A logo with a purple arrow

AI-generated content may be incorrect.

**COORDENADORIA DE ENGENHARIA DA COMPUTAÇÃO**

**RAFAEL AZZOLINI**

**RAPHAEL AUGUSTO SANTOS**

**ESTRUTURAÇÃO DA APLICABILIDADE DE DADOS SENSÍVEIS EM UM AMBIENTE DE RECURSOS HUMANOS UTILIZANDO GERAÇÃO AUMENTADA DE RECUPERAÇÃO**

**LLEIDA, ES**

**2025**

A logo with a purple arrow

AI-generated content may be incorrect.

**RAFAEL AZZOLINI**

**RAPHAEL AUGUSTO SANTOS**

**ESTRUTURAÇÃO DA APLICABILIDADE DE DADOS SENSÍVEIS EM UM AMBIENTE DE RECURSOS HUMANOS UTILIZANDO GERAÇÃO AUMENTADA DE RECUPERAÇÃO**

Trabalho de conclusão de curso apresentado ao Centro Universitário Facens como exigência parcial para obtenção do diploma de graduação em Engenharia da Computação.

Orientador (a): Prof. **Jordi Planes Cid**.

**LLEIDA, ES**

**2025**

**FICHA CATALOGRÁFICA ELABORADA**

**PELA “BIBLIOTECA FACENS”**

Bibliotecária responsável

Eliane da Rocha CRB 8062/8a

C172e

Vieira, Thiago Pinto.

Estruturação da aplicabilidade de dados sensíveis em um

ambiente de recursos humanos utilizando geração aumentada de

recuperação / por Gabriel Martins Mancio de Camargo

Rafael Azzolini. - Sorocaba, SP: [s.n.], 2024.

83.

Trabalho de Conclusão de Curso (Graduação ou Pós) – Centro

Universitário Facens – Coordenadoria de Engenharia da Computação

– Curso de Engenharia da Computação, 2024.

Orientador: Prof. Me. Fabrício Torquato Leite.

1. Inteligência artificial. 2. Geração aumentada de recuperação. 3.

Dados sensíveis. I. Anzolini, Rafael. II. Centro Universitário Facens.

III. Título.

CDD 621.39

**RAFAEL AZZOLINI**

**RAPHAEL AUGUSTO SANTOS**

**ESTRUTURAÇÃO DA APLICABILIDADE DE DADOS SENSÍVEIS EM UM AMBIENTE DE RECURSOS HUMANOS UTILIZANDO GERAÇÃO AUMENTADA DE RECUPERAÇÃO**

Trabalho de conclusão de curso apresentado ao Centro Universitário Facens como exigência parcial para obtenção do certificado de conclusão do curso de Engenharia da Computação.

Orientador (a): Prof. **Jordi Planes Cid**.

BANCA EXAMINADORA

Prof. Me. Fabrício Torquato Leite

Afiliação (Centro Universitário Facens)

Prof. Me. Lucas Nunes Monteiro

Afiliação (Centro Universitário Facens)

Prof. Allan Marconato Marum

Afiliação (Centro Universitário Facens)

I dedicate this work to everyone who was part of this journey of professional development, and to those who contributed, directly or indirectly, to my goal being successfully achieved, amid a process permeated by joy.

**ACKNOWLEDGEMENTS**

We want to thank this educational institution for preparing us for the job market, and all the teachers, coordinators, classmates, and staff in general who have been with us on this journey in some way. To our families, who have placed great trust and credibility in us and our plans from the beginning of this journey to the present moment. To our advisors for their encouragement, for the time they donated so that we could find the best way to complete our project, and for any suggestions for improvement. To all the members of the college library for their attention and loving support.

Our sincere thanks.

*“Innovation is the key to improving the world****”.***

*Bill Gates*

**ABSTRACT**

The growth of generative Artificial Intelligences (AIs) has revolutionized data generation, as well as the automation and assistance of processes across various fields. However, it has also raised concerns regarding the protection and safe, controlled use of information. The lack of transparency and control methods over how data is utilized highlights the need to explore approaches that enable this dynamic. Integrating ethical principles and security measures from the inception of these systems is essential for ensuring user trust. This project aims to explore and study the potential use of generative artificial intelligence in conjunction with sensitive data sources through the implementation of an intelligent human resources assistant. This assistant seeks to streamline response delivery based on the access level of the requesting user. The project development leveraged technologies such as Large Language Models (LLMs) alongside programming languages like JavaScript, using libraries such as React, Node, Tailwind, Libsodium, and Python, while employing the LangChain framework. These tools facilitated the development of an application using the Retrieval-Augmented Generation (RAG) technique, which enhances the response generation capabilities of these models. The results demonstrated satisfactory performance across various request scenarios. Thus, through this project, the authors aimed to fully develop an intelligent assistant with access control, contributing to a better understanding of the different possibilities for data manipulation and protection when using LLMs.

**Keywords**: Generative AI, Access control, Human resources assistant, Deep learning algorithms, Large language model, Retrieval-augmented generation.

**LIST OF ILLUSTRATIONS**

[FIGURA 1.](about:blank) Áreas de envolvimento da PLN21

[FIGURA 2.](about:blank) Estrutura genérica de uma *RAG* 24

FIGURA 3. Relação entre os projetos 28

FIGURA 4. Modelo genérico de requisição 29

FIGURA 5. Modelo de requisição com métodos de segurança 30

FIGURA 6. Recursos utilizados 31

FIGURA 7. Definições do modelo ACID 32

FIGURA 8. Esquema da função de criptografia do *token* 37

FIGURA 9. Fluxograma da aplicação 39

FIGURA 10. Esquema de fluxo de interfaces 40

FIGURA 11. Design da interface de *login* 41

FIGURA 12. Design da interface de cadastro 42

FIGURA 13. Interface do assistente 44

FIGURA 14. Modelo do banco de dados 45

FIGURA 15. Relacionamento de tabelas com tabela intermediária 47

FIGURA 16. Modelo de estrutura do *Agent* 50

FIGURA 17. Fluxo do prompt baseado na técnica *few-shot* 52

FIGURA 18. Estrutura de exemplo utilizada no *prompt* 53

FIGURA 19. Modelo de estrutura do *Agent* de controle 56

FIGURA 20. Interface de administração de usuários 61

FIGURA 21. Exemplo de *dataset* utilizado 62

FIGURA 22. Valores do *dataset* na plataforma 63

FIGURA 23. Valor de resposta da requisição 63

FIGURA 24. Exemplo de dados relacionados na plataforma 64

FIGURA 25. Exemplo de *dataset* contendo requisições conhecidas 65

FIGURA 26. Acurácia de resultados obtidos com requisições previamente conhecidas 65

FIGURA 27. Exemplo de *dataset* contendo requisições similares 66

FIGURA 28. Acurácia de resultados obtidos com requisições similares 66

FIGURA 29. Exemplo de *dataset* contendo requisições desconhecidas 67

FIGURA 30. Acurácia de resultados obtidos com requisições previamente desconhecidas 68

FIGURA 31. Exemplo de *dataset* contendo requisições fora de contexto 69

FIGURA 32. Acurácia de resultados obtidos com requisições fora do contexto do projeto 69

FIGURA 33. Exemplo de *dataset* contendo requisições não permitidas 70

FIGURA 34. Acurácia de resultados com requisições não permitidas 71

FIGURA 35. Exemplo de *dataset* contendo requisições com *injection* 72

FIGURA 36. Acurácia de resultados com requisições contendo injeção 72

[FIGURA 37.](about:blank) Relatório geral de vulnerabilidades e riscos 73

[FIGURA 38.](about:blank) Vulnerabilidades e confiabilidade da análise conforme contexto 74

