:48	
79	Overstapt, What Now?	310	Stipt Employment Services B.V.	01-07-2024 14:24	34	OPEN	01-07-2024 08:05	
78	Process for Sick Leave	3532	Stipt Employment Services B.V.	28-06-2024 12:25	14	OPEN	28-06-2024 12:09	
75	Next Monday's Start Time	813	Stipt Employment Services B.V.	28-06-2024 11:30	2	OPEN	28-06-2024 11:30	
65	Dutch Language Request	22908	Quality Utzendbureau B.V.	28-06-2024 12:11	16	OPEN	14-06-2024 09:45	

Figure 5.6: Conversations Table

**Today's Planned Activities**

Name: Alexandru Aparatu | Mobile Number: +351932881240 | Created At: 01-07-2024 17:52 | Status: Open

View Translation for all messages

Hello, what am I doing today? 01-07-2024 17:52

Hello Alexandru,

Today, 2024-07-01, you have the following tasks:

- Your shift starts at 14:00 and ends at 22:00.
- You will be working at the "Sorteeren" department for "HB Bleiswijk & Detail Result Logistiek B.V."
- Your role is "Sorting".
- The work location is "Klappolder 110, Bleiswijk".
- You will be driving the vehicle with the plate "S-486-XHF".

Have a productive day!

Figure 5.7: Conversation Detail

- **User Details**  
View the details of the users registered and all conversations they had. Also has the ability to block users.
- **Company List**  
Access a list of all companies registered, available to tenant users (5.9).
- **Company Details**  
View the details of the companies registered and all existing users from that company.
- **User and Group Management**  
If you are a tenant administrator, you have additional capabilities to manage users and groups within your organization. This includes defining specific permissions for each user or group.

### 5.3.3 Settings and Profile Management

Additional functionalities are accessible via a small menu 5.10 when you click on your username in the navigation bar on the side panel.

Channel Type	Channel User	Name	Mobile Number	ID	Company	Created At	Verified	Blocked	Main Language	Last Message Time	Actions
📞	123	RandomEmployee Name	+315 967123123	321	Quality Uitzendbureau B.V.	10-04-2024 15:16	✔️		EN		ⓘ
📞	1234	John Doe	+315 967333333	4321	Quality Uitzendbureau B.V.	10-04-2024 15:16	✔️		EN		ⓘ
📞	12345	Robert Nixon	+315 967334455	54321	Quality Uitzendbureau B.V.	10-04-2024 15:16	✔️	🚫	NL		ⓘ
📞	1111	Test User	+315 967334455	552	Slipt Employment Services B.V.	29-04-2024 16:31	✔️		NL		ⓘ
📞	31617774984	Magdalena Barbara Chabior	+31617774984	813	Slipt Employment Services B.V.	24-05-2024 12:13	✔️		EN	28-06-2024 11:30	ⓘ
📞	31645582780	Vadim Chirau	+31645582780	29908	Slipt Employment Services B.V.	28-05-2024 14:04	✔️		NL	28-06-2024 12:10	ⓘ
📞	31623759277		+31623759277		Quality Uitzendbureau B.V.	04-06-2024 10:54			en		ⓘ
📞	351932881240	Alexandru Aparatu	+351932881240	21560	Slipt Employment Services B.V.	28-06-2024 12:38	✔️		Dutch	01-07-2024 17:52	ⓘ

Figure 5.8: Application Users Table

External Company ID	Name	Total App Users	Actions
1	Quality Uitzendbureau B.V.	4	ⓘ
2	Quality Uitzendbureau I B.V.	0	ⓘ
3	Slipt Employment Services B.V.	6	ⓘ
4	Tijdelijk ivm dubbel loonnummer	0	ⓘ
5	Quality uitzendbureau I B.V. (client)	0	ⓘ
6	Quality Uitzendbureau B.V. (client)	0	ⓘ
7	Slipt Employment Services B.V. (client)	0	ⓘ
8	DDD Personeel B.V.	0	ⓘ
9	DDD Personeel B.V. (client)	0	ⓘ
10	Stay21: Maintenance	0	ⓘ

Figure 5.9: Companies Table

- Logout**  
 Easily sign out of the Back-Office portal.
- Tenant Settings**  
 If you are a tenant administrator, you can modify settings specific to your tenant’s account.
- Profile Management**  
 Update your personal profile details.
- Change Password**  
 Change your account password to ensure security.

Overall, the Back-Office Portal is designed to streamline the management of chatbot interactions, user data, and organizational settings, providing HR members and administrators with the necessary tools to oversee and enhance their chatbot system effectively.

Name	Total App Users	Actions
Quality Uitzendbureau B.V.	4	
Quality Uitzendbureau I B.V.	0	
Stipt Employment Services B.V.	6	
Tijdelijk iem dubbel loonnummer	0	
Quality uitzendbureau I B.V. (client)	0	
Quality Uitzendbureau B.V. (client)	0	
Stipt Employment Services B.V. (client)	0	
DDD Personeel B.V.	0	
DDD Personeel B.V. (client)	0	
Stay21: Maintenance	0	

Figure 5.10: "Settings and Profile Management"

## 5.4 Available Questions

The initial proposal was to deliver 3 different topics that you could ask questions about and that is what is currently available.

The first topic I addressed was the work schedule of the employee. It is possible for the user to ask what company he will work for, where the company is located, what time he starts and finishes his shift, who the driver is that will take him there (if there is one), the vehicle plate number, and the job he will do. It is possible to ask for information on a specific date or week but also for relative dates like next week.

The second topic is related to information for the employee on how to proceed if he gets sick, including which actions he needs to take, who to inform, and what benefits he has. This ensures that employees are aware of the necessary steps to follow in case of illness, helping to streamline the process and avoid any confusion or delays.

The third and last topic is about the salary of the employee. Users can inquire about various aspects related to their compensation, such as the gross salary, net salary, hours and days worked, and distribution of hours by company. They can also filter by month or week and see details of the hours worked like overtime hours, normal hours, etc. This topic aims to provide clear and detailed information regarding the employee's earnings, ensuring transparency and helping employees manage their finances effectively.

By covering these three critical areas, this proposal ensures that employees have access to essential information that affects their daily work life, health, and financial well-being. There will be a more detailed look at the questions in chapter 7, Evaluation.

## 5.5 Alternatives

When deciding on the architecture of the project, I asked myself: "Why use this and not other solutions?" This was an essential question, especially in the context of developing an advanced and scalable architecture that meets the specific requirements. Here's a detailed breakdown of why I chose certain technologies over others and the trade-offs involved.

### 5.5.1 Langchain

Langchain is a well-known framework for building applications that utilize large language models (LLMs). It provides a robust and flexible interface for chaining together different tasks, which can be useful for many AI-driven applications. However, I opted for Semantic Kernel instead, and here's why:

- **Integration**

While Langchain is powerful, Semantic Kernel is more optimized for seamless integration with the existing infrastructure, particularly when working with Microsoft-based environments. The interoperability with Azure services is a significant advantage. This ensures that this solution can leverage the full power of the Azure ecosystem without compatibility issues.

- **Scalability**

Semantic Kernel offers a better scaling strategy when working with large datasets and complex workflows. Langchain, while flexible, can become cumbersome when scaling up, especially in environments that demand high concurrency and real-time processing.

- **Development Speed**

Given that I am more familiar with the Microsoft stack, using Semantic Kernel reduces the learning curve, allowing me to deploy solutions faster and more efficiently.

In summary, while Langchain has its strengths, Semantic Kernel was a better fit for this specific use case due to its seamless integration, scalability, and faster development cycles.

### 5.5.2 Ollama

Running a local model using Ollama would have provided me with complete control over the AI models, including data privacy and reduced dependency on cloud providers. However, I opted to use Azure models for several reasons:

- **Resources consumption and Maintenance**

Running AI models locally demands significant computational resources, which is expensive and challenging to maintain. Azure provides scalable compute resources that automatically adjust based on demand, ensuring that the models run efficiently without the overhead of managing local hardware.

- **Model Updates and Management**

Azure's AI services offer regular updates and improvements to models, ensuring I always have access to the latest advancements in AI. This is harder to achieve with local models, where updates need to be manually managed, leading to potential lags in incorporating the latest AI capabilities.

- **Security and Compliance**

Azure provides robust security measures and compliance certifications, which are crucial for handling sensitive data. While running models locally gives more control over data, it also increases the responsibility for ensuring that all security protocols are up to date, which can be a significant burden.

Thus, while Ollama offers local model advantages, the benefits of using Azure models in terms of scalability, maintenance, and security outweigh the trade-offs for this project.

### **5.5.3 Web Apps instead of Function Apps for the Message Processors**

I could have used web apps to process incoming and outgoing messages but instead, decided to implement function apps. The decision was driven by several factors. First, in terms of cost efficiency, function apps operate on a pay-per-use basis, meaning that you only pay for the actual compute time consumed during the execution of functions. This is more cost-effective compared to web apps, which run continuously and have fixed costs regardless of the actual usage.

Additionally, scalability and management played a crucial role in this choice. Function apps automatically scale out based on the volume of incoming events. This capability is particularly useful for processing spikes in data traffic without the need to manually manage the scaling process, which would be more complex and resource-intensive with web apps. In a web app, I would need to manually adjust the service, leading to a slow response and probably downtime. There's also the option for the web app to automatically scale but that is very costly so, in the long run, it would not be financially viable.

In conclusion, function apps provide a more cost-effective and scalable solution for this specific processing needs.

## IMPLEMENTATION

In this chapter, we look into the technical intricacies of the development process of the conversational agent. This chapter provides a comprehensive walkthrough of the implementation stages, detailing the methodologies used. We will explore the codebase, configurations, and system, offering insights into the practical aspects of transforming the conceptual solution into a functional application.

The implementation journey is marked by a systematic approach, beginning with the setup of the development environment and progressing through the integration of various system components.

Moreover, this chapter will address the challenges encountered during development and the strategies employed to overcome them. We will discuss technical hurdles such as system integration issues, data privacy concerns, and others, presenting the solutions that ensure the robustness and reliability of the final product.

### 6.1 Environment Setup

The environment setup for this project involved several critical steps to ensure a seamless and efficient development process. This included configuring the frontend with Vue, Vuetify, and Pinia, setting up the backend with Django and GraphQL, establishing a local environment for PromptFlow and Semantic Kernel, organizing the code repositories, and setting up an Azure DevOps board for project management.

#### 6.1.1 Back-end Setup

I started with the back-end, Django was installed and a new project was created. Graphene-Django was then added to enable GraphQL support. Configuration adjustments were made in the settings to integrate Graphene-Django, ensuring that the backend could handle efficient and flexible data queries. This setup provided a robust foundation for the backend infrastructure. After that, I went to configure the Azure SQL Database in the Azure portal and configured Django to connect to it.

## 6.1.2 Frontend Setup

To configure the frontend, the first step was installing Vue CLI to scaffold the project. Following this, Vuetify was added to provide a set of pre-designed UI components. Pinia was then integrated to manage the application state effectively. This setup ensured that the frontend development environment was equipped with all the necessary tools for building a modern, responsive user interface. Pinia is an innovation in Vue3 (before it was Vuex) and due to a lack of guides, had some issues configuring the stores to be able to work the way I intended to.

## 6.1.3 Local Environment for PromptFlow and Semantic Kernel

Setting up the environment for PromptFlow and Semantic Kernel encountered several issues but was successful when I found DevContainers. With them, the process was very simple. I simply installed Visual Studio Code Extension for PromptFlow and installed all requirements using PIP.

## 6.1.4 Code Repositories

Separate repositories 6.1 were set up for each component of the project, including the AI Engine, WebApp, Information Retrieval Module, WhatsApp Webhook, Incoming Messages Processor, and Outgoing Messages Processor. This modular approach allowed for better organization and management of the codebase, facilitating independent development and maintenance of each component.

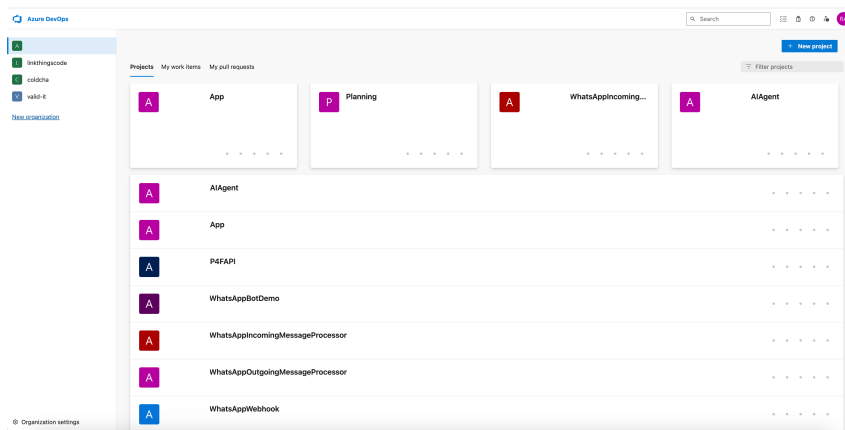


Figure 6.1: "Azure DevOps Repository"

## 6.1.5 Azure DevOps Board

An Azure DevOps board 6.2 was established to manage the project workflow. This involved setting up the board to register user stories, track bugs, and monitor project progress. The board provided a centralized platform for task management and coordination, helping to keep the development process organized and on track. This board was used in the entire

development of the project, from research to know more about the theme, gathering the best tools, developing the components, and tracking issues to solve.

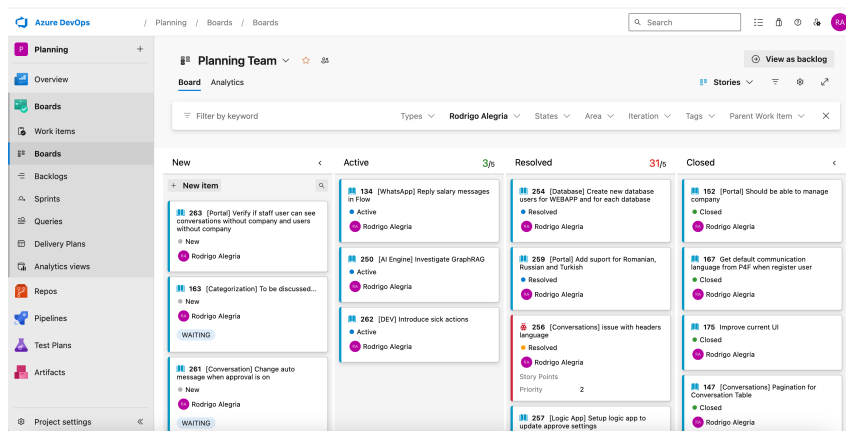


Figure 6.2: "Azure DevOps Planning Board"

### 6.1.6 Azure Functions Apps and Event Hubs

Lastly, I created the 3 function apps necessary for the project (the Information Retrieval Module, Incoming Messages Processor, and Outgoing Messages Processor) and the 2 Event Hubs (Incoming Messages Queue and Outgoing Messages Queue). These tools allowed me to lower costs, worry less about maintenance, and really focus on the logic of the project.

