
Université Ibn Tofail
Faculty of Sciences, Kenitra

Master's Final Year Project Report

Master in Artificial Intelligence and Virtual Reality

Development of a Customer Service Chatbot for a Smart Home Rental Business

Host Company: SOJORI HOLDING - Casablanca

Prepared by: Mr. Mouad MIFTAH IDRISSE

Supervised by: Mr. Anass NOURI (FSK-UIT)
Mr. Tawfiq GOUACH (SOJORI Holding)

Presented on September 20, 2024, before the jury composed of:

- Mr Anass NOURI (FSK-UIT)
- Mr Tarik BOUJIHA (ENSAK-UIT)
- Mr Rochdi MESSOUSSI (FSK-UIT)
- Mme Raja TOUAHNI (FSK-UIT)

Acknowledgements

I would like to begin by expressing my deepest gratitude to Allah for providing me with the strength, perseverance, and guidance throughout this journey.

I also extend my sincere gratitude to the SOJORI Holding team for providing me with the opportunity to complete my End of Studies internship with their esteemed company.

My heartfelt thanks go to my supervisors, Mr. Anass Nouri and Mr. Tawfiq Gouach, whose guidance and support were invaluable throughout the entire process.

I wish to thank the professors and staff of the Master's program in Artificial Intelligence and Virtual Reality at the Faculty of Sciences - Ibn Tofail University for their continuous encouragement and for equipping me with the knowledge and training necessary to succeed in this project.

I am deeply thankful to my parents, whose unwavering support, love, and encouragement have been a constant source of strength.

Lastly, I extend my appreciation to my colleagues, friends, and family for their ongoing support and motivation.

Abstract

This project was conducted as part of my End of Studies internship for the Master's degree in Artificial Intelligence and Virtual Reality at the Faculty of Sciences - Ibn Tofail University. The goal of this project was to enhance the customer experience in smart home rental services by integrating artificial intelligence solutions. Hosted by SOJORI Holding, the project involved developing a multi-platform chatbot system, integrated with modern tools such as FastAPI, OpenAI models, and LangChain to handle customer queries more effectively. The chatbot automates essential processes such as online check-in, property access, and communication with tenants through WhatsApp and the company's website. The project aimed to improve efficiency, customer satisfaction, and scalability by employing state-of-the-art technologies like Retrieval-Augmented Generation (RAG) and LangGraph for data retrieval and response generation.

Keywords: Smart home rental, Customer service automation, AI chatbot, WhatsApp API, OpenAI, LangChain, RAG, LLMs, LangGraph.

Résumé

Ce projet a été réalisé dans le cadre de mon stage de fin d'études pour le Master en Intelligence Artificielle et Réalité Virtuelle à la Faculté des Sciences - Université Ibn Tofail. L'objectif de ce projet était d'améliorer l'expérience client dans le domaine de la location de maisons intelligentes en intégrant des solutions basées sur l'intelligence artificielle. Hébergé par SOJORI Holding, le projet a consisté à développer un système de chatbot multi-plateforme, intégré avec des outils modernes tels que FastAPI, les modèles d'OpenAI et LangChain, afin de traiter les demandes des clients de manière plus efficace. Le chatbot automatise des processus essentiels tels que l'enregistrement en ligne, l'accès aux propriétés, et la communication avec les locataires via WhatsApp et le site web de l'entreprise. Le projet visait à améliorer l'efficacité, la satisfaction client et l'évolutivité en employant des technologies de pointe comme la Génération Augmentée par la Récupération (RAG) et LangGraph pour la récupération de données et la génération de réponses.

Mots-clés : Location de maisons intelligentes, Automatisation du service client, Chatbot IA, API WhatsApp, OpenAI, LangChain, RAG, LLMs, LangGraph.

List of Abbreviations

AI	Artificial Intelligence
NLP	Natural Language Processing
ICT	Information and Communication Technology
IoT	Internet of Things
LLM	Large Language Model
RAG	Retrieval-Augmented Generation
API	Application Programming Interface
GPT	Generative Pre-trained Transformer
AWS	Amazon Web Services
PMS	Property Management System
VS Code	Visual Studio Code
DB	Database
ML	Machine Learning
JSON	JavaScript Object Notation
CRM	Customer Relationship Management

Table of Contents

Acknowledgements	2
Abstract	3
Résumé	4
List of Abbreviations	5
Table of Contents	6
List of Figures	9
List of Tables	11
General Introduction	1
Chapter I: General context	2
I.1 Company presentation	2
I.1.1 SOJORI HOLDING Company	2
I.1.2 Core Services	2
I.1.3 Structure and Teams	3
I.2 Project Presentation	3
I.2.1 Context and motivation	3
I.2.2 Problem Statement	4
I.2.3 Objectives	4
I.3 Project management tools	5
I.3.1 ClickUp	5
I.3.2 Slack	6
Chapter II: Fundamental Concepts	7
II.1 Natural Language Processing (NLP)	7
II.1.1 Key Techniques in NLP	7
II.1.2 Word Embeddings:	8
II.2 Generative artificial intelligence	10
II.2.1 The Evolution of Generative artificial intelligence:	11
II.2.2 Types of Generative AI:	12
II.2.3 Applications of Generative AI:	12
II.3 Large Language Models (LLMs):	13
II.3.1 How LLMs Work:	13
II.3.2 Training and Fine-Tuning of LLMs	14
II.3.3 Examples of Powerful LLMs:	14
II.3.4 Applications of Large Language Models	16
II.4 Prompt Engineering	16

II.4.1	Key Uses of Prompt Engineering:	16
II.4.2	Techniques for Effective Prompt Engineering:.....	17
II.5	Retrieval-Augmented Generation (RAG)	17
II.5.1	Key Components of RAG:.....	17
II.6	Chatbots	18
II.6.1	Evolution of chatbots	18
II.6.2	Types of Chatbots	19
II.6.3	Benefits of Chatbots.....	20
II.6.4	Traditional Chatbot Vs Conversational AI	20
Chapter III: Tools and Technologies		21
III.1	Development Tools and Environments.....	21
III.1.1	Python	21
III.1.2	Visual studio Code.....	21
III.1.3	GitHub.....	22
III.1.4	Next.js	22
III.2	Backend Frameworks and Libraries	22
III.2.1	FastAPI	22
III.2.2	LangChain and LangGraph.....	22
III.2.3	Docker.....	24
III.3	Databases and Storage	24
III.3.1	MongoDB database.....	24
III.3.2	Amazon Web Service.....	25
III.3.3	Redis	25
III.4	APIs and Integrations.....	26
III.4.1	OpenAI APIs.....	26
III.4.2	Meta Cloud API	28
III.4.3	Hostaway.....	30
Chapter IV: System design and implementation.....		32
IV.1	Database Preparation.....	32
IV.1.1	HostAway API Integration.....	32
IV.1.2	Custom Database Design and Field Analysis	33
IV.2	Exploring existing solutions.....	34
IV.2.1	Evaluating Chatbot Building Platforms	34
IV.2.2	Evaluating Pre-Built Chatbots: HostAI.....	35
IV.2.3	Implementation of OpenAI Assistants API.....	35
IV.3	Building the Initial RAG System	36
IV.3.1	Database Preparation for RAG.....	36

IV.3.2	Query Processing and Workflow	37
IV.3.3	Response Generation with LLM	38
IV.3.4	Role of LangChain in RAG:	40
IV.3.5	Challenges and Limitations.....	40
IV.4	Transition to Advanced RAG System.....	41
IV.4.1	Development of Graph-Based RAG System.....	41
IV.4.2	Multi-Channel integration.....	43
IV.5	Dashboard Chat and Session handling.....	52
IV.5.1	State Handling.....	52
IV.5.2	Storing chat History	52
IV.5.3	Dashboard Visualization	53
	General Conclusion and Perspectives	54
	References.....	55

List of Figures

Figure 1 Sojori holding's logo.....	2
Figure 2 ClickUp logo	5
Figure 3 Screenshot of ClickUp tasks.....	5
Figure 4 Slack logo	6
Figure 5 Applications of Natural Language Processing	7
Figure 6 Vector Space Representation of Gender Relations in Word Embeddings.....	8
Figure 7 Embeddings Across Data Types.....	9
Figure 8 Query Processing in a Vector Database.....	9
Figure 9 Interconnections Between Generative AI and Other AI Subfield.	10
Figure 10 The Evolution of Generative AI	11
Figure 11 Types of Generative AI	12
Figure 12 Generative AI Use Cases	12
Figure 13 Interconnections Between Generative AI and Large Language Models	13
Figure 14 Generative AI Vs Large Language Models	13
Figure 15 Token Prediction Process in a Large Language Model	14
Figure 16 Example components of Prompt Engineering for Language Model Input and Output	17
Figure 17 Workflow of a Retrieval-Augmented Generation (RAG) System.....	18
Figure 18 History of chatbots.....	18
Figure 19 Difference between Conversational AI and Traditional chatbots.....	20
Figure 20 Python logo.....	21
Figure 21 VS Code Logo	21
Figure 22 Github Logo.....	22
Figure 23 MongoDB Data Model and Document Structure	24
Figure 24 Redis Data Types and Key-Value Storage Model	25
Figure 25 OpenAI Assistant API playground interface	27
Figure 26 Example of WhatsApp Templates for Customer Notifications	28
Figure 27 Example of FLOW JSON and its result	29
Figure 28 Example of WhatsApp Flows for Booking and Customer Engagement	30
Figure 29 Hostawat logo	30
Figure 30 Overview of Hostaway's Dashboard.....	31
Figure 31 Example of Hostaway API Listing Fields	32
Figure 32 Preparation of Reservation Fields for Database Integration.....	33
Figure 33 Custom Guest Behavior Fields for Enhanced Data Analysis	34
Figure 34 Summary Structure for Listings	36

Figure 35 Summary Structure for Reservations.....	37
Figure 36 Summary Structure for Calendar Entries.....	37
Figure 37 Query Processing Workflow in RAG System	38
Figure 38 Prompt structure for Llm in the initial rag system	38
Figure 39 Workflow of the initial Retrieval-Augmented Generation (RAG) System	40
Figure 40 Langgraph system design	41
Figure 41 Example of Handling a General Query Without Reservation	43
Figure 42 Workflow for RAG-Based Inquiry Handling	44
Figure 43 Example of the chatbot system handling question requires reservation	45
Figure 44 Response Workflow for Static Information Categories.....	45
Figure 45 Example of Room Service Inquiry Handling	47
Figure 46 Example of a WhatsApp Flow JSON for Language Selection.....	47
Figure 47 Example of Language determination Results Based on User's Phone Number	48
Figure 48 Language Change Workflow in WhatsApp Backend.....	48
Figure 49 Example of Language Change Workflow in WhatsApp Chatbot Interface	49
Figure 50 Workflow for Handling Requests via WhatsApp Flows	49
Figure 51 WhatsApp Interface for Selecting Check-in, Checkout, and Cleaning Options.....	50
Figure 52 Workflow for Online Check-in Process Using WhatsApp Flow and OpenAI.....	50
Figure 53 Guest and registration document selection flow	51
Figure 54 Example of online check-in form	51
Figure 55 Example of a user's chat history stored in MongoDB.....	53
Figure 56 Real-time Chat Visualization and User Interaction Dashboard.....	53

List of Tables

Table 1 Key Techniques in Natural Language Processing	8
Table 2 Comparison of existing LLMs	15
Table 3 Different Benefits of Chatbots	20
Table 4 Existing Chatbot Building Platforms	34
Table 5 Comparison of AI Models	39

General Introduction

In the evolving landscape of digital interactions, businesses are increasingly adopting AI-driven solutions to streamline customer service and enhance operational efficiency. The property management sector, particularly, has seen a rising need for systems capable of managing guest inquiries, reservations, and property access seamlessly. Sojori Holding embarked on a project to develop a chatbot system tailored to automate responses to common inquiries and manage more complex tasks with minimal human intervention.

The project aims to create a personalized, dynamic chatbot that aligns with Sojori Holding's identity and is capable of handling a diverse range of customer inquiries across platforms such as WhatsApp and the company's website. By automating tasks like booking updates, property information requests, and guest verification, the system provides a seamless customer experience. It is designed to integrate smoothly with Sojori's existing infrastructure, ensuring high-quality service across multiple channels.

In addition to providing accurate and context-aware responses, the chatbot system is scalable, enabling future customization and potential expansion into other areas of business. The project not only focuses on enhancing customer service but also on optimizing operational workflows through AI-based automation, ensuring the system reflects Sojori's unique operational needs and brand identity.

This report is organized into the following structure:

1. **General Introduction** provides an overview of the project, introducing the hosting company, Sojori Holding, and the motivation behind the chatbot system's development. It outlines the key challenges in automating customer service, particularly in the smart home rental sector, and the objectives the project seeks to achieve.
2. **Fundamental Concepts** explains the theoretical and technical concepts that underpin the project, covering key areas such as Natural Language Processing (NLP), Generative AI, Large Language Models (LLMs), Retrieval-Augmented Generation (RAG), and prompt engineering.
3. **Tools and Technologies** presents the development tools, libraries, and frameworks used throughout the project. It discusses the programming environments, backend frameworks (such as FastAPI, LangChain, and LangGraph), and the databases (MongoDB) that form the backbone of the chatbot system.
4. **System Design and Implementation** delves into the methodology and practical implementation of the chatbot. This chapter covers the workflow of integrating AI and RAG systems, as well as the multichannel communication setup (WhatsApp and web) used to deliver personalized customer interactions.

Chapter I: General context

This chapter gives an overview of the project, starting off with a presentation of the host organization, SOJORI HOLDING, and how it's seeking to improve tenants' experience in the home rental scene. It continues to then introduce the project by diving into the context and motivation behind it. The chapter then details the objectives and provides an outline of the planning and management strategies that guided its development.

I.1 Company presentation

I.1.1 SOJORI HOLDING Company



Figure 1 Sojori holding's logo

SOJORI HOLDING is an emerging startup specializing in the smart home rental industry. The company was founded to meet the growing demand for a seamless and efficient rental experience by transforming conventional homes into fully automated smart houses through advanced ICT (Information and Communication Technology) solutions.

SOJORI's primary mission is to enhance the rental experience, especially for families and larger groups, by combining the comfort of traditional rentals with the convenience typically found in hotels. They aim to automate the entire rental process—ranging from booking and check-in to property management—through AI and IoT technologies. Their vision is to revolutionize the Moroccan rental market by providing the luxury of a hotel experience while maintaining the privacy and flexibility of a private home.

I.1.2 Core Services

SOJORI's core offerings focus on smart home rentals and the development of advanced management systems for its own properties. By integrating technologies such as keyless entry, voice-activated controls, automated customer support, and energy-efficient systems, SOJORI aims to streamline the rental process for its customers. In addition to providing an enhanced renter experience, SOJORI is working to create management tools that allow its team to oversee and manage all aspects of their properties, from energy consumption to customer interactions, in a seamless and efficient manner.

I.1.3 Structure and Teams

As an upcoming startup, SOJORI operates with a streamlined organizational structure, focusing on the following key teams:

- **AI Team:** Responsible for integrating artificial intelligence into customer interactions and property management.
- **Back-End Development:** Manages the infrastructure and databases that support the smart home systems.
- **Front-End Development:** Focuses on designing and maintaining user interfaces across SOJORI's web and mobile applications.
- **Rental Services:** This team handles property management, customer service, and the logistical aspects of rental operations, including the management of IoT devices for smart homes.

I.2 Project Presentation

I.2.1 Context and motivation

Artificial Intelligence (AI) is increasingly being adopted into various fields, transforming the way healthcare, finance, manufacturing, and many other industries operate. This is because of its ability to analyze large amounts of data, recognize patterns, and make well-informed decisions. In customer service, for instance, AI increases efficiency by automating repetitive tasks and providing fast, accurate responses, improving user experience and satisfaction. This makes AI an essential tool for companies aiming to offer seamless customer support.

Simultaneously, the concept of smart homes has evolved significantly with the development of the Internet of Things (IoT). Smart home technology initially aimed at providing convenience, security, and energy efficiency through automation and remote control. Over time, it has evolved into more integrated systems that adapt to user needs, improving daily life. Today, smart homes use interconnected devices—such as thermostats, lighting, and security cameras—to create a unified and responsive living environment.

As smart homes become more common, tenants now expect the same level of sophistication and responsiveness to extend beyond the physical home and improve the entire rental process. As the expectations continued to grow, it became clear that integrating AI into the rental process was the next step. AI can automate administrative tasks, provide personalized recommendations for tenants, and facilitate communication between tenants and property managers. This creates a more cohesive and efficient rental experience, providing a more seamless and satisfying experience for all parties involved.

I.2.2 Problem Statement

As the smart home rental business continues to grow, the increasing expectations of tenants are becoming more apparent. They are expecting to be exposed to not only new and enhanced in-home automation solutions but also dynamic interaction with property management systems. This involves effortlessly managing inquiries, service requests, and other essential tasks, all customized to meet their specific needs and preferences.

However, what is offered as automated customer care services in the industry frequently falls short. These systems often result in a one-size-fits-all approach that is incohesive and lacks the flexibility needed to address the different requirements of tenants and properties. Additionally, there is a noticeable lack in the ability of these systems to integrate various processes and manage interactions across multiple platforms.

Moreover, the absence of context-aware support often leads to tenants facing difficulties when trying to communicate their specific needs or seeking assistance during their stay. This can shake the confidence in the service provided and can taint the overall experience of living in a smart home rental. The challenge is not just about integrating technology but also about creating a cohesive and responsive support system that aligns with the high standards of smart home living.

These issues highlight the pressing need to rethink how customer service is provided in the smart home rental space to better meet the evolving expectations of tenants.

I.2.3 Objectives

To address the specific challenges in the smart home rental industry and meet the evolving expectations of modern tenants, this project focuses on developing a comprehensive AI-powered solution by tackling specific objectives that aim to enhance the overall tenant experience. The primary objectives of the project are:

- **Develop a unified, multilingual chatbot system:** Create a multilingual AI-powered chatbot that operates across multiple platforms, including the website and WhatsApp, allowing tenants to engage with property management in their desired language and preferred communication channel, maintaining consistent and personalized communication and support for a diverse tenant base.
- **Automate essential tenant processes and requests:** Implement automation for key tenant processes such as online check-in, service requests (including room service), access management, and in-home automation tasks. That way, the tenants can also address their needs effectively through the chat-bot.

- Enhance personalization and context-aware responses:** Integrate advanced AI techniques to ensure that the chatbot delivers unique, context-aware responses to each tenant based on their individual profile and needs.
- Customize the chatbot to meet the company's needs and provide 24/7 service:** Tailor the AI-powered chatbot to align with the company’s branding, operational procedures, and specific tenant services. Additionally, equip the chatbot with the ability to provide continuous, 24/7 customer support and monitoring capabilities, allowing managers to store and review all interactions to analyze tenant behavior and feedback, ensuring a high level of service quality and consistency.