FIGURA 39. *Status* retornado pela requisição de teste de injeção 75

FIGURA 40. Tempo utilizado como critério 75

**LIST OF ACRONYMS**

ACID Atomicidade, Consistência, Isolamento e Durabilidade

*API*  *Application Programming Interface*

CNN Rede Neural Convolucional

*CORS*  *Cross-Origin Resource Sharing*

*CSP Content Security Policy*

*CSS Cascading Style Sheets*

*DDL Data Definition Language*

*DML* *Data Manipulation Language*

DLNNs Redes Neurais de Aprendizado Profundo

*FAISS Facebook AI Similarity Search*

*GPT* *Generative Pre-Trained Transformers*

*HTTP* *Hypertext Transfer Protocol*

IA Inteligência Artificial

*JSON* *JavaScript Object Notation*

JS *JavaScript*

*LLMs* *Large Language Models*

MS Microsoft

*OWASP ZAP Zed Attack Proxy*

PLN Processamento de Linguagem Natural

*RAG*  *Retrieval-Augmented Generation*

RH Recursos Humanos

RNN Rede Neural Recorrente

SGBD Sistema de gerenciamento de banco de dados

*SQL* *Structured Query Language*

**SUMMARY**

[**1 INTRODUCTION 1**](#_3znysh7)**5**

[**2 THEORETICAL FRAMEWORK**](#_1mrcu09) **18**

[**2.1 ARTIFICIAL INTELLIGENCE**](#_2et92p0) **18**

[**2.2 *MACHINE LEARNING***](#_tyjcwt) **19**

[**2.3 NATURAL LANGUAGE PROCESSING 2**](#_1t3h5sf)**0**

[**2.4 *LARGE LANGUAGE MODELS* 2**](#_4d34og8)1

[**2.5 *RETRIEVAL-AUGMENTED GENERATION* 2**](#_2s8eyo1)**3**

**3 RELATED WORKS 25**

**3.1 WORKS USED AS REFERENCE 25**

**4 METODOLOGIA 29**

**4.1 COMPARAÇÃO ENTRE AS ARQUITETURAS 29**

**4.2 ARQUITETURA 31**

4.2.1 Banco de dados31

4.2.2 *Large Language Model* e *framework* para a *técnica de RAG*33

4.2.3 Abordagem dos *Agents* 33

4.2.4 Abordagem da interface 34

4.2.5 Abordagem de controle e segurança 35

**4.3 METODOLOGIA DE TESTES****37**

**5** [**DESENVOLVIMENTO**](#_26in1rg) **39**

**5**[**.1 DEFINIÇÃO DE INTERFACES**](#_lnxbz9) **39**

**5**[**.2**](#_lnxbz9) **SEGURANÇA E CONTROLE DE ACESSOS EM SISTEMAS DE AUTENTICAÇÃO****40**

**5**[**.3**](#_lnxbz9) **IMPLEMENTAÇÃO DE TELAS DE LOGIN E CADASTRO: USABILIDADE E SEGURANÇA****40**

5[.3.1 Tela de login](#_35nkun2) 41

5[.3.2 Tela de cadastro](#_35nkun2) 41

**5**[**.4 INTERFACE DO ASSISTENTE APLICANDO CONCEITOS DO *CHATGPT***](#_lnxbz9) **43**

**5**[**.5 PAINEL DE CONTROLE DE ADMINISTRADORES**](#_lnxbz9) **44**

**5**[**.6 DESENVOLVIMENTO DE BANCO DE DADOS**](#_lnxbz9) **44**

5[.6.1 Tabelas utilizadas](#_35nkun2) 45

**5**[**.7 PREPARAÇÃO DO AMBIENTE DE *BACK-END***](#_lnxbz9) **48**

**5**[**.8 CONEXÃO DO BANCO COM AMBIENTE EM PYTHON**](#_lnxbz9) **48**

**5**[**.9 CRIAÇÃO DO *AGENT* BASEADO NO MODELO DE LINGUAGEM**](#_lnxbz9) **49**

5[.9.1 Definição do *Large Language Model*](#_35nkun2) 50

5[.9.2 Definição do *prompt* do *Agent*](#_35nkun2) 51

5[.9.2.1 Criação do conjunto de instruções do *prompt*](#_35nkun2) 52

5[.9.2.2 Criação do seletor de exemplos](#_35nkun2) 54

5[.9.3 Definição do *Agent type*](#_35nkun2) 54

5[.9.3.1 Utilização de *tools*](#_35nkun2) 55

**5**[**.10 CONTROLE DE ACESSO E SEGURANÇA NA EXECUÇÃO DO *AGENT***](#_lnxbz9) 55

5[.10.1 Implementação de *token* de usuário](#_35nkun2) 55

5[.10.2 Implementação de *Agent* interpretador para controle de acesso](#_35nkun2) 56

5[.10.2.1 Criação de *tool* de tratamento de texto](#_35nkun2) 57

5[.10.2.2 Criação do *prompt* de controle](#_35nkun2) 57

5[.10.2.3 Criação do *Agent*](#_35nkun2) 58

5[.10.3 Implementação de *tags* e parâmetros *header* da requisição](#_35nkun2) 58

**6** [**RESULTADOS**](#_32hioqz) **60**

**6**[**.1 CARACTERIZAÇÃO DA INTERFACE**](#_1hmsyys) **60**

**6**[**.2 AVALIAÇÃO DOS MODELOS**](#_41mghml) **61**

**6**[**.3** **TESTES E COLETA DE DADOS**](#_2grqrue) **64**

6[.3.1 Teste com requisições previamente conhecidas](#_vx1227) 64

6[.3.2 Teste com requisições similares às previamente conhecidas](#_3fwokq0)66

6[.3.3 Teste com requisições desconhecidas](#_3fwokq0) 67

6.3.4 Teste de alucinação 68

6.3.5 Teste do *Agent* de validação de acesso à informação 70

6.3.6 Teste de requisições com *injection* 71

**6.4 TESTES DE SEGURANÇA E VULNERABILIDADE 73**

**6.5 ANÁLISE DOS RESULTADOS E CONCLUSÕES 76**

6.5.1 Resultados das requisições 76

6.5.2 Resultados da avaliação de segurança 77

**7** [**CONSIDERAÇÕES FINAIS**](#_1mrcu09) **79**

[**REFERÊNCIAS**](#_37m2jsg) **81**

**1 INTRODUCTION**

In contemporary times, the concept of technology is undergoing a process of evolution and transformation, that is, the intensity of transformations combined with the increasing rate of technological development highlights the speed with which this area of ​​knowledge grows and develops (Grinnin; Grinnin; Korotayev, 2020).

In general, the entire human history, especially during the last centuries, can be considered (albeit with significant reservations) as a history of achievements of science and technology, especially information technologies (Kurzweil, 2001, apud Grinnin; Grinnin; Korotayev, 2020, our translation).

Furthermore, technological growth is one of the most important factors contributing to the transformations and development of society (Grinnin; Grinnin; Korotayev, 2020). As a result of this accelerated phase of progress, the emergence of new areas of development was made possible; in addition, it became possible to greatly improve areas of emerging technologies, such as the branch of artificial intelligence (AI), which was the basic concept explored in this work.

AI-based technologies have undergone a major development process in several areas, especially when considering the period that encompasses the current decade (Habbal, Ali; Abuzaraida, 2024). Among the main areas of AI developed, we can mention generative AI, which enables the creation of images, texts and videos in a synthetic way, as well as the training of chatbots based on Large Language Models (LLMs), such as ChatGPT or even Gemini.

The concept of accelerated growth in the field of generative AI, like any other, occurs to keep up with challenges related to its implementation, that is, the focus on development to the detriment of possible consequences leads to the emergence of gaps, often left aside at first, which end up becoming problems to be addressed in the future. This can be observed through the increase in concerns related to risks, security, and trust about AI-based systems (Habbal, Ali; Abuzaraida, 2024). In the context of this project, we sought to study and explore the premise of the possibility of using generative AI systems in conjunction with data that has different levels of sensitivity by uniting them with methods that enable control of access to information.

During this work, the main focus was on the concept of using LLMs for specific purposes, with the main focus being the development of intelligent assistants through the application of the Retrieval-Augmented Generation (RAG) process, which allows, in a simplified manner, to use the entire context analysis process previously trained by different models and submit it to the specific input of a set of data, to direct the model's understanding of a given subject, enabling its use in the general approach to a given context of interest, which, in this project, was the Human Resources (HR) sector and its functions.

One of the main questions related to the application of these models and techniques is precisely how the data to which the model was presented is manipulated, and how it would be possible to control the provision of this information, considering that the function of the model is solely to seek an answer, ignoring aspects such as the veracity and confidentiality of the data.

In view of this challenge and considering the notable growth of AI systems and the lack of control, the authors sought to explore the possibilities of the safe use of sensitive data provided to large-scale models, to enable their use in a judicious manner.

Therefore, the main proposal adopted was the implementation of an HR assistant, with the aim of enabling the provision of information based on requests made by users, in addition to enabling the implementation of a data provision control, to handle the variety of data with different levels of sensitivity that this sector uses.

To this end, the LLM models were presented to a set of data derived from a database containing information used by the sector in question, which allowed the model to be adapted based on an HR environment, opening up paths to complement the general activities of the sector, in addition to allowing the provision of requested information, in a way that always considers the confidentiality of the information that was requested.

Therefore, by implementing the assistant, we sought to achieve a balance between the use of generative AI and the efficiency of data supply control, quality, and security, especially sensitive data, while also developing a possibility of improving the quality of life of HR departments by creating possibilities for dynamic interaction with the department.

Next, in this monograph, Chapter 2 will address the theoretical framework adopted to carry out the project. Chapter 3 will present the works used as reference. In addition, Chapter 4 will present the entire methodology used and the decisions made in the adoption of technologies, methods, and metrics. Then, Chapter 5 describes the entire process of developing the assistant application and the main algorithms used. Furthermore, Chapter 6 will discuss the results obtained. Finally, Chapter 7 will develop the conclusions obtained throughout the entire project from the authors' perspective, as well as suggestions for future implementations.

**2 THEORETICAL FRAMEWORK**

The main characteristic of Artificial Intelligence (AI) is its ability to learn from large volumes of data, or big data, which is used to develop complex models through machine learning. This process allows AI to identify patterns and make predictions based on the information absorbed. However, the quality and representativeness of data are extremely important aspects that directly impact the performance and impartiality of these systems. According to Garcia (2020, apud Barbosa, 2022), among the main ethical challenges associated with the use of AI are the biases and prejudices that these technologies can replicate and even amplify.

During the development of this project, the concept of data quality received special attention, especially in the form of access to information and the security of viewers.

**2.1 ARTIFICIAL INTELLIGENCE**

Artificial Intelligence (AI) can be seen as an attempt to replicate human reasoning and analysis capabilities through a combination of computer science, algorithm development, mathematical models, and computational systems. This combination allows machines, through the processing of binary data, to approach complex human skills, such as decision-making, problem-solving, and even learning new patterns. Today, this new approach has an increasingly deep and widespread presence in everyday life, being incorporated into applications ranging from simple tools, such as virtual assistants and content recommendation filters, to advanced technologies for analyzing large volumes of data (big data) and predictive algorithms in areas such as health, finance, and security. The need for AI arose during the Second World War when the need for machines to be able to think like humans was essential.

According to Pires (2021 and Pereira, 2021), the historical context states that AI follows four lines of thought where:

I. Systems that think like human beings: "The new and interesting effort to make systems think like minds, in the full and literal sense”.

II. Systems that act like human beings: "The art of creating systems that perform functions that require intelligence when performed by people”.

III. Systems that think rationally: "The study of mental faculties through their use of computational models”.

IV. And systems that act rationally: "Computational intelligence is the study of the design of intelligent agents”.

According to Oliveira (2021 apud Pereira, 2021), AI is used in the most diverse situations, not only where it can replicate human thought, but also in contexts that involve solving problems that range from identifying the linguistic intelligence of texts in the most varied formats to the vision and processing of images.

Thus, it is logical to think that AI is present in the most diverse sectors, acting in a way that seeks to transform operations into increasingly productive and efficient ones, eliminating repetition and generating the full use of available resources in the sectors employed, developing their greatest potential and employing substantial results in the creation and solution of current and future problems.

**2.2 *MACHINE LEARNING***

The term machine learning, according to Tavares (2020 apud Barbosa, 2022), seeks to use certain data patterns to enable computers to create knowledge using algorithms and data patterns, so that without receiving information or direct commands from a person, the machine can make decisions, solve problems, and identify errors in a simplified manner, a large part of the activity of human intelligence. Thus, data and algorithms enable the machine to produce information and perform certain actions independently.

The “learning” process of these systems is influenced by multiple factors, observable or unobservable in the physical world, subject to effects from external sources. The machine learning technique that can deal with the complexity of the real world is called “deep learning”: a mathematical-statistical function that maps sets of input values ​​to output values ​​through representations expressed in terms of other, simpler representations, identified in distinct layers (KAUFMAN, 2021, p. 75).