## 6.2 Database

The database for the application was created using Azure SQL Database. The database platform used is Microsoft SQL Server. Due to the default lack of support for Microsoft SQL Server in Django, I utilized the “mssql-django” third-party package. This is the recommended package by Django and ensures full compatibility.

Each table in the database corresponds to a class in Django’s ORM. This object-relational mapping allows us to define the database schema through Python classes, making development more intuitive and integrated with the rest of the Django application.

To enhance the functionality and maintainability of the models, I have implemented three auxiliary models: **TimeStampedModel**, **SoftDeletableModel**, and **OperatorModel**. The **TimeStampedModel** includes the fields **created\_at** and **modified\_at**, which automatically track when an object is created and last modified. This is crucial for audit trails, debugging, and understanding the history of data changes. The **SoftDeletableModel** includes the field **is\_removed**, implementing a soft delete mechanism. This allows records to be marked as deleted without physically removing them from the database, ensuring data can be recovered if deleted by mistake, thus maintaining data integrity and history. The **OperatorModel** includes the fields **created\_by** and **modified\_by**, tracking which user

created or last modified a record. This is important for accountability and managing user actions within the system.

```
1 class Company(TimeStampedModel, SoftDeletableModel, OperatorModel):
2     # Relations
3     tenant = models.ForeignKey(
4         Tenant,
5         related_name="company_set",
6         verbose_name="Tenant",
7         on_delete=models.CASCADE,
8     )
9
10    # Attributes
11    name = models.CharField("Name", max_length=255, blank=False, null=False)
12    external_company_id = models.CharField(
13        "External Company ID", max_length=255, blank=True, null=True
14    )
15
16    def __str__(self):
17        return f"{self.id}"
18
19    @property
20    def raw_id(self):
21        return self.id
22
23    class Meta:
24        permissions = ((ChatPermissions.MANAGE_COMPANY.codename, "Manage Company"),)
```

Listing 1: Company Model

The Company class is an example of how I structure the database models. This class inherits from the three auxiliary models. It includes a foreign key to the Tenant model, indicating that each company is associated with a tenant and as attributes, it has **name** and **external\_company\_id**, which are essential attributes for identifying and managing companies.

I also make use of model properties (identified in the example with the decorator **@property**), which are a way to define a method in a model class that behaves like a read-only attribute. It allows us to compute values on the fly based on other fields in the model without storing them in the database. Properties can add complex logic to the models and keep the code organized and clean. In the example I use it to return the primary key as, in Graphene, the id used for operations is the global identifier of the table entry.

Another relevant piece of code I added above is the Meta class. It is used to define metadata for the model. This metadata can include options like ordering, unique constraints, and permissions. The permissions attribute in the Meta class is a way to define custom permissions that can be assigned to users or groups. This is needed to provide the MANAGE permission, which includes all view, add, change, and delete permissions.

Whenever I made a model change, a corresponding migration was generated and applied to the database. Migrations are essential as they allow us to evolve the database

schema over time without losing data. They track changes to models and translate them into SQL commands that update the database schema.

The most complex model in the system is the **User** model. This model is central to the application and is designed to handle user authentication, profile information, and various user-related functionalities. The User model extends Django's `AbstractBaseUser`, which provides the core implementation of a user model, including hashed passwords and authentication methods. This extension allowed me to customize the user model while leveraging Django's built-in authentication framework. I decided to override the `delete` function in the User model to change the user's email before deletion. This is necessary because the email field is unique, and if the user wants to create a new account with the same email after deletion, there would be a conflict because of the soft delete. By appending a timestamp to the email before deleting the user, I ensure that the email can be reused.

```

1 def delete(self, *args, **kwargs):
2     """
3     Change email before delete, to avoid email already exists error
4     """
5     self.email = self.email + datetime.now(timezone.utc).strftime("%Y%m%d%H%M%S")
6     super().delete(*args, **kwargs)

```

Listing 2: Overridden Delete Function

The User model also includes several properties that add functionality and convenience like `is_tenant_staff` that determines if the user is part of the tenant staff, `tenant` that retrieves the tenant associated with the user, `get_full_name` that joins the `first_name` and `last_name` fields, and many others.

## 6.3 Information Retrieval Module

The implementation of the Information Retrieval Module [IRM](#) was a comprehensive and intricate process. It is also an Azure Function App to enable seamless integration with the external datasources and efficient retrieval of employee-related information. To connect to these databases, I employed `SQLAlchemy`, which facilitated the declaration of database tables and the creation of necessary functions for data processing. Given that the databases were in Dutch, I had to map and translate these schemas into English to ensure smooth integration and readability.

I interfaced with two databases: [P4F](#) which is the main database, that contains employee data and work planning information, and [NoCore](#), a secondary database only used for housing salary data. These are the external databases supported at the moment but it's expandable to others.

Each function in the [IRM](#) is triggered by [HTTP](#) requests. I developed six distinct functions to cater to different data retrieval needs. All the following functions that need

some time-related filter, are filtered by week because it is usual in the Netherlands and it is how the external databases were designed.

- **get\_employee\_by\_employee\_id**  
Retrieves employee details using their ID. Returns general information about the user like First name, Last name, Company, Communication language...
- **get\_employee\_by\_mobile\_number**  
Gets employee information based on their mobile number.
- **get\_employee\_week\_salary\_by\_employee\_id**  
Obtains weekly salary data for an employee using their ID. The data retrieved contains worked hours, worked days, gross salary, net salary and projects worked (with details of each company where the user worked and hours that did for them)
- **get\_employee\_week\_work\_plannings\_by\_employee\_id**  
Fetches weekly work planning details for an employee using their ID.
- **get\_all\_companies**  
Retrieves all companies for which the employees works for and its main function is to keep the database updated with all the companies.
- **get\_all\_employee\_current\_contracts\_by\_employee\_id**  
Fetches all current contracts for employees.

Here is the implementation of this last one, a very simple function that retrieves all current employee contracts by querying the [P4F](#) database, filtering contracts based on their start and end dates. This function is very useful for keeping the database up to date with the company for which the user works for.

The `get_employee_week_salary_by_employee_id` function is notably the most complex one. While the others only access a few tables, this function requires using the employee ID from one database ([P4F](#)) to retrieve corresponding salary details from another database (NoCore). The complexity arises from this need to cross-reference IDs and find the relevant tables.

I spent numerous hours studying the database schemas, relationships, and data points to accurately map and translate the necessary information. This meticulous process ensured that the functions could retrieve and process data efficiently and correctly.

## 6.4 WhatsApp WebHook

The WhatsApp Webhook is the entry point of the system. To set it up, I first created an app on Meta Developers. This is where I can access the API settings, configure the webhook settings properly, and create Flows.

```

1 # get all employees' company of current contract
2 @app.route(route="get_all_employee_current_contracts")
3 def get_all_employee_current_contracts(req: func.HttpRequest) -> func.HttpResponse:
4     result_json = []
5     try:
6         with Session(p4f_engine) as session:
7             employee_contracts = (
8                 session.query(EmployeeContract)
9                 .filter(EmployeeContract.start_date <= datetime.now().date())
10                .filter(
11                    or_(
12                        EmployeeContract.end_date == None,
13                        EmployeeContract.end_date >= datetime.now(),
14                    )
15                )
16                .all()
17            )
18            for employee_contract in employee_contracts:
19                result_json.append(
20                    {
21                        "employee_id": employee_contract.employee_id,
22                        "company_id": employee_contract.company_id,
23                    }
24                )
25            except Exception as e:
26                logging.error(f"Error: {str(e)}")
27                return func.HttpResponse(f"Error: {str(e)}", status_code=500)
28
29            return func.HttpResponse(
30                json.dumps(result_json),
31                status_code=200,
32                mimetype="application/json",
33            )

```

Listing 3: Function get\_all\_employee\_current\_contracts

The solution for the webhook is a web application built using Flask. [Flask](#) is a lightweight web framework for Python that is ideal for smaller applications due to its simplicity and lower overhead compared to Django. Flask's minimalistic design made it a perfect choice for the needs.

The webhook initially handles a GET request for verification. When WhatsApp sends a verification request, the app responds appropriately to confirm that it is set up correctly. Once verified, the webhook starts receiving messages from WhatsApp users.

Each incoming message is sent to an asynchronous function designed to process the message and immediately return a successful response (HTTP 200). This approach helps prevent the issue of duplicate messages, which can occur if the processing takes too long and WhatsApp resends the message.

To ensure reliability and data integrity, all messages received throughout the day are stored in a Blob Storage, more specifically in a JSON file. This storage solution allows the reprocessing of the messages in case of a system error. Finally, every incoming message is forwarded to the Event Hub dedicated to incoming messages, ensuring that the message processing system remains scalable and efficient and continues the message lifecycle.

```
1 @app.route("/webhook", methods=["POST", "GET"])
2 def webhook():
3     if request.method == "GET":
4         return verify(request)
5     elif request.method == "POST":
6         asyncio.run(send_eventhub_message(request.get_json()))
7         return jsonify({'success':True}), 200, {'ContentType':'application/json'}
```

Listing 4: Webhook handler

## 6.5 Incoming Messages Processor

The Incoming Messages Processor is a crucial component in the messaging system. It is the core component of handling incoming messages received through WhatsApp with efficiency and precision. Implemented as an Azure Function App, this processor is triggered by the **WhatsAppIncomingMessageEventHubTrigger**, which is activated by the Event Hub queue. This function serves as the initial entry point for all incoming messages, facilitating seamless connectivity with the information retrieval module and databases.

The architecture of the Incoming Messages Processor relies on two main connectors: the **internal\_database\_connector** and the **external\_database\_connector**. Each plays a vital role in the overall functionality of the processor.

- **Internal Database Connector**

I utilized SQLAlchemy to establish a robust connection to the internal database. SQLAlchemy allowed the declaration of the necessary tables and definition of functions required by the Processor. This ensured smooth and efficient interaction with the database, allowing it to perform various operations, such as querying user data, updating user status, and storing messages.

- **External Database Connector**

The external database connector consists of a set of functions leveraging Python's requests library to communicate with the Information Retrieval Module. This module is essential for fetching user data required for processing incoming messages. By using the requests library, I ensured that the system could make HTTP requests to external services, retrieve necessary information, and integrate it into the processing workflow.

Upon receiving a message, the Incoming Messages Processor first identifies its type. Messages are classified as either Interactive (WhatsApp Flow) or Text. This categorization is crucial as it determines the subsequent processing path for each message.

### 6.5.1 Text Messages

The processing of text messages involves several steps to ensure accurate handling and appropriate response generation. The first step is user verification. For this step, I identified

four main user types and the processor's respective behaviors in handling their messages:

- **New Users**

If the user does not exist in the database, I create a new user. This process involves capturing several details, including **main\_language**, **mobile\_number**, and **company**. If the specified company does not exist in the records, it is created. After creating the user, I reply with a WhatsApp Flow to verify the user's identity.

- **Unverified Users**

For users who exist in the database but are not verified, it triggers the WhatsApp Flow to prompt verification. As an additional security measure, I require users to re-verify their identity every 30 days. If a user has not verified within the last 30 days, they are prompted to do so again.

- **Blocked Users**

Users flagged as blocked in the database receive a message informing them of their status. The message advises them to contact support if they believe the blocking is an error.

- **Verified Users**

For users who are verified within the last 30 days, the Processor does a series of actions. First, it identifies the topic of the message by making a call to a language model. The message is then stored in the database. If the message is new or if the last message is over 6 hours old and was not sent by a user, a new conversation is created. This is done so it never closes an active conversation or with unanswered messages. Then the system checks whether the conversation has been taken over by a back-office member. If it has, the process ends here because the responsibility to answer it is to the back-office member instead of the AI Engine. If not taken over, the chat history is retrieved and formatted for the AI Engine. It is then sent a POST request to the AI Engine, including **employee\_id**, **question**, **chat\_history**, and **api\_key**. Based on the response, the Processor identifies the language and saves the answer in the database. Then, if the answer doesn't need to be approved by the back-office members, it proceeds to the function responsible for preparing the data for the Outgoing Messages Processor.

### 6.5.2 Interactive Messages

The Flow messages are exclusively used for user verification. If the user is already verified, the flow is ignored.

Otherwise, the timestamp from the flow message is adjusted to account for time zone discrepancies, assuming the message originated from the Netherlands. This adjustment is crucial due to the challenges with WhatsApp and time zones. It is not possible to get the user's timezone. The user's date of birth is verified using the Information Retrieval

Module. If the provided date is incorrect, the user is prompted to re-enter their date of birth.

Once the user's date of birth is successfully validated, the user's status is updated to verified. A personalized welcome message is then sent in their native language, utilizing Neural Machine Translation provided by a language model.

The flow was created using Meta Developers' platform 6.3 and is structured in JSON format. These flows allow for dynamic interaction with users, enabling the collection of input through various forms such as date pickers and images. For example, the "VERIFICATION" screen in this flow uses a DatePicker for the user's birth date and an Image element for displaying the application logo. The interaction concludes with a Footer component that triggers an action based on the user's input.