I.3 Project management tools

I.3.1 ClickUp



Figure 2 ClickUp logo

During the development process, we relied on ClickUp as our main project management tool, enabling efficient task tracking, setting deadlines, assigning responsibilities, and real-time progress monitoring. Its intuitive interface helped the team stay focused on project goals, while its flexible features made it simple to adapt to the dynamic needs, whether dealing with straightforward to-do lists or managing more complex project timelines.

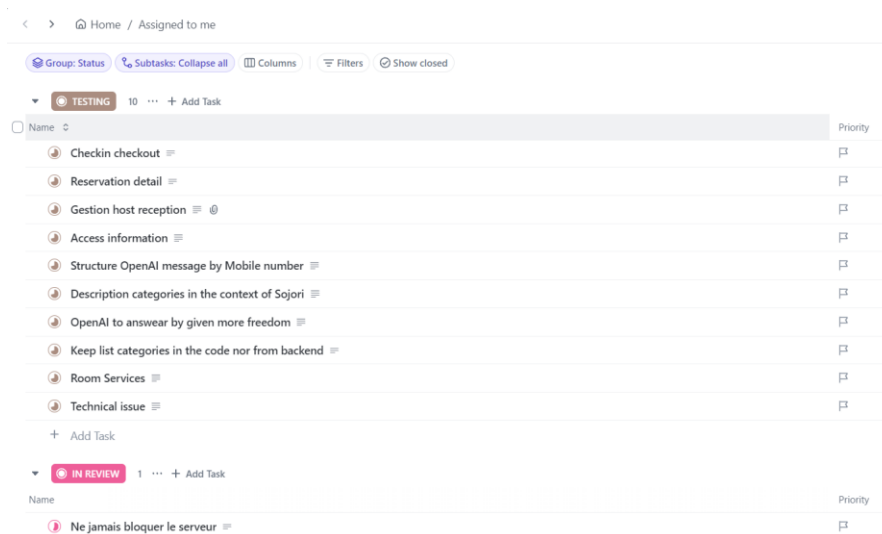


Figure 3 Screenshot of ClickUp tasks

I.3.2 Slack



Figure 4 Slack logo

We relied on Slack for seamless communication and real-time collaboration throughout the project. Slack enabled our team to exchange messages, share files, and engage in discussions related to project tasks and progress. By utilizing various channels within Slack, we were able to keep conversations organized and focused on specific topics. The platform's integration with other tools, including ClickUp, further improved our workflow, allowing us to stay connected and productive.

Chapter II: Fundamental Concepts

This chapter delves into the foundational concepts relevant to the project. It provides an overview of state-of-the-art advancements in generative AI, large language models (LLMs), embeddings, and chatbots. By exploring these areas, the chapter aims to establish an understanding of the methodologies that are integral to the project's development and implementation. It also highlights the evolution and current applications of these technologies, setting the stage for their practical use in the project.

II.1 Natural Language Processing (NLP)

Natural language processing (NLP) is a machine learning technology that allows computers to interpret, manipulate, and comprehend human language. It combines linguistics, computer science, and machine learning to efficiently process and analyze a large amount of natural language data. This technology is essential for a variety of applications, such as sentiment analysis, translation services, chatbots, and more.

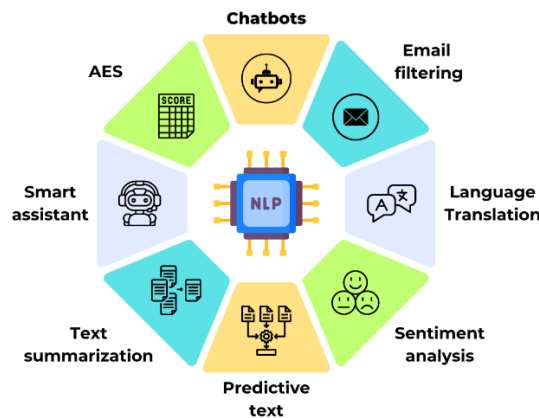


Figure 5 Applications of Natural Language Processing

II.1.1 Key Techniques in NLP

NLP uses several key techniques to convert raw text data into meaningful information that machines can process. These techniques include:

Sentence Segmentation	This is the first step in many NLP tasks and involves the division of text into separate sentences, enabling further analysis at the sentence level.
Word Tokenization	Breaking down text into individual words or tokens. This step simplifies language data processing by working with smaller, more manageable pieces.

Stemming	The process of reducing words to their base or root form. For example, the word "running" becomes "run." This helps combine different word forms into a single entity, which is useful for text analysis.
Lemmatization	Similar to stemming, but it reduces words to their dictionary form. It would lower "better" to "good." Lemmatization includes context and morphology, making it more accurate than stemming.
Stop Word Analysis	Removing frequent words like "and," "the," and "is" that provide less meaning in the analysis. This helps in focusing on the more significant words in the text.
Dependency Parsing	This approach helps understand the relationship between words in a sentence by analyzing the grammatical structure and word interdependencies.
Part-of-Speech Tagging	Assigning parts of speech (such as noun, verb, or adjective) to each word in a sentence. Grammatical knowledge aids parsing and sentiment analysis.

Table 1 Key Techniques in Natural Language Processing

II.1.2 Word Embeddings:

Word embeddings are a core component of natural language processing (NLP) and machine learning that transform textual data into numerical form. Unlike traditional one-hot encoding, which represents words as binary vectors in a sparse vector space, word embeddings use dense vector representations. Training models on large text corpora generates these embeddings, aiding in their understanding of the semantic meanings and relationships between words.

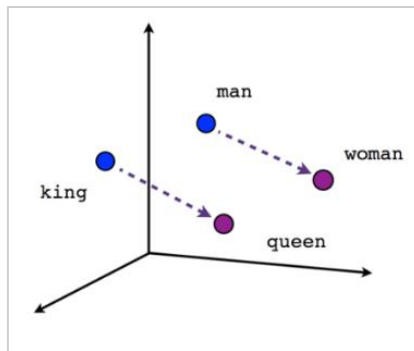


Figure 6 Vector Space Representation of Gender Relations in Word Embeddings

The key idea behind word embeddings is to represent words within a continuous vector space, positioning semantically similar words near each other. For example, in this vector space, the words "king" and "queen" may be close because they share similar contexts, while "king" and "apple" would be distant due to their unrelated contexts. This spatial representation effectively captures linguistic relationships such as synonyms, antonyms, and analogies. This can be seen in the vector difference between "king" and "queen," which parallels the difference between "man" and "woman."

II.1.2.1 Vector Embeddings:

Vector embeddings extend the concept of word embeddings to various data types, such as images, audio, and documents. In a multi-dimensional space, each vector represents a data point's key features, allowing for analysis and similarity searches across different data types.

The process of creating vector embeddings involves an embedding model that transforms raw data into fixed-size vectors. These vectors preserve data attributes and relationships as compact numerical representations. For instance, image vectors can capture visual characteristics, while audio vectors can represent sound patterns.

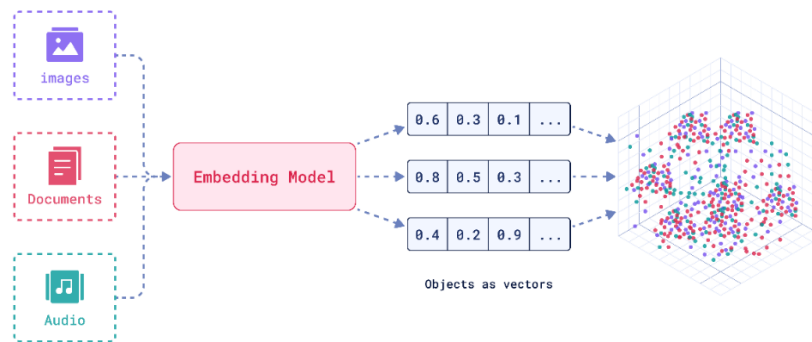


Figure 7 Embeddings Across Data Types

II.1.2.2 Vector Databases:

Vector databases are specialized systems designed to effectively manage the storing, indexing, and searching of vector embeddings. Vector databases, unlike traditional databases that manage structured data using rows and columns, optimize to handle the unique challenges of vector data, including handling large dimensionality and performing fast similarity searches.

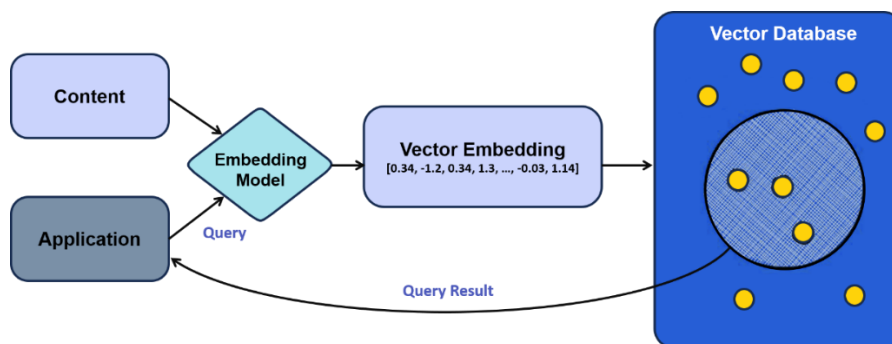


Figure 8 Query Processing in a Vector Database

In a vector database, data points are represented as vectors within a multi-dimensional space. When a query is made, the database employs similarity metrics, such as cosine similarity, Euclidean distance, or Manhattan distance, to measure the proximity between the query vector and the stored vectors.

This method identifies the vectors that are closest to the query, effectively retrieving the most similar items from the database.

For example, in an image search application, a user might input a photo, which the system converts into a vector. The system then retrieves visually similar images by comparing the vector of the input photo with the vectors stored in the database, enhancing the user experience in digital asset management and multimedia search.

II.1.2.3 Applications of Vector Embeddings and Vector Databases:

- Content-Based Recommendation Systems:** These systems can recommend items that are similar to those a user has previously interacted with. A movie recommendation system, for example, might embed movies as vectors based on their features, then recommend similar movies based on their proximity in vector space.
- Image and audio retrieval:** Vector embeddings enable efficient indexing and searching of multimedia content. The system transforms a user's uploaded picture or audio sample into a vector and then retrieves relevant items from the database, enhancing digital asset management and searching capabilities.
- Anomaly Detection:** By embedding data points as vectors, security and fraud detection systems can identify outliers or abnormalities that deviate from regular patterns, enabling the detection of fraud or security breaches.

II.2 Generative artificial intelligence

Generative AI is a subset of artificial intelligence focused on creating new content by learning from existing data. Generative AI models aim to generate new outputs like text, images, audio, video, animations, and 3D models. These models leverage complex algorithms and neural networks to learn the underlying structure and statistical distribution of the training data. As a result, they can generate new content that aligns with learned patterns while being original.

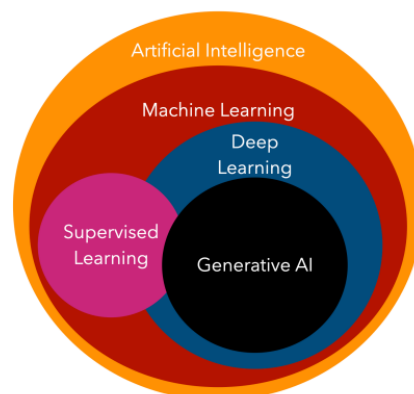
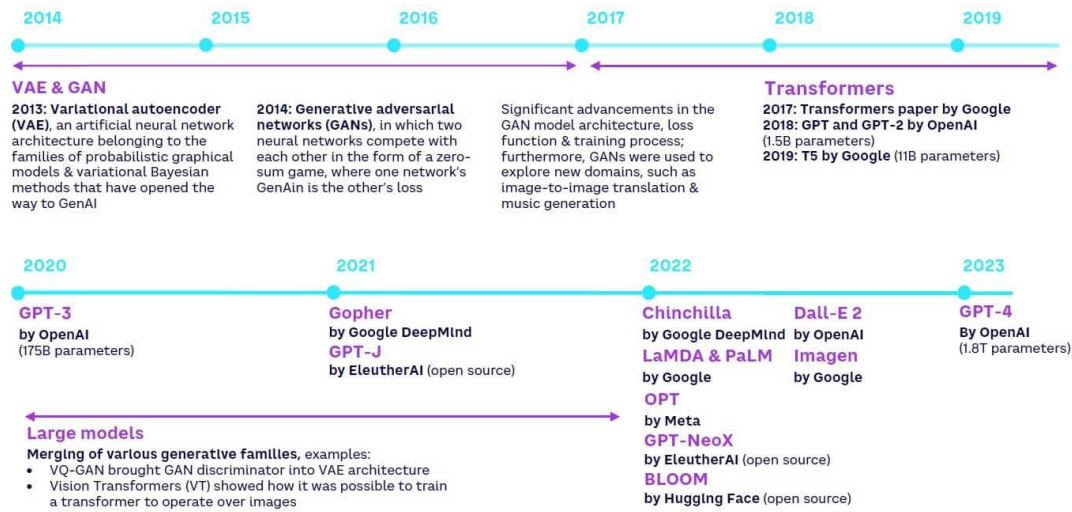


Figure 9 Interconnections Between Generative AI and Other AI Subfield.

II.2.1 The Evolution of Generative artificial intelligence:



Source: Arthur D. Little; Foster, David. *Generative Deep Learning*. O'Reilly Media, 2019; LinkedIn

Figure 10 The Evolution of Generative AI

The development of Generative AI has witnessed significant advancements over the past decade, driven by notable innovations in artificial intelligence. The timeline above captures key developments in generative AI models:

- **2013:** Variational Autoencoders (VAEs) and Generative Adversarial Networks (GANs) were introduced. VAEs allow generative models to learn latent space representations, and GANs use adversarial training to make synthetic data that looks like real data.
- **2017:** Breakthrough with Transformers, which revolutionized natural language processing by introducing the attention mechanism, laying the groundwork for advanced generative models like BERT, GPT, and T5.
- **2018-2019:** Development of GPT models that enhanced language generation through unsupervised learning and T5 that unified NLP tasks into a text-to-text format, improving task performance.
- **2020:** Introduction of GPT-3, With 175 billion parameters, GPT-3 became the largest and most powerful language model at the time of its release. It could generate human-like text across various prompts and exhibited capabilities in tasks like translation, summarization, and even basic reasoning, making it a versatile tool for numerous applications.
- **2021:** Google DeepMind developed an advanced language model, exploring scaling laws for language models and introducing GPT-J as an open-source alternative to GPT-3, thereby promoting research accessibility.
- **2022:** Models like Chinchilla were released with a focus on training efficiency. LaMDA and PaLM improved conversational AI. Both Dall-E 2 and Imagen improved text-to-image

synthesis. Also, open-source models like OPT, GPT-NeoX, and BLOOM encouraged inclusion and multilingual abilities.

- **2023:** Release of GPT-4, with 1.8 trillion parameters, represents one of the most advanced generative models to date, pushing the boundaries of text generation and demonstrating its capabilities across diverse applications.

II.2.2 Types of Generative AI:

Based on the data they produce; we can classify generative AI into several categories:

- **Language-Based Generative AI:** Models that generate text-based content, such as essays, stories, or code. Examples include GPT-3 and GPT-4.
- **Visual Generative AI:** Models that focus on creating visual content, such as images, videos, and 3D models, often using GANs to generate highly realistic visuals.
- **Auditory Generative AI:** Models that generate audio content, such as music and speech.
- **Synthetic Data Generation:** Models that create artificial datasets to train other AI models.

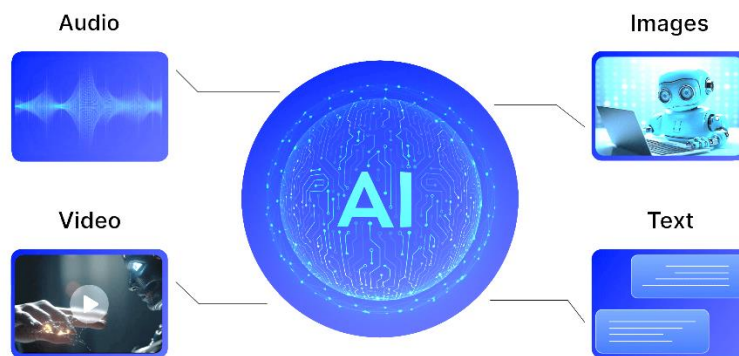


Figure 11 Types of Generative AI

II.2.3 Applications of Generative AI:

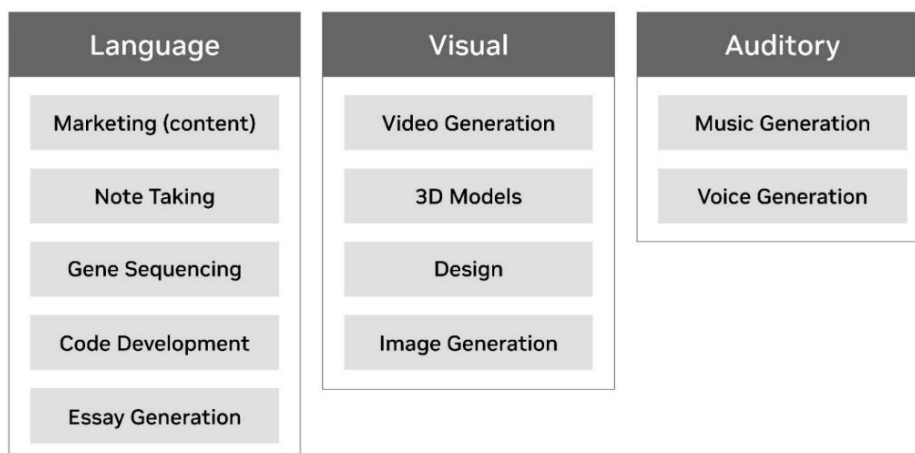


Figure 12 Generative AI Use Cases

II.3 Large Language Models (LLMs):

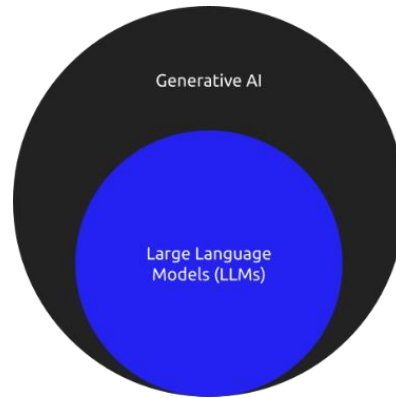


Figure 13 Interconnections Between Generative AI and Large Language Models

Generative AI and Large Language Models (LLMs) are key components of modern artificial intelligence. Generative AI refers to a broad category of models that create new content across various formats, such as text, images, and audio. LLMs, a subset of generative AI, specifically focus on generating text-based outputs by leveraging large datasets and complex neural networks.



Generative AI

- Creates many outputs
- Myriad of generative AI tools
- Built on LLMs but also other types of machine learning models

VS.