This form of machine learning is also called Deep Learning Neural Networks (DLNNs). Its main inspiration is based on the functioning of the human brain, using training on extensive and labeled data sets, to understand complex and hierarchical representations of data.

**2.3 *NATURAL LANGUAGE PROCESSING***

According to Beysolow (2018 apud Barbosa, 2022), Natural Language Processing (NLP) is the field of AI that deals with the manipulation and interpretation of data in natural language, whether spoken or written, as used by humans. The goal of NLP is to enable machines to understand and interact with human language effectively, overcoming the complexity of specific expressions and structures that are part of everyday language use.

In the case of spoken language, NLP deals with challenges such as speech recognition, understanding intonations, and detecting patterns to interpret implicit meanings through vocal cues. These elements are essential for the machine to interpret not only the literal content of words but also the context and implicit meanings, enabling more natural interaction.

Exploring natural language in its various forms, NLP seeks to understand the literal meaning of words, and capture nuances, contexts and intentions present in human communication, including sentiment analysis, identification of relevant information, automatic translation, generation of summaries, among several other applications that aim to facilitate interaction between humans and machines through language.

**2.4 *LARGE LANGUAGE MODELS***

Large Language Models (LLMs) fall into the category of base models, where they are extensively trained with large volumes of data, generating the ability to understand and generate natural language in a wide variety of activities and contexts. These models are designed to perform various NLP tasks efficiently, including automated text generation, chatbot-like interaction to answer user questions and text translation, and even sentiment analysis and automatic summaries. Thus, NLP is adopted in several applications and contexts, where its effectiveness is evident in the ability of the models to generate interpretable outputs satisfactorily. This is due to the robust architecture of the systems, which are trained through supervised learning on large sets of textual data. Figure 1 shows some of the main areas of application of LLMs today.

Figure 1 – PLN's areas of involvement

*A diagram of different applications

AI-generated content may be incorrect.*

[[1]](#footnote-1)Source: Available at: https://www.doi.org/. Acessed on: March 31, 2025.

Today, the most advanced models considered state-of-the-art have tens of billions of adjustable parameters during training. Among these models, we can highlight the Generative Pre-Trained Transformers (GPT), a model pre-trained on large sets of textual data, allowing fine-tuning for specific tasks, such as language generation, sentiment analysis, language modeling, machine translation, and text classification. Its Transformer architecture represents a significant advance compared to previous NLP approaches such as the Recurrent Neural Network (RNN) and the Convolutional Neural Network (CNN).

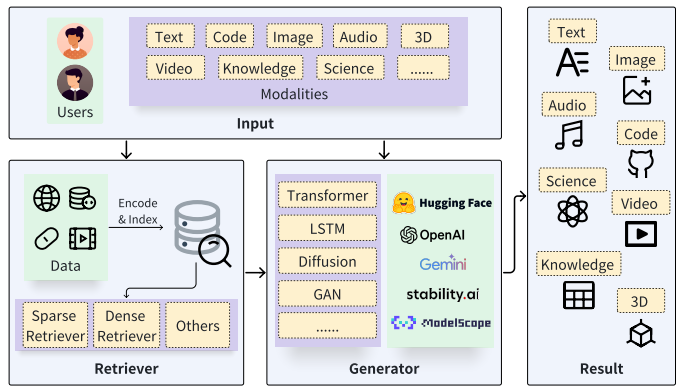
The LLaMa developed by Meta represents another advanced LLM model. Its objective is to improve the response of chatbots to user questions, mainly addressing the need for flexibility and specialization through incremental learning, in addition to being free to use. Falcon LLM proposes a significant change compared to the approaches of other AIs, where there are data sets with 180B, 40B, 7.5B, and 1.3B models, thus allowing the democratization of access to AI technology so that individuals and organizations can run these models on their systems.

ChatGPT is an example of an LLM in the form of a powerful and innovative tool, made available by the company OpenAI in 2022. It gained great notoriety due to its ability to provide assertive solutions and its ease of use. Among the applications of ChatGPT, it has proven capable of answering a wide range of questions and topics, summarizing texts, writing code, and providing information on various subjects, thus proving itself as a valuable tool in the search for efficiency, saving time, and providing an acceptable response. Despite its many advantages, ChatGPT in its 3.5-Turbo version has some limiting points, such as the number of tokens, that is, the number of interactions that the user has, in addition to presenting limits on the size of the responses. Sometimes it presents an incorrect response, it is necessary to check the assertiveness of its responses.

**2.5 *RETRIEVAL-AUGMENTED GENERATION***

The RAG concept, first proposed by Lewis et al. (2020), has been receiving increasing attention due to its ability to minimize hallucinations in LLMs, where false, incorrect, or irrelevant information is generated due to missing data in the LLM training sets, thus affecting the reliability and applicability of the models.

When considering the aspects of LLMs, one limitation that arises in their use is the quality of the data used in their training, which is often generalized or outdated. This is especially relevant in the case of a chatbot, where the lack of assertiveness can hinder the provision of accurate information about the organization's products or services. In addition, another recurring problem when using LLMs is hallucinations, a limitation caused by the absence of data in the LLM training sets, which can be significant in contexts where the accuracy and relevance of information are critical. In Figure 2, it is possible to observe the interaction between the user and the system in the functioning of the RAG architecture, where the user's inputs are used to search for information, which varies between different formats, such as text, code, image, video, context, among others. Thus, these inputs are used in the retriever (Retriever) and in the generator (Generator), where the retriever extracts relevant information from the available data sources, and the generator interacts with the returned results, producing responses in different formats.

Figure 2 – Generic structure of a RAG

[[2]](#footnote-2)Source: Available at: https://www.doi.org/. Accessed on: 31 March. 2025.

By using RAG in conjunction with LLM, this problem tends to be solved, since the combination of text generation and information retrieval, whether from databases or documents, allows the model to consult information from reliable and structured sources. In this way, the responses generated tend to be more accurate, relevant, and based on real data that are pertinent in the context of the RAG application.

**3 RELATED WORKS**

To develop and use metrics and methods that would contribute to the development of the project, research was conducted on articles and projects that addressed the theme of the work, considering aspects such as implementation, methodologies, and definition of results.

The main research documents sought were scientific publications, such as monographs, theses, articles, among others, with an emphasis on the year 2019 onwards.

In addition, the main topics sought in the research were those related mainly to applications with large language models, approaches to the Retrieval-Augmented Generation technique, enhancement of LLMs, and security and access control methodologies.

When observing RAG applications, its application focuses on scenarios where the LLM model must provide information with a higher level of context or where the information is continuously updated. However, it is extremely important to be aware of the quality and timeliness of the data sources. The works used as reference for the project will be presented below.

**3.1 TRABALHOS UTILIZADOS COMO REFERÊNCIA**

O trabalho de Proença (2024) utiliza uma implementação da técnica de *RAG* para auxiliar no processo de levantamento bibliográfico para revisões de literatura durante o processo do desenvolvimento de trabalhos acadêmicos. Por meio desse método, foram obtidos ótimos resultados de acurácia tanto no processo de recolhimento das informações quanto no processo de geração de respostas, com o parâmetro de acurácia em torno de 91%.

Os autores Chen, Lin, Han e Sun (2024) expõem em seu artigo métodos de análise de desempenho da técnica de *Retrieval-Augmented Generation* quando utilizada em conjunto com diferentes LLMs, considerando aspectos como alucinação, integração da informação e ruído nos dados de referência. A aplicação de seu projeto demonstrou bons resultados na acurácia da geração de respostas baseadas no conjunto de dados mesmo em situações onde existiu muito ruído junto aos dados consultados.

O trabalho de Ganesh, Purwar e B. (2024) explora a utilização de modelos de classificação de texto em conjunto com LLMs de modo a potencializar a técnica de *RAG* ao expandir as informações de contexto na geração das respostas.

O artigo de Soong *et al.* (2024) desenvolve todo o processo de implementação do processo de *Retrieval-Augmented Generation* com o intuito de melhorar a geração de respostas criadas pelos modelos *GPT 3* e *4* considerando o contexto de aplicações na biomedicina. A acurácia obtida com respostas fornecidas pelo modelo baseado em RAG foi bem superior às fornecidas por simples LLMs utilizados na comparação, com valores até 25% superiores.

O projeto de Izacard *et al.* (2023) aborda o desenvolvimento de um modelo utilizando a técnica de RAG em conjunto com um método de *few-shot* para geração de respostas baseadas em um pequeno conjunto de exemplos. A utilização desse método permitiu o modelo obter melhores resultados de acurácia, cerca de 3%, quando comparado com modelos que utilizam até 50 vezes mais parâmetros para treinamento.

O projeto de Lara *et al.* (2023) explora o desenvolvimento de um assistente de atendimento humanizado por meio da manipulação de *large language models* em conjunto com o *framework* *Langchain.* A análise e validação de seu projeto se deu de forma qualitativa, por meio de interpretações das decisões tomadas pelo modelo considerando diferentes situações, o qual demonstrou certo nível de dificuldade ao trabalhar com requisições mais complexas, em detrimento de requisições simples.

Vidivelli, Ramachandran e Dharunbalaji (2024) promovem um estudo do desenvolvimento de *chatbots* utilizando a abordagem do *Langchain* por meio da aplicação da técnica de *Retrieval-Augmented Generation*, com o intuito de aprimorar a previsão e relevância das interações com os usuários. O modelo implementado apresentou cerca de 97% de acurácia na geração de respostas as quais foi previamente instruído. Além disso, por meio da implementação da técnica de RAG, o modelo obteve melhores resultados das métricas de *BLEU* e *ROUGE*, que comparam um texto gerado com um texto de referência, quando comparado à sua implementação sem a técnica de RAG, alcançando cerca de 23 a 36 pontos de diferença.

O artigo de Yoshioka e Silva (2023) aborda o aprofundamento do conhecimento em segurança de informação e a importância da análise de vulnerabilidades, propondo soluções de controle e segurança ao utilizar domínios e métodos de cabeçalho para proteção.