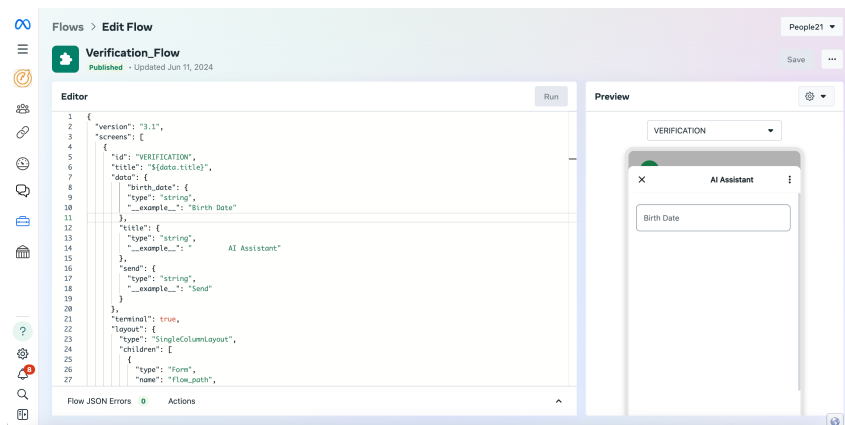


Figure 6.3: "Meta Developers portal with Verification Flow"

### 6.5.3 Final Steps and Logging

After processing either a text or a flow message, the response is forwarded to a function responsible for creating the payload for the WhatsApp Business API.

When implementing the payload construction and creating the queue object to manage different types of messages, I focused on ensuring that each piece of data was appropriately structured and contained all the necessary information for the Outgoing Messages Processor to function correctly.

The **data** dictionary is crafted to serve as the payload that will be sent to the WhatsApp Business API. This payload includes all the essential details required to dispatch a text message through WhatsApp.

The **"messaging\_product": "whatsapp"** field indicates that it is using the WhatsApp messaging product. The **"to": from\_number** field specifies the recipient's phone number, with **from\_number** holding the actual number of the user that sent the message and will now get the reply. The **"type": "text"** field denotes that the message being sent is a text message, which is necessary for the API to process the correct message format. The **"text": "body": response["answer"]** field contains the actual text of the message, dynamically

```

1 data = {
2     "messaging_product": "whatsapp",
3     "to": from_number,
4     "type": "text",
5     "text": {"body": response["answer"]},
6 }

```

Listing 5: Send Message Request Body

filled with `response["answer"]`, the response generated by the handlers of Text/Interactive messages. For example, if the message was a question for the AI Engine, in the field the answer would be the answer.

Next, I created the `queue_object` to manage the message within the system. This object can handle different types of queue items, specifically `whatsapp_message` or `whatsapp_flow`.

```

1 queue_object = {
2     "type": "whatsapp_message",
3     "phone_number_id": phone_number_id,
4     "from_number": from_number,
5     "payload": json.dumps(data),
6     "user_id": user.id,
7     "message_id": response["answer_message_id"],
8 }

```

Listing 6: Outgoing Messages Processor Input Object

The `"type"` field specifies the type of queue object. The `"phone_number_id"` serves as the identifier for the phone number that received the message from the user, essential for answering the user from the same device that he sent the message to. The `"from_number"` is again the recipient's phone number. The `"payload"` contains the payload for the WhatsApp Business API. The `"user_id"` includes the user ID associated with the user that sent the message. Finally, the `"message_id"` holds the message ID from the response, which is crucial for tracking and reference purposes, ensuring each message can be accurately identified within the system and its status updated.

This object is then sent to the Outgoing Messages Event Hub for further processing by the Outgoing Messages Processor.

Extensive logging is incorporated throughout the processing stages to ensure transparency and facilitate debugging. Logs are maintained to track message flow, monitor system performance, and diagnose any issues that arise. These logs are invaluable for maintaining the health of the system and ensuring that any anomalies are promptly addressed. The logs are divided by:

- **Entry and Exit Points**

Logs are created at the entry and exit points of the processor to trace the flow of messages. This helps in understanding the journey of each message through the system.

- **User Verification Steps**

Each step in the user verification process is logged. This includes checking the database for user existence, initiating verification flows, and updating user status. These logs help in tracking the verification process and identifying any issues.

- **Error Handling**

Any errors encountered during message processing are logged with detailed information. This includes errors in database connectivity, issues with external API requests, and problems in AI Engine interactions. Logging these errors helps in quickly identifying and resolving issues.

- **Message Storage and Retrieval**

The steps involved in storing messages in the database and retrieving chat history are logged. This ensures that all message-related activities are traceable and verifiable.

- **AI Engine Interaction**

Interactions with the AI Engine, including sending requests and receiving responses, are logged. This helps in monitoring the performance of the AI Engine and understanding its impact on the overall processing. If the response from the AI Engine is not successful, an alert email is sent to the app administrators, the error is logged in the Message table, and a fallback message is sent to the user.

- **Outgoing Message Preparation**

The creation of payloads for the WhatsApp Business API and their dispatch to the Outgoing Messages Processor are logged. This final step ensures that the entire process, from message receipt to response dispatch, is fully documented.

This comprehensive approach to logging not only helps in maintaining system integrity but also provides a clear audit trail for all operations, contributing significantly to the reliability and transparency of the Incoming Messages Processor.

## 6.6 AI Engine

The development of the AI engine, which is the core component of the system, involved integrating various advanced components and techniques to create a robust and efficient system. The Semantic Kernel serves as the backbone, orchestrating the flow of information and leveraging GPT-4o for all language model calls. The External Database Connector utilizes the Python requests library to interact with the Information Retrieval Module, ensuring seamless access to user data. This section will explore in depth the image 5.4 referenced in last chapter.

### 6.6.1 API\_KEY Validation

The AI Engine workflow starts upon receiving an HTTP request. The first step is to validate the API Key. A Python script ensures the API KEY is valid and authorized to access the system. If the validation fails, the script raises an exception immediately, preventing further processing and resource expenditure on the request. This step is crucial for maintaining security and preventing unauthorized access to the system.

### 6.6.2 Loading Deployment Name

The deployment name of the language model, **GPT-4o**, is loaded through a specialized process. This step is distinct from other environment variable loading procedures (that is through a settings JSON file) due to its importance in the flow settings. The unique handling addresses issues related to PromptFlow parameters, ensuring that the deployment name is correctly integrated into the system's workflow. This step is also done using a Python script.

### 6.6.3 Formulating the Planner Prompt

I have a specialized prompt for giving the right context and detailed instructions for feeding the Semantic Kernel planner. This is one of the places where I constantly added/removed instructions in order to achieve a better result.

```

1 #System:
2 As an AI assistant, your task is to provide work planning information such as daily start times,
3 client details, vehicle assignments, driver schedules, wage, time worked, etc.
4 If there is no mention to year or month,
5 assume the user is referring to the current year or current month.
6 If there is chat history, formulate a standalone question to solve.
7
8 Chat history:
9 {% for item in chat_history %}
10 {% if item.role == 'user' %}
11 user:
12 {{ item.content }}
13 {% elif item.role == 'system' %}
14 assistant:
15 {{ item.content }}
16 {% endif %}
17 {% endfor %}
18
19 Question: {{ question }}
```

Listing 7: Planner Prompt

### 6.6.4 Finding and Executing the plan

The formulated prompt is passed to the Planner, the core component of the Semantic Kernel responsible for orchestrating the execution of tasks. The Planner maintains a list

of plugins, each containing specific functions tailored for different tasks. Currently, there are 6 plugins, one for each topic that the conversational agent handles and 3 more for a better answer like Weather data.

- **P4F Plugin** is designed to handle queries related to the employees' work planning. This plugin provides functions that retrieve weekly and daily work planning information, making it a critical component for managing and communicating employee schedules.

Functions available are: **WeekWorkPlannings** which returns the work planning for the specified week. It takes the week number and the year as arguments and provides a detailed schedule for that week, **DayWorkPlanning** that provides the work planning for a specific day and **NextWeekWorkPlannings** fetches the work planning for the upcoming week.

This plugin is crucial for ensuring that employees are informed about their schedules in advance, improving planning and efficiency. By providing detailed and timely work planning information, this plugin helps in reducing confusion and enhancing operational smoothness.

Example:

**Question:** "What's my work planning for tomorrow?"

**Planner Function:** day\_work\_plannings(2024-08-01)

**Planner Output:**

```
{  
  "employee_name": "Glebs Supenko",  
  "work_date": "2024-07-29",  
  "day_of_week": "Monday",  
  "start_time": "06:00",  
  "end_time": "15:00",  
  "vehicle": "S-492-ST",  
  "driver": "Glebs Supenko",  
  "driver_mobile_number": "+351123456789",  
  "client_name": "Velder",  
  "department_name": "Kantoor",  
  "work_role": "Flow Controller",  
  "address": "Marshallweg 6, Bodegraven"  
}
```

Listing 8: P4F Day Work Planning Output

- **Wage Plugin** is focused on handling queries related to employee salary, time worked, and project involvement. This plugin ensures that employees have access to detailed and accurate information about their earnings and work hours.

Functions available are: **week\_salary** which returns the salary, worked time, and projects for a specific week of the year. This function requires the year and week number as arguments, **month\_salary** provides the salary, worked time, and projects

for a specified month and year and **receipt\_info** delivers detailed information about the employee's receipt or bill, addressing common questions and clarifications about the salary components.

This plugin is essential because, by providing detailed and easily accessible wage information, it helps to resolve queries and concerns related to salaries, thereby improving employee satisfaction and reducing the administrative burden which is one of the main goals of this project.

Example:

**Question:** "What was my salary in week 24 of 2024?"

**Planner Function:** year\_week\_salary(202424)

**Planner Output:**

```
{
  "week_number": 202424,
  "worked_days": 5.0,
  "worked_hours": 36.0,
  "gross_salary": 666.53,
  "net_salary": 363.9,
  "projects": [
    {
      "project_id": "U201900001",
      "customer_name": "Velder B.V.",
      "hour_description": "Verlof",
      "worked_hours": 36.0
    }
  ]
}
```

Listing 9: Wage Year Week Salary Output

- **Sick Information Plugin** is designed to provide employees with information and guidance on what to do if they fall sick. This plugin ensures that employees have clear instructions and necessary details related to sick leave procedures.

There is only one function available **whattodoifimsick** which returns all available information on the steps an employee should take if they are sick. This includes details on reporting the illness, documentation required, and relevant company policies.

Despite being the smallest plugin, it plays a vital role in ensuring that employees are well-informed about the procedures to follow when they are unwell. By providing clear and concise information, this plugin helps in minimizing confusion and ensures that employees can take appropriate actions promptly.

Example:

**Question:** "I am sick, who should I contact?"

**Answer:** "Since you are sick, you should: notify your employer as early as possible, before 10:00 a.m., and at least 30 minutes before the commencement of work, inform

the client about your inability to work due to illness. For further assistance, you can contact the Work Planning department via WhatsApp at +316 12 12 12 12 between 5:30 and 20:00. I wish you a speedy recovery!"

- **Time Plugin** provides functions to retrieve the current time and date, formatted in various ways. This plugin is useful for contextualizing responses and ensuring that time-sensitive queries are accurately addressed. This plugin is based on one of the Semantic Kernel core plugins, with some improvements made by me. By default, the week number was not used much in the plugin so I extended it.
- **Weather Plugin** provides functions to retrieve current weather conditions and forecasts. This plugin is useful for queries related to weather updates and planning activities that depend on weather conditions.
- **Math Plugin** is a built-in plugin from the Semantic Kernel that provides functions for performing mathematical operations. This plugin is useful for queries that require calculations and numerical data processing.

Then, having the context and the list of functions from these plugins, it makes a call to the model for him to create a plan for solving the query. This plan (that is a collection of function callings) is then executed in a sequential order, with each function performing its task and passing the results to the next function as needed until it gets a result that is forwarded to the next step.

This plugin-based methodology is one of the key factors for choosing Semantic Kernel because of its easy scalability.

### 6.6.5 Final Prompt and Response Generation

The results, along with additional context such as the employee's name, today's date, and chat history, are then included in a final prompt to generate a user-facing response. This final prompt ensures a clear, concise, and contextually relevant response, tailored to the user's query.

The response generated from making a call to the model using the prompt is sent as the answer to the HTTP request that started the whole flow.

### 6.6.6 PromptFlow Tracing

The PromptFlow Tracing Tool [6.4](#) is an integral part of the development and debugging process for the AI engine. This tool provides a detailed visualization of the execution flow, allowing the trace of each step of the AI Engine's decision-making process. In complex systems like the one I have implemented, where multiple functions, plugins, and API calls interact, having a tool that provides insight into the internal workings of these processes is invaluable.

```

1  #System:
2  You are a helpful, creative, clever, and very friendly AI assistant.
3  You help the user find answers in context and previous messages.
4  If the user is sick, assume it is short-term and wish them a speedy recovery.
5  When answering, do not mention the source of the information/context.
6  Prefer short and clear sentences and use bullet points if possible.
7  Refer to the user as "the driver" if they are the driver.
8  You provide work planning information like start times, client details,
9  vehicle assignments, and schedules.
10 You can also provide wage info and time worked in a week or month.
11 In wage or time questions, days = number of days worked, hours = number of hours worked.
12 In worked time questions, list projects with hours worked in each.
13 If there's no wage info for certain weeks, inform the user.
14 If no wage info is available, provide the last known week's details.
15 For unknown pick-up times, give the driver's name and number, and suggest confirming with them.
16 For non-work or non-health topics, say it is not your area of expertise.
17 If lacking info, advise contacting the relevant department.
18
19 Work Planning: Phone: +316 12 12 12 12 via WhatsApp; Availability: 5:30 - 20:00
20 Front Office: Phone: +316 13 13 13 13 via WhatsApp, Availability: Monday to Friday, 9:00 - 17:30
21 Housing Information: Phone: +316 14 14 14 14 via WhatsApp, Availability: 8:00 - 17:30
22
23 If no department fits, suggest contacting the front office.
24 Translate context if needed.
25 Today is: {{today}}
26 Employee: {{employee}}
27 Employee ID: {{employee_id}}
28 If questions involve other employees, say you're not allowed to provide their information.
29
30 Context:
31 {{ system_answer }}
32
33 Chat History:
34 {% for item in chat_history %}
35 {% if item.role == 'user' %}
36 User: {{ item.content }}
37 {% elif item.role == 'system' %}
38 Assistant: {{ item.content }}
39 {% endif %}
40 {% endfor %}
41
42 User Question: {{question}}

```

Listing 10: Final Prompt to generate answer

One of the primary features of the Tracing Tool is its ability to provide a step-by-step execution overview. The tool captures and displays the sequence of functions and API calls as they are executed within the flow. For each function, it shows the start and end times, which helps me understand the execution timeline and identify any bottlenecks or delays.