Large Language Models

- Create text-only outputs
- Myriad of parameters to understand and produce text
- Foundation for text-based generative AI tools like ChatGPT

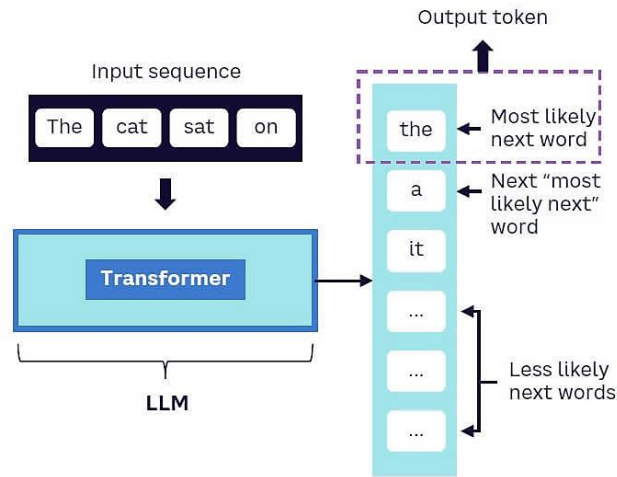
Figure 14 Generative AI Vs Large Language Models

II.3.1 How LLMs Work:

A fundamental component of LLMs is the transformer architecture, a neural network model that processes input data in parallel rather than in a sequential manner. This architecture consists of an encoder and a decoder:

- **Encoder:** This component reads the input text and creates a representation of the input data, capturing its semantic meaning and the relationships between words and phrases.
- **Decoder:** The decoder takes the encoded input and generates output text, predicting the next word or token based on the context provided by the input.

Transformers use attention mechanisms to enhance their ability to handle context effectively. When generating each word, attention mechanisms allow the model to focus on specific parts of the input, which helps to understand context and generate more coherent and contextually relevant text.



Source: "Introduction to Large Language Models." Google Cloud, 2023

Figure 15 Token Prediction Process in a Large Language Model

II.3.2 Training and Fine-Tuning of LLMs

Training an LLM involves feeding the model large amounts of text data to process it and learn the statistical properties of language. The training data helps the model understand language, meaning, and even world knowledge, enabling it to correctly predict the next word in a sequence.

LLMs are commonly trained using a self-supervised learning approach, where the model learns to predict parts of the input data itself (e.g., the next word in a sentence) without requiring explicit labels. This method allows LLMs to effectively acquire knowledge from unstructured data.

To fine-tune an LLM, there are several approaches:

- **Zero-Shot Learning:** Based on the prompt's provided context, the model can perform a task without specific training.
- **Few-Shot Learning:** We provide the model with a few examples of a task to improve its performance on that task.
- **Fine-Tuning:** To optimize the model's performance for specific applications, we further train it on a smaller, task-specific dataset.

II.3.3 Examples of Powerful LLMs:

Model	Provider	Open-Source	Speed	Quality
GPT-4	OpenAI	No	★☆☆	★★★★★
GPT-3.5-turbo	OpenAI	No	★★★	★★★★☆

GPT-3	OpenAI	No	★★☆	★★★★☆
Claude-instant	Anthropic	Yes	★★★	★★☆☆☆
Claude	Anthropic	Yes	★★☆	★★★★☆
T5	Google	Yes	★★☆	★★☆☆☆
BERT	Google	Yes	★★★	★★☆☆☆
command-xlarge	Cohere	No	★★☆	★★☆☆☆
command-medium	Cohere	No	★★★	★★☆☆☆
ada, babbage, curie	OpenAI	No	★★★	★★☆☆☆
T5	Google	Yes	★★☆	★★☆☆☆
PaLM	Google	Yes	★★☆	★★☆☆☆
LLaMA	Meta AI	Yes	★★☆	★★☆☆☆
CTRL	Salesforce	Yes	★★★	★★☆☆☆
Dolly 2.0	Databricks	Yes	★★☆	★★☆☆☆

Table 2 Comparison of existing LLMs

Among the various LLMs, OpenAI's models, particularly GPT-3 and GPT-4, stand out as leaders in the field. Their ability to generate high-quality and contextually relevant text has established them as the industry standard for a wide range of applications.

The main characteristics of GPT models include:

- **Transformer Architecture:** GPTs are based on the transformer architecture, which enables them to handle long sequences of text efficiently.
- **Pre-training and Fine-tuning:** GPT models undergo two main phases: pre-training and fine-tuning. During pre-training, the model learns general language patterns from a massive dataset. In the fine-tuning phase, the model is adjusted to perform specific tasks by training on a smaller, task-specific dataset.
- **Scalability:** GPT models are known for their large scale, with GPT-3 having 175 billion parameters, which allows GPTs to generate highly sophisticated and human-like text across a wide range of applications, from writing to generating code and answering questions.
- **Versatility:** With their wide range of applications in natural language processing, GPTs are incredibly versatile. They can handle tasks such as text completion, translation, summarization, and even act as conversational agents like ChatGPT.

II.3.4 Applications of Large Language Models

- **Copywriting:** LLMs can generate marketing content, articles, and other written materials, saving time and resources for content creators.
- **Knowledge Base Answering:** LLMs can retrieve and generate responses based on vast amounts of stored information, making them valuable for customer service and information retrieval.
- **Text Classification:** LLMs can classify text based on sentiment, topic, or other criteria, aiding in tasks like sentiment analysis and document categorization.
- **Code Generation:** LLMs are capable of generating code snippets in various programming languages, assisting developers in automating repetitive coding tasks.
- **Conversational AI:** LLMs power chatbots and virtual assistants, enabling them to understand user intent and provide relevant responses in natural language.

II.4 Prompt Engineering

Prompt engineering is a key technique for optimizing the use of generative AI models, particularly large language models (LLMs). It involves crafting and refining inputs is crucial in guiding the AI model to generate specific and relevant outputs.

The effectiveness of these models heavily depends on the quality and precision of the prompts, ensuring that AI understands the context and intent behind queries, which directly impact the accuracy and relevance of the responses generated.

II.4.1 Key Uses of Prompt Engineering:

- **Guiding Output:** Prompts direct the AI to produce specific outputs, such as detailed reports, creative writing, or mimicking a particular style (formal or conversational).
- **Imparting Personality:** prompts can adjust the AI's tone, making it respond humorously, formally, or empathetically, enhancing the user experience in applications like customer service.
- **Providing Instructions:** Prompts can specify the structure, style, or depth of the response, giving users the power to direct how the AI approaches a task, such as providing a step-by-step outline or summarizing information.
- **Setting Context:** Including contextual details in prompts ensures the AI's responses are relevant and context-aware. Additionally, this approach can introduce new information that the models haven't been trained on.

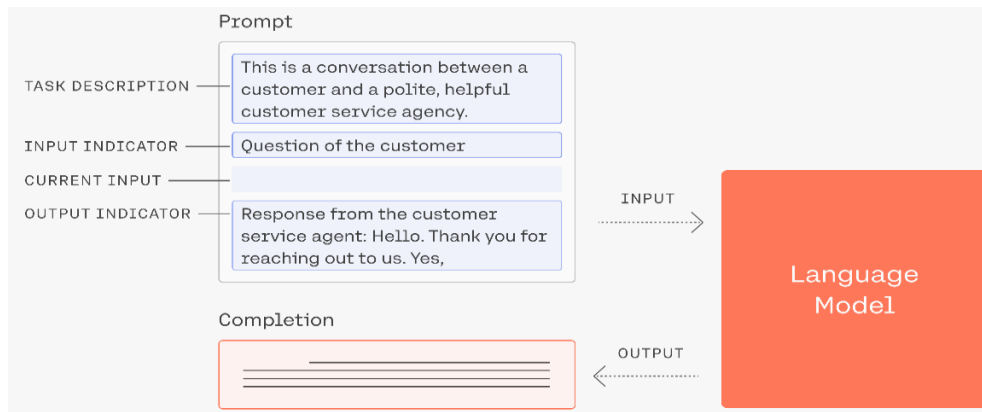


Figure 16 Example components of Prompt Engineering for Language Model Input and Output

II.4.2 Techniques for Effective Prompt Engineering:

Prompt engineering is a powerful tool for leveraging the full potential of LLMs across various applications. A properly crafted prompt has the potential to serve as the basis for creative applications or features. In order to achieve optimal results from generative AI models, it is essential to prioritize a number of important techniques:

- **Clarity and Specificity:** The prompt should clearly outline the expected responses from the AI. Vague or ambiguous prompts often lead to generic or irrelevant responses.
- **Iterative Refinement:** Developing the perfect prompt is often an iterative process. By continuously refining and tweaking prompts based on the AI's output, users can achieve more precise and satisfactory results.
- **Contextual Relevance and Instructions:** Include relevant context, examples, and direct instructions in the prompt to ensure the AI understands the format, style, and specific requirements of the desired output. This guarantees that the AI tailors its response to the particular situation or use case.

II.5 Retrieval-Augmented Generation (RAG)

Large language models (LLMs) are highly effective at generating human-like text and understanding complex queries, but their knowledge is limited to the data they were trained on, which can become outdated or lack specific information. Retrieval-Augmented Generation (RAG) addresses this limitation by enhancing LLMs with real-time retrieval of relevant information from external sources, such as vector databases. This technique ensures that the responses generated are not only coherent but also accurate and up-to-date, especially when specialized or current information is needed.

II.5.1 Key Components of RAG:

- **Embedding and embedding models:** Convert input text into numerical vectors that capture semantic meaning.

- **Vector Database:** Stores and retrieves these vectors based on similarity to ensure relevant information is accessed.
- **Large Language Models (LLMs):** Generate text based on the retrieved information provided within the prompt.

By integrating these components, RAG systems are able to deliver responses that are not only contextually relevant, but also informed by the latest available data.

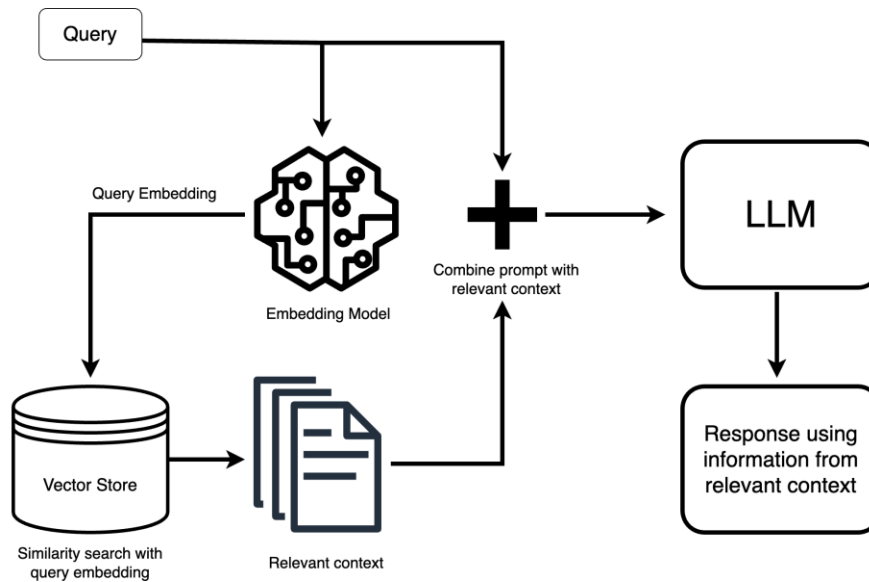


Figure 17 Workflow of a Retrieval-Augmented Generation (RAG) System

II.6 Chatbots

As artificial intelligence has evolved, one of its most impactful applications has been in the development of **chatbots**. These are software programs designed to simulate human-like conversations through text or voice interactions. Leveraging advances in natural language processing (NLP) and machine learning, chatbots can handle a wide range of tasks, from answering basic questions to managing complex interactions.

II.6.1 Evolution of chatbots

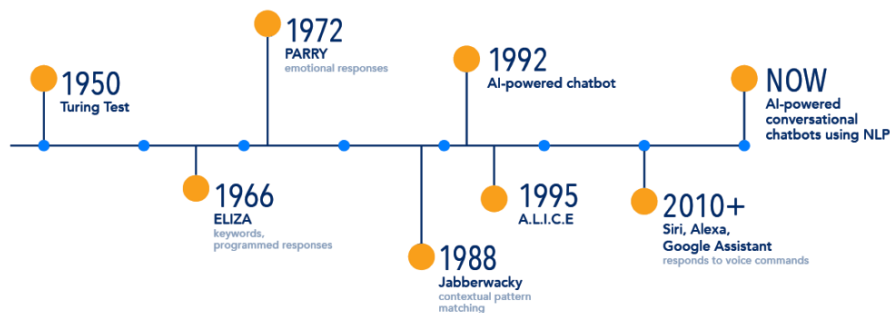


Figure 18 History of chatbots

Chatbots have come a long way since their inception, evolving through various stages of development. The journey began with the Turing Test in 1950, a concept introduced by Alan Turing to determine whether a machine could exhibit human-like intelligence. This set the foundation for future developments in artificial intelligence and conversational agents. In 1966, ELIZA, one of the earliest chatbots, was created to simulate a psychotherapist using simple pattern matching and substitution methodologies. Fast forward to 1972, PARRY was developed, mimicking a person with paranoid schizophrenia, showcasing more advanced natural language processing capabilities.

The evolution continued with more sophisticated chatbots like ALICE in 1995, which used natural language processing to carry out more meaningful conversations, and Jabberwacky in 1988, focusing on creating more human-like interactions through learning from past conversations. By the 2010s, we saw the emergence of advanced AI-powered conversational chatbots such as Siri, Alexa, and Google Assistant, capable of understanding voice commands and executing tasks. Today, chatbots utilize sophisticated natural language processing (NLP) and machine learning (ML) techniques, allowing them to handle complex queries and provide personalized responses in real-time.

II.6.2 Types of Chatbots

Chatbots can be broadly categorized based on their underlying technologies and functionalities:

Rule-Based Chatbots: These are the simplest form of chatbots that operate on predefined rules and responses. They are suitable for straightforward tasks but lack the flexibility to handle complex queries or understand nuanced language.

Conversational AI Chatbots: These advanced chatbots leverage AI and NLP to understand and generate responses based on the input they receive. They are capable of managing multi-turn dialogues, understanding context, and adapting their responses accordingly, making them ideal for more dynamic and interactive applications.

Contextual Chatbots: These chatbots utilize machine learning to remember past interactions and use this data to provide relevant and context-aware responses. They are designed to learn and improve over time, making them more effective in handling complex user interactions.

Voice-Enabled Chatbots: As the name suggests, these chatbots can process and respond to voice inputs. They are commonly integrated into virtual assistants like Siri and Alexa, enabling hands-free operation and accessibility.

Hybrid Chatbots: These combine rule-based and AI-driven approaches to offer more flexibility and adaptability in handling diverse user interactions. They can manage simple queries using rule-based logic while employing AI for more complex, contextual conversations.

II.6.3 Benefits of Chatbots

Chatbots provide several advantages that make them valuable tools for businesses and users alike:

Benefit	Description
24/7 Availability	Chatbots are available around the clock, ensuring continuous support and service without downtime.
Multilingual	They can communicate in multiple languages, making them accessible to a global audience.
Consistency in Answers	Chatbots provide uniform responses, ensuring that every customer receives the same information.
Seamless Transaction	They can handle transactions smoothly, improving the customer experience and reducing friction.
Instant Response	Chatbots provide immediate answers to user queries, enhancing customer satisfaction.
Omni-channel	They can be integrated across various platforms, offering consistent support through different channels.
Personalized Interaction	Chatbots can tailor their responses based on user data, providing a more personalized experience.

Table 3 Different Benefits of Chatbots

II.6.4 Traditional Chatbot Vs Conversational AI

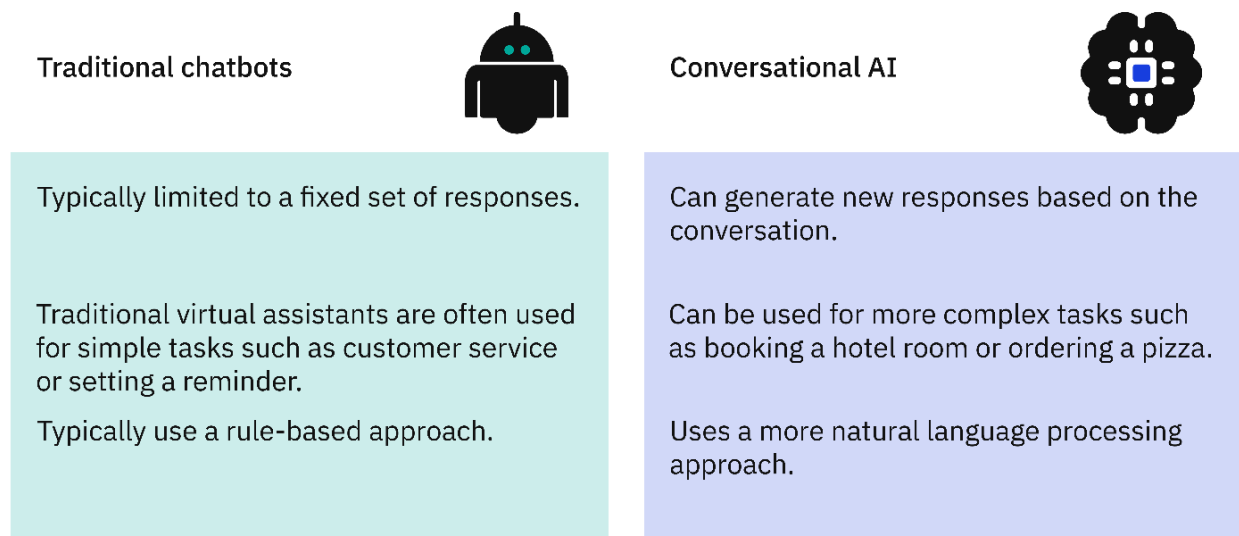


Figure 19 Difference between Conversational AI and Traditional chatbots

Overall, the progress of chatbots from simple rule-based systems to advanced conversational agents has greatly expanded their abilities and potential uses in a wide range of industries. With the help of AI and NLP, modern chatbots have the ability to offer highly personalized, efficient, and engaging experiences, resulting in increased user satisfaction and improved operational efficiency.

Chapter III: Tools and Technologies

This chapter outlines the tools and technologies used in the development of the chatbot system. We discuss the role of FastAPI for the backend, the integration of MongoDB for database management, and the utilization of OpenAI APIs for natural language understanding and response generation. The chapter also covers LangChain and LangGraph, which were critical in managing multi-turn conversations and ensuring the chatbot's scalability and flexibility.

III.1 Development Tools and Environments

III.1.1 Python



Figure 20 Python logo

Python is a high-level, interpreted programming language known for its simplicity and readability, making it accessible to developers of all levels. It is the primary language for our project due to its extensive libraries and frameworks for machine learning and data science, which are essential for AI development.