O trabalho de McAvinue e Dev (2024) desenvolve um processo de análise comparativa na geração de respostas e recomendações de dois LLMs utilizando as métricas e ferramentas de avaliação da plataforma *Langsmith.* Ambos os modelos testados apresentaram resultados máximos de acurácia em determinados aspectos, como o processo de geração da recomendação e neutralidade de respostas considerando o perfil do usuário.

A pesquisa de Alfarizi, Najie, Afif e Ashari (2022) aborda a utilização da ferramenta OWASP ZAP na detecção de vulnerabilidades e suas possíveis soluções dentro de um sistema. A utilização da ferramenta apresentou resultados significativos na detecção de vulnerabilidades, possibilitando a tomada de decisão e adoção de medidas para combatê-las.

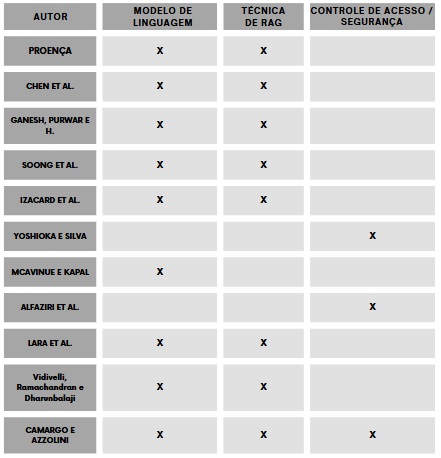
Alves, Coelho e Cunha (2021) desenvolveram em seu projeto um *framework* de criptografia baseado na biblioteca *libsodium*, por meio da análise de cada um dos principais métodos, considerando aspectos como tempo de processamento e modelo de manipulação da informação criptografada. Sua pesquisa demonstrou explicitamente que modelos mais simples levam menos tempo para manipular a informação em comparação a modelos mais complexos, que apresentam quase o dobro do tempo para a mesma tarefa.

Assim como os projetos expostos, este trabalho visou desenvolver uma abordagem da técnica de *RAG* que integre a estrutura de um LLM no desenvolvimento de um assistente, utilizando uma rede já estabelecida. Entretanto, a abordagem de segurança e controle de acesso às respostas do modelo que contém informações sensíveis foi implementada de maneira a complementar todas as características individuais presentes nos projetos de referência.

Além disso, nessa etapa, a atenção ao acesso dos dados é enfatizada de forma mais incisiva, visando estabelecer um diferencial na manipulação de informações sensíveis.

A tabela da figura 3 a seguir ilustra a relação das abordagens de todos os projetos em comparação com este projeto. As linhas representam os autores de cada um dos projetos e as colunas representam as principais características abordadas considerando todos os trabalhos apresentados.

Figura 3 - Relação entre os projetos



Fonte: elaborada pelo autor.

No Capítulo 4 de Metodologia, a abordagem das métricas das referências ocorre de forma mais detalhada, explicando a escolha dos métodos adotados para desenvolvimento.

**4 METODOLOGIA**

O capítulo a seguir abordará com um pouco mais de profundidade os métodos, ferramentas e validações utilizadas no desenvolvimento do projeto.

Atualmente é possível encontrar diversos modelos de assistentes inteligentes baseados na técnica de RAG, eles são utilizados em áreas como medicina, design, ou mesmo obras artísticas e, além disso, esses mesmos assistentes demonstram ótimos resultados quando submetidos a validações que testam sua eficácia.

Entretanto, as informações fornecidas por tais mecanismos auxiliares não é submetida a nenhuma verificação ou controle de segurança, de tal forma que quaisquer informações são passíveis de exposição a qualquer usuário que utilize a aplicação, deixando clara a limitação e ausência de segurança presente nos mais variados projetos.

Portanto, buscou-se neste projeto a implementação de métodos de segurança que pudessem trabalhar e manipular dados de modo seguro e transparente, e ao mesmo tempo expandir as possibilidades e aplicabilidades da utilização de assistentes aliados a modelos LLM, principalmente em contextos que envolvam dados sensíveis.

**4.1 COMPARAÇÃO ENTRE AS ARQUITETURAS**

Como mencionado anteriormente, o principal ponto de divergência entre a arquitetura adotada no projeto e as demais arquiteturas está na adoção de técnicas de controle de acesso e segurança com o intuito de desenvolver a proteção e manipulação dos dados.

As figuras 4e 5 a seguir demonstram de maneira simples e objetiva as divergências entre os modelos.

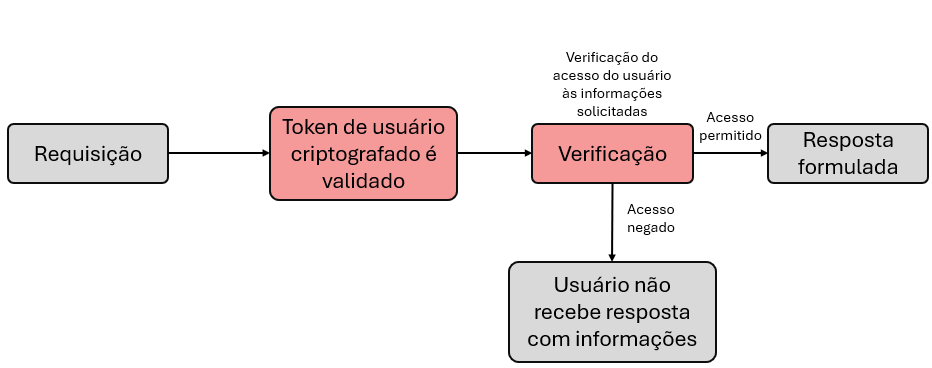
Figura 4 - Modelo genérico de requisição



Fonte: elaborada pelo autor.

A figura 5 a seguir deixa explícita a inserção e funcionalidade dos métodos de segurança dentro de uma requisição.

Figura 5 - Modelo de requisição com métodos de segurança



Fonte: elaborada pelo autor.

Como é possível observar na figura 5, o processo de transporte de dados para o LLM e posterior geração de resposta é submetido a dois processos de validação de segurança e controle para responder a uma requisição, ou seja, num primeiro momento é realizada a verificação do *token* de acesso daquele usuário que realiza a solicitação, pois é por meio dele que é permitido o acesso ao assistente e, posteriormente, é verificado se aquele usuário pode acessar aqueles dados, visto que existem tipos diferentes de perfis, que possuem níveis de acessos diferentes.

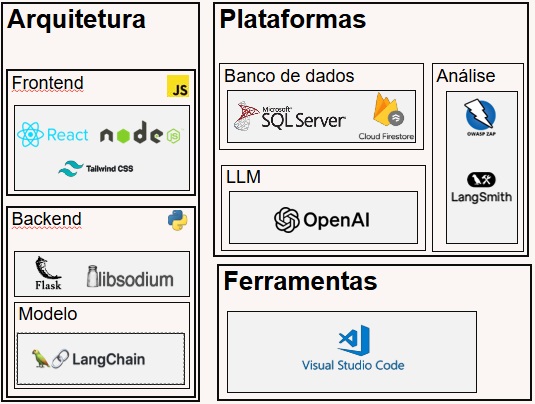
Por outro lado, a inserção de uma camada de métodos de segurança no interior do processo de requisição pode acarretar na diminuição do desempenho da aplicação quando comparada a aplicações as quais o fornecimento das informações é realizado de forma direta.

Além disso, o foco principal na segurança limita o processo de escalabilidade, visto que a centralização e controle das informações são aspectos cruciais na manutenção da confiabilidade dos dados, o que não se torna um problema em aplicações que abordam apenas o fornecimento dos dados os quais foram apresentadas.

**4.2 ARQUITETURA**

A seguir, será abordada a questão da metodologia aplicada na arquitetura do projeto. De maneira complementar, a figura 6 a seguir ilustra de forma sumarizada os recursos utilizados no projeto.

Figura 6 - Recursos utilizados



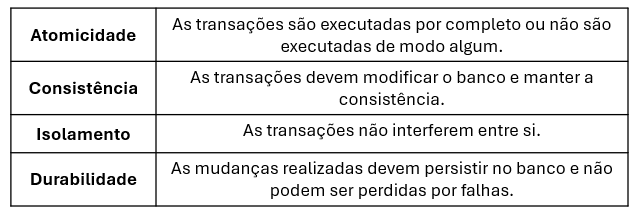
Fonte: elaborada pelo autor.

**4.2.1 BANCO DE DADOS**

Em um primeiro momento, foi definido o banco de dados SQL como fonte de informações do projeto. A escolha do banco no modelo relacional foi definida ao considerar características como a rigidez da estrutura, a conformidade com os modelos ACID e a escalabilidade.

Considerando a premissa da aplicação da necessidade de segurança dos dados, a rigidez ou ainda a pouca flexibilidade do esquema de dados relacional torna sua estrutura bem definida e fortemente conectada, de modo que contribua com o controle de integridade e manutenção dos dados. Nessa mesma lógica, a estrutura relacional entra em grande conformidade com o modelo de bancos ACID, que consiste nas características de atomicidade, consistência, isolamento e durabilidade. Dessa forma, as transações e todo processo de manipulação dos dados é feito com foco principal na sua persistência e confiabilidade. A figura 7 abaixo ilustra de maneira objetiva as características do modelo ACID.

Figura 7 - Definições do modelo ACID



Fonte: elaborada pelo autor.

Apesar da alta escalabilidade oferecida por modelos não relacionais parecer um aspecto positivo num primeiro momento, esta pode se tornar um problema no que se diz respeito à dificuldade de controle dos dados e sua integridade de modo geral, dificultando o processo, outro ponto que influenciou fortemente na escolha do modelo de banco.

Além disso, algumas características como o conhecimento prévio, o que permite uma manipulação mais eficiente dos dados, a facilidade de utilização e possibilidade de explorar a problemática do projeto de maneira mais próxima à realidade, na qual são majoritariamente utilizados bancos de dados relacionais, foram fatores que também contribuíram na decisão.

**4.2.2 *LARGE LANGUAGE MODEL* E *FRAMEWORK* PARA A TÉCNICA DE *RAG***

Posteriormente à escolha do banco, iniciou-se processo de definição do LLM a ser utilizado no desenvolvimento do projeto. Considerando o custo de viabilização do projeto e fatores como a facilidade de aplicação, atividade da comunidade e crescimento nos últimos meses, foi adotada a *API* da OpenAI como modelo de linguagem aplicado no projeto.