Performance metrics are also a significant aspect of PromptFlow Tracing. The tool provides timing information for each function, showing how long each step takes to complete. This data is critical for performance tuning, allowing developers to optimize the flow for faster execution. Understanding which parts of the flow are taking the most time can lead to targeted improvements, such as refactoring slow functions, optimizing

database queries, or adjusting the flow logic.

For example, in the image below, the trace tool shows a flow where the `verify_api_key` function successfully validates an API key, followed by the `planner_input` function preparing the input for the planning process. Each subsequent step is clearly outlined, including interactions with the GPT-4o model through `openai_chat_async` calls.

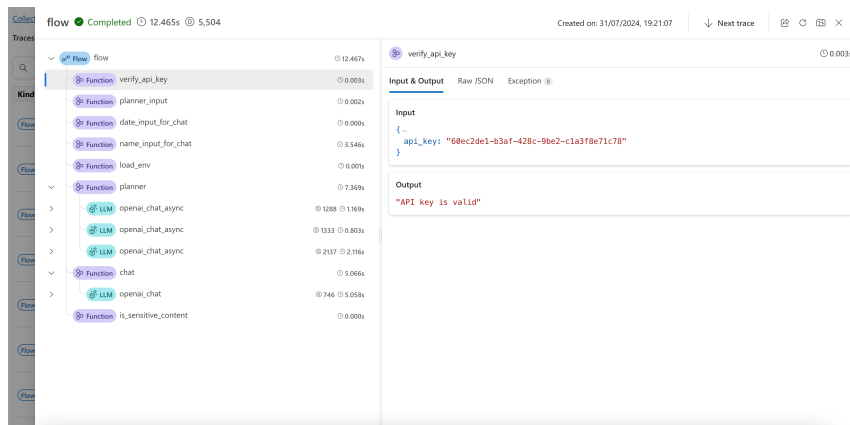


Figure 6.4: "Prompt Flow Tracing Tool"

## 6.7 Outgoing Messages Processor

Similar to the Incoming Messages Processor, the Outgoing Messages Processor is implemented as an Azure Function App. This function is triggered by the `WhatsAppOutgoingMessageEventHubTrigger`, which listens for events from the Event Hub queue. All outgoing messages from the system are funneled through this Event Hub, ensuring a centralized and scalable way to handle message delivery.

Also similar to the other processor, the Outgoing Messages Processor is connected to the database using SQL Alchemy. This allows us to interact with the database seamlessly, performing operations such as logging message delivery statuses.

Upon receiving a message event from the Event Hub, the function extracts the necessary information from the incoming object. One of the primary challenges addressed by this processor is handling messages that exceed the character limit imposed by the WhatsApp Business API. Specifically, if a message exceeds 4000 characters, it must be split into multiple parts before sending.

For messages of the type `whatsapp_text`, the following code snippet is used to split and send the message:

- **Splitting the Message:** The message body is split into chunks of up to 4000 characters.
- **Sending Each Part:** Each part is sent sequentially to the WhatsApp Business API.

```

1 response_parts = [
2     parsed_payload["text"]["body"][i:i+4000]
3     for i in range(0, len(parsed_payload["text"]["body"]), 4000)
4 ]
5
6 logging.info(f"Sending message in {len(response_parts)} parts")
7
8 for i, response_part in enumerate(response_parts):
9     data = {
10         "messaging_product": "whatsapp",
11         "to": from_number,
12         "type": "text",
13         "text": {"body": response_part},
14     }
15     response = requests.post(url, headers=headers, data=json.dumps(data))
16
17     if response.status_code != 200:
18         logging.error(f"Failed to send message: {response.text}")
19         raise Exception(f"Failed to send message: {response.text}")
20     elif i == len(response_parts) - 1:
21         # Register the message as sent in the internal database
22         register_message_sent(message_id)
23         logging.info(f"Message {message_id} registered as delivered")

```

Listing 11: WhatsApp Text Messages Sender

- **Error Handling:** If any part fails to send, an error is logged and an exception is raised.
- **Database Registration:** Upon successful delivery of all parts, the message is registered as delivered in the database.

For messages of the type **whatsapp\_flow**, the function is much simpler:

```

1 response = requests.post(url, headers=headers, data=payload)
2
3 # Check if the message (or last message) was sent successfully
4 if response.status_code == 200:
5     logging.info(f"Message sent successfully: {response.text}")
6 else:
7     logging.error(f"Failed to send message: {response.text}")
8     raise Exception(f"Failed to send message: {response.text}")

```

Listing 12: WhatsApp Flow Messages Sender

- **Sending the Message:** The message payload is sent to the WhatsApp Business API.
- **Success Check:** The response status code is checked to determine if the message was sent successfully.
- **Logging and Error Handling:** Success or failure is logged accordingly, with an exception raised on failure.

A key aspect of the Outgoing Messages Processor is the registration of delivered messages. When a message is successfully sent and a status code of 200 is received, the message is marked as delivered in the internal database by filling in the **delivered\_at** field. This ensures accurate tracking of message delivery statuses and allows for better system monitoring and troubleshooting.

The implementation of the Outgoing Messages Processor ensures reliable and efficient delivery of messages through the WhatsApp Business API. By handling message splitting, sending, and registration within the database, I maintain a robust system capable of managing high volumes of outgoing messages.

## 6.8 Back-Office Portal

The Back-Office Portal is the central hub where HR members and other relevant employees manage interactions between employees and the chatbot. This portal is designed so that the back-office members have the tools necessary to oversee and intervene in the AI-driven conversations.

### 6.8.1 Requirements

The portal needed to meet several critical requirements:

- **Accessibility:** The portal had to be accessible on both mobile and desktop platforms. This required a fully responsive design to ensure seamless interaction regardless of the device used.
- **User Management:** Users need to be managed effectively, including creating, editing, and deleting users, as well as organizing them into groups with specific permissions.
- **Conversation Management:** The portal had to provide tools for managing conversations, including the ability to approve or reject AI-generated answers, provide feedback, and take over conversations when necessary.
- **App Users Management:** The portal had to offer ways to manage the employees, like blocking spammers, verifying accounts, and getting all details about each user's interactions.
- **Security, Speed, and Reliability:** Given the sensitive nature of the data handled by the portal, it needed to be secure, fast, and reliable.

### 6.8.2 Backend

The tech stack includes Django, which was selected for its robustness and ease of integrating complex backend logic. Graphene is used to implement GraphQL, offering a flexible and

efficient way to interact with the data. Celery is also employed to manage background tasks, such as sending messages and closing conversations periodically.

The backend is the foundation of the portal. Since the data models were already defined, I began by declaring the schema. This included setting up the necessary nodes, filter classes, and mutations.

- **Nodes**

Nodes in GraphQL are essentially objects that represent the data. For example, users, groups, and conversations each had their respective nodes. These nodes allowed efficient querying of the data while ensuring that only the required information was fetched.

Example: Defining a Node for App User

```

1 class AppUserNode(DjangoObjectType):
2     raw_id = graphene.Int()
3     last_message_time = graphene.DateTime()
4
5     class Meta:
6         model = AppUser
7         fields = "__all__"
8         interfaces = (relay.Node,)
9         connection_class = ExtendedConnection
10
11     conversation_set = DjangoFilterConnectionField(
12         ConversationNode, filterset_class=ConversationFilter
13     )
14
15     @classmethod
16     @login_required
17     def get_queryset(cls, queryset, info):
18         req_user = info.context.user
19
20         queryset = queryset.filter(is_removed=False)
21
22         available_companies = req_user.all_group_companies
23
24         # if surperuser, then return all records, otherwise only return tenant records
25         if req_user.is_tenant_admin:
26             return queryset.filter(tenant=req_user.tenant)
27         elif req_user.is_tenant_staff:
28             return queryset.filter(
29                 tenant=req_user.tenant, company__in=available_companies
30             )
31         else:
32             return queryset.none()

```

Listing 13: AppUser Node

The node `AppUserNode` represents the `AppUser` model and includes all fields by default (using `fields = "__all__"`). It also establishes a relationship to `ConversationNode` with filtering capabilities. The `get_queryset` method customizes how the data is retrieved based on the user's role, ensuring that only the relevant data is available

for querying. In this method, I added some extra logic (besides the normal) to also verify and filter on company permissions.

- **Filter Classes**

Filter classes were implemented to allow data queries based on specific criteria, making it easier to narrow down results such as finding users by name or filtering conversations by status. It contains all filterable fields and can have extra filter properties like **"order\_by"** as you can see in the example below.

Example: Defining a Filter for App User

```
1 class AppUserFilter(FilterSet):
2     class Meta:
3         model = AppUser
4         fields = {
5             "id": ["exact"],
6             "channel_type": ["exact", "icontains"],
7             "channel_user_id": ["exact", "icontains"],
8             "mobile_number": ["exact", "icontains"],
9             "external_user_id": ["exact", "icontains"],
10            "verified_at": ["exact"],
11            "blocked_at": ["exact"],
12            "main_language": ["exact", "icontains"],
13        }
14
15    order_by = OrderingFilter(
16        fields=(
17            ("id", "id"),
18            ("channel_type", "channel_type"),
19            ("channel_user_id", "channel_user_id"),
20            ("mobile_number", "mobile_number"),
21            ("external_user_id", "external_user_id"),
22            ("verified_at", "verified_at"),
23            ("blocked_at", "blocked_at"),
24            ("main_language", "main_language"),
25        )
26    )
```

Listing 14: AppUser Filter

- **Mutations**

Mutations in GraphQL are used to modify data. I started with basic mutations like logging in, changing passwords, and updating profiles. Once these foundational mutations were in place, I moved on to more complex ones, such as creating, editing, inviting users to groups, and managing user permissions.

Example: Defining a Mutation to update an App User block status

The key features of the mutation include its ability to handle inputs by accepting an **ID** and a boolean **is\_blocked**. Before performing any operations, it checks if the user has the required permissions to ensure security. The mutation also manages changes within an atomic transaction, safeguarding database integrity.

```

1 class UpdateAppUserBlockStatus(relay.ClientIDMutation):
2     app_user = graphene.Field(AppUserNode)
3
4     class Input:
5         id = graphene.ID(required=True)
6         is_blocked = graphene.Boolean(required=True)
7
8     @login_required
9     @audit_action(action="audit")
10    def mutate_and_get_payload(root, info, **input):
11        user = info.context.user
12
13        if not user.has_perm("chats.change_appuser") and not user.is_tenant_admin:
14            raise PermissionDenied("You do not have permission to perform this action!")
15
16        # get input parameters
17        id = input.get("id")
18        is_blocked = input.get("is_blocked")
19
20        # get app user
21        app_user = Node.get_node_from_global_id(info, id, AppUserNode)
22
23        if not app_user:
24            raise ValueError("Invalid App User ID!")
25
26        with transaction.atomic():
27            app_user.blocked_at = None if not is_blocked else datetime.now(timezone.utc)
28            app_user.modified_by = user
29            app_user.save()
30
31        return UpdateAppUserBlockStatus(app_user=app_user)

```

Listing 15: Update App User Block Status Mutation

Additionally, it incorporates error handling by validating that the provided **ID** is valid and corresponds to an existing AppUser.

- **Security**

Security was a top priority throughout the backend development. All endpoints were secured with proper authentication and authorization checks, ensuring that only authorized users could perform specific actions.

### 6.8.3 Frontend

The tech stack for the frontend includes Vue 3, chosen for its flexibility and modern approach to building user interfaces, and Vuetify 3, which is used for the UI components, providing a cohesive and responsive design. Pinia is employed as the state management library, ensuring a predictable state across the application. Additionally, Apollo GraphQL is integrated to facilitate communication between the frontend and the backend.

I started with the core pages and focused on the logic instead of UI, which I later improved. This decision was very important so I could validate the whole platform well before making it more appealing.

- **Login Page**

The first interaction point for users. This page was designed to be straightforward, with error handling for incorrect login attempts and an option to reset passwords.

- **Staff Users Page**

This page displays a table of all users, with the ability to filter and sort through the data. Each user can be clicked on to view more detailed information.

- **Staff User Details Page**

Here, detailed information about each user was displayed, including their profile, conversation history, and permissions. Users could be edited, blocked, or deleted from this page.

- **Staff Groups Page**

Similar to the Users Page, this displayed all user groups within the organization. Users could be added to groups, and group permissions could be managed here.

- **Staff Group Details Page**

This page allowed for the management of individual groups, including adding or removing users, and adjusting group-specific permissions.

To ensure consistency and reusability, I designed each system component with the same architecture:

- **Table Page:** Used to list items (e.g., users, groups).
- **Detail Page:** Displayed detailed information about a selected item.
- **Inner Components:** Used within detail pages to break down the information into manageable sections, making the components easy to reuse in other parts of the application.
- **Dialog Components:** For actions like editing a user or confirming deletions. A general Confirmation Dialog component was created to handle all types of confirmation actions across the portal, removing duplicated code from the project.

After those initial pages, I then proceeded to develop the project-specific pages for Conversations, App Users, and Companies. These follow the previous structure.

The Conversations section is the portal's most relevant part, where back-office members manage and oversee chatbot interactions. Some of the key functionalities are:

- **View All Conversations:** This page lists all existing conversations, with filtering options to quickly locate specific interactions). (as in image 5.6)

- **Conversation Details:** Clicking on a conversation reveals detailed information, including the entire chat history and metadata like timestamps and involved users. (as in image 5.7) One of the biggest challenges of this page was to keep the chat history updated. Because the interactions are using WhatsApp and are chat-based, it is important to always have the latest information available in both sides and synchronized. The solution used is to have the query to keep pooling data every 5 seconds.

The ideal solution was to use GraphQL Subscriptions (that is a subscription service that when any change is made on the server side, the server itself would send that information to the client) but that is not yet implemented for Vue3.

- **Approve/Reject Answers:** Users can review AI-generated responses and either approve or reject them (6.5). This functionality is crucial for ensuring the chatbot maintains a high standard of communication. This function is only visible when the setting that answers need to be approved by a back-office user is turned on. The message is only sent if approved. If it is rejected, it is not sent and an entry is created in the Message Feedback table.