III.1.2 Visual studio Code



Figure 21 VS Code Logo

Visual Studio Code (VS Code) is a free, open-source code editor developed by Microsoft. It is popular among developers for its lightweight design and extensive functionality. VS Code is highly customizable, with a wide range of extensions that enhance productivity and support various programming languages and frameworks. All of these features, including an incorporated terminal, Git integration, and debugging tools, simplify the processes of coding, testing, and debugging, making it suitable for diverse development tasks, including web development, data science, and AI.

III.1.3 GitHub



Figure 22 Github Logo

GitHub is a web-based platform that uses Git, a distributed version control system, for code collaboration and management. It provides repositories for hosting code, tracking changes, collaborating through pull requests, and conducting peer reviews with comments and inline code suggestions. Its collaboration features and use of branches enabled us to work on different features simultaneously, ensuring that code remains clean, organized, and well-documented.

III.1.4 Next.js

Next.js is an open-source React framework that supports server-side rendering (SSR) and static site generation (SSG) for web applications. Developed by Vercel, it improves performance and SEO by rendering pages on the server rather than in the client's browser, leading to faster load times and a better user experience, particularly for dynamic websites with frequently changing content. Its integration with React allows developers to use React's component-based architecture while gaining additional performance benefits.

III.2 Backend Frameworks and Libraries

III.2.1 FastAPI

FastAPI is a high-performance web framework for building APIs with Python, designed for ease of use and efficiency. It supports asynchronous programming, making it suitable for applications requiring high concurrency and low latency. FastAPI also automatically handles data validation and serialization, reducing the amount of code developers need to write. These features make it a great choice for creating reliable and scalable backend systems.

III.2.2 LangChain and LangGraph

LangChain:

LangChain is a library that simplifies the development of applications powered by large language models (LLMs). It simplifies the integration of LLMs, making it easier to handle complex workflows, external data retrieval, and interactive conversations. With a comprehensive set of tools, LangChain enables developers to build sophisticated, dynamic, and context-aware systems for various tasks such as chatbots, content generation, and question-answering.

Key Components of LangChain:

- **LLM Interface:** LangChain provides a simple API to connect with various LLMs, such as GPT or Bard, allowing developers to interact with these models without having to manage the complexities of integration.
- **Prompt Templates:** Prompt templates are reusable structures that help format queries in a way that the LLM can understand and respond to appropriately. Developers can create these templates to standardize inputs across different tasks, ensuring accurate and consistent responses.
- **Chains:** The core concept in LangChain, chains are sequences of operations that connect user inputs to the model's output. A chain can include multiple steps, such as fetching data, transforming it, querying the LLM, and formatting the response. This allows developers to break down complex tasks into manageable steps.
- **Agents:** Agents take chains a step further by introducing dynamic decision-making into workflows. They can assess the input and determine the appropriate next step or tool to use. Agents enable applications to handle more complex queries or tasks that involve multiple steps, such as managing diverse types of data sources or processing user interactions in real-time.
- **Memory:** LangChain supports memory in applications, allowing them to recall previous interactions and maintain context across conversations. This is crucial for building advanced conversational agents that need to respond based on past exchanges, whether in a simple recent context or through more detailed historical analysis.
- **Retrieval Modules:** LangChain facilitates the creation of Retrieval-Augmented Generation (RAG) systems, where the model can access external data, such as internal documents, to generate more accurate and contextually relevant responses. This feature enhances LLMs' capabilities by allowing them to incorporate up-to-date or domain-specific information without needing to be retrained.

LangGraph:

LangGraph extends LangChain by enabling more complex workflows that involve cyclical, dynamic interactions. It introduces the ability to create **graph-based workflows** where nodes (representing tasks or agents) and edges (representing the flow of data or decisions) can loop back, allowing for iterative processes, which makes it ideal for complex AI-driven systems such as dynamic chatbots, automated reasoning, and interactive applications.

Key Features of LangGraph:

- **StateGraph:** At the core of LangGraph is the StateGraph, where each node represents a task or decision point, and the state evolves as the process progresses. This state is updated dynamically as each node contributes to the overall outcome, enabling sophisticated multi-step interactions.
- **Cyclic Workflows:** Unlike traditional chains, LangGraph allows for cycles, meaning that tasks can be repeated or revisited based on the results of previous steps.

- Agent Executors:** LangGraph enhances agent-based workflows by allowing agents to execute tasks in a loop, reasoning at each step about what to do next. This is particularly valuable in scenarios like chatbot systems or RAG implementations, where an LLM may need to reconsider and refine its approach to a problem repeatedly.
- Conditional Flows:** In LangGraph, developers can define conditional edges, where the flow between nodes is determined by the output of previous tasks. This adds flexibility to workflows, allowing the application to adapt dynamically to different scenarios and input variations.

III.2.3 Docker

Docker is a platform that automates the deployment of applications in lightweight, portable containers. These containers package all necessary components, including code, runtime, libraries, and system dependencies, ensuring consistency across different environments and preventing compatibility issues. In this project, Docker was used to containerize backend services, enabling seamless deployment, which improved the development workflow.

III.3 Databases and Storage

III.3.1 MongoDB database

MongoDB is a NoSQL database known for its scalability, flexibility, and ease of use. Unlike traditional relational databases, it stores data in flexible, JSON-like documents with dynamic schemas that can easily adapt to changing data structures. This makes MongoDB ideal for applications that need quick development and frequent updates.

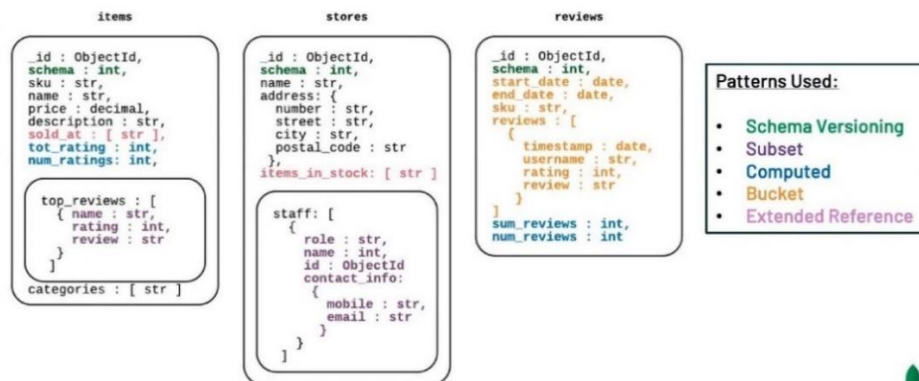


Figure 23 MongoDB Data Model and Document Structure

Widely used across industries, MongoDB efficiently handles large volumes of unstructured data. It's well-suited for use cases like Internet of Things (IoT), mobile apps, real-time analytics, content management systems, and personalization engines, all of which benefit from its ability to manage diverse data types and provide fast, scalable solutions.

For cloud-based deployments, MongoDB offers MongoDB Atlas, a fully managed cloud database service that simplifies operations by automating tasks such as provisioning, backups, and scaling. MongoDB Atlas ensures high availability, security, and seamless integration with major cloud providers, handling database management complexities so teams can focus on development, making it an excellent choice for companies looking to build scalable, secure, and reliable cloud applications.

III.3.2 Amazon Web Service

Amazon Web Services (AWS) is a comprehensive cloud computing platform offering a range of services, including computing power, storage options, and databases. In this Project, AWS was used for its reliable infrastructure and its flexibility to adjust resources dynamically based on demand. AWS services, such as Amazon S3 for storage, Amazon EC2 for computing, and Amazon RDS for managed databases, provided a robust and adaptable environment for managing the project’s backend, while offering global reach and high availability to ensure reliable performance for users worldwide.

III.3.3 Redis

Redis (Remote Dictionary Server) is an open-source, in-memory data structure store that functions as a high-performance database, cache, and message broker. Unlike traditional relational databases that store data on disk, Redis operates entirely in memory, resulting in exceptionally fast read and write operations. Its speed and versatility make it ideal for use cases requiring low-latency data access, such as caching, real-time analytics, session management, and high-speed data processing.

Classified under the NoSQL database category, Redis stores data in the form of key-value pairs where each key is a unique identifier, and the value can be one of several supported data types. These data types include strings, hashes, lists, sets, sorted sets, bitmaps, hyperloglogs, and streams, each of which has specific operations and commands tailored to efficiently manage data in various scenarios. For example, strings can be manipulated with operations like SET and GET, lists can be used to create queues or stacks with commands like LPUSH and RPOP, while sets handle collections of unique elements with commands like SADD and SREM.

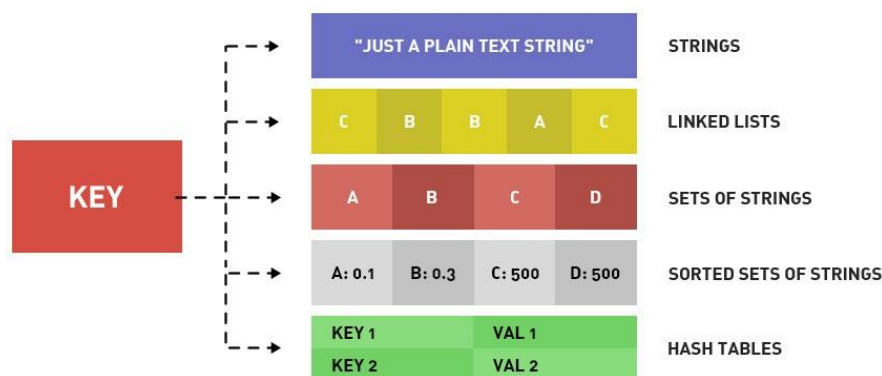


Figure 24 Redis Data Types and Key-Value Storage Model

III.4 APIs and Integrations

III.4.1 OpenAI APIs

OpenAI offers a range of APIs that allow developers to integrate powerful language models into their applications, catering to different needs, from simple text generation to more structured and context-rich interactions, enabling a wide array of use cases.

III.4.1.1 Standard Chat Completion API

The **OpenAI Chat Completion API** utilizes models like GPT-4 to generate conversational responses based on user input. This API is designed for tasks that require coherent and contextually relevant text generation, much like interacting with ChatGPT in a conversational format.

Capabilities of the Chat Completion API:

- **Natural Language Understanding:** The API is proficient in understanding and processing natural language inputs, making it suitable for conversational applications.
- **Content Generation:** It excels at generating human-like text, making it ideal for creative writing, customer support, and any application requiring dynamic text responses.
- **Customizable Responses:** By modifying the prompts, developers can tailor the responses to fit specific needs, guiding the model's behavior to match desired interaction styles.
- **Context Management:** The Chat Completion API can maintain context across multiple turns within a session, similar to chatting with ChatGPT. However, it primarily operates in a session-based manner, without retaining long-term context or utilizing external documents.

III.4.1.2 OpenAI Assistant API

The **OpenAI Assistant API** builds upon the capabilities of the standard Chat Completion API, adding advanced tools for managing more structured interactions. This API is ideal for applications that require detailed context management, dynamic guidance, and the ability to reference external documents to provide accurate and informed responses.

Key Features of the OpenAI Assistant API:

- **Instructions for Customization:** Developers can provide specific instructions that dictate how the assistant should behave and respond, allowing for highly tailored interactions.
- **Dynamic Context Management:** The Assistant API maintains a more sophisticated level of context throughout interactions, useful for complex multi-turn dialogues that require understanding of previous exchanges and coherent progression.

- **Grounding with Documents:** A standout feature of the Assistant API is its ability to store and reference external documents, allowing it to retrieve and incorporate relevant information from these documents into its responses, making it suitable for applications that require data retrieval.
- **Multi-Modal and Multi-Language Support:** The API supports multiple languages and can potentially handle various media types, depending on the integration, broadening its applicability across different use cases.
- **Seamless Integration:** Designed for cross-platform compatibility, the Assistant API ensures a uniform user experience across different digital environments.

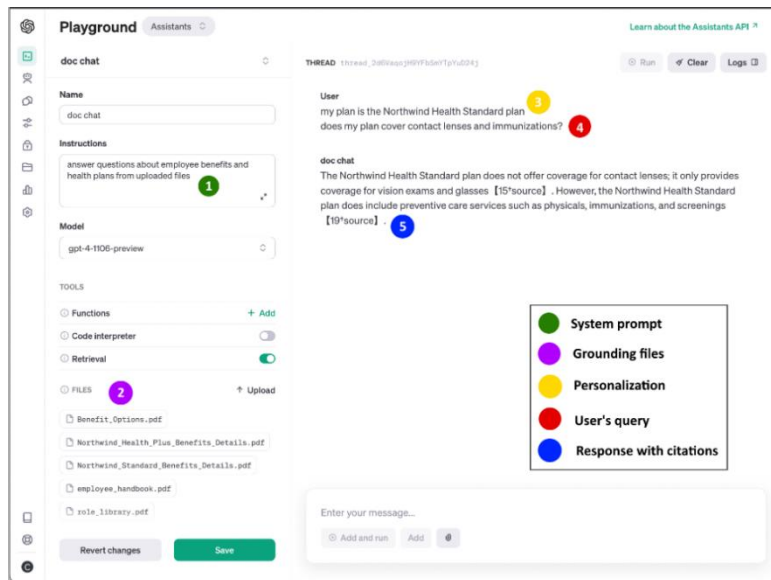


Figure 25 OpenAI Assistant API playground interface

The previous figure demonstrates the OpenAI Assistant API playground interface, highlighting how instructions, grounding documents, and user inputs are managed to generate context-aware responses. This visual representation helps in understanding how the API operates and can be customized for different applications.

III.4.1.3 OpenAI Whisper

OpenAI Whisper is a speech recognition system that converts spoken language into written text. We integrated this tool into our project to enable voice-based interactions, allowing users to communicate with the chatbot via spoken commands. Whisper's accurate transcription of speech supports a more natural and accessible user interface for those who prefer voice over text inputs. Its integration with the OpenAI API ensures seamless processing of voice commands, enhancing the overall functionality and user experience of the chatbot system.

III.4.2 Meta Cloud API

Meta's Cloud API is a powerful platform that enables businesses to integrate Meta services, such as Facebook Messenger, Instagram Direct, and WhatsApp, directly into their applications. This integration supports the automation of messaging, customer interactions, and other key functions across these platforms, making it a vital tool for businesses seeking to improve their customer service and operational efficiency. For example, a business could use Meta Cloud API to manage customer inquiries on Facebook Messenger, send real-time updates via WhatsApp, and engage customers through Instagram—all from a single interface.

III.4.2.1 WhatsApp Cloud API

The WhatsApp Cloud API is a specialized extension of Meta's Cloud API, designed specifically for integrating WhatsApp's messaging capabilities into business systems. It automates and streamlines tasks such as order confirmations, customer support, and sales processes with minimal human intervention. With support for multiple programming languages and rich media capabilities, this versatile API adapts to businesses of all sizes. Its flexibility enables companies to sustain continuous communication with customers on WhatsApp, improving both engagement and satisfaction.

The WhatsApp Cloud API offers powerful features that help businesses automate interactions and enhance communication. Some of the key features include:

III.4.2.2 WhatsApp Templates:

WhatsApp Templates are pre-approved message formats that businesses can use to initiate conversations or send notifications outside the standard 24-hour messaging window. They are essential for maintaining consistent communication, ensuring that important updates reach customers promptly.

For instance, a business might use a WhatsApp template to send order confirmations, appointment reminders, or promotional offers. These templates can be customized with variables such as the customer's name or order details, allowing for personalized and direct interactions.

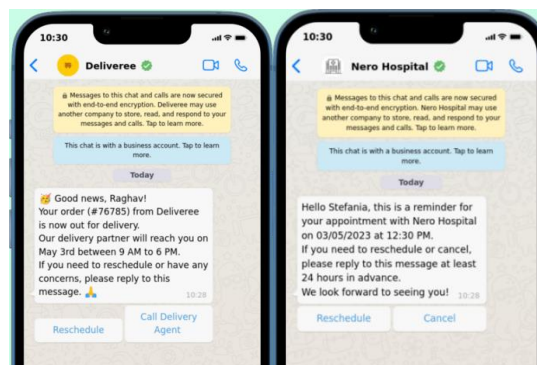


Figure 26 Example of WhatsApp Templates for Customer Notifications

III.4.2.3 WhatsApp Flows

WhatsApp Flows are a sophisticated feature of the WhatsApp Business API, designed to create interactive and automated conversation pathways. These flows allow businesses to guide users through a series of predefined steps, automating customer interactions to ensure a smooth user experience. Unlike simple messaging sequences, WhatsApp Flows incorporate various components—such as text, images, buttons, and input fields—creating a cohesive journey within the chat interface.

These workflows are defined using Flow JSON, a structured format developed by Meta to help build and customize interactions within WhatsApp. Businesses can utilize this format to access WhatsApp Flows features through a custom JSON object, which specifies the structure of the flow, including the screen sequence, user input fields, and actions triggered by user responses. The workflows are fully initiated, executed, and managed entirely inside WhatsApp, enabling the creation of complex multi-screen interactions that guide users through processes such as booking, customer service inquiries, and product returns.

For example, a Flow JSON might specify a screen asking for the user’s first and last names, followed by another screen asking for their email address. Depending on the inputs, the flow can proceed to different screens or trigger specific actions like sending a confirmation message or logging the data in a backend system.

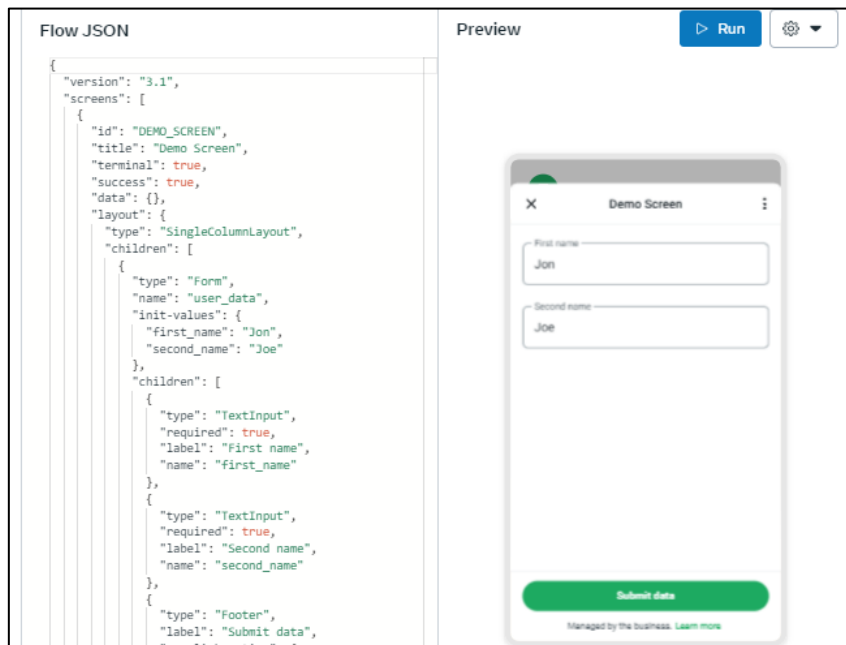


Figure 27 Example of FLOW JSON and its result

The key advantage of using WhatsApp Flows lies in their ability to provide a richer user experience through structured and interactive conversation pathways. By leveraging the WhatsApp Cloud API and

Flow JSON, businesses can streamline customer interactions, automate routine tasks, and enhance overall engagement on the platform.