Tendo em vista a necessidade da utilização de um LLM para desenvolvimento do assistente e utilizando a mesma abordagem presente nos trabalhos de Lara *et al.* (2023) e de Vidivelli, Ramachandran e Dharunbalaji (2024), optou-se por utilizar a biblioteca python *LangChain***,** considerando sua principal caraterística: a facilidade de implementar aplicações em conjunto com LLMs.

A *LangChain* é uma ferramenta de desenvolvimento muito versátil, e oferece diversos mecanismos e *features* que possibilitam um melhor desempenho na comunicação e utilização de *LLMs* de maneira geral, o que contribuiu fortemente com a escolha de sua utilização no projeto. Além disso, ela também destaca-se na simplicidade de utilização de seus métodos e sua robusta documentação.

Dentre as principais ferramentas da *LangChain* utilizadas podem ser citados os *Agents*, *Prompts*, *tools*, ferramentas e plataformas de validação, como o *Langsmith,* que serão melhor detalhadas a seguir.

**4.2.3 ABORDAGEM DOS *AGENTS***

Definidos o LLM e o banco de dados utilizado, o assistente teve seu início de desenvolvimento por meio da criação de um *Agent* de consulta. Um *Agent* consiste em uma ferramenta que possibilita que LLMs executem ações em uma determinada ordem de acordo com a situação de requisição que receberam previamente, eles atuam no papel de operários esperando uma ordem (no caso do projeto, uma pergunta ou requisição) para executar as tarefas que julgarem necessárias.

Por exemplo, os *Agents* implementados no projeto devem interpretar a pergunta realizada pelo usuário, entender quais ferramentas utilizar e definir o melhor caminho a tomar para realizar uma consulta de dados que possa fornecer uma resposta satisfatória. Considerando a fonte de dados escolhida para o projeto, o *Agent* definido para implementação do assistente foi o *Agent* SQL, que além de possibilitar uma abordagem mais flexível em tabelas SQL, também oferece funções como verificação de *schema* (estrutura) de tabelas e recuperação de erros recorrentes, de forma nativa.

Após a construção do *Agent*, foi realizada a definição e implementação de um *Prompt* específico e customizado, de modo a otimizar e direcionar de maneira efetiva o comportamento do *Agent* e sua tomada de decisão.

Um *prompt* é uma ferramenta que permite a definição de instruções e a entrada de dados configurados pelo usuário de maneira a auxiliar o *LLM* no entendimento do contexto e na geração de respostas mais coerentes. Os *prompts* são altamente utilizados em conjunto com os *Agents,* pois é por meio deles que são fornecidas todas as regras e caminhos que podem ser seguidos de forma detalhada após o recebimento de uma requisição. De maneira simples, o *Agent* é criado com base em um *prompt* que deverá seguir.

No contexto do projeto, o *prompt* carrega informações técnicas e comportamentais, definindo a função de busca na base de dados, instruções relacionadas à realização da coleta de informações, processos de verificação e decisões a serem tomadas caso a requisição fuja do contexto geral do projeto. Além disso, com base no trabalho de Izacard *et al.* (2022), foi adotada a técnica de few-shot na definição do *prompt*, com o intuito de possibilitar o aprendizado por meio do fornecimento de exemplos, os quais servem de comparação no momento de geração da resposta.

**4.2.4 ABORDAGEM DA INTERFACE**

Terminadas as configurações e implementações principais dos assistentes e do banco de dados para consumo, iniciou-se a implementação da interface do usuário final. Para o desenvolvimento da interface foi definida a biblioteca React JS ao considerar a experiência prévia dos autores com a tecnologia.

O *wireframe* da aplicação e a interface de usuário foram idealizados de modo que fossem simples e fortemente intuitivos, visto que em cenários reais diferentes tipos de usuários podem entrar em contato com a aplicação, cada um com um distinto nível de afinidade em relação à tecnologia.

Portanto, ela conta com uma tela inicial de apresentação da aplicação, cadastro, *login* de usuário, *dashboard* de controle de perfil, bem como uma tela de interação direta com o assistente. Além disso, a tela de interação conta com funcionalidades como registro de respostas anteriores e gravação de conversas.

**4.2.5 ABORDAGEM DE CONTROLE E SEGURANÇA**

Um dos pontos principais da aplicação é a possibilidade de utilizar dados com com os mais diversos patamares de sensibilidade de maneira mais segura, portanto fez-se necessária a implementação de métodos e técnicas de segurança e controle de acesso.

A utilização do *Agent* de consulta é baseada na definição de 2 perfis de usuário, sendo eles o perfil de funcionário, que tem acesso restrito aos dados, e o perfil de gestor, que possui acesso completo aos dados. Para a realização desse controle de acesso, foi implementado um segundo *Agent*, complementar ao primeiro.

O *Agent* de controle de acesso foi criado utilizando os mesmos métodos de criação e utilização do *Agent* de consulta, porém com instruções em seu *prompt* focalizadas na verificação do teor e da temática da pergunta, de modo que o modelo possa interpretar a pergunta feita, classificá-la e responder se ela contém dados que não podem ser fornecidos para um funcionário. A utilização do *Agent* de controle está condicionada a um *token* de usuário implementado na aplicação, que guarda o valor do perfil do usuário que realiza a requisição.

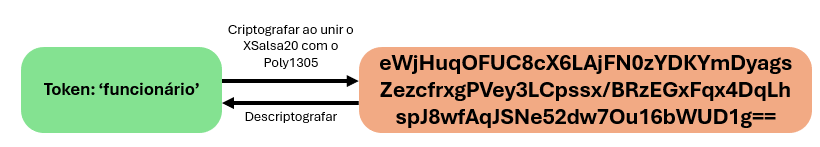
De maneira complementar, outro mecanismo de segurança utilizado no projeto foi o de criptografar e descriptografar o *token* do usuário que interage com a aplicação, de modo que tornou-se possível saber qual perfil de usuário está ativo na aplicação e ao mesmo tempo esconder esse dado por meio de criptografia.

O processo de codificação dos dados foi executado foi baseado no apresentado no projeto de Alves, Coelho e Cunha (2021) por meio da biblioteca *javascript* denominada *libsodium* em conjunto com a biblioteca *python* *pynacl*, as quais fornecem métodos de criptografia de dados e interoperabilidade entre as linguagens.

O método de criptografia utilizado foi o de criptografia simétrica*.* Que consiste na utilização de apenas uma chave entre emissor e receptor, os quais a utilizam para estabelecer a conexão segura entre os dados obtidos no processo de comunicação, sem mencionar que o método oferece extrema rapidez e eficácia, o que somado à facilidade de utilização e implementação em detrimento de métodos assimétricos que trabalham com duas chaves, tornou-se uma escolha viável de abordagem no projeto.

Os dados foram criptografados por meio de um algoritmo de fluxo de chave denominado XSalsa20-Poly1305. O algoritmo XSalsa20 permite a transformação das informações em uma sequência contínua de *bytes* que é combinada com o texto a ser criptografado, o que faz com que o fluxo seja mais eficiente para dados de diferentes tamanhos. Além disso, o algoritmo também usa um *nonce,* número aleatório e arbitrário que só pode ser utilizado uma vez, que é unido à chave de criptografia para produzir uma *keystream*, que é utilizada para efetivamente criptografar a mensagem. O método de criptografia é o mesmo para cada dado, entretanto, a utilização do *nonce* garante que um mesmo dado resulte em uma *string* criptografada diferente em cada execução, o que se torna útil na aplicação do projeto, visto que os valores de *tokens* podem ser ou funcionário ou gestor para cada usuário. Ademais, o XSalsa20 também é utilizado em conjunto com o algoritmo Poly1305, cuja função é executar um processo interno de autenticação que permite verificar a integridade e autenticidade de uma mensagem criptografada. Um exemplo do processo de criptografia do *token* pode ser observado na figura 8 a seguir.

Figura 8 - Esquema da função de criptografia do *token*



Fonte: elaborada pelo autor.

De maneira resumida, no momento de executar uma requisição ao assistente, é verificado inicialmente o *token* e posteriormente avaliado o nível de acesso do usuário em relação à informação solicitada, de modo que fiquem bem definidas as informações que serão fornecidas para cada usuário.

Por fim, com base no estudo abordado por Yoshioka e Silva (2023), foram utilizados métodos de segurança no cabeçalho das requisições, com o intuito de proteger os dados utilizados entre as requisições e as demais informações contidas em todo o processo de comunicação entre as tecnologias do projeto, que serão melhor desenvolvidas no Capítulo 5.

**4.3 METODOLOGIA DE TESTES**

Posteriormente à definição e construção da metodologia de arquitetura da aplicação, iniciou-se o processo de definição dos métodos de testes e validações, que pudessem conferir maior confiabilidade ao projeto.

O principal método adotado foi o de comparação entre as respostas geradas pelo assistente com as respostas já esperadas para determinadas consultas, foram realizados testes com requisições conhecidas, similares às conhecidas, não conhecidas e de injeção de *script*. A avaliação foi realizada com base no projeto de McAvinue e Dev (2024) por meio da utilização do parâmetro de acurácia obtido com a implementação do método *Response Evaluation* da ferramenta *Langsmith* contida na própria biblioteca *LangChain.*

Essa ferramenta consiste em um conjunto de métodos que possibilitam a avaliação das respostas obtidas por meio da utilização de *large language models*, além disso, seus métodos utilizam o próprio LLM para gerar *scores* ao relacionar um conjunto de exemplos de perguntas e respostas às ações e retornos do *Agent* baseado nos modelos, realizando uma comparação direta entre as informações.

Ademais, outro método de validação utilizado com base nessa ferramenta foi a verificação do desempenho do *Agent* de controle por meio da relação do acesso do usuário com a informação solicitada. Foram realizadas requisições de diferentes tipos de dados utilizando *logins* de usuários com diferentes níveis de acesso, sejam eles de funcionário ou de gestor.

Previamente conhecidas as situações em que deveriam ser fornecidas as respostas ou não, foi possível validar a funcionalidade de bloqueio de fornecimento da informação baseado no nível de acesso daquele que as solicitou.

Por fim, o último método de validação adotado por meio do *Langsmith* foi a verificação de ausência de alucinação do modelo, ao realizar perguntas das mais variadas áreas do conhecimento, as quais o modelo foi previamente instruído a não responder. Dessa forma, utilizando um conjunto de dados reduzido, foi possível realizar uma cobertura completa de funcionamento da aplicação, partindo do acesso ao fim da requisição.