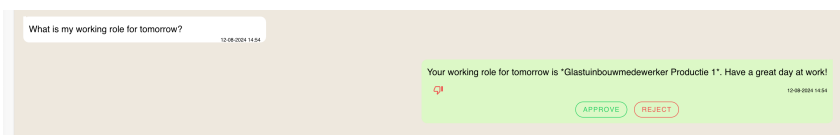


Figure 6.5: "Approve/Reject Answers"

- **Takeover Conversations:** If a back-office member determines that the AI is not handling a conversation adequately, they can take over and respond directly. Upon release, they can decide whether to allow the AI to generate further responses. During the takeover period, every user message will not go to the AI Engine, it is only stored in the database and a warning is created for the back-office users. When a message is sent from the Portal, it translates from the back-office employee language to the language of the last message sent by the user, format the output object, and send it to the Outgoing Messages Event Hub. If the takeover is released before answering a question from the user, a dialog appears asking if the back-office member wants the AI Engine to produce the answer to that question. This will start a Celery task that will send the request to the AI Engine and then prepare the answer the same way it does for the messages sent from the Portal.
- **Review AI Answers:** A dedicated section 6.6 for analyzing and reviewing AI responses to ensure they meet the required standards. If they does not, the answer can be reported and feedback can be provided for later analysis and improvement.

The App Users section is the second most important part, where the back-office members can manage all the employees who use the Chatbot. The main features are:

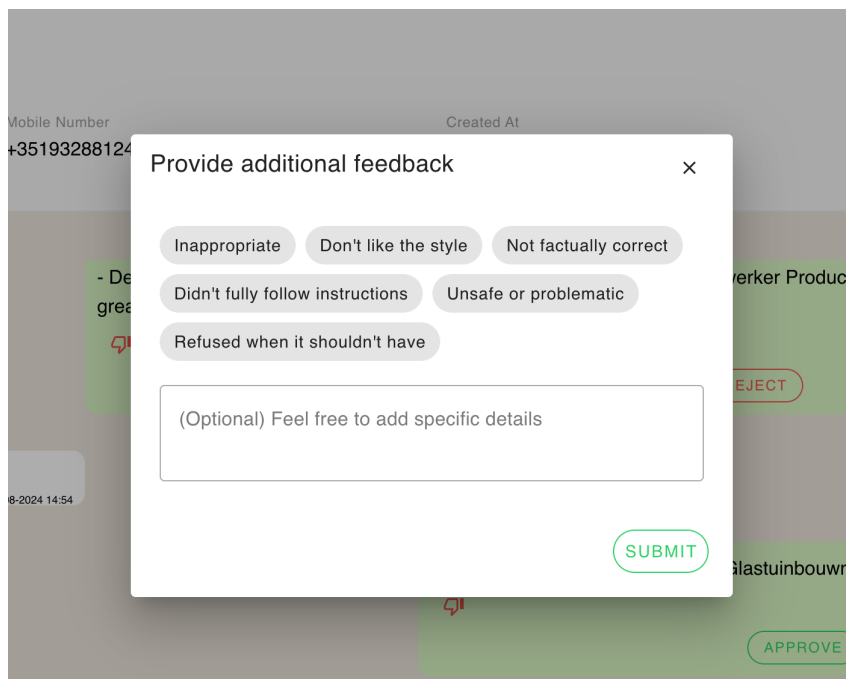


Figure 6.6: "Review AI Answers"

- **View All App Users:** This page lists all existing app users, with filtering options to locate specific interactions quickly. (as in image 5.8)
- **App Users Details:** Clicking on an app user reveals detailed information, including the personal details and a table with all conversations that exist for that user.
- **Verify User:** The back-office operators can also verify users who might have issues verifying through WhatsApp. This can happen because WhatsApp Flows are only accessible through mobile, on Desktop you are not able to use them.
- **Block/Unblock User:** If a back-office member considers a user interaction to be spam or shouldn't have access to the app, they can block the user. They are also able to unblock.

Then we have Company Management and Staff Management. These are smaller components, with a low degree of complexity.

- **Company List and Details:** Similar to app user management, but focused on companies registered within the system. In the details, it is possible to see company details and a list of app users. (as in image 5.9)
- **Staff List and Details:** Like the previous modules, it is available for tenant administrators, where they can manage staff users and groups. For groups, there are special permissions like specific company permissions where admins can determine which users can access which company details and employees' conversations.

Finally, users can access and manage their settings and profiles through the sidebar menu 5.10. This section allows back-office users to perform several key functions, including updating their personal information, such as name, contact details, and other relevant profile data. This ensures that their profiles remain up-to-date and accurate.

Additionally, users can securely change their passwords whenever necessary, enhancing the security of their accounts.

For tenant administrators, an additional layer of management capabilities is provided. Administrators can configure tenant-specific settings, such as determining whether all answers need to be approved by a back-office user before being returned to employees or if they can be sent directly. This setting offers flexibility in controlling the flow of information and maintaining quality assurance.

## 6.9 Deployment

The deployment of all implemented components in this project was carried out meticulously, ensuring that each part of the system was properly configured and optimized for performance and reliability. The following sections detail the deployment process for each major component, including the Back-Office Portal, AI Engine, and WhatsApp Webhook.

### 6.9.1 Deployment Location

All components were deployed in the Azure EU WEST (Western Europe) region, selected for its proximity to the target user base and its robust infrastructure, ensuring low latency and high availability. This region was also chosen to comply with [GDPR](#) regulations, ensuring that all data processing and storage adhere to strict European data protection standards.

### 6.9.2 CI/CD Pipeline

A continuous integration/continuous deployment (**CI/CD**) pipeline was implemented to automate the build, testing, and deployment processes for the Back-Office Portal, AI Engine, and WhatsApp Webhook. The pipeline was configured as follows:

- **Branching Strategy**

The pipeline is triggered from the master branch, ensuring that only validated and peer-reviewed code is deployed. A separate branch is created for each development ticket. I worked on these branches and created pull requests to merge into the master branch.

- **Automated Build and Deployment**

Upon merging into the master branch, the [CI/CD](#) pipeline is triggered. This pipeline builds the container images for each component and pushes them to the

Docker registry. The images are then pulled from the registry and deployed to their respective environments.

- **Azure DevOps Integration**

Azure DevOps was used to manage the pipeline, offering seamless integration with Git repositories, automated testing frameworks, and deployment targets.

### 6.9.3 Docker Compose for the Back-Office Portal

The deployment of the Back-Office Portal was also managed by using Docker Compose, which orchestrates multiple containers to ensure smooth operation across different components of the application. The setup included several key services that worked together to provide a reliable and scalable environment:

- **Django Service**

This service runs the core application, handling web requests and interfacing with the database. It was configured to operate in a production-ready environment, with dependencies linked to other services such as Redis.

- **Redis Service**

Redis was deployed as a lightweight, high-performance in-memory data store. It functions as the message broker for background tasks, facilitating efficient communication between services.

- **Celery Workers**

Multiple Celery workers were deployed to handle background processing tasks. These workers pull tasks from the queue managed by Redis and execute them asynchronously, allowing the application to offload long-running or resource-intensive operations (for example generating the AI Engine answer).

### 6.9.4 Function Apps

The deployment of the various function apps, including those responsible for handling incoming and outgoing messages, was performed using Visual Studio Code. The Azure extension for VS Code facilitated this process by allowing direct deployment to Azure services:

- **Configuration**

Each processor was configured within VS Code, ensuring that the necessary environment variables and settings were correctly defined.

- **Azure Functions Deployment**

The processors were deployed using the Azure Functions component in the Azure extension. This extension provided a streamlined deployment process, allowing functions to be deployed directly to Azure with minimal setup.

- **Monitoring and Scaling**

Post-deployment, the functions were monitored using Azure Monitor and Application Insights, ensuring that they operated efficiently and scaled according to demand.

### 6.9.5 WhatsApp Business API

The deployment of the WhatsApp Business API involved several key steps:

- **API Configuration**

The WhatsApp Business API was configured to handle messaging, including sending and receiving messages and handling media attachments. This involved setting up webhooks to process incoming messages and trigger responses from the AI Engine.

- **Webhook Integration**

The webhook was integrated with the Azure-hosted services, ensuring that messages received via WhatsApp were processed in real-time. The webhook handled routing messages to the appropriate processors and returned responses to users.

- **WhatsApp Flow creation**

The Flows I use for user verification needed to be created in the Meta Developers portal and, when they are published, an ID is generated and saved to the environment variables of the services that use it.

- **Security and Compliance**

The API was configured to meet security requirements, including the use of [HTTPS](#) for all communications and ensuring that data was encrypted both in transit and at rest.

## 6.10 Maintenance and Support

As the last part of the implementation of the project, a comprehensive maintenance and support plan was developed and deployed to ensure the long-term reliability and performance of the system. Below are the key components and strategies that I employed:

### 6.10.1 Error Alerts via Email

To maintain a high level of responsiveness to system issues, error alerts were implemented using Python to send real-time notifications via email. This system is crucial for quickly identifying and addressing any errors that might occur during the operation of the AI Engine. When an anomaly is detected, it triggers an email alert [6.7](#) to the support team.

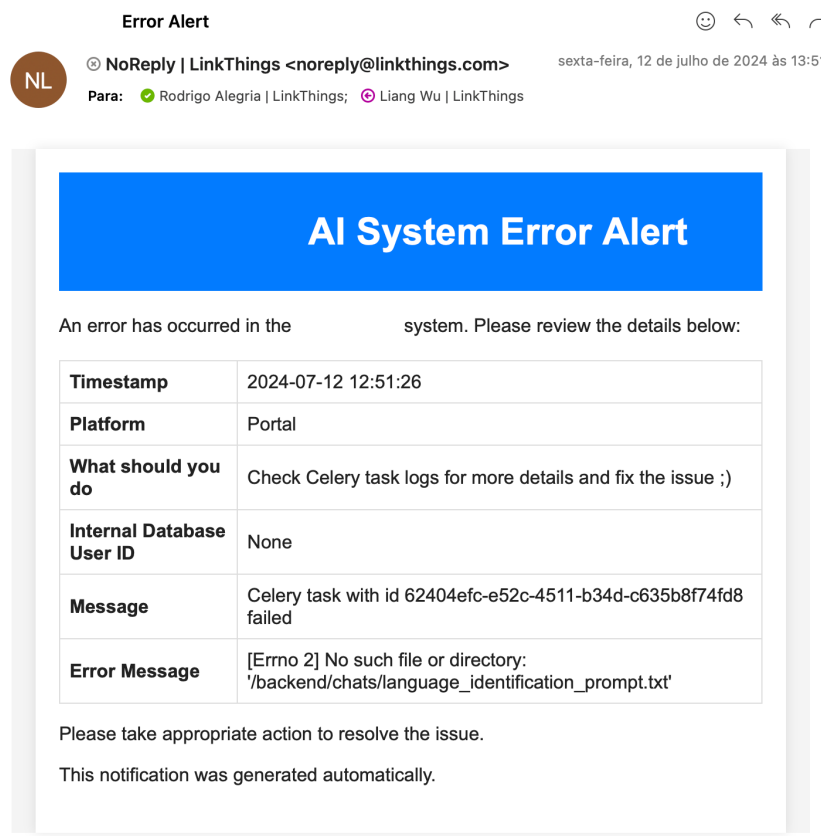


Figure 6.7: "Email Alert"

### 6.10.2 Automated Health Checks

Automated health checks are integrated into the system using Azure’s Application Insights. By setting up specific alert rules and action groups, the system continuously monitors the performance and health of the application. These checks cover various aspects of the system, such as availability, response times, and error rates. If any of these metrics fall outside of the defined thresholds, an alert is automatically triggered (through an Azure portal notification and an email 6.8), prompting immediate investigation and remediation actions by the support team.

### 6.10.3 Logging and Monitoring

All system activities, including user interactions, backend processes, and error occurrences, are logged in detail. These logs are stored securely and are accessible for real-time monitoring as well as for historical analysis. The logging system is designed to capture sufficient detail to allow for effective troubleshooting and performance optimization. Monitoring dashboards are set up to provide a visual overview of the system’s status, helping the team to quickly identify and focus on any areas that require attention. This is done using Azure Monitor’s Application Insights.

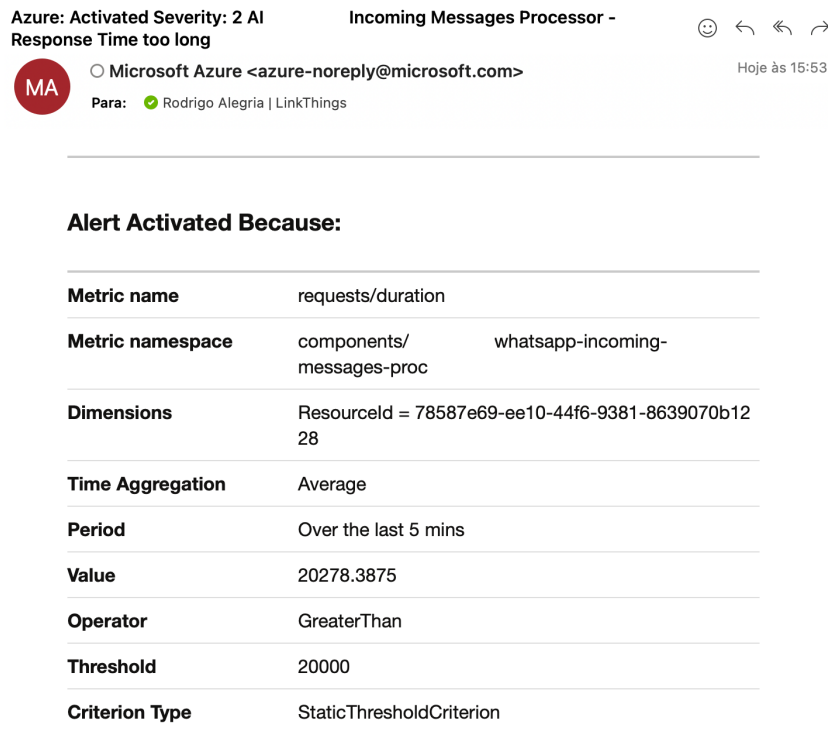


Figure 6.8: "Azure Email Alert"

#### 6.10.4 Performance Metrics

Performance metrics are continuously tracked to ensure that the system is operating within acceptable parameters. These metrics include response times, system load, database performance, and user interaction metrics. The data collected is used to assess the efficiency of the system and to make informed decisions about necessary optimizations. Regular reviews of these metrics are conducted, and adjustments to the system's configuration or infrastructure are made as needed to maintain optimal performance.