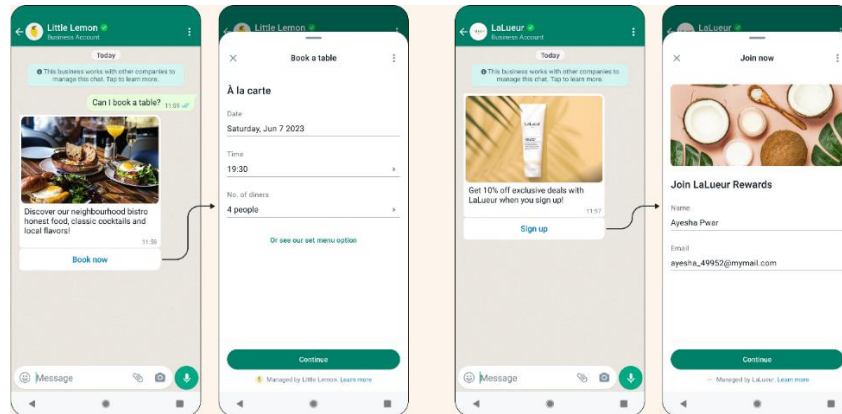


Figure 28 Example of WhatsApp Flows for Booking and Customer Engagement

III.4.2.4 Webhooks

Webhooks are a powerful component of the WhatsApp Cloud API, designed to facilitate real-time communication between WhatsApp and other integrated systems. A webhook is essentially an HTTP callback that is triggered by specific events, such as receiving a new message or updating the status of a message. And when using WhatsApp flows and templates, webhooks captures customer's interactions in real time, such as clicking a button, submitting a form, or sending a response. This allows businesses to automate responses or actions based on these events, enhancing the efficiency and customization of customer interactions.

For example, businesses can configure webhooks to automatically send a follow-up message if a customer does not respond within a specified timeframe, or to update a database when a customer's query is resolved.

III.4.3 Hostaway



Figure 29 Hostaway logo

Hostaway is an all-in-one property management software (PMS) and channel manager designed specifically for vacation rental businesses. It serves as a comprehensive solution for property managers and owners by centralizing and automating various aspects of rental operations. Hostaway integrates seamlessly with major booking platforms like Airbnb, Booking.com, Vrbo, and Expedia, allowing users to manage multiple properties across different channels from a single interface.

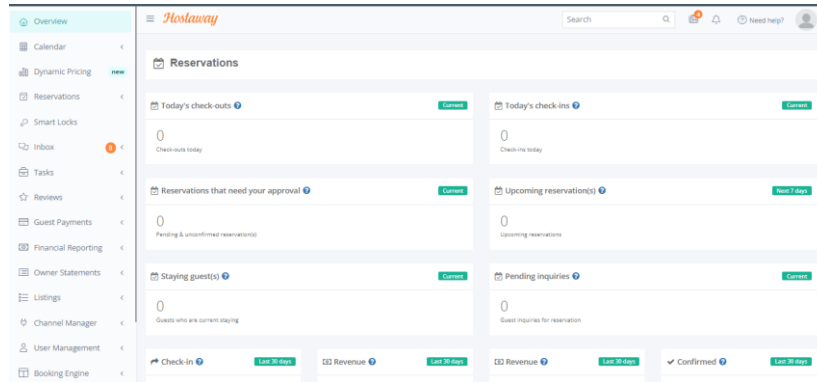


Figure 30 Overview of Hostaway's Dashboard

III.4.3.1 Key Features of Hostaway

Hostaway simplifies property management by offering a wide range of features:

- Centralized Property Management:** As a unified platform, Hostaway allows property managers to oversee listings, reservations, guest communications, and payments across multiple booking sites, reducing the need for multiple platforms and streamlining operations.
- Channel Manager:** Hostaway's capabilities ensure real-time synchronization of property listings, availability, and pricing across all connected booking platforms, preventing double bookings and optimizing revenue.
- Automated Messaging and Communication:** Hostaway automates guest communications by sending pre-configured messages for various stages of the booking process, such as confirmations, check-in instructions, and check-out reminders.
- Advanced Analytics and Reporting:** The platform provides in-depth analytics and reporting tools that allow property managers to monitor performance metrics like occupancy rates, revenue, and booking sources, enabling data-driven decision-making to improve profitability.
- API Integration:** The Hostaway API provides developers with the ability to access and manipulate data stored within the Hostaway platform. This API is essential for creating custom integrations and applications that can extend the functionality of Hostaway to better suit the unique needs of each property management business. This feature is particularly beneficial for property managers who need to synchronize data with other software systems, such as accounting or customer relationship management (CRM) tools.
- Task Management and Automation:** Hostaway includes tools for managing and automating tasks such as housekeeping, maintenance, and other operational duties. Property managers can assign tasks, track their progress, and receive notifications upon completion, ensuring that properties are well-maintained and guest-ready at all times.

Chapter IV: System design and implementation

Chapter 4 dives into the technical details of the system’s design and implementation. It begins with the preparation of the database, followed by the integration of APIs, and concludes with the construction of the chatbot’s Retrieval-Augmented Generation (RAG) system. This chapter also addresses the challenges faced during the development process and how they were overcome by transitioning to a more advanced system leveraging LangGraph for routing queries.

IV.1 Database Preparation

To develop our AI-powered customer service chatbot for property management, establishing a robust and flexible database was essential. The database needed to maintain data integrity, ensure scalability, and allow seamless integration with various property management platforms. This setup enables real-time updates and accurate responses to user inquiries.

IV.1.1 HostAway API Integration

HostAway serves as our main property management system, offering comprehensive API capabilities to manage property listings, reservations, and calendars. By integrating the hostAway API, we made sure that our database stayed in sync with the hostAway system, providing our customer support with easily accessible, up-to-date information.

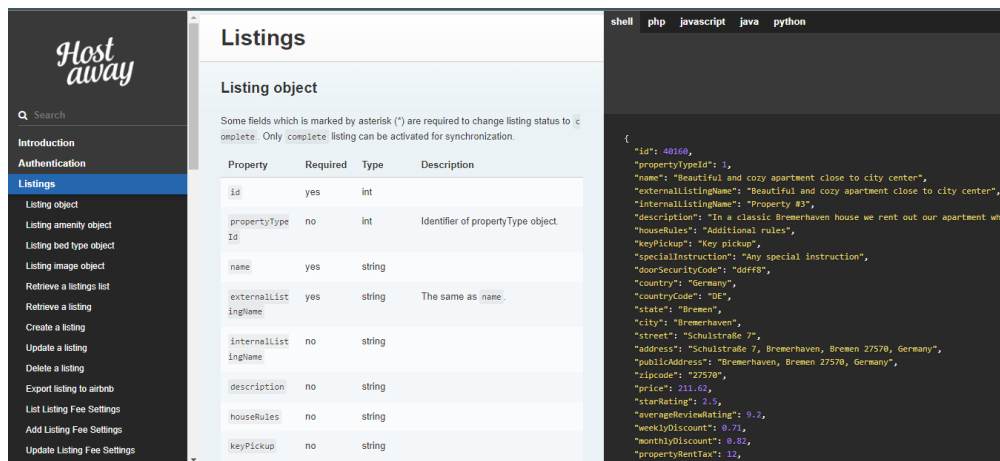


Figure 31 Example of Hostaway API Listing Fields

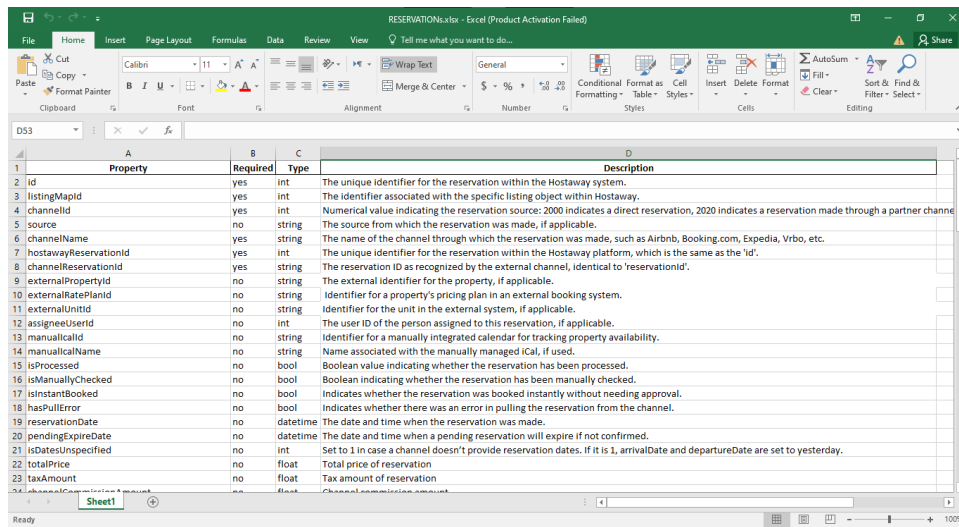
In order to get insight into what our database should be prepared to receive from Hostaway, we conducted a detailed analysis of the HostAway API to ensure a thorough understanding of its data structure and functionality, focusing on key data points such as property specifications and reservation details.

IV.1.2 Custom Database Design and Field Analysis

Based on our analysis of HostAway, along with insights from other platforms like Booking.com and Airbnb, we started the process of creating a customized database structure that suits our needs. This involved aligning our fields with common attributes across platforms, such as "property name," "check-in date," and "guest contact information," while also creating custom fields specific to our particular services, like "cleaning schedule" of each property, which customers could check and personalize during their stay.

The database that was initially created consists of:

- **Listings Table:** Stores property details, including descriptions, amenities, pricing, and custom fields for additional features unique to our operations.
- **Reservations Table:** Tracks booking details, guest information, check-in and check-out dates, payment status, and any special requests or notes, allowing flexibility for additional information.
- **Calendar Table:** Manages availability and scheduling, ensuring accurate booking information across all platforms.

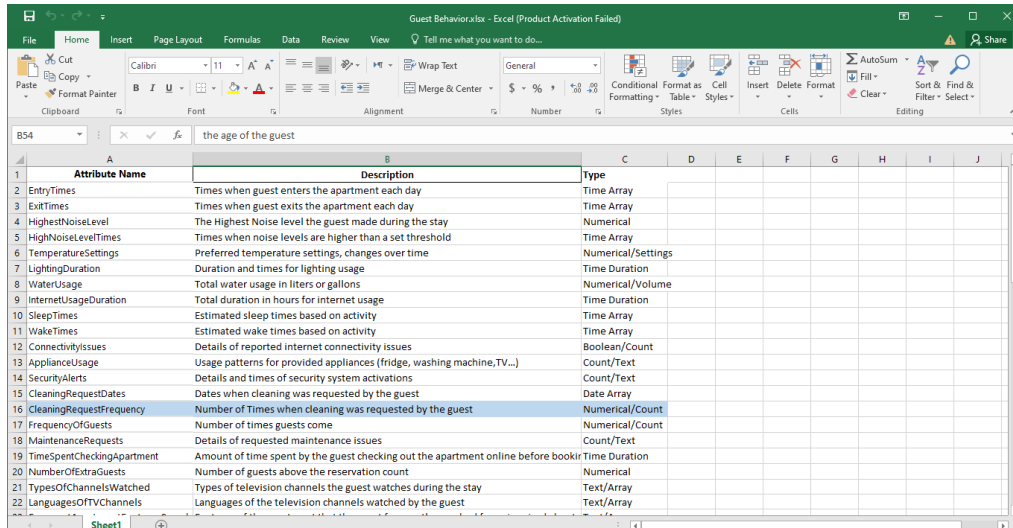


Property	Required	Type	Description
id	yes	int	The unique identifier for the reservation within the Hostaway system.
listingMapid	yes	int	The identifier associated with the specific listing object within Hostaway.
channelId	yes	int	Numerical value indicating the reservation source: 2000 indicates a direct reservation, 2020 indicates a reservation made through a partner channel.
source	no	string	The source from which the reservation was made, if applicable.
channelName	yes	string	The name of the channel through which the reservation was made, such as Airbnb, Booking.com, Expedia, Vrbo, etc.
hostawayReservationId	yes	int	The unique identifier for the reservation within the Hostaway platform, which is the same as the 'id'.
channelReservationId	yes	string	The reservation ID as recognized by the external channel, identical to 'reservationId'.
externalPropertyId	no	string	The external identifier for the property, if applicable.
externalRatePlanId	no	string	Identifier for a property's pricing plan in an external booking system.
externalUnitId	no	string	Identifier for the unit in the external system, if applicable.
assigneeUserId	no	int	The user ID of the person assigned to this reservation, if applicable.
manualId	no	string	Identifier for a manually integrated calendar for tracking property availability.
manualName	no	string	Name associated with the manually managed iCal, if used.
isProcessed	no	bool	Boolean value indicating whether the reservation has been processed.
isManuallyChecked	no	bool	Boolean indicating whether the reservation has been manually checked.
isInstantBooked	no	bool	Indicates whether the reservation was booked instantly without needing approval.
hasPullError	no	bool	Indicates whether there was an error in pulling the reservation from the channel.
reservationDate	no	datetime	The date and time when the reservation was made.
pendingExpireDate	no	datetime	The date and time when a pending reservation will expire if not confirmed.
isDatesUnspecified	no	int	Set to 1 in case a channel doesn't provide reservation dates. If it is 1, arrivalDate and departureDate are set to yesterday.
totalPrice	no	float	Total price of reservation
taxAmount	no	float	Tax amount of reservation
channelCommissionAmount	no	float	Channel commission amount

Figure 32 Preparation of Reservation Fields for Database Integration

In addition, we developed other tables to support our management needs and future use cases, such as a Guest Behavior Table for a potential recommendation system, a Tasks Table for managing operational tasks like cleaning requests, and a Chatbot History Table for tracking interactions and improving.

By designing the database, we ensured seamless integration of data from both HostAway and our own system, providing a unified source of information for the chatbot to access.



Attribute Name	Description	Type
EntryTimes	Times when guest enters the apartment each day	Time Array
ExitTimes	Times when guest exits the apartment each day	Time Array
HighestNoiseLevel	The Highest Noise level the guest made during the stay	Numerical
HighNoiseLevelTimes	Times when noise levels are higher than a set threshold	Time Array
TemperatureSettings	Preferred temperature settings, changes over time	Numerical/Settings
LightingDuration	Duration and times for lighting usage	Time Duration
WaterUsage	Total water usage in liters or gallons	Numerical/Volume
InternetUsageDuration	Total duration in hours for internet usage	Time Duration
SleepTimes	Estimated sleep times based on activity	Time Array
WakeTimes	Estimated wake times based on activity	Time Array
ConnectivityIssues	Details of reported internet connectivity issues	Boolean/Count
ApplianceUsage	Usage patterns for provided appliances (fridge, washing machine,TV...)	Count/Text
SecurityAlerts	Details and times of security system activations	Count/Text
CleaningRequestDates	Dates when cleaning was requested by the guest	Date Array
CleaningRequestFrequency	Number of Times when cleaning was requested by the guest	Numerical/Count
FrequencyOfGuests	Number of times guests come	Numerical/Count
MaintenanceRequests	Details of requested maintenance issues	Count/Text
TimeSpentCheckingApartment	Amount of time spent by the guest checking out the apartment online before booking	Time Duration
NumberOfExtraGuests	Number of guests above the reservation count	Numerical
TypesOfChannelsWatched	Types of television channels the guest watches during the stay	Text/Array
LanguagesOfTVChannels	Languages of the television channels watched by the guest	Text/Array

Figure 33 Custom Guest Behavior Fields for Enhanced Data Analysis

IV.2 Exploring existing solutions

In developing an AI-powered customer service chatbot, our goal was to provide a seamless, multi-platform communication experience for users. The process involved evaluating existing solutions and eventually creating a tailored chatbot that could meet our specific needs.

IV.2.1 Evaluating Chatbot Building Platforms

Our initial strategy involved investigating chatbot development frameworks that could seamlessly integrate with our existing systems, aiming to avoid building a solution entirely from scratch. However, most of the frameworks we evaluated were not able to manage the specific workflows or provide the detailed data integration required for our property management operations.

Chatbot Frameworks Considered:




Framework	Key Features	Limitations
 Dialogflow	Known for Google Cloud integration and natural language understanding.	Lacked the property management features we required.
 Microsoft Bot Framework	Offers robust chatbot-building tools.	Did not provide seamless multi-platform support or handle real-time data.
 ManyChat	Useful for basic automation and marketing.	Unsuitable for complex property management with dynamic data.

Table 4 Existing Chatbot Building Platforms

IV.2.2 Evaluating Pre-Built Chatbots: HostAI

During our research, we explored several pre-built chatbot solutions, including HostAI. Designed specifically for property management, HostAI integrates advanced AI with property-specific information to deliver automated, context-aware responses to guest inquiries.

Features and Capabilities of HostAI:

- Its 24/7 availability and automated responses allow it to efficiently handle common inquiries about Wi-Fi, check-ins, and dining options, reducing the need for human intervention.
- HostAI's seamless integration with HostAway allows it to pull real-time data on property listings, reservations, and guest communications to provide guests with accurate information.
- The chatbot interface allows for the management of booking inquiries, modifications, and cancellations, while also providing real-time updates.
- HostAI boosts revenue and guest satisfaction by personalizing responses and suggesting additional services based on guest preferences and past interactions, such as local tours or late checkouts.

Despite these features, HostAI did not fully meet our requirements due to its limited integration capabilities. While it integrates seamlessly with HostAway messaging, it is heavily dependent on this platform and lacks the flexibility to operate across other channels. Our business model requires communication across multiple platforms, such as our website, social media, and messaging apps. Therefore, it was unsuitable for our communication strategy.

IV.2.3 Implementation of OpenAI Assistants API

After evaluating pre-built solutions, we implemented OpenAI's Assistants API, configuring it with specific instructions and data files to handle a wide range of inquiries.

However, while this setup worked well for standard queries, it had notable limitations that made it unsuitable for our large-scale system. The primary issue was the lack of real-time data handling. The OpenAI Assistant developed relied on static data files, necessitating manual updates to property information, bookings, or guest inquiries, a process that was both time-consuming and prone to errors. This also limited the assistant's context awareness, reducing its ability to handle dynamic queries effectively.