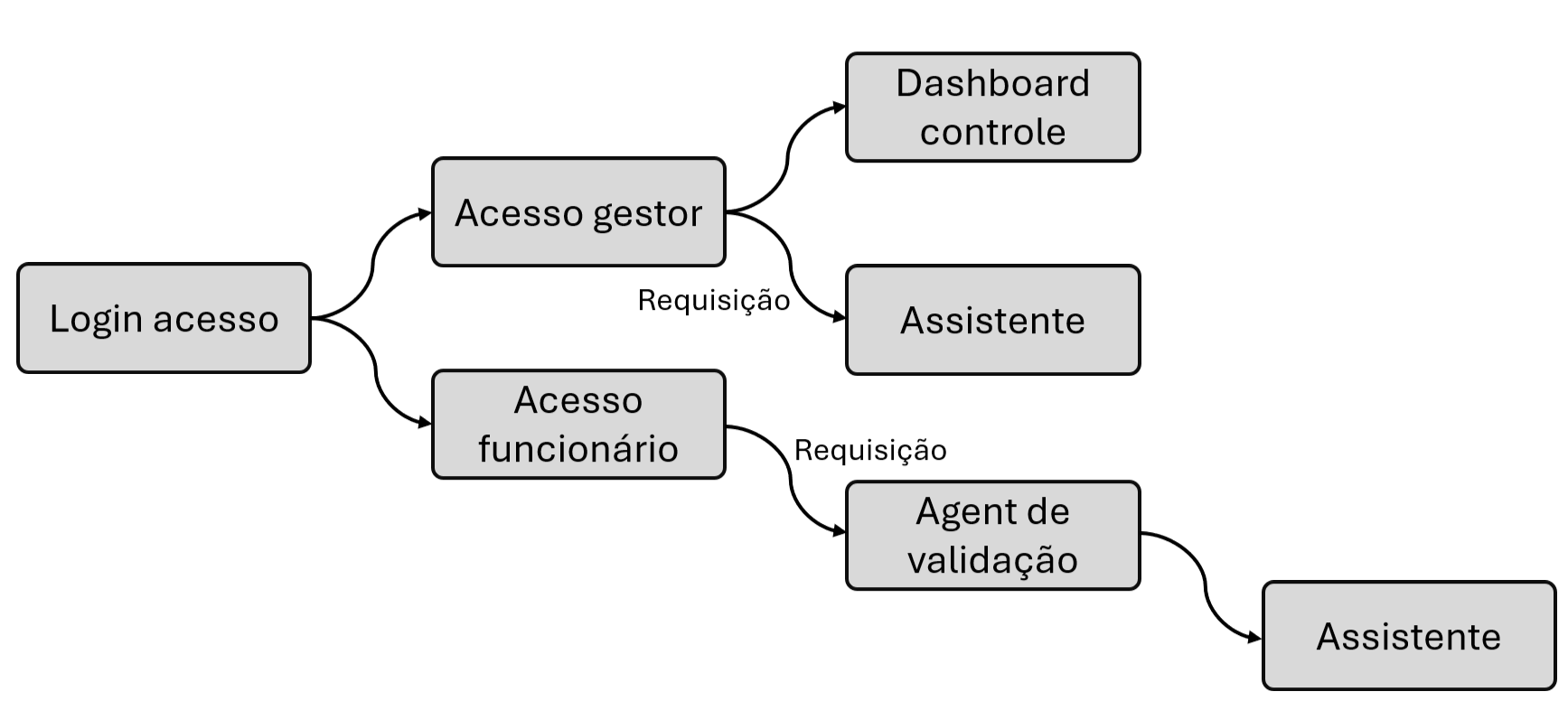
Outro ponto de suma importância no processo de testes foi a utilização do software OWASP ZAP com base no abordagem utilizada por Alfarizi *et al.* (2022). A OWASP é uma comunidade que oferece métodos e ferramentas relacionados à área de segurança de aplicações web. Uma de suas principais ferramentas fornecidas é a ZAP, um software que consegue realizar diretamente verificações de segurança, detectar vulnerabilidades dentro de uma aplicação e ainda fornecer caminhos para a solução dos problemas, funcionando como um intermediador entre o navegador e a aplicação.

**5. DESENVOLVIMENTO**

O capítulo a seguir abordará de forma detalhada e técnica todo o processo de desenvolvimento e implementação do projeto, partindo da definição das tecnologias, criação de interface, construção do banco até a criação dos *Agents.*

De maneira complementar, o fluxograma presente na figura 9 simplifica o funcionamento geral da aplicação.

Figura 9 - Fluxograma da aplicação

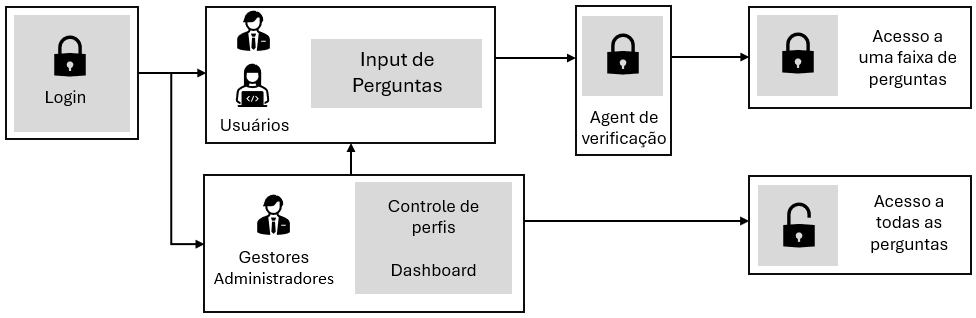


Fonte: elaborada pelo autor.

**5.1 DEFINIÇÃO DE INTERFACES**

Considerando as funcionalidades ao decorrer da elaboração do projeto, foi preciso identificar e desenhar cada um dos pontos de interação do usuário com o assistente, e como cada um deles se comportaria a depender do contexto apresentado.

Assim, ao decorrer do levantamento de requisitos foi identificado os sistemas que estabelecem relação de comunicação com a interface principal. A figura 10 ilustra o esquema do fluxo de navegação entre as interfaces.

Figura 10 - Esquema de fluxo de interfaces  
  
Fonte: elaborada pelo autor.

**5.2 SEGURANÇA E CONTROLE DE ACESSOS EM SISTEMAS DE AUTENTICAÇÃO**

Observando o funcionamento principal do assistente, um dos conceitos que levanta maior observação é relativo a segurança, que devido ao constante fluxo de consultas de informações de terceiros, são sensíveis por princípio. Desta forma surge a necessidade de desenvolver um ambiente seguro, implementando uma forma controlada de acessos, onde as informações fossem consultadas de maneira coordenada e auditada, assim era notável a criação de um sistema de autenticação de usuários. Com o surgimento desta necessidade, era indispensável o desenvolvimento de um sistema de *login*, incluindo uma forma de criação de registros de usuários, assim como um para o gerenciamento de acessos, onde os administradores da aplicação teriam controle dos perfis de acesso de cada usuário registrado.

**5.3 IMPLEMENTAÇÃO DE TELAS DE LOGIN E CADASTRO: USABILIDADE E SEGURANÇA**

A interface de *login* mencionada anteriormente é responsável por uma das etapas cruciais durante o uso do assistente, o acesso ao ambiente seguro da aplicação parte desta, onde os usuários que estejam devidamente cadastrados poderão utilizar o assistente.

**5.3.1 TELA DE *LOGIN***

O desenvolvimento da interface de *login* seguiu com o sistema de autenticação de *e-mail* e senha utilizando o Firebase, uma solução confiável para garantir a segurança do sistema. A escolha por este formato se deu pelo equilíbrio oferecido entre segurança e facilidade de implementação. Para integrar o Firebase ao projeto, o primeiro passo foi instalar a biblioteca com o comando *npm install firebase* e configurar os arquivos necessários. Na figura 11é possível observar o resultado da interface implementando os recursos citados.

Figura 11 - Design da interface de login

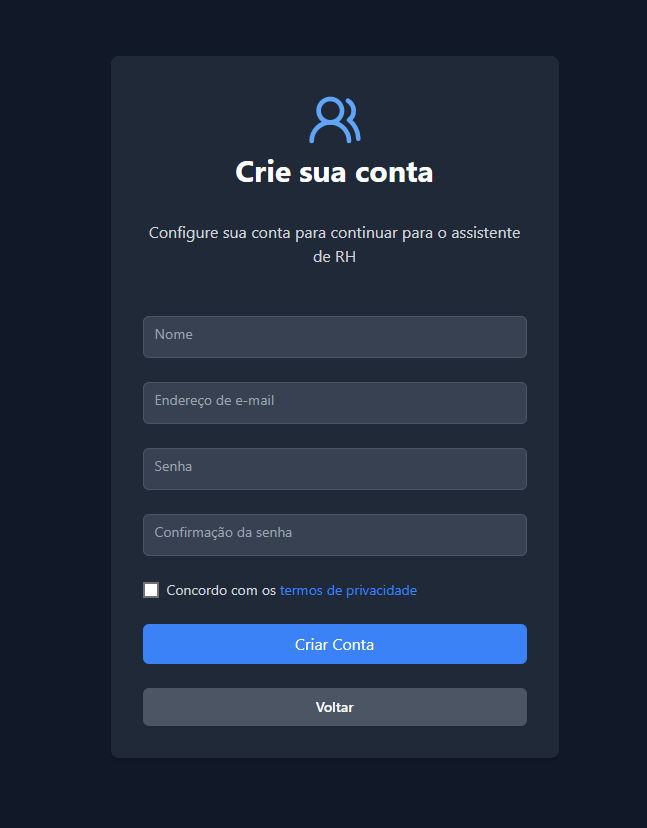


Fonte: elaborada pelo autor.

A criação da interface fez uso intensivo do conceito de componentes, uma característica central do React e um dos principais motivos para sua escolha como framework neste projeto. O uso de componentes permite a construção de uma aplicação modular, facilitando a manutenção e a manipulação do código, além de promover a reutilização de partes da interface de maneira eficiente.