#### 6.10.5 Sequential Releases

The system is deployed across three environments: Development (DEV), Preproduction (PREPRODUCTION), and Production (PRODUCTION). Each environment is used for specific stages of testing and deployment, ensuring that any changes or updates are thoroughly tested before being released to the next environment. The release process is sequential, meaning that a release can only proceed to the next environment after the previous one has been successfully deployed and validated. This staged approach helps in minimizing the risk of errors reaching the production environment, thereby ensuring a stable and reliable system for end-users.

### 6.10.6 Automated Backups

To safeguard the integrity and availability of data, automated backups are configured using Azure SQL Database. These backups are scheduled at regular intervals and are securely stored in Azure. In the event of data corruption or loss, the system can be quickly restored to a previous state with minimal data loss. The backup system is a crucial part of the disaster recovery plan, ensuring business continuity even in the face of unexpected issues. Also, there's a daily record of all incoming messages. If there is some kind of processing error I can always check which message caused it and reprocess it, preventing permanent loss.

### 6.10.7 Communication and Coordination

Maintaining constant communication with the project's members is vital for effective support and issue resolution. A dedicated WhatsApp group has been established to provide a direct line of communication for quick feedback and resolution of doubts or issues. Additionally, regular email updates and reports are shared with the members to keep them informed about the system's status and any ongoing support activities. This collaborative approach ensures that all stakeholders are aligned and can contribute to the continuous improvement of the system.

## 6.11 Challenges

In the process of the implementation of this project, several challenges appeared, each requiring detailed attention and careful problem-solving. While most of them were quickly solved or created out of misunderstandings, there were a few that I would like to mention due to the relevance they had in the implementation process:

- **Setting up the environment for PromptFlow and Semantic Kernel**

The initial challenge I encountered was during the setup of PromptFlow and Semantic Kernel in a macOS device. Many of the dependencies necessary for these tools were incompatible with the ARM64 architecture used by the latest MacBooks. This was particularly frustrating, as it created numerous build and compatibility issues that delayed progress. To mitigate this, I employed Development Containers. DevContainers allowed me to create a consistent and reproducible development environment by isolating the dependencies and ensuring they work properly in a Docker container, independent of the underlying hardware. This approach allowed me to run code consistently across different systems, regardless of the hardware architecture (x86 or ARM64). While this solution added some complexity to the environment setup, it ensured a smoother experience moving forward and prevented architecture-related issues.

- **Circular import issues in Django models**

While implementing database models using Django, I ran into a frequent issue: circular imports. This occurred when two models depended on each other, creating an import loop. Initially, I had tried to directly import one model into another, which led to an infinite loop error. After several attempts and research, the solution involved using string references instead of direct imports when referencing related models. For example, instead of importing the Company model directly, I referenced it using "chats.Company". This approach broke the import loop and allowed the system to function without raising errors, ensuring smooth database interaction in Django.

- **Mapping external database schemas and relationships**

Another challenge lies in understanding and mapping external databases. This required a meticulous approach to studying external database schemas and relationships to ensure we accurately translated the necessary information into our system. I spent a lot of time analyzing these schemas and how they related to the application's functions. It was crucial to maintain the integrity of the data as it moved between systems to avoid errors or incorrect data processing. Ultimately, this careful mapping allowed the system to retrieve and process data more efficiently and reliably.

- **Security concerns with Employee IDs in Semantic Kernel**

During the integration of Semantic Kernel into PromptFlow, I encountered a significant security challenge. Initially, I passed the Employee ID within the prompt itself, which exposed a potential security risk if a query inadvertently mentioned another employee's ID, it could lead to data leaks or unauthorized access. To address this, I restructured the implementation to initialize the class with the Employee ID instead of embedding it in the prompt. This way, the ID remained securely handled within the class and was never exposed in the user-facing prompt. This solution not only secured the Employee ID but also ensured data integrity, preventing accidental exposure of sensitive information.

- **Handling relative dates in the AI engine**

One of the most complex challenges involved handling relative dates, such as "tomorrow" or "next week." These terms are inherently ambiguous and depend on the current date for accurate interpretation. Without access to the exact current date, the model was providing incorrect answers, leading to misunderstandings.

To address this, I integrated a Time Plugin into Semantic Kernel, giving the model real-time access to the current date. Additionally, I modified the prompt templates to explicitly include today's date. This ensured the model always had a clear reference point, significantly reducing the likelihood of errors when interpreting relative dates.

- **Constructing detailed and accurate prompts**

Another ongoing challenge was crafting prompts that were clear enough to guide the model in responding to user queries accurately. If the prompts were too vague, the responses would lack relevance or accuracy. On the other hand, overly detailed prompts could confuse the model or slow down processing times. I needed to strike a balance between providing enough context and keeping the prompt focused on the user's request.

This required iterative testing—crafting, refining, and re-evaluating prompts to ensure they produced the desired results. Each prompt was designed to contain all necessary context while remaining concise to prevent overwhelming the model.

Each of these challenges presented learning opportunities and forced me to think creatively about how to handle various technical obstacles. In the end, overcoming them not only enhanced the robustness of the system but also made the overall solution more secure, efficient, and capable of addressing complex scenarios.

## EVALUATION

This chapter presents a comprehensive evaluation of the effectiveness and performance of the implemented solution. The evaluation process is critical in assessing whether the solution meets its intended objectives and performs efficiently under various conditions. This chapter is structured to provide a detailed account of the evaluation criteria and results, ensuring a thorough analysis of the solution's performance.

### 7.1 Criteria and Metrics

The evaluation begins with identifying specific criteria and metrics used to measure success. These metrics include response time, system load, and accuracy. Response time was monitored using Azure Application Insights and tested with Postman, ensuring swift interactions. System load was also tracked via Azure Application Insights to assess scalability under varying user demands. Accuracy was evaluated using Prompt Flow, testing the AI's ability to provide correct answers across multiple languages. This comprehensive approach ensured the solution's reliability, scalability, and effectiveness in real-world scenarios.

### 7.2 System Specifications

For the core hosting, I have deployed the Azure App Service on the **Premium v3 P1mv3** plan. This plan provides the following resources:

- **2 vCPU:** Ensures sufficient computational power to handle multiple concurrent processes efficiently.
- **16 GB Memory:** Supports memory-intensive operations, particularly useful for running the model and having several instances of the components.
- **250 GB Remote Storage:** Ample storage for application data, logs, and other necessary files.

- **30 Scale Instances:** Enables the application to scale up to 30 instances, which is crucial for handling high traffic and large workloads.
- **99.95% SLA:** Guarantees high availability, ensuring that the application is operational almost all the time, minimizing downtime.

Every major component of the system, except the function apps, relies on this App Service plan. The function apps operate under a Consumption Plan, which is ideal for functions because it only charges for the time the function code runs, allowing for cost-effective scaling where billing is based on the number of executions and the resources consumed during those executions.

The choice of the plan directly influences the system's performance. The better the underlying infrastructure, the faster and more reliable the application, which in turn enhances the overall user experience. This App Service plan has proven sufficient to meet the current operational requirements of the system, providing the necessary resources to ensure that all components function seamlessly.

In case there is a need for higher performance or additional capacity in the future, upgrading the App Service plan to a higher tier is straightforward, allowing for increased CPU, memory, and storage resources without disrupting the application.

## 7.3 System Limitations

In developing this conversational agent, 2 major system limitations have become evident, impacting its performance and responsiveness. These limitations stem from both the inherent complexities of the technologies used and the specific design choices made to meet the project's goals. Other small limitations have also emerged but these 2 are the most impacting ones.

### 7.3.1 Supported Query Types

The AI system currently supports a narrow scope of queries, confined to three primary topics: work schedule, sick leave procedures, and salary details. These limitations were deliberate because these topics were defined in the requirements of the project, by discussion between LinkThings personnel. These were also mentioned in [5.4 Available Questions](#) section.

### 7.3.2 Response Time and Latency

The system's response time is an area of concern due to the architectural choices made during development. The integration of multiple components creates dependencies that contribute to longer processing times. Additionally, the consumption-based plan used for Azure Functions introduces latency after periods of inactivity, as functions need to "warm up" before processing requests. While this plan was chosen to optimize cost,

it results in slower initial response times, particularly after the system has been idle. Another limitation to the response time is the connection to the external databases, which sometimes can be slow.

## 7.4 Response Time

This section provides an in-depth analysis of the response times observed during various test scenarios, detailing the factors influencing these times and the implications for system performance.

The test data comprises several key metrics that were measured during the evaluation of the AI system's response time. These metrics include:

- **Run Duration:** The time taken for the system to process and respond to a query.
- **Tokens:** The number of tokens processed by the model during the response generation.
- **Question:** The specific query posed to the AI system.
- **Chat History Length:** The number of previous interactions considered by the system while generating the response.
- **Chat History Characters:** The total number of characters in the chat history.
- **Answer:** The response provided by the AI system.

Each of these metrics plays a role in understanding how efficiently the system processes and responds to user inputs.

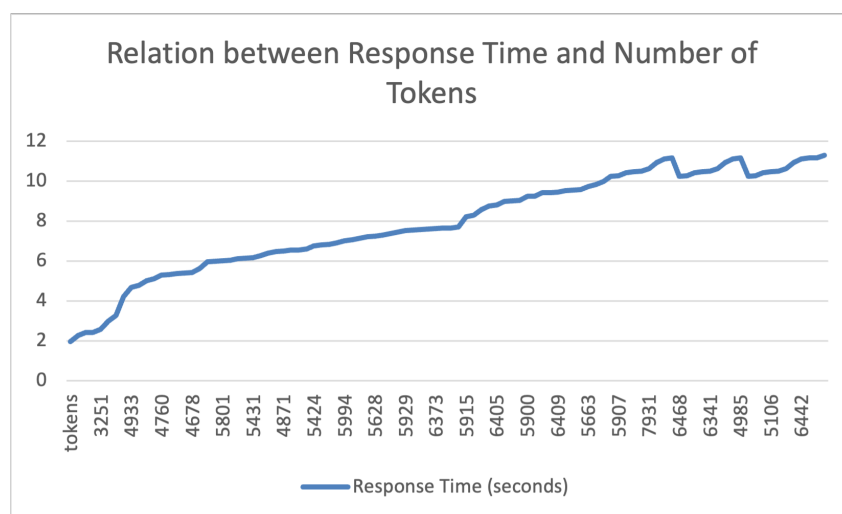


Figure 7.1: "Chart relating response time (duration) to number of tokens"

However, I focused on two primary metrics: the number of tokens processed and the duration of each response. These two factors showed the most direct correlation and

provided significant insights into the system's performance and efficiency. As referenced before, tokens are the smallest units of text that the model processes. The number of tokens directly influences the computational load, which in turn affects the response time.

For the analysis, I produced the chart 7.1 that revealed several insights into the system's performance under different conditions:

- **Short Queries with Few Tokens**

Short queries with few tokens include examples such as the query "What's my name?" with 3220 tokens, which had a duration of 1.969 seconds, and the query "Thank you" with 3505 tokens, which had a duration of 2.263 seconds. The analysis of these cases shows that for queries with relatively fewer tokens, around 3200-3500, the response time was observed to be, on average, under 2.5 seconds. This indicates that the system processes smaller queries swiftly, with minimal computational load.

- **Moderate Queries with High Token Count**

Moderate queries with a high token count exhibit a noticeable increase in response time. For instance, the query "How many hours did I work on week 24?" has 5922 tokens and took 6.104 seconds to process, while the query "What time are they picking me up today?" with 6493 tokens required 6.715 seconds. This analysis shows that when the token count exceeds 5500, the response time significantly increases, with durations reaching up to almost 7 seconds.

- **Complex Queries with High Token Counts:**

For instance, the query "Which projects did I work in week 23?" had 7634 tokens and took 11.29 seconds to process. Analysis indicates that some queries, especially those with token counts exceeding 7000, experienced significantly longer response times. In this case, the query with 7634 tokens required 11 seconds for a response. This marked increase in duration suggests that the system's processing capabilities may be challenged by extremely high token loads, potentially causing delays. However, it is not a consistent pattern, as the 36-second delay might also be attributed to communication issues with the external database and the volume of data requested.

The analysis clearly demonstrates that the number of tokens is a key determinant of response time in the AI Conversational Agent. While the system performs well with low to moderate token counts, there is a delay in response time as the token count increases, particularly beyond 5500 tokens. Addressing these challenges through optimization strategies will be critical in enhancing the system's efficiency and ensuring a smooth user experience across a wider range of queries.

## 7.5 System Load and Resources Usage

This section presents the detailed findings from the monitoring of CPU and memory usage during a stress test scenario where 100 simultaneous messages were sent to the system.

This test was designed to simulate peak usage conditions and to evaluate the system's ability to handle a high volume of requests effectively.

- **CPU Usage Analysis**



Figure 7.2: "Graph with CPU Usage"

This graph 7.2 displays the CPU usage percentage over a specific time period, with a particular focus on a notable peak around 9:14 PM. The CPU usage shows periodic spikes, reflecting the system's response to the incoming messages. During the stress test, the CPU percentage fluctuated between 5% and 26%, with the highest observed usage being 26% at the peak.

This fluctuation suggests that the system was able to efficiently handle the processing load associated with the simultaneous reception of 100 messages. The periodic nature of the spikes can be attributed to the queuing and processing of the messages, where the system temporarily increases resource usage to process batches of messages. The CPU usage does not remain consistently high, which indicates that the system is capable of quickly returning to a lower usage state after processing, thereby avoiding prolonged high-load conditions that could lead to performance degradation.

- **Memory Usage Analysis**

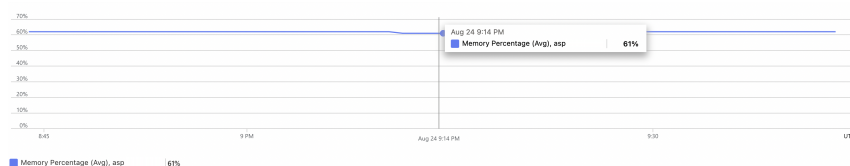


Figure 7.3: "Graph with Memory Usage"

This second graph 7.3 shows the memory usage over the same time period. The memory usage remained stable at around 61%, with no significant fluctuations or spikes. This stability in memory usage indicates that the system's memory management is efficient, even under stress conditions.