Recognizing these challenges and the need for real-time data integration led us to develop a more flexible system using Retrieval-Augmented Generation (RAG) to dynamically retrieve information from our MongoDB database.

IV.3 Building the Initial RAG System

To address the limitations of static data in pre-built chatbot solutions, we adopted Retrieval-Augmented Generation (RAG). This approach enhanced our AI-powered chatbot by combining retrieval-based techniques with generative models, enabling it to access real-time information from our database. As a result, the chatbot can provide contextually relevant, accurate, and personalized responses based on the most current data.

IV.3.1 Database Preparation for RAG

To build an effective retrieval-augmented generation (RAG) system, the first step was to set up a robust vector database to store and manage embedded data. Since our data was already housed in MongoDB, we utilized MongoDB Atlas Vector Search for its seamless integration with our existing infrastructure and its efficient handling of high-dimensional vector data. This choice ensured a unified system that enabled fast, efficient similarity searches to retrieve contextually relevant records in response to user queries.

With the vector database in place, the next step was data embedding. Data embedding involves converting text data from our various tables, such as Listings, Reservations, and Calendar, into vector representations.

IV.3.1.1 Data Consolidation:

To prepare the data for embedding, we focused on key fields from each table that were most relevant to user queries. This consolidation into "summary" fields ensured that all pertinent details were included in the embeddings, making them more effective for the RAG system.

For the Listings, we combined essential fields such as property type, name, location, amenities, pricing, etc., into a summary, to create a comprehensive overview of each listing.

```

f"{property_type} named '{name}' located at {address}, {city}, {country}. "
f"\nDescription: {description}.\n"
f"This {property_type.lower()} can accommodate up to {capacity} guests"
f"with {bedrooms} bedrooms, {beds} beds, and {bathrooms} bathrooms.\n"
f"Amenities include: {amenities}. The nightly rate is ${price}"
f"with a minimum stay of {min_nights} nights and a maximum of {max_nights} nights. "
f"Weekly discount: {weekly_discount}, Monthly discount: {monthly_discount}.\n"
f"Check-in starts at {check_in_start} and check-out is by {check_out_time}.\n"
f"Door security code: {door_security_code}. Cancellation policy: {cancellation_policy}.\n"
f"Star rating: {star_rating}. WiFi username: '{wifi_username}', password: '{wifi_password}'.\n"
f"Special instructions: {special_instruction}. House rules: {house_rules}. House manual: {house_manual}.\n"
f"Contact: {contact_name}, Phone: {contact_phone}, Email: {contact_email}.\n"
f"Average review rating: {average_review_rating} from {total_reviews} reviews.\n"
f"Access details: Parking instructions: {parking_instructions}"
f"Building instructions: {building_instructions}, Apartment instructions: {apartment_instructions}.\n"
f"Enjoy your stay in our comfortable and well-equipped accommodation.\n"
    
```

Figure 34 Summary Structure for Listings

Similarly, in the Reservations table, we summarized fields like guest details, booking dates, payment status, etc., to provide a concise yet informative snapshot of each reservation.

```
f"Reservation Details: {reservation_details}, {channel_details}",
f"Guest Info: {guest_name}, {contact_info}, City: {format_value('guestCity')}",
f"Booking Details: {booking_dates}, Nights: {format_value('nights')}, Guests: {format_value('numberOfGuests')}",
f"Payment Info: {payment_info}, {security_deposit}",
f"Verification & Security: {verification_info}, {additional_security}",
f"Notes: {format_value('guestNote')}",
f"Agreement Status: {format_value('reservationAgreement')}",
f"Updated On: {format_value('updatedAt')}
```

Figure 35 Summary Structure for Reservations

For the Calendar entries, we focused on summarizing availability, pricing, stay requirements, etc., offering a clear view of booking possibilities.

```
f"Listing ID {listing_id} on {date} is {available_status}. "
f"The price per unit is ${price}. "
f"Minimum stay required is {minimum_stay} night(s), and the maximum stay allowed is {maximum_stay} night(s)."
```

Figure 36 Summary Structure for Calendar Entries

IV.3.1.2 Data Embedding and Indexing

After preparing the relevant text data, we proceeded with the embedding and indexing process to prepare the vector database for efficient retrieval.

- **Data Preprocessing:** We cleaned and normalized the text to remove any inconsistencies, ensuring uniformity across the data and preventing irregularities from interfering with the embedding process.
- **Splitting and Embedding:** To handle larger text blocks more efficiently, we split the text into smaller chunks. We then passed these chunks through the embedding model, transforming them into vector representations that capture the semantic meaning of the content.
- **Indexing Configuration:** To ensure efficient retrieval, we set up MongoDB Atlas Vector Search to index the embedded vector. Indexing organizes the vector data by creating indexes, allowing the system to quickly locate and retrieve relevant embeddings in response to user queries.

IV.3.2 Query Processing and Workflow

As for processing the user's question, it involves several steps to fit seamlessly into our system. The following diagram illustrates the key stages:

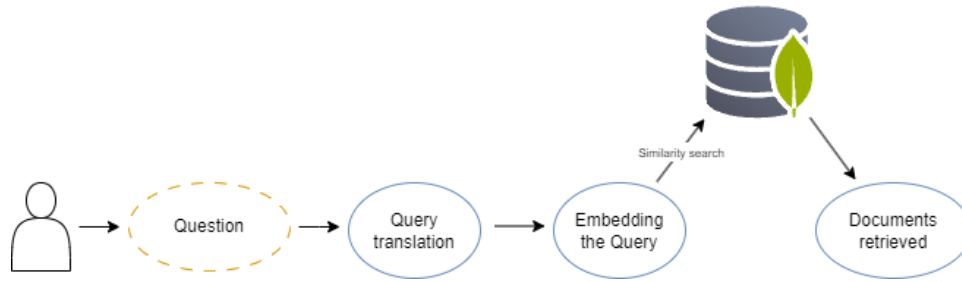


Figure 37 Query Processing Workflow in RAG System

First, the query is translated into English to align with the language used in our database, ensuring consistency in embedding and search operations. The translated query is then embedded into a vector format, enabling a similarity search. Finally, the MongoDB Atlas Vector Search database retrieves the top 10 to 15 most similar documents, balancing the need for sufficient context while avoiding an overload of information.

IV.3.3 Response Generation with LLM

The final step in the RAG workflow involves generating responses using a Large Language Model (LLM). This step is critical because it converts the retrieved data into well-structured and meaningful responses for the users. The effectiveness of this stage largely depends on prompt engineering.

IV.3.3.1 Prompt Engineering

The prompt serves as a set of instructions that dictate how the LLM should interpret the input data and generate output. In our RAG system, the prompt structure is carefully designed to include identity of the assistant, history, user’s question and the retrieved information. The prompt consists of several components that collectively guide the LLM:

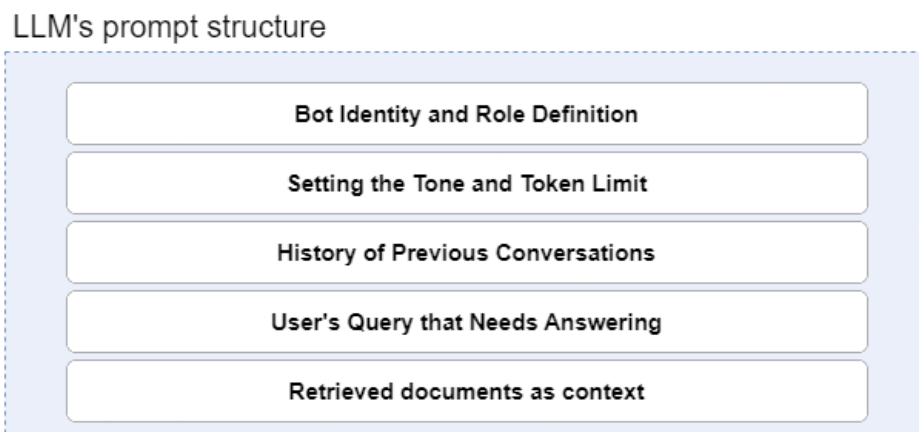


Figure 38 Prompt structure for llm in the initial rag system

IV.3.3.2 Model choosing and Evaluation

In the LLM phase of our RAG system, we evaluated different AI models to determine the most suitable for our needs. We tested both LLaMA and OpenAI models to assess their performance in generating high-quality responses for our chatbot.

LLaMA, an open-source model, offers a good balance of speed and cost-effectiveness as it is free to use. However, In our initial tests, without the ability to fine-tune due to limited data, LLaMA did not perform well with complex queries and lacked the depth in understanding and response accuracy needed for our use case. This limitation highlighted that, while LLaMA is fast and efficient for rapid interaction scenarios, it falls short in delivering high-quality, nuanced responses without further customization.

On the other hand, **OpenAI models** performed better in understanding complex user inputs and generating context-aware, coherent responses. OpenAI's advanced natural language processing capabilities allowed us to implement effective prompt engineering, which compensated for the lack of fine-tuning. This flexibility enabled us to tailor the chatbot's behavior according to our specific needs without extensive retraining. While OpenAI models are slightly slower than LLaMA due to the depth of processing, their ability to handle sophisticated conversations with high accuracy.

Model	Speed	Quality	Open Source
LLaMA	Fast	Moderate	Yes
OpenAI API	Moderate	High	No

Table 5 Comparison of AI Models

This table summarizes the differences between the two models, highlighting OpenAI's superior quality in response generation. Despite not being free, OpenAI aligned with our main focus on delivering high-quality responses. We prioritized precise and high-quality outputs, making OpenAI the best choice for our needs.

IV.3.3.3 Embedding Model

For generating vector embeddings, we chose OpenAI's Ada model. This decision was driven by a desire to maintain consistency throughout our system, as we were already using OpenAI's models for text generation. The Ada model excels in capturing semantic meaning, ensuring compatibility with our overall architecture.

The Ada model was used not only to embed the data into our vector database but also to embed user queries when they arrived, ensuring efficient and accurate similarity searches.

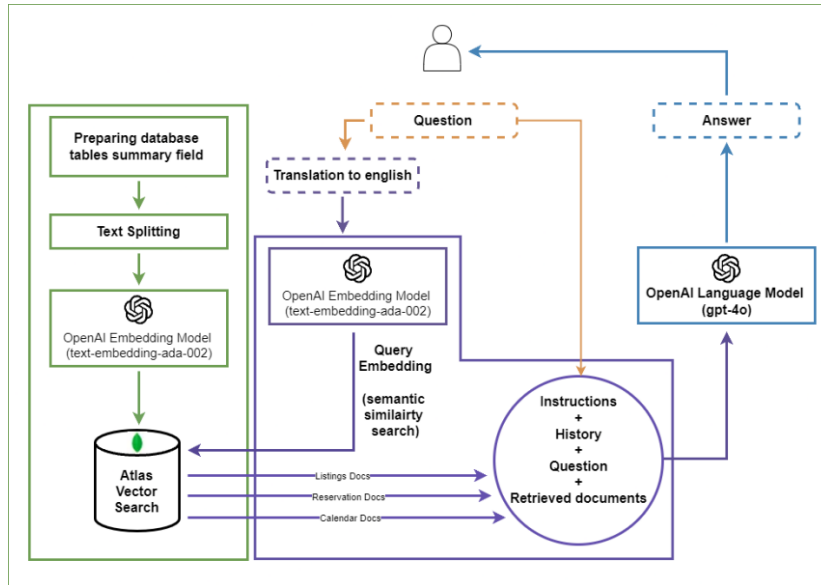


Figure 39 Workflow of the initial Retrieval-Augmented Generation (RAG) System

IV.3.4 Role of LangChain in RAG:

To manage the complex interactions within our RAG system, LangChain's **ConversationalRetrievalChain** and **ConversationBufferMemory** played critical roles in handling multi-turn conversations. The **ConversationalRetrievalChain** creates a seamless flow from the user's query to the final response generated by the LLM, rephrasing questions based on previous context and supplying the model with retrieved documents. Meanwhile, **ConversationBufferMemory** preserves the conversation history across multiple interactions, enabling the system to reference past exchanges during prompt engineering and maintain dialogue continuity.

This approach ensures that the system remains responsive not only to the current query but also aware of the ongoing conversation, resulting in more personalized and accurate responses.

IV.3.5 Challenges and Limitations

The primary challenge with the initial RAG implementation was a lack of precision and relevance in the retrieved information. Since the system pulled data from multiple tables, it often surfaced details that weren't contextually relevant or personalized to the user's query, resulting in overly long contexts that weren't necessary for generating accurate responses. For example, a query about a specific reservation might also retrieve general information about listings or unrelated guest details, overwhelming the language model and leading to responses with additional information that the user didn't request.

In summary, these challenges underscored the limitations of a one-size-fits-all retrieval strategy in a complex domain like property management. They highlighted the need for a more refined and specialized approach that retrieves only essential information based on the nature of the query while managing context more efficiently.

IV.4 Transition to Advanced RAG System

To address these limitations, we moved to a more refined approach with query classification and routing. By classifying queries based on their intent, the system could retrieve only the necessary information, narrowing down the data to what's most relevant.

IV.4.1 Development of Graph-Based RAG System

This system was achieved using a graph-based routing mechanism with LangGraph, which breaks down the query processing workflow into distinct, interconnected tasks. Each task, represented by a node, performs a specific function within the system, while the edges define the flow of information between these tasks.

LangGraph extends the capabilities of LangChain by enabling the creation and management of complex, stateful, multi-agent workflows in AI applications that utilize LLMs effectively. In this setup, each node represents an LLM agent, and the edges represent communication channels between the agents. This graph structure ensures clear and manageable workflows, where each agent performs specific tasks and passes information to other agents as needed, allowing the chatbot to handle multiple interactions simultaneously, efficiently routing queries based on their classification.

IV.4.1.1 System Architecture Overview

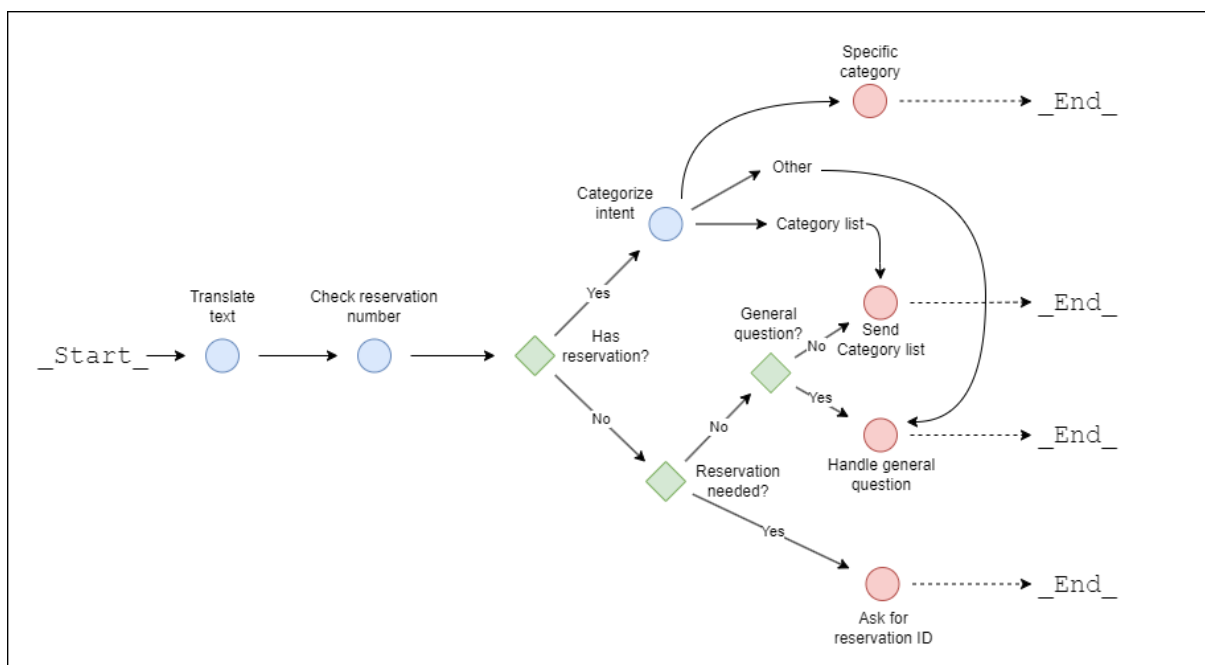


Figure 40 Langgraph system design

The system uses OpenAI's GPT-4 model as the core decision-maker, handling translation, query categorization, and response generation to ensure accurate and context-aware outputs.

The workflow of our graph-based system follows these steps:

Initial Query Handling:

- **Translate Text Node:** The first step involves translating the incoming query into English using the GPT-4 model. This ensures consistency in language, aligning the query with our database schema for accurate processing.
- **Check Reservation Number Node:** After translation, the system checks if the user has an associated reservation number. If a reservation is found, the query is processed as a "query with reservation." If not, it is treated as a "query without reservation."

Handling Queries Without Reservation:

- **General Question Check:** For users without a reservation, the system first determines if the query is a general question. If it is, the chatbot provides a general response using OpenAI's model. If the query is more specific, the system may ask for a reservation ID to provide a more tailored response.
- **Category Classification:** If no reservation ID is needed, the system categorizes the query based on its content. This classification helps in directing the query to the appropriate node, ensuring a relevant and accurate response.

Handling Queries With Reservation:

- **Categorization and Routing:** For queries associated with a reservation, the system categorizes the query into predefined categories such as booking details, check-in instructions, or property information. This step ensures the query is routed to the correct data nodes for accurate information retrieval.

Fallback and Miscellaneous Nodes:

- **Send Categories Node:** If the system cannot directly categorize a query, it sends a list of available categories to the user, prompting them to specify their request further.
- **Default Node:** For uncategorized queries, the system generates a default response using the LLM, providing a generic yet informative answer.

This system efficiently manages various types of queries based on their classification, ensuring personalized responses for both general inquiries and reservation-specific questions.