**5.3.2 TELA DE CADASTRO**

A interface para o cadastro de novos usuários também foi desenvolvida seguindo a abordagem de componentes do React, aproveitando-se das vantagens de modularidade e reutilização de código. A funcionalidade de registro foi implementada de forma integrada com o Firebase, onde é criado um registro com o e-mail e senha no Firebase, e outro no Firestore com nome, e-mail, posição, aceite de termos de uso e *token* criptografado para controle de acessos do *back-end*, abordado com mais detalhes nas seções seguintes. A partir da figura 12 é possível identificar o resultado da tela de cadastro.

Figura 12 - *Design* da interface de cadastro

Fonte: elaborada pelo autor.

Durante o processo de desenvolvimento, foi buscado garantir que o formulário de cadastro fosse intuitivo e dinâmico, trazendo naturalidade no momento de preenchimento. Além disso, foi implementada a validação de campos, garantindo a integridade dos dados, evitando registros inválidos e promovendo a segurança do sistema desde o início do processo de criação de contas.

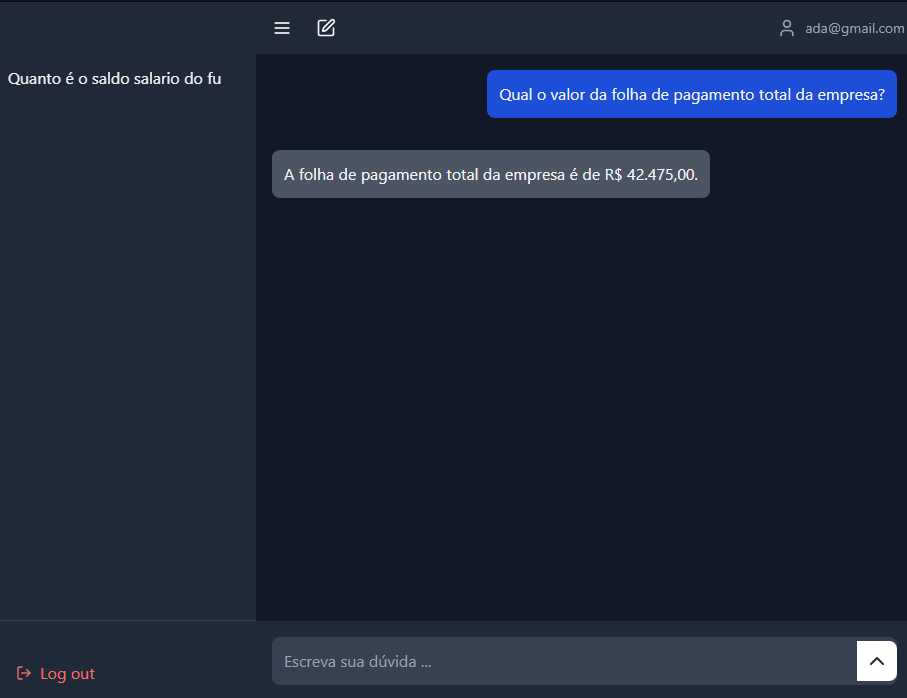
É importante mencionar que durante o processo de criação de novos usuários, é gerado um *token* utilizando a palavra chave funcionário, que, ao ser criptografado e armazenado, assegura o controle de acesso a informações sensíveis. Esse processo será detalhado em seções posteriores.

Inicialmente, todos os novos registros são classificados como funcionários padrão. No entanto, essa posição pode ser alterada pelo administrador geral da aplicação ou com posições hierárquicas superiores.

**5.4 INTERFACE DO ASSISTENTE APLICANDO CONCEITOS DO *CHATGPT***

Com o avanço no desenvolvimento e parametrização da interface, foi implementada a criação de elementos complementares para a seção principal do *chat* de perguntas. Seguindo um conceito já adotado pelo *ChatGPT* e visando uma experiência mais acessível e familiar para o usuário, foi implementado um menu lateral expansível, que armazena o histórico de conversas anteriores. Nesse menu, cada tópico é representado pela primeira pergunta feita na respectiva sessão, permitindo que o usuário acesse facilmente suas dúvidas anteriores, respondidas ou não, e ele pode ser acessado a qualquer momento pelo usuário que realizou as mesmas, assim criando dinamismo na transição de perguntas que os usuários apresentam.   
 Além disso, outros aspectos semelhantes ao modelo da *OpenAI* são: exibe o e-mail do usuário logado, permite o *logout* dos perfis e possibilita a criação de novas sessões de *chat*. Essa última funcionalidade também se integra ao menu lateral, criando uma experiência de navegação intuitiva e coesa. Na figura 13, são notáveis estas características**.**

Figura 13 - Interface do assistente



Fonte: elaborada pelo autor.

**5.5 PAINEL DE CONTROLE DE ADMINISTRADORES**

Como mencionado anteriormente, a aplicação do assistente conta com uma interface de controle, o painel de administração, onde pode ser acessado de forma dinâmica pelos usuários com a devida permissão atribuída no momento de criação da aplicação. Esse painel permite que administradores e gestores gerenciem permissões de acesso dos usuários com a classificação de funcionário, possibilitando o ajuste das funções e acessos conforme necessário.

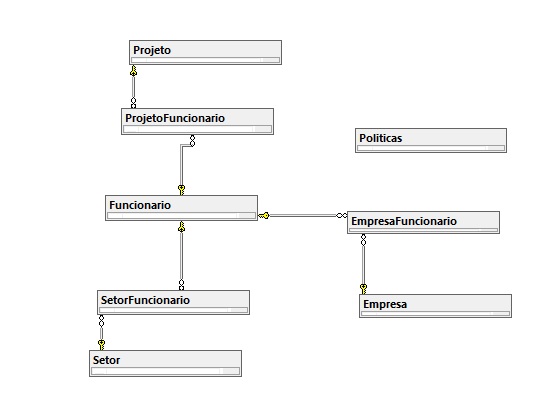
Dessa forma, o painel de administração desempenha um papel essencial na gestão e no controle de acessos ao *chat* principal, definindo como as informações são demonstradas aos funcionários, reforçando a segurança e a organização da plataforma.

**5.6 DESENVOLVIMENTO DE BANCO DE DADOS**

Considerando a abordagem do projeto de ser utilizado em um ambiente empresarial com foco no setor de recursos humanos, o primeiro passo foi a criação de um banco de dados que pudesse simular, de maneira aproximada, uma base utilizada por esse setor, e que contivesse toda a fonte de informação utilizada no decorrer do desenvolvimento.

O modelo exemplificado na figura 14 a seguir foi a versão final definida para o conjunto de dados.

Figura 14 - Modelo do banco de dados



Fonte: elaborada pelo autor.

Como pode ser observado, o elemento central definido no conjunto de dados foi a tabela funcionário, visto que a ideia principal está centralizada em utilizar e trabalhar com os dados dos funcionários e como eles se relacionam com as demais informações.

**5.6.1 TABELAS UTILIZADAS**

Como mencionado acima, a tabela funcionário representa o item principal do conjunto de dados, pois é nela que existem todas as relações e referências com as demais tabelas. Além disso, sua estrutura é composta por dados que envolvem números de documentos, como CPF ou RG, valores de salário, características como raça, religião, orientação sexual, entre outras informações, as quais, indiscutivelmente, não podem ser fornecidas de forma indiscriminada, visto que são classificadas como dados pessoais sensíveis (Santos, 2021) e, no caso de dados de pagamentos, salários entre outros relacionados à empresa, pode ocorrer a exposição desnecessária do trabalhador, caso sejam mal utilizados.

Vale ressaltar que as informações utilizadas em toda a dinâmica do funcionário foram levantadas com base no artigo de Santos (2021), o qual aborda de maneira mais específica a utilização de dados considerando um contexto de trabalho, ou seja, considerando a relação empregado e empregador, e como as informações podem ser utilizadas em cada situação.

Ademais, de maneira a complementar a quantidade de informações, foram criadas as tabelas de projeto, setor e empresa, que serão melhor discutidas a seguir.

De modo a diversificar a base de dados, foi implementada a tabela de empresas. Esta tabela conta com informações básicas sobre diferentes empresas complementares ou mesmo parceiras que podem ser geridas pelo mesmo setor de recursos humanos, como dados sobre nome fantasia, capital social, natureza jurídica e situação. Assim como a tabela de funcionário, esta tabela possui relações por meio de tabelas intermediárias, de modo que se possa obter dados sobre funcionários ligados a uma determinada empresa em questão.

A tabela de projetos possui informações referentes a todos os projetos realizados, sejam eles projetos já finalizados ou mesmo projetos em andamento. Os dados contidos nessa tabela englobam características como nome, custo e datas de início e fim. Ela possui relação direta com a tabela de funcionários, por meio da tabela intermediária denominada projetoFuncionario.

Dessa forma, é possível obter informações referentes aos projetos de interesse ou ainda fazer levantamentos de funcionários trabalhando em diferentes projetos de maneira a facilitar o processo de gestão e alocação de pessoas para as diversas atividades. Outrossim, a tabela de setor foi desenvolvida para adicionar informação e permitir a correlação dos dados de funcionários e empresas com diferentes setores.

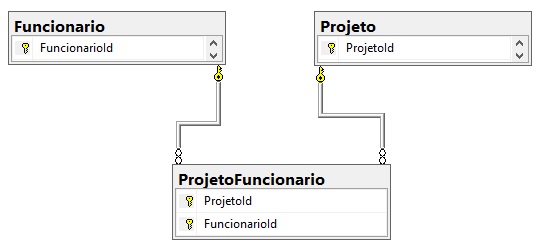
A tabela de políticas foi desenvolvida pensando na utilização direta de um funcionário, como um meio pontual de obter informações simples sobre como funcionam determinadas políticas da empresa, como férias, folgas, entre outros. Portanto, os dados contidos nessa tabelas englobam as diferentes maneiras que a empresa aborda suas políticas.

De modo simplificado, o processo de desenvolvimento do banco de dados se deu por meio da execução de *scripts* padrões da linguagem *MS SQL Server*, definidos como *scripts* de *Data definition language* (*DDL*), que consistem em comandos de criação de objetos.

Ademais, esses *scripts* foram utilizados em conjunto com *scripts* de *Data manipulation language* (*DML*), que configuram instruções de manipulação dos dados com métodos de inserção e atualização.