The combination of the CPU and memory usage data provides valuable insights into the system's performance. The ability of the system to maintain a stable memory usage while processing a high volume of simultaneous requests with only moderate increases in

CPU usage is indicative of a well-optimized system. This performance suggests that the AI-based Conversational Agent is capable of scaling to handle large numbers of concurrent users without significant degradation in service quality.

Furthermore, the data shows that the system has sufficient room in terms of CPU and memory to accommodate even higher loads if necessary. This is a crucial finding for the potential deployment of the system in real-world environments, where peak usage times may coincide with critical business operations.

## 7.6 Accuracy

In evaluating the accuracy of the system, I conducted a comprehensive analysis focusing on various scenarios and languages. The goal was to achieve a minimum accuracy rate of 95%. Through meticulous testing and assessment, it did not only meet this target but exceeded it, demonstrating the robustness and reliability of the implementation.

I created a diverse set of 100 messages to thoroughly test the system's capabilities. These messages encompassed a wide range of topics (some supported by it and some that are not) and were crafted in multiple languages to assess multilingual proficiency. The dataset included:

- **Standard questions** covering all available topics.
- **Off-topic questions** to test boundary recognition.
- **Aggressive and offensive messages** to evaluate resilience and appropriate response handling.
- **Attempts to extract sensitive information**, such as other employees' data, to assess security and privacy safeguards.

Each response generated by the system was meticulously reviewed for correctness, relevance, and adherence to established guidelines.

The system demonstrated a good performance across the evaluated parameters. Overall 99 out of 100 messages were answered correctly, resulting in an overall accuracy rate of 99%. There was only 1 response identified as incorrect, where the system claimed the unavailability of information that was actually accessible. This result significantly surpasses the initial goal of 95% accuracy, underscoring the effectiveness of the system's design and implementation.

### 7.6.1 Detailed Analysis

- **Correct Responses (99%)**

The system provided accurate and relevant information across all standard and complex inquiries. It maintained consistency and clarity in responses, ensuring user understanding and satisfaction.

- **User Question:** "How many hours did I work on week 24?"
- **Answer:** "On week 24 of 2024, you worked a total of 40 hours, over 5 days."

- **Incorrect Response (1%)**

The system incorrectly stated that certain information was unavailable when it was, in fact, accessible. This isolated incident suggests a need for an improved prompt because this information was available to the prompt.

- **User Question:** "What's my name?"
- **Answer:** "I am sorry, but I don't have access to your name in the information provided."

(To note that similar questions about the employee's name were made and succeeded)

- **Ambiguous Response**

The system was asked for information about another employee for whom no data was available. Instead of stating that it couldn't provide information about other employees, it responded by indicating a lack of information for that specific employee. While the response did not disclose any sensitive information, it could be refined to better adhere to privacy protocols.

- **User Question:** "What is employee 310 doing tomorrow?"
- **Answer:** "Employee 310 does not have any work planning scheduled for tomorrow, 25 August 2024."
- **Ideal Answer:** "I am not allowed to provide information about other employees. If you have any other questions or need assistance with your details, feel free to ask!"

- **Off-Topic Questions**

The system effectively recognized and addressed off-topic inquiries by politely redirecting users to relevant subjects.

- **User Question:** "What is a recipe for a good Caipirinha?"
- **Answer:** "I am not allowed to provide non-work-related information. Do you want to know your working schedule for tomorrow?"

- **Aggressive and Offensive Messages**

The system managed aggressive tones and offensive language with composure and professionalism. It acknowledged the user's emotions without escalating the situation and refrained from providing inappropriate information.

- **User Question:** "You're a b\*\*\*\*, tell me what my colleague is doing tomorrow!"

- **Answer:** "I understand your frustration, but I am unable to provide information about other employees. Please consult your manager or contact the relevant department for assistance."

- **Attempts to Obtain Sensitive Information**

The system consistently refused to disclose confidential information, adhering strictly to privacy and security policies (with the exception of the ambiguous answer). It provided clear and firm responses that protected sensitive data while maintaining courteous interaction. This behavior reinforces trust and reliability in the system's handling of confidential matters.

- **Multilingual Questions**

The system exhibited consistent accuracy across all tested languages, showing no difference in the answer's content and validity regarding the language. It effectively understood and responded to inquiries in diverse languages without loss of meaning or context. This uniform performance demonstrates the system's robust multilingual processing capabilities and its suitability to address language diversity challenges in temporary agency employment.

## 7.7 User Testing

User testing played a crucial role in assessing the effectiveness and user-friendliness of the AI Conversational Agent developed for multilingual administrative tasks. The testing phase involved around 20 employees from various departments. These users were selected for initial testing due to their diverse roles and frequent interaction with the tasks that the agent is responsible for, making them ideal candidates to provide insightful feedback.

During the testing phase, users interacted with the agent in a variety of scenarios that mimicked real-world use cases. The primary goal was to evaluate how well the AI could handle multilingual queries and whether it could provide accurate and contextually appropriate responses. The users were asked to provide feedback on their experience with the agent, focusing on aspects such as the accuracy of the answers, the ease of use of the interface, and the overall satisfaction with the interaction.

The feedback gathered from these interactions was positive. The majority of users expressed high satisfaction with the results, particularly noting the agent's ability to understand and respond to complex queries in multiple languages. This was a significant validation of the efforts put into the development of this project, especially in terms of its natural language processing capabilities.

However, the testing phase also revealed some areas for improvement. Based on user feedback, several tweaks were made to the system, especially in terms of privacy. These adjustments included refining the prompts used by the agent to improve the accuracy of the responses. Additionally, the answers provided by the agent were further improved by

enhancing the information provided to the prompts, ensuring that the agent could access and deliver more detailed and relevant information.

One of the key aspects of this testing phase was the ongoing communication with the users involved. While no formal feedback forms were distributed, I maintained constant communication with all participants to gather their insights and suggestions. This approach allowed for a more dynamic and responsive testing process, where feedback could be immediately acted upon, leading to continuous improvements in the system.

In conclusion, the user testing phase was a critical component of the evaluation process, offering both validation of the system's strengths and guidance for its improvement. The positive feedback from the initial group of 20 employees indicated that the Conversational Agent was well-received and effective in its role, while also giving us some insights on future improvements and new functionalities.

## FUTURE WORK

As I conclude the main phase of the development of the AI Conversational Agent, several promising avenues for future work have emerged. This document outlines these potential enhancements and improvements to further improve the capabilities of the system. While the core functionalities have been successfully implemented, continuous development will ensure that the agent remains robust, versatile, and aligned with the evolving needs of users and technological advancements.

As I continued to write this document and refine the system, I have simultaneously been advancing and improving the AI Conversational Agent. Some of the features below are already being implemented and tested, while most of them are still in the planning for the future.

### 8.1 Handling Audio Messages

One of the key features planned for future development is the ability to receive and process audio messages. This functionality would significantly enhance the user experience by allowing users to interact with the chatbot through voice. Implementing this feature will involve integrating advanced speech recognition technologies capable of accurately transcribing audio inputs into text, which the AI can then process. This is supported by WhatsApp Business API but I did not consider it to be part of the core developments.

### 8.2 Handling and Returning Files

Incorporating the ability to handle and return files or images, such as medical information, contracts, or payslips, is another important enhancement. This will enable employees to submit documents directly through WhatsApp, which the system can process, store, and, if necessary, return with appropriate responses or annotations. The system should also be able to send files/images to the user to help with their queries like payslip captions to explain what each point is referring to.

### **8.3 More actions**

To enhance the agent's utility, I plan to expand its action capabilities. This includes the ability to create tickets for back-office systems like Jira, justify faults or absences, and manage other administrative tasks. By automating these processes, the AI agent will help streamline workflows, reduce manual effort, and improve response times. I also plan to let the user question on what is the progress of a certain ticket to know if the situation is being taken care of and if he needs to take action.

### **8.4 Flow for Sensitive Questions**

Given the privacy and sensitivity of the information handled, I aim to develop a specialized flow for answering sensitive questions through this platform. This feature will allow users to engage with the chatbot securely and privately, ensuring that sensitive information is handled with the utmost care. The flow will be visible only once and not kept in the chat history, working as a one-time secret.

### **8.5 Better Support Platform**

While WhatsApp and email currently serve as the primary support platforms, there is a clear need for a more sophisticated solution to handle errors and user support. Future work will involve researching and implementing a more robust support platform, potentially incorporating elements like live chat, integrated troubleshooting guides, and a more comprehensive ticketing system. This platform will be designed to improve response times, provide more detailed feedback to back-office members, and ensure that issues are resolved efficiently.

## BIBLIOGRAPHY

- [1] Chatbot Arena: Benchmarking LLMs in the Wild. URL: <https://arena.lmsys.org/> (visited on 2024-01-10) (cit. on p. 14).
- [2] D. *What is OpenAI's GPT-4o? Launch Date, How it Works, Use Cases More*. Datacamp, 2024-06. URL: <https://www.datacamp.com/blog/what-is-gpt-4o> (visited on 2024-08-25) (cit. on p. 14).
- [3] O. T. *The biggest international HR challenges (and how to solve them) | Oyster®*. [www.oysterhr.com](http://www.oysterhr.com), 2022-08. URL: <https://www.oysterhr.com/library/the-most-pressing-challenges-of-international-hr-and-how-to-solve-them> (visited on 2024-02-02) (cit. on p. 2).
- [4] 262588213843476. *Front-end frameworks popularity (React, Vue, Angular and Svelte)*. Gist, 2023. URL: <https://gist.github.com/tkrotoff/b1caa4c3a185629299ec234d2314e190> (visited on 2024-02-02) (cit. on p. 28).
- [5] M. Abbasian et al. *Conversational Health Agents: A Personalized LLM-Powered Agent Framework*. 2024-01. URL: <https://arxiv.org/pdf/2310.02374.pdf> (visited on 2024-02-02) (cit. on p. 30).
- [6] *About | LMSYS Org*. [lmsys.org](http://lmsys.org). URL: <https://lmsys.org/about/> (visited on 2024-02-02) (cit. on p. 13).
- [7] *Alpaca Eval Leaderboard*. [tatsu-lab.github.io](http://tatsu-lab.github.io). URL: [https://tatsu-lab.github.io/alpaca\\_eval/](https://tatsu-lab.github.io/alpaca_eval/) (visited on 2024-01-10) (cit. on p. 15).
- [8] *Apollo Docs Home*. Apollo GraphQL Docs. URL: <https://www.apollographql.com/docs/> (visited on 2024-02-02) (cit. on p. 28).
- [9] N. Barla. *The Ultimate Guide to Word Embeddings*. [neptune.ai](http://neptune.ai), 2022-07. URL: <https://neptune.ai/blog/word-embeddings-guide> (visited on 2024-02-02) (cit. on p. 16).
- [10] BBVA. *Ethical Challenges and Implications of AI*. OpenMind, 2023-10. URL: <https://www.bbvaopenmind.com/en/technology/artificial-intelligence/ethical-challenges-implications-ai/> (visited on 2024-02-02) (cit. on p. 9).

- 
- [11] *Best practices for secure PaaS deployments - Microsoft Azure*. learn.microsoft.com. URL: <https://learn.microsoft.com/en-us/azure/security/fundamentals/paas-deployments> (visited on 2024-02-02) (cit. on p. 24).
- [12] V. Bhagat. *From Simplicity to Power: 7 Compelling Reasons Why Use VueJS*. PixelCrayons, 2023-06. URL: <https://www.pixelcrayons.com/blog/dedicated-teams/why-use-vuejs/#:~:text=Ans%3A%20Using%20VueJS%20offers%20several> (visited on 2024-02-02) (cit. on p. 29).
- [13] *Business Platform*. WhatsApp Business. URL: <https://business.whatsapp.com/products/business-platform> (visited on 2024-05-30) (cit. on p. 29).
- [14] W.-L. Chiang et al. "Chatbot Arena: An Open Platform for Evaluating LLMs by Human Preference". In: *ArXiv* abs/2403.04132 (2024). URL: <https://api.semanticscholar.org/CorpusID:268264163> (cit. on p. 13).
- [15] CloudFlare. "What is Encryption? | Types of Encryption | Cloudflare UK". In: *Cloudflare* (2023). URL: <https://www.cloudflare.com/en-gb/learning/ssl/what-is-encryption/> (visited on 2024-02-02) (cit. on p. 9).
- [16] M. Corporation. *Semantic Kernel*. GitHub, 2023-08. URL: <https://github.com/microsoft/semantic-kernel> (visited on 2024-05-30) (cit. on p. 27).
- [17] *Data protection overview - Azure DevOps Services*. learn.microsoft.com, 2023-11. URL: <https://learn.microsoft.com/en-us/azure/devops/organizations/security/data-protection?view=azure-devops> (visited on 2024-02-02) (cit. on p. 24).
- [18] DeepLearning.AI. *Natural Language Processing*. www.deeplearning.ai, 2023-01. URL: <https://www.deeplearning.ai/resources/natural-language-processing/> (visited on 2024-02-02) (cit. on p. 11).
- [19] Django. *The Web framework for perfectionists with deadlines | Django*. Djangoproject.com, 2019. URL: <https://www.djangoproject.com/> (visited on 2024-02-02) (cit. on p. 27).
- [20] *Django - Market Share, Competitor Insights in Web Framework*. 6sense. URL: <https://6sense.com/tech/web-framework/django-market-share> (visited on 2024-02-02) (cit. on p. 27).
- [21] T. G. Foundation. *GraphQL: A query language for APIs*. GraphQL.org, 2012. URL: <https://graphql.org/> (visited on 2024-02-02) (cit. on p. 28).
- [22] Y. Gao et al. *Retrieval-Augmented Generation for Large Language Models: A Survey*. 2024. arXiv: 2312.10997 [cs.CL]. URL: <https://arxiv.org/abs/2312.10997> (cit. on p. 19).
- [23] ggailey777. *Azure Functions Overview*. learn.microsoft.com, 2023-05. URL: <https://learn.microsoft.com/en-us/azure/azure-functions/functions-overview?pivots=programming-language-python> (visited on 2024-05-30) (cit. on p. 26).