Below is an example of how the system handles a general query without a reservation:

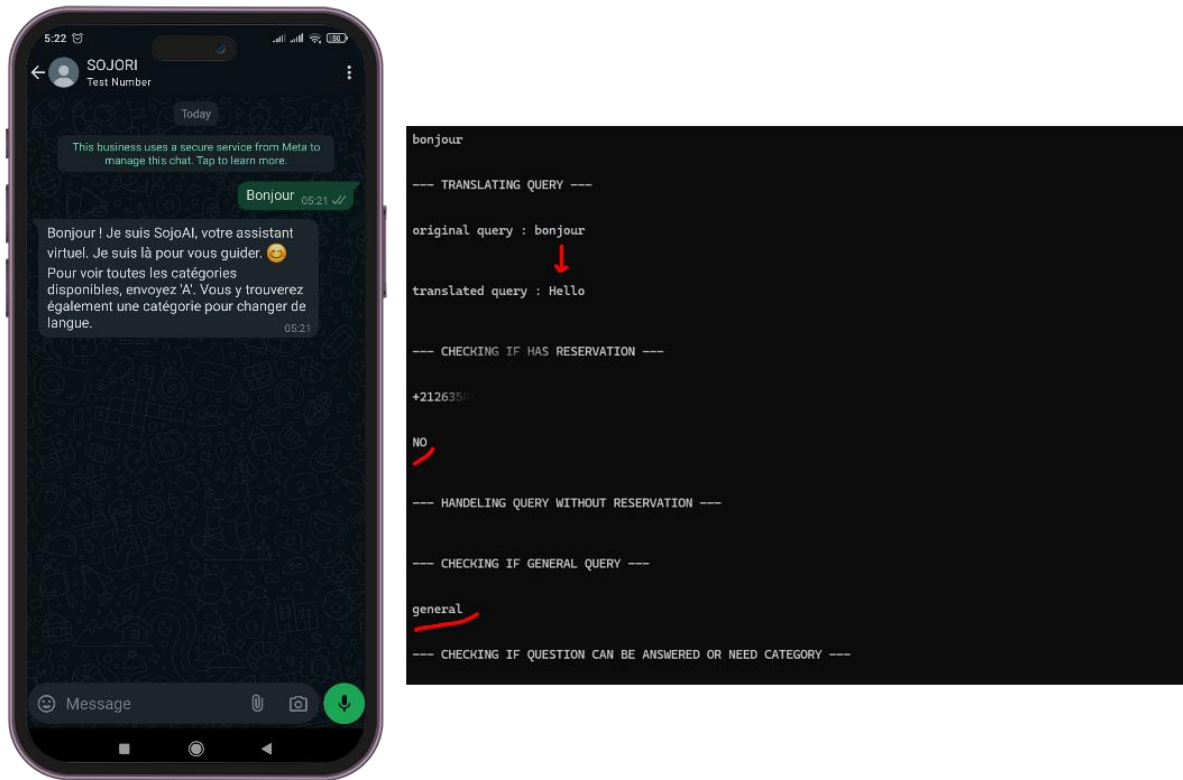


Figure 41 Example of Handling a General Query Without Reservation

As shown in this figure, the system first translates the user's query to English, then checks if the user has an associated reservation. Since no reservation is found in this case, the query is processed as a general question.

IV.4.2 Multi-Channel integration

IV.4.2.1 Multi-Channel Integration and Deployment

To meet diverse user needs and enhance customer service, our chatbot system aim to be integrated across multiple platforms, including the website, WhatsApp, and in home voice activated device. This multi-channel deployment ensures that users receive consistent support and information, regardless of their preferred communication channel, thereby broadening our reach and improving overall user satisfaction.

To achieve seamless multi-channel integration, we developed an API using FastAPI. This API is designed to efficiently receive and process user queries from various communication platforms, such as the website, WhatsApp. It serves as a centralized hub, routing user queries through our graph-based system for processing and response generation. This approach ensures that each interaction is managed consistently and accurately, regardless of the channel through which it originates. By leveraging this API, we ensure that the chatbot adapts to the diverse communication preferences of our users.

IV.4.2.2 Unified Workflows for All Channels

IV.4.2.2.1 LLM with Data Retrieval Workflows

For certain categories of customer inquiries, we employ a Retrieval-Augmented Generation (RAG) system to provide precise and relevant responses. This approach combines retrieving information from our database and generating tailored responses using OpenAI's api model. The categories that utilize this RAG system include:

- **Reservation Inquiries:** For questions related to a customer's reservation, we retrieve specific booking details such as reservation status, arrival and departure dates, and other relevant information.
- **Company Information:** For queries about Sojori, we retrieve a detailed company description from the database to provide accurate information.
- **Property Information:** For inquiries concerning property details, we fetch comprehensive information about the property of interest, including amenities, policies, and other relevant details.
- **Access Codes:** For requests regarding access codes to properties, we retrieve the necessary door codes for entry.
- **Access Information:** For questions about how to access the property, we provide instructions for parking, building entry, and apartment access.

After retrieving the appropriate data, we use prompt engineering to define a prompt template for OpenAI's language model. This template outlines the task and role, presents the user's question, includes the retrieved information, and provides precise instructions to ensure the response is direct and tailored to the user's needs.

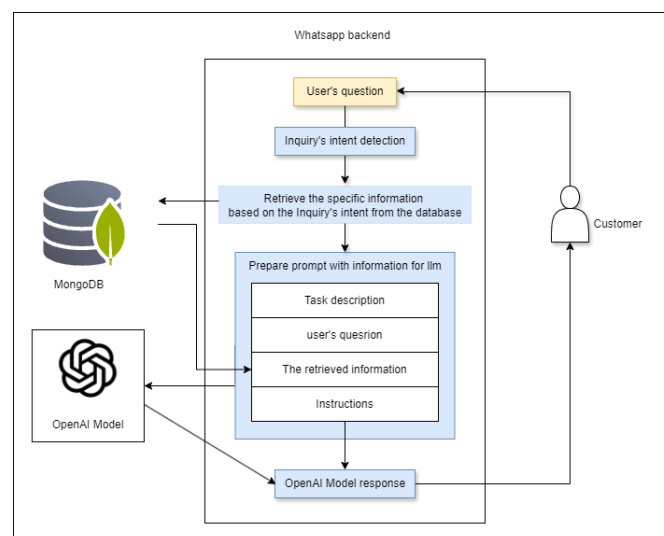
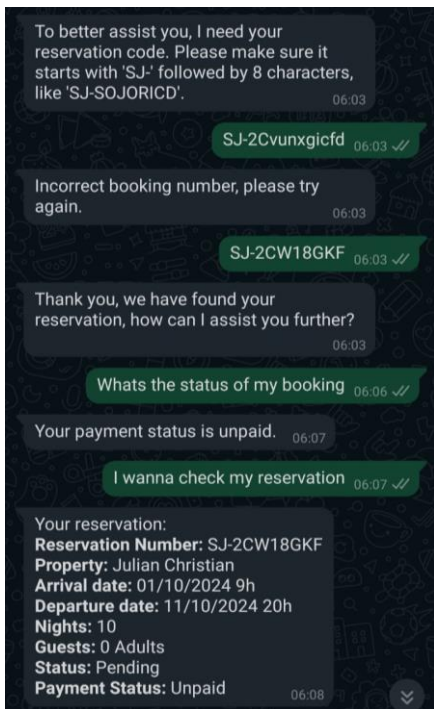


Figure 42 Workflow for RAG-Based Inquiry Handling

Example of Booking Inquiry Handling Using LLM with Data Retrieval:



As shown in the figure, when a customer's query requires a reservation, and the system does not already have the information, the chatbot first prompts the user to provide a valid reservation ID. If not, the system prompts for the correct one. Once confirmed, the chatbot retrieves relevant reservation details from the database, and generates a personalized response using an LLM with data retrieval, as demonstrated in the example.

Figure 43 Example of the chatbot system handling question requires reservation

IV.4.2.2.2 LLM Without Data Retrieval Workflows

In some categories of customer inquiries, there is no need to retrieve real-time data since we rely on predefined, static information. For these types of queries, we use an automated response system supported by OpenAI's language model (LLM). Once we classify the query's intent into one of these specific categories, we prepare a prompt with clear category-specific guidelines tailored for the LLM to generate an appropriate response.

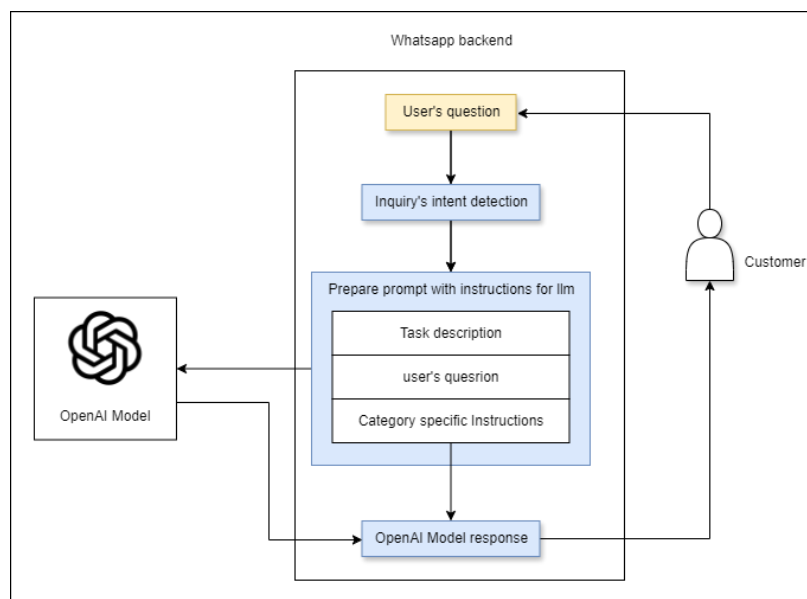


Figure 44 Response Workflow for Static Information Categories

The system follows these steps for each specific category:

Useful Numbers:

If the customer asks for useful numbers, the system provides instructions to the LLM containing static information about essential contact numbers, police, ambulance, tourist police, Firefighters, then the OpenAI model generates a response with a list of the useful numbers requested or all of them.

Room Service:

For room service inquiries, once the intent is detected, the system instructs the LLM to analyze the user's request and generate a structured response.

Response Example:

"Room Service:

You have requested the following items/services:

- Towels: 2
- Soap.

Please confirm if the list is correct or cancel your room service request. If there are any changes or additional requests, please specify them. Thank you!"

After the user confirms the request, a task is created in the database with details about the room service request, marked with urgency as 'Urgent.' to notify the staff.

Technical Issues:

For technical issues, the system may involve two separate processes, depending on the user's input.

- If an image is sent, one OpenAI agent analyzes the image to determine the issue and asks for additional details if needed.
- If a text description is provided, another agent processes the description and acknowledges receipt, indicating the issue will be addressed.

A task object for the technical issue is then created in the database, marked as 'Urgent,' and includes details from the user's input and any analyzed image data.

Example of Room Service Inquiry Handling:

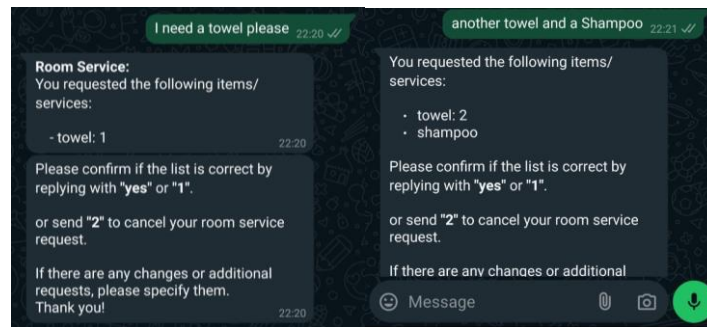


Figure 45 Example of Room Service Inquiry Handling

IV.4.2.3 WhatsApp-Specific Workflows:

For certain categories, we have developed workflows specifically tailored to WhatsApp, leveraging its automation tools and user-friendly interface to create a more intuitive and engaging experience. In contrast, the workflows for these same categories on the website follow the unified approach, either using the LLM with RAG for complex queries or providing static instructions for simpler tasks.

To implement WhatsApp-specific workflows, we utilized WhatsApp Manager to create flows that guide user interactions, simplifying processes and enhancing user experience. One such example is the flow built for choosing a language, implemented using the Flow JSON structure:

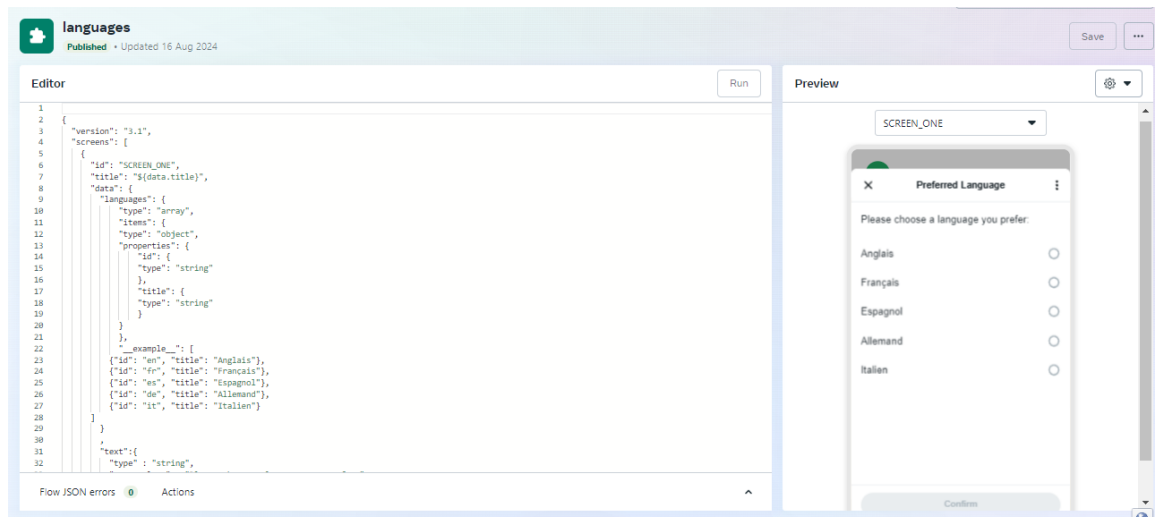


Figure 46 Example of a WhatsApp Flow JSON for Language Selection

This led us to develop specialized workflows, with a seamless and efficient user experience on WhatsApp for the following categories:

- **Language Choosing Node workflow:**

In our WhatsApp communication, we prioritize engaging with users in their preferred language. During the initial interaction, we utilize an OpenAI agent to determine the most appropriate language based on the user's phone number and the language of their first message. This detected language is set as the default for all subsequent interactions. However, if we notice that the user frequently sends messages in a different language, we dynamically adjust the language settings to better align with their preferences.

<p>United States Phone Number</p> <pre>print(language_detection_agent("+12025550143"))</pre> <p>en</p>	<p>German Phone Number</p> <pre>language_detection_agent("+4915123456789")</pre> <p>'de'</p>
<p>French Phone Number</p> <pre>language_detection_agent("+33123456789")</pre> <p>'fr'</p>	<p>Italian Phone Number</p> <pre>language_detection_agent("+393471234567")</pre> <p>'it'</p>
<p>Spanish Phone Number</p> <pre>language_detection_agent("+34612345678")</pre> <p>'es'</p>	<p>Moroccan Phone Number</p> <pre>language_detection_agent("+212354698712")</pre> <p>'fr'</p>

Figure 47 Example of Language determination Results Based on User's Phone Number

After the initial language is set, users have the option to change it by sending a message requesting a language change. When the system detects that, the user's intent is related to selecting or changing the language, it follows the workflow outlined in the diagram below.

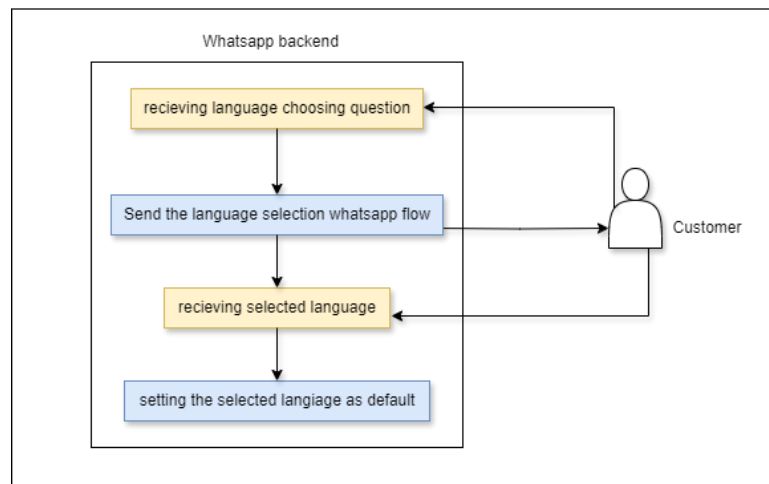


Figure 48 Language Change Workflow in WhatsApp Backend

This workflow includes receiving the language change request, sending the appropriate WhatsApp flow for language selection. As shown in the example, the user chooses 'English,' the language preference is updated, and subsequent communications continue in English.

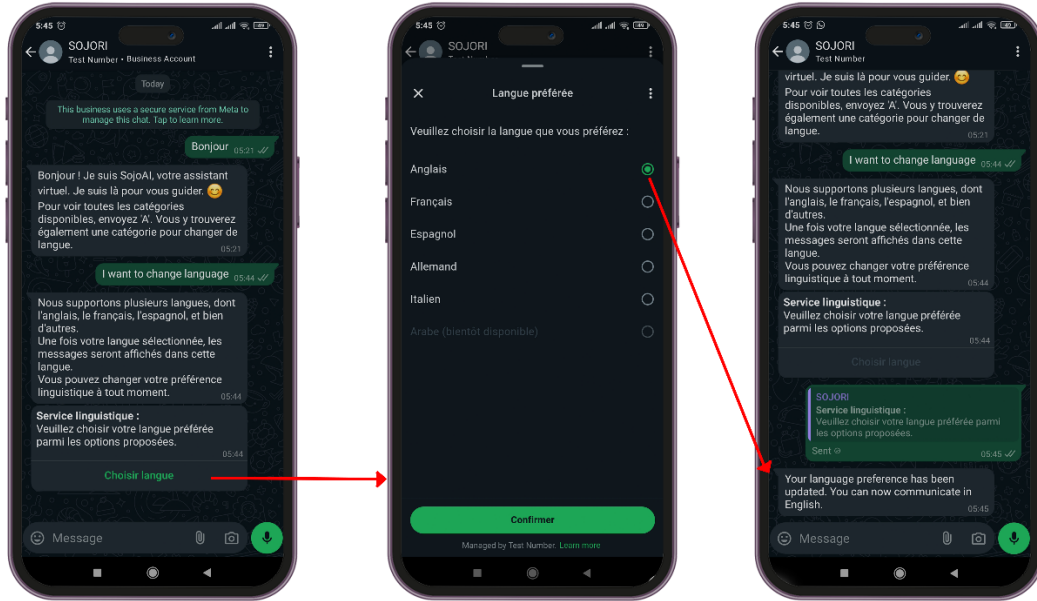


Figure 49 Example of Language Change Workflow in WhatsApp Chatbot Interface

- **Check-in, checkout and house cleaning:**

For categories such as arrival, departure, and House Cleaning, if the user's question intent is classified under any of these categories, the WhatsApp backend automatically triggers the corresponding WhatsApp flow based on the identified category. This flow sends a tailored message to the user on WhatsApp, asking for the necessary details or preferences.

Once the user's response is received through the WhatsApp flow, the system updates the MongoDB database with the appropriate task type. This could include scheduling a specific arrival time, requesting a late departure, setting an early arrival, or choosing a preferred cleaning time. This ensures that the user's preferences are accurately recorded and processed.