Além dos *scripts* utilizados, outro ponto destacável da implementação da base de dados é a criação de tabelas intermediárias que permitiram a correlação entre duas ou mais tabelas, possibilitando assim o relacionamento de dados com cardinalidade classificada como NxN (N para N, ou muitos para muitos), onde vários registros de uma tabela podem ser vinculados a vários registros de outra tabela. Um exemplo desse tipo pode ser observado na figura 15 abaixo, que demonstra a conexão da tabela de funcionários com a tabela de projetos sendo realizada por uma tabela intermediária denominada projetoFuncionario, que possui as chaves (informação de identificação) de cada uma das duas tabelas.

Figura 15 - Relacionamento de tabelas com tabela intermediária



Fonte: elaborada pelo autor.

**5.7 PREPARAÇÃO DO AMBIENTE DE *BACK-END***

O primeiro passo da definição foi a criação de uma instância da aplicação Flask, de modo a informar onde está sendo realizada a execução do *framework* e possibilitar a localização dos recursos relacionados à aplicação.

Além disso, foi utilizado o pacote *flask-cors* para habilitar o suporte *Cross-Origin Resource Sharing* (*CORS*) na aplicação definida pela instância criada, de modo a permitir a comunicação entre os diferentes domínios utilizados.

Complementando a definição do ambiente, foi criada uma rota de *API* em Flask com método POST cuja função é receber e processar dados de requisição do usuário fornecidos pelo *front-end* em formato *JavaScript Object Notation* (*JSON*). Primeiramente, os dados são coletados e é feita a extração da pergunta feita pelo usuário, a requisição é então enviada para os *Agents,* que serão melhor discutidos nas próximas seções, para que a processem e definam uma resposta apropriada, a qual será retornada diretamente para o *front-end*. Vale ressaltar que no *header* da rota foram definidos diferentes *cabeçalhos* com o intuito de diminuir vulnerabilidades e aumentar a segurança. Essas definições serão melhor abordadas na seção de segurança.

Além disso, também foi implementado o *decorator after\_request* na aplicação, com o intuito de possibilitar a remoção de informações do cabeçalho da requisição da *API*. Essa implementação também será melhor detalhada na seção de segurança no final do capítulo.

**5.8 CONEXÃO DO BANCO COM O AMBIENTE EM PYTHON**

Posteriormente à especificação, criação e carregamento do banco de dados, foi iniciado o processo de conexão da base com o ambiente em python, para que fosse possível o acesso e manipulação dos dados por meio do *back-end*.

Primeiramente, foi realizada a importação do módulo *SQLDatabase* diretamente da biblioteca do *LangChain*, esse módulo funciona como uma abstração do *SQLAlchemy.*

O *SQLAlchemy* consiste em uma biblioteca python de mapeamento de objetos relacionais, que permite a conexão com bancos de dados SQL de forma simples.

Resumidamente, o módulo *SQLDatabase* realiza todo o processo de conexão utilizando as ferramentas presentes no *SQLAlchemy*, necessitando apenas de uma *string* de referência ao banco ao qual se deseja conectar. A *string* é composta por 4 elementos principais, que são: o sistema de gerenciamento (SGBD), o servidor, o banco de dados e o *driver* utilizado no banco.

De forma explicativa, os componentes da *string* são definidos por:

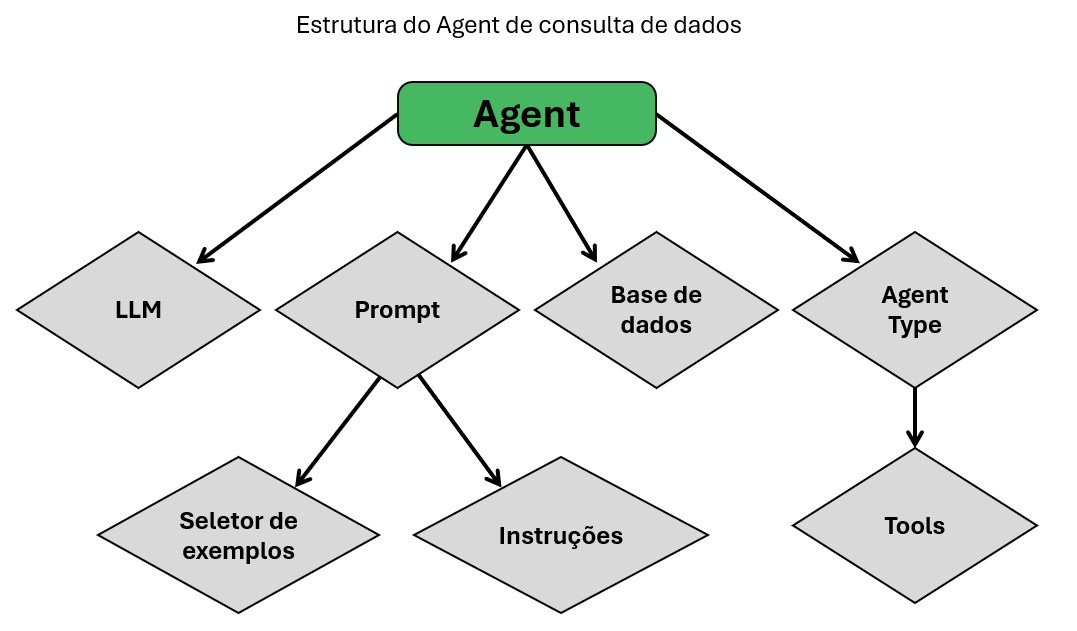
* Sistema de gerenciamento: consiste no *software* utilizado para gerenciamento da base de dados;
* Servidor: servidor o qual a base de dados está hospedada;
* Banco de dados: banco de dados utilizado pela aplicação;
* *Driver*: estabelece o protocolo de comunicação com a base de dados.

**5.9 CRIAÇÃO DO *AGENT* BASEADO NO MODELO DE LINGUAGEM**

Estabelecida a conexão com o banco de dados, o próximo passo foi a criação do *Agent* que realiza as consultas e manipula as informações na base de dados, ou seja, o elemento central de comunicação com o usuário final.

Primeiramente foi realizada a importação do método *create\_sql\_agent* da biblioteca *LangChain*. Basicamente, esse método permite a implementação de um *Agent* focalizado em utilizar um conjunto de dados SQL. Sua implementação exige o fornecimento de alguns parâmetros, os quais, no contexto do projeto, foram o *large language model*, a base de dados, o *prompt* e o tipo de agente. A figura 16 ilustra a estrutura e os componentes utilizados, que serão melhor discutidos a seguir.

Figura 16 - Modelo de estrutura do *Agent*



Fonte: elaborada pelo autor.

**5.9.1 DEFINIÇÃO DO *LARGE LANGUAGE MODEL***

O primeiro parâmetro fornecido para o método de criação do *Agent* foi o *large language model* propriamente dito.

Para isso, foi necessária a importação do método *ChatOpenAI* da biblioteca *LangChain,* pois é por meio dele que se define um modelo de linguagem que será utilizado por meio do consumo da *API* da *OpenAI,* a qual foi utilizada no desenvolvimento da aplicação. Assim como os demais, esse método exige o fornecimento de alguns parâmetros, sendo que os utilizados no projeto foram os parâmetros de modelo, temperatura e Chave de *API*.

O parâmetro de modelo ou *model* diz respeito ao modelo que será utilizado. O modelo escolhido foi o *gpt-3.5-turbo*, visto que este apresentou comportamento satisfatório durante a primeira realização de testes de verificação ao realizar consultas básicas, e ainda exige poucos créditos de consumo da *API* da *OpenAI*.

Vale ressaltar que para utilização de qualquer modelo e *API* da *OpenAI* é necessária a criação de uma conta no ambiente de *API* próprio da empresa, para posterior obtenção de uma chave e a possibilidade de inserir créditos para consumo do serviço.

O parâmetro *temperature* é aquele que define o nível de aleatoriedade na geração das respostas do modelo, ele pode ser definido num intervalo de 0 a 1, sendo que 0 representa nenhuma aleatoriedade e 1 a máxima aleatoriedade. Caso o parâmetro seja definido em 1, uma mesma pergunta pode gerar inúmeras respostas diferentes, enquanto que a definição em 0, gera a mesma resposta para a mesma pergunta. No contexto do projeto, foi optado por não explorar a geração de respostas de forma aleatória ao definir o parâmetro como 0, com o intuito de manter constante o comportamento do modelo e simplificar sua utilização e previsão.

Por fim, o parâmetro de chave de *API* é obrigatório e necessita do fornecimento de uma chave válida para consumo do serviço.

**5.9.2 DEFINIÇÃO DO *PROMPT* DO *AGENT***

Como já mencionado, o *prompt* consiste na requisição completa que será passada para o modelo, contendo o conjunto de instruções, os meios de comportamento e a requisição feita pelo usuário.

Para a criação do *prompt* estruturado denominado *full\_prompt,* foi necessária a importação do método *ChatPromptTemplate.from\_messages*, que permite a customização e estruturação de *prompts* para assistentes de forma personalizada.

O consumo desse método foi executado por meio do fornecimento de uma lista como parâmetro, a qual contém 3 itens, sendo eles o modelo de *prompt*, a variável de entrada de usuário e um marcador.

A variável de entrada consiste na pergunta feita pelo usuário que será incluída no *prompt*. De maneira simples, a requisição do usuário será armazenada e passada para o modelo com a devida identificação para posterior interpretação.

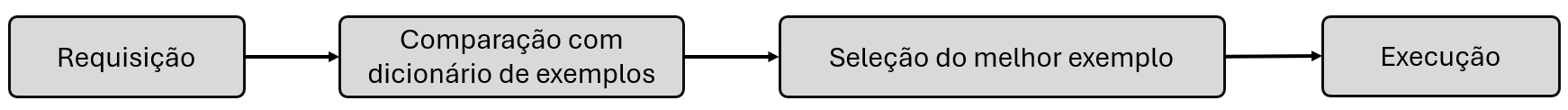
O marcador consiste em uma definição implementada pelo método *MessagesPlaceholder,* pois é por meio dele que se torna possível a gravação de mensagens anteriores trocadas pelo modelo.

Finalmente, o modelo de *prompt* consiste na utilização do método *SystemMessagePromptTemplate*, também da biblioteca do *LangChain*, que recebe o conjunto bruto de instruções passadas