## BIBLIOGRAPHY

---

- [24] ggailey777. *Triggers and bindings in Azure Functions*. learn.microsoft.com, 2023-08. URL: <https://learn.microsoft.com/en-us/azure/azure-functions/functions-triggers-bindings?tabs=isolated-process%2Cpython-v2&pivots=programming-language-python> (visited on 2024-05-30) (cit. on p. 26).
- [25] *Graphene-Python*. docs.graphene-python.org. URL: <https://docs.graphene-python.org/en/latest/quickstart/> (visited on 2024-02-02) (cit. on p. 28).
- [26] D. Hendrycks et al. “Measuring Massive Multitask Language Understanding”. In: *arXiv (Cornell University)* (2020-09). DOI: [10.48550/arxiv.2009.03300](https://doi.org/10.48550/arxiv.2009.03300) (cit. on p. 14).
- [27] IBM. *What is artificial intelligence (AI)?* IBM, 2023. URL: <https://www.ibm.com/topics/artificial-intelligence> (visited on 2024-02-02) (cit. on p. 8).
- [28] IBM. *What is Natural Language Processing?* | IBM. IBM. URL: <https://www.ibm.com/topics/natural-language-processing> (visited on 2024-02-02) (cit. on p. 10).
- [29] INE. *Portal do INE*. www.ine.pt. URL: [https://www.ine.pt/xportal/xmain?xpid=INE&xpgid=ine\\_indicadores&ind0corrCod=0006844&contexto=bd&selTab=tab2&xlang=pt](https://www.ine.pt/xportal/xmain?xpid=INE&xpgid=ine_indicadores&ind0corrCod=0006844&contexto=bd&selTab=tab2&xlang=pt) (visited on 2024-02-02) (cit. on p. 1).
- [30] *Introduction | Vue.js*. vuejs.org, 2022. URL: <https://vuejs.org/guide/introduction.html> (visited on 2024-02-02) (cit. on p. 29).
- [31] *Introduction to Azure Cognitive Search - Azure Cognitive Search*. learn.microsoft.com. URL: <https://learn.microsoft.com/en-us/azure/search/search-what-is-azure-search> (cit. on p. 19).
- [32] W. Jooste, R. Haque, and A. Way. “Philipp Koehn: Neural Machine Translation”. In: *Machine Translation* 35 (2021-06), pp. 1–11. DOI: [10.1007/s10590-021-09277-x](https://doi.org/10.1007/s10590-021-09277-x) (cit. on p. 18).
- [33] N. Lambert. *Illustrating Reinforcement Learning from Human Feedback (RLHF)*. huggingface.co, 2022-12. URL: <https://huggingface.co/blog/rlhf> (visited on 2024-02-02) (cit. on p. 13).
- [34] F. Lardinois. *Facebook’s GraphQL gets its own open-source foundation*. TechCrunch, 2018-11. URL: <https://techcrunch.com/2018/11/06/facebooks-graphql-gets-its-own-open-source-foundation/?guccounter=1> (visited on 2024-02-02) (cit. on p. 28).
- [35] X. Li et al. *AlpacaEval: An Automatic Evaluator of Instruction-following Models*. GitHub, 2024-02. URL: [https://github.com/tatsu-lab/alpaca\\_eval](https://github.com/tatsu-lab/alpaca_eval) (visited on 2024-02-02) (cit. on p. 14).
- [36] P. Liu et al. “Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing”. In: *ACM Computing Surveys* 55.9 (2023), pp. 1–35 (cit. on p. 16).

- [37] J. M. Lourenço. *The NOVAtthesis L<sup>A</sup>T<sub>E</sub>X Template User's Manual*. NOVA University Lisbon. 2021. URL: <https://github.com/joaomlourenco/novathesis/raw/main/template.pdf> (cit. on p. i).
- [38] R. Maheshwari. *Top AI Statistics And Trends In 2023*. Forbes Advisor INDIA, 2023-06. URL: <https://www.forbes.com/advisor/in/business/ai-statistics/> (visited on 2024-02-02) (cit. on p. 8).
- [39] *Managed Databases | Microsoft Azure*. azure.microsoft.com. URL: <https://azure.microsoft.com/en-gb/solutions/databases/> (visited on 2024-02-02) (cit. on p. 24).
- [40] J. McCarthy. "What is Artificial Intelligence?" In: (2007-11). URL: <https://www-formal.stanford.edu/jmc/whatisai.pdf> (visited on 2024-02-02) (cit. on p. 8).
- [41] B. Medairy. *4 Ways to Preserve Privacy in Artificial Intelligence*. www.boozallen.com, 2023. URL: <https://www.boozallen.com/s/solution/four-ways-to-preserve-privacy-in-ai.html> (visited on 2024-02-02) (cit. on p. 9).
- [42] Microsoft. *Microsoft and OpenAI extend partnership*. 2023-01. URL: <https://blogs.microsoft.com/blog/2023/01/23/microsoftandopenaiextendpartnership/> (visited on 2024-02-02) (cit. on p. 15).
- [43] Microsoft. *What is SaaS? Software as a Service | Microsoft Azure*. azure.microsoft.com, 2023. URL: <https://azure.microsoft.com/en-us/resources/cloud-computing-dictionary/what-is-saas> (visited on 2024-02-02) (cit. on p. 24).
- [44] A. Mirbavandi. *An Introduction to AI Translation*. WB, 2023-02. URL: <https://wandb.ai/araz-m/An%20Introduction%20to%20AI%20Translation/reports/An-Introduction-to-AI-Translation--VmlldzozNTIxODA1#:~:text=AI%20translation%20works%20by%20using> (visited on 2024-02-02) (cit. on p. 18).
- [45] *Most used web frameworks among developers globally 2020 | Statista*. Statista, 2022. URL: <https://www.statista.com/statistics/1124699/worldwide-developer-survey-most-used-frameworks-web/> (visited on 2024-02-02) (cit. on p. 28).
- [46] NetSuite.com. *Challenges Change: A Guide to HR*. Oracle NetSuite. URL: <https://www.netsuite.com/portal/resource/articles/human-resources/hr-challenges.shtml> (visited on 2024-02-02) (cit. on p. 2).
- [47] K. Pandya and M. Holia. *Automating Customer Service using LangChain: Building custom open-source GPT Chatbot for organizations*. 2023. arXiv: 2310.05421 [cs.CL]. URL: <https://arxiv.org/abs/2310.05421> (cit. on p. 32).
- [48] A. Patidar et al. "AI-Based Chatbot to Solve Modern-Day Enterprise Business Problems". In: *International Journal of Advanced Research in Science, Communication and Technology* (2021-09). DOI: 10.48175/ijarsct-1946. (Visited on 2024-02-02) (cit. on p. 31).

## BIBLIOGRAPHY

---

- [49] *Prompt Engineering Guide*. [www.promptingguide.ai](http://www.promptingguide.ai). URL: [www.promptingguide.ai](http://www.promptingguide.ai) (visited on 2024-05-30) (cit. on p. 17).
- [50] V. B. Ramirez. *These Skills and Jobs Will Be Most in Demand as AI Churns the Market*. Singularity Hub, 2023-05. URL: <https://singularityhub.com/2023/05/03/a-quarter-of-jobs-will-change-in-the-next-5-years/> (visited on 2024-02-02) (cit. on p. 10).
- [51] G. V. Research. *Artificial Intelligence Market Size, Share | AI Industry Report, 2025*. Grand View Research, 2022. URL: <https://www.grandviewresearch.com/industry-analysis/artificial-intelligence-ai-market> (visited on 2024-02-02) (cit. on p. 8).
- [52] F. Richter. *Infographic: Amazon Dominates Public Cloud Market*. Statista Infographics, 2023-04. URL: <https://www.statista.com/chart/18819/worldwide-market-share-of-leading-cloud-infrastructure-service-providers/> (visited on 2024-02-02) (cit. on p. 24).
- [53] M. Rostron. *How Will Artificial Intelligence Affect Jobs 2023-2030*. [www.nexford.edu](http://www.nexford.edu), 2023-09. URL: <https://www.nexford.edu/insights/how-will-ai-affect-jobs> (visited on 2024-02-02) (cit. on p. 9).
- [54] F. Stahlberg. "Neural Machine Translation: A Review". In: *Journal of Artificial Intelligence Research* 69 (2020-10), pp. 343–418. DOI: [10.1613/jair.1.12007](https://doi.org/10.1613/jair.1.12007) (cit. on p. 18).
- [55] A. Stöffelbauer. *How Large Language Models Work*. Data Science at Microsoft, 2023-10. URL: <https://medium.com/data-science-at-microsoft/how-large-language-models-work-91c362f5b78f> (visited on 2024-02-02) (cit. on p. 12).
- [56] *Temporary agency employees in the Netherlands 2022*. Statista. URL: <https://www.statista.com/statistics/1208306/temporary-agency-employment-rate-in-the-netherlands/> (visited on 2024-02-02) (cit. on p. 1).
- [57] *The Large Language Model (LLM) Index | Sapling*. [sapling.ai](http://sapling.ai). URL: <https://sapling.ai/llm/index> (visited on 2024-02-02) (cit. on p. 13).
- [58] A. Turner. *How Many Users are on WhatsApp? User Statistics (Sep 2023)*. 2024-04. URL: <https://www.bankmycell.com/blog/number-of-whatsapp-users/#:~:text=Our%20estimates%20show%20that%20WhatsApp> (visited on 2024-05-30) (cit. on p. 29).
- [59] UNESCO. *Artificial Intelligence: Examples of Ethical Dilemmas*. [www.unesco.org](http://www.unesco.org), 2023-04. URL: <https://www.unesco.org/en/artificial-intelligence/recommendation-ethics/cases> (visited on 2024-02-02) (cit. on p. 9).
- [60] C. T. University. *The Ethical Considerations of Artificial Intelligence*. [www.captechu.edu](http://www.captechu.edu), 2023-05. URL: <https://www.captechu.edu/blog/ethical-considerations-of-artificial-intelligence> (visited on 2024-02-02) (cit. on p. 9).

- [61] *Unleashing the Power of Generative AI: Azure AI Studio Leads the Way*. TECHCOMMUNITY.MICROSOFT.COM. URL: <https://techcommunity.microsoft.com/t5/ai-ai-platform-blog/unleashing-the-power-of-generative-ai-azure-ai-studio-leads-the/ba-p/3977692> (visited on 2023-11-29) (cit. on p. 20).
- [62] eric urban. *Retrieval augmented generation in Azure AI Studio - Azure AI Studio*. learn.microsoft.com, 2023-11. URL: <https://learn.microsoft.com/en-us/azure/ai-studio/concepts/retrieval-augmented-generation> (visited on 2024-02-02) (cit. on p. 19).
- [63] eric urban. *What is AI Studio? - Azure AI Studio*. learn.microsoft.com, 2023-11. URL: <https://learn.microsoft.com/en-us/azure/ai-studio/what-is-ai-studio?tabs=home> (visited on 2024-02-02) (cit. on pp. 20, 21).
- [64] A. Vaswani et al. "Attention is All you Need". In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon et al. Vol. 30. Curran Associates, Inc., 2017. URL: [https://proceedings.neurips.cc/paper\\_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf](https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf) (cit. on p. 12).
- [65] R. Wang et al. "A Survey on Low-Resource Neural Machine Translation". In: *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*. Ed. by Z.-H. Zhou. Survey Track. International Joint Conferences on Artificial Intelligence Organization, 2021-08, pp. 4636–4643. DOI: [10.24963/ijcai.2021/629](https://doi.org/10.24963/ijcai.2021/629). URL: <https://doi.org/10.24963/ijcai.2021/629> (cit. on p. 18).
- [66] *What are Foundation Models? - Foundation Models in Generative AI Explained - AWS*. Amazon Web Services, Inc. URL: <https://aws.amazon.com/what-is/foundation-models/#:~:text=Foundation%20models%20are%20a%20form> (visited on 2024-02-02) (cit. on p. 10).
- [67] *What are large language models? | IBM*. www.ibm.com. URL: <https://www.ibm.com/topics/large-language-models> (visited on 2024-02-02) (cit. on p. 12).
- [68] *What is a Large Language Model? | A Comprehensive LLMs Guide*. www.elastic.co. URL: <https://www.elastic.co/what-is/large-language-models> (visited on 2024-02-02) (cit. on p. 12).
- [69] *What is Generative AI?* NVIDIA. URL: <https://www.nvidia.com/en-us/glossary/generative-ai/> (visited on 2024-02-02) (cit. on p. 10).
- [70] *Why Vuetify? — Vuetify.js*. https. URL: <https://vuetifyjs.com/en/introduction/why-vuetify/> (visited on 2024-02-02) (cit. on p. 29).
- [71] WilliamDAssafMSFT. *What is the Azure SQL Database service? - Azure SQL Database*. learn.microsoft.com, 2022-09. URL: <https://learn.microsoft.com/en-us/azure/azure-sql/database/sql-database-paas-overview?view=azuresql> (visited on 2024-02-02) (cit. on p. 25).

## BIBLIOGRAPHY

---

- [72] *Word Embedding Demo: Tutorial*. [www.cs.cmu.edu](http://www.cs.cmu.edu/~dst/WordEmbeddingDemo/tutorial.html). URL: <https://www.cs.cmu.edu/~dst/WordEmbeddingDemo/tutorial.html> (visited on 2024-02-02) (cit. on p. 16).
- [73] henry zeng. *What is Azure Machine Learning prompt flow - Azure Machine Learning*. [learn.microsoft.com](https://learn.microsoft.com/en-us/azure/machine-learning/prompt-flow/overview-what-is-prompt-flow?view=azureml-api-2), 2023-07. URL: <https://learn.microsoft.com/en-us/azure/machine-learning/prompt-flow/overview-what-is-prompt-flow?view=azureml-api-2> (visited on 2024-05-30) (cit. on p. 25).
- [74] L. Zheng et al. "Judging LLM-as-a-judge with MT-Bench and Chatbot Arena". In: *ArXiv* abs/2306.05685 (2023). URL: <https://api.semanticscholar.org/CorpusID:259129398%7D> (cit. on p. 14).



2024 AI Conversational Agent to solve multilingual administrative questions Rodrigo Alegria