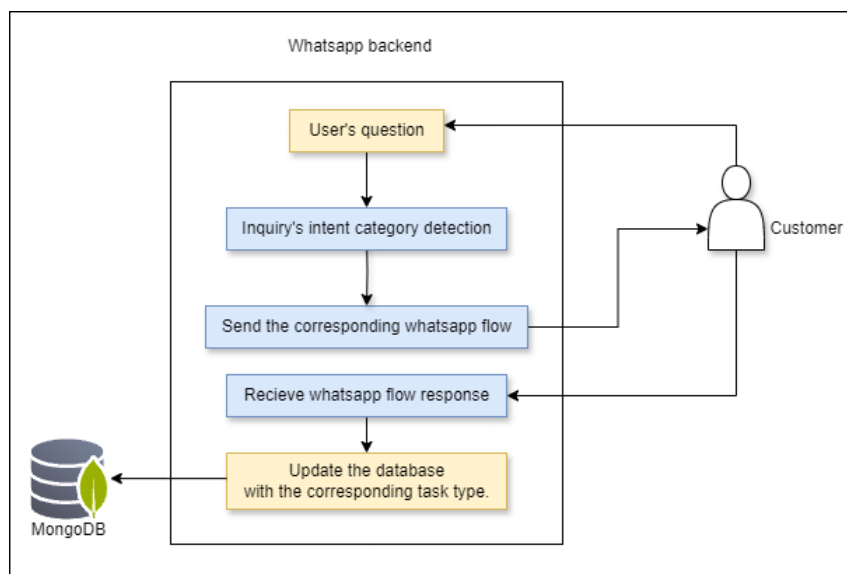


Figure 50 Workflow for Handling Requests via WhatsApp Flows

Below is an example of the WhatsApp flows and the interfaces as they appear to users.

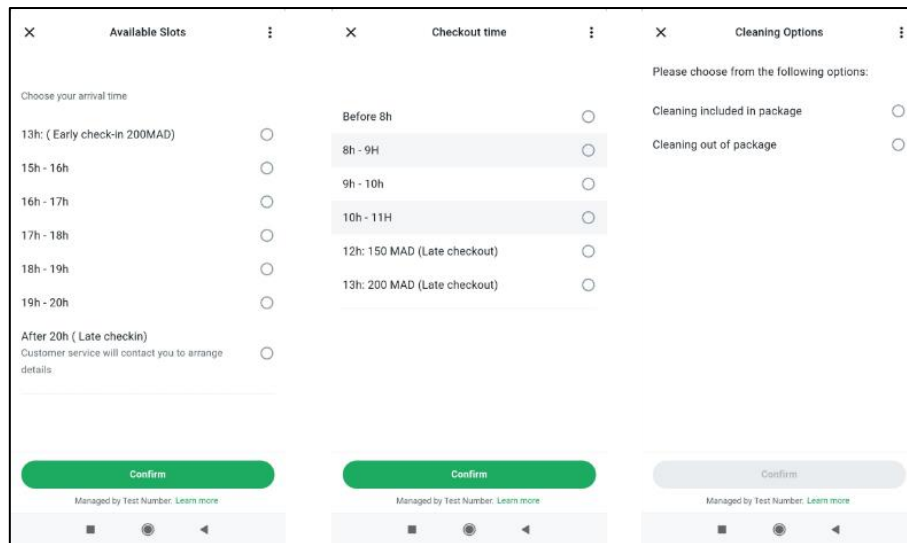


Figure 51 WhatsApp Interface for Selecting Check-in, Checkout, and Cleaning Options

- **Online check-in process:**

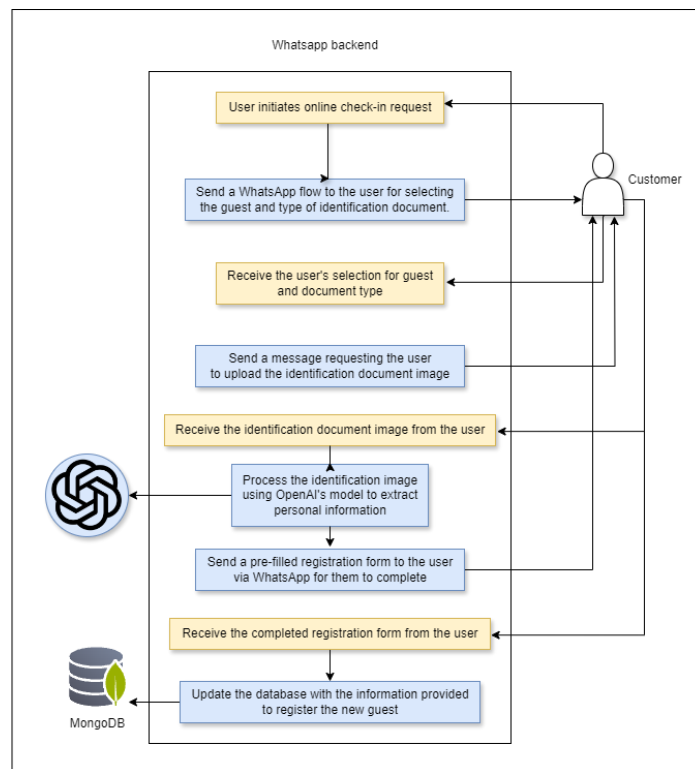


Figure 52 Workflow for Online Check-in Process Using WhatsApp Flow and OpenAI

For a more complex process like online check-in or guest registration, the workflow begins when a customer requests to initiate the online registration process. We start by sending a WhatsApp flow interface that displays the user's booking details, such as dates, the number of guests, and the verification

status of registered guests. This interface also allows the user to select which guest they want to register and specify the type of identification document they will provide.

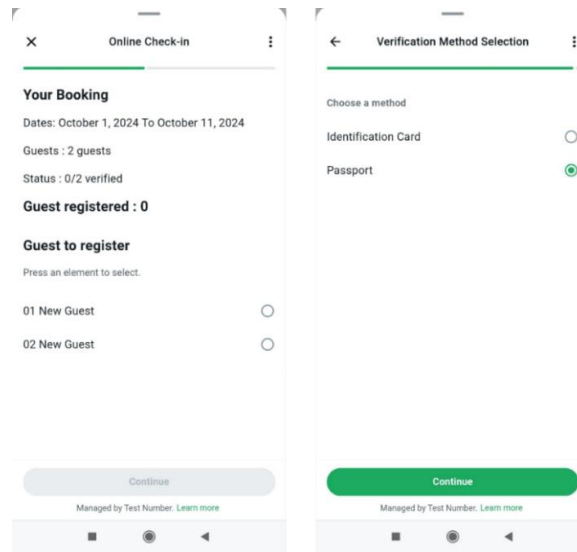


Figure 53 Guest and registration document selection flow

Once the user has selected the guest and document type, we send a message instructing them to upload an image of the identification document. Upon receiving the image, we pass it through an OpenAI language generation model, where we use prompt engineering to ensure the model returns the extracted personal information in a specific JSON format. This JSON format is designed to be compatible with the WhatsApp flow system, allowing us to pre-fill the registration form. The prefilled form is then sent back to the customer via WhatsApp, where they can review, correct, and finalize the details.

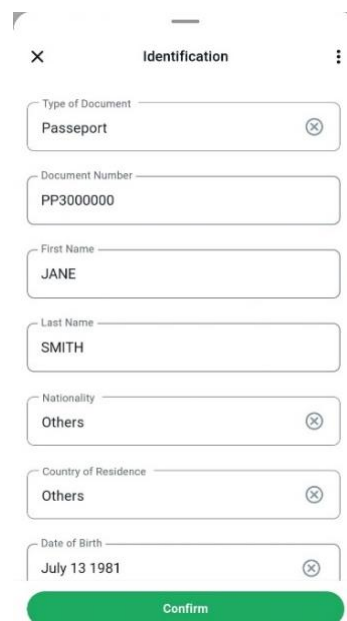


Figure 54 Example of online check-in form

After the customer completes and submits the registration form, we update our database with the newly registered guest's information, ensuring all details related to their reservation are accurately recorded. This automated process streamlines the online check-in, making it efficient and user-friendly.

IV.5 Dashboard Chat and Session handling

In the implementation of our chatbot system, handling user states and managing chat sessions are critical for delivering a smooth and efficient user experience.

IV.5.1 State Handling

The state refers to what the chatbot expects next from the user, such as waiting for an image, a reservation ID, or other specific inputs depending on the ongoing conversation. To manage these states effectively, we use Redis, a high-performance in-memory database. Redis allows us to store and quickly access the state of each user session due to its speed and efficiency in handling data in memory.

Using Redis for State Management:

- **User State Tracking:** Each user interaction is assigned a unique identifier. Redis stores the current state of each user's interaction, allowing the chatbot to know what the next expected action is. For example, if the chatbot is waiting for an image, this state is stored in Redis, and the system will know to expect an image next rather than a text response.
- **Session Continuity:** Redis enables quick read and write operations, which ensures that user sessions are managed in real-time. If a user disconnects and reconnects, Redis helps maintain session continuity by recalling the last known state.

IV.5.2 Storing chat History

Apart from managing states, storing chat history is crucial for reviewing interactions and improving service. We use Redis to temporarily hold a portion of the chat history for quick access and efficient response generation. Redis allows us to retrieve relevant context swiftly before providing an answer. After each interaction, both the user's question and the chatbot's response are stored directly in our primary database. This approach ensures that all interactions are documented for future analysis and visualization while maintaining fast response times during active sessions.

```

{
  "_id": "66d19b4cc9ea9f07999e5947", # Unique identifier for the user
  "mobile_number": "+212649457373", # User's mobile number
  "openai_language": "fr", # Proposed Language by OpenAI based on initial messages
  "confirm_language": "", # Confirmed Language by the user, if available
  "user_id": "", # Unique user ID
  "OpenAI": True, # Status of OpenAI interaction (True/False)
  "username": "Abdelhamid", # User's name
  "last_date_sent": "2024-08-30 16:53:58", # Timestamp of the Last message sent
  "last_date_received": "2024-08-30 16:53:56", # Timestamp of the Last message received
  "history": [{
    "start_date": "2024-08-30 11:13:31", # Start date of the interaction
    "end_date": "2024-08-30 16:53:58", # End date of the interaction
    "topics_discussed": {
      "topics": [], # List of topics discussed
      "summary": "" # Summary of the conversation
    }
  }],
  "reservations": [{
    "reservation_id": ObjectId('66cf50ebc39428002e528272')
    "listing_id": ObjectId('666c55f78ed9ac002e5cc33e')
    "status": "Pending"
    "check_in_date": "2024-09-10T00:00:00.000+00:00"
    "check_out_date": "2024-09-12T00:00:00.000+00:00"
  }], # List of user's reservations
  "Tasks": [], # List of tasks associated with the user
  "conversations": [
    {
      "date": "2024-08-30 11:13:32",
      "origin": "Received",
      "source": "customer",
      "type": "Text",
      "content": "hi",
      "location": {"longitude": 0, "latitude": 0},
      "url": "",
      "sentiment": "",
      "intent": "",
      "Category": "",
      "urgency_level": ""
    }
  ]
}
  
```

Figure 55 Example of a user's chat history stored in MongoDB

IV.5.3 Dashboard Visualization

To visualize the stored chat history and session data, our dashboard consolidates all user interactions, whether they originate from the website or WhatsApp. This comprehensive aggregation allows administrators to view every conversation in real-time, facilitating immediate responses and manual interventions when necessary. Additionally, the dashboard displays detailed information related to each user, including their reservations and tasks, providing a complete overview of the customer's interaction history and current needs.

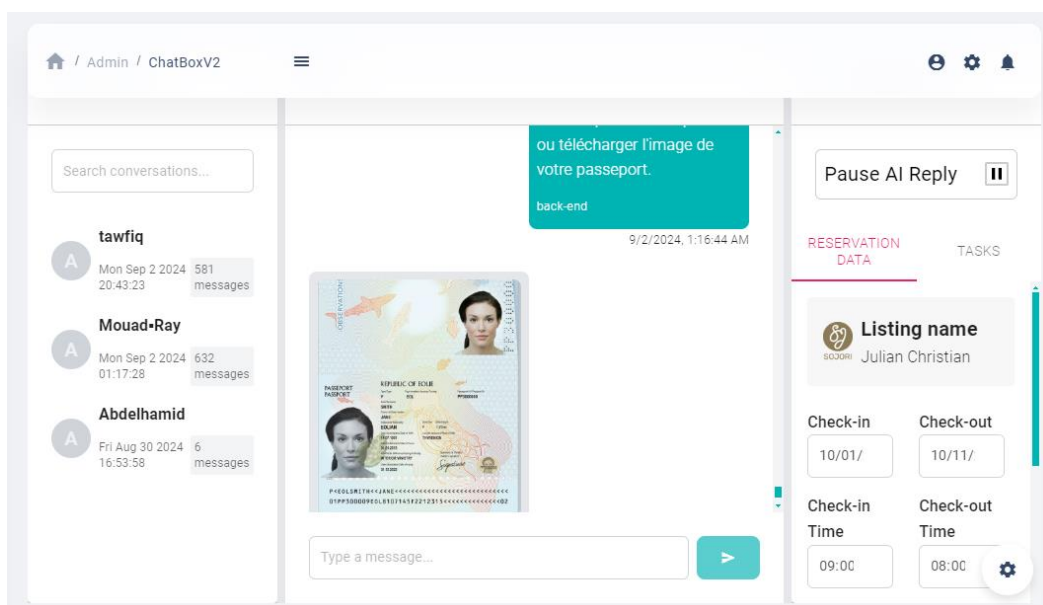


Figure 56 Real-time Chat Visualization and User Interaction Dashboard

General Conclusion and Perspectives

The development of the chatbot system for Sojori Holding marks a significant step in advancing customer service and property management. By leveraging AI technologies like Large Language Models (LLMs) and a Retrieval-Augmented Generation (RAG) system, the solution has automated the handling of complex customer inquiries across multiple platforms, including WhatsApp and the company's website. This system not only streamlines interactions by providing instant, contextually aware responses but also supports a personalized experience that reflects Sojori's commitment to enhancing customer satisfaction. Through the integration of efficient backend systems and state-of-the-art AI tools, the project has met its primary objective of creating a scalable, intelligent, and adaptable chatbot tailored to the company's needs. The successful implementation highlights the potential for AI-driven automation to transform customer service in smart home rentals and beyond.

In conclusion, this product is still in the development phase, marking just the beginning of the journey towards creating a fully optimized chatbot system. Looking ahead, there are several enhancements planned:

- **Voice Integration through smart speaker:** To enhance accessibility and ease of use, we plan on integrating in-home voice-activated assistance that will allow tenants to interact with the chatbot using voice commands, creating a seamless smart home experience. The in-home voice-activated bot will be part of the same multichannel system, allowing users to receive consistent customer service across all platforms.
- **Fine-tuning the AI Model:** As more data on customer interactions is gathered, there will be opportunities to fine-tune the chatbot's language models. This will further improve response accuracy and relevance, enhancing the overall user experience by tailoring the responses more closely to individual preferences and query types.
- **Human Intervention in the Admin Dashboard:** The Pause-AI feature will be activated on the admin dashboard, enabling human agents to intervene when necessary. The goal is to allow agents to pause AI-driven responses and take over conversations manually when complex or sensitive issues arise, all in order to improve customer satisfaction.

References

1. MongoDB. (n.d.). MongoDB Atlas Vector Search. Retrieved from MongoDB Platform:
<https://www.mongodb.com/products/platform/atlas-vector-search>
2. Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., ... & Riedel, S. (2020). Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. Retrieved from:
<https://arxiv.org/pdf/2005.11401>
3. Amazon Web Services. (n.d.). What is Prompt Engineering?. Retrieved from AWS:
<https://aws.amazon.com/what-is/prompt-engineering/>
4. Hostaway. (n.d.). *Hostaway Main Page* : <https://www.hostaway.com/>
5. Hostaway. (n.d.). *Hostaway API Documentation*. Retrieved from:
<https://api.hostaway.com/documentation>
6. HostAI. (n.d.). Bot Communication Tool. Retrieved from: <https://www.hostai.app/>
7. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N.,...& Polosukhin, I. (2017). *Attention Is All You Need*. Retrieved from: <https://arxiv.org/pdf/1706.03762>
8. OpenAI. (n.d.). *OpenAI Assistant Overview*. Retrieved from OpenAI Platform Documentation:
<https://platform.openai.com/docs/assistants/overview>
9. Meta for Developers. (n.d.). *WhatsApp Webhooks*. Retrieved from:
<https://developers.facebook.com/docs/whatsapp/webhooks/>
10. Meta for Developers. (n.d.). *WhatsApp Cloud API Overview*. Retrieved from:
<https://developers.facebook.com/docs/whatsapp/cloud-api/overview/>
11. Banh, L., & Strobel, G. (2023). *Generative artificial intelligence*. *Electronic Markets*, 33(63).
<https://link.springer.com/content/pdf/10.1007/s12525-023-00680-1.pdf>
12. Hugging Face. (n.d.). *LangGraph for Advanced RAG*. Retrieved from Hugging Face Cookbook:
https://huggingface.co/learn/cookbook/en/advanced_rag
13. Banh, L., & Strobel, G. (2023). *Generative artificial intelligence*. *Electronic Markets*, 33(63).
<https://arxiv.org/pdf/2407.19994>
14. LangChain. (n.d.). *LangChain Documentation*. Retrieved from:
<https://python.langchain.com/v0.2/docs/introduction/>

This project, conducted as part of an end-of-studies internship for a Master's degree in Artificial Intelligence and Virtual Reality at the Faculty of Sciences - Ibn Tofail University, aims to develop a customer service chatbot to enhance the tenant experience in smart home rentals. Hosted by SOJORI Holding, the chatbot leverages advanced technologies such as FastAPI, OpenAI, and LangChain, incorporating features like online check-in, property access, and communication through WhatsApp and the company's website. By combining artificial intelligence and real-time data retrieval systems, the project enhances efficiency, customer satisfaction, and service scalability.

Ce projet, réalisé dans le cadre d'un stage de fin d'études en Master Intelligence Artificielle et Réalité Virtuelle à la Faculté des Sciences - Université Ibn Tofail, vise à développer un chatbot de service client pour améliorer l'expérience des locataires dans le domaine de la location de maisons intelligentes. Hébergé par SOJORI Holding, le chatbot utilise des technologies de pointe telles que FastAPI, OpenAI, et LangChain, et intègre des fonctionnalités comme l'enregistrement en ligne, l'accès aux propriétés et la communication via WhatsApp et le site web de l'entreprise. En combinant l'intelligence artificielle et des systèmes de récupération de données en temps réel, le projet améliore l'efficacité, la satisfaction des clients, et l'évolutivité du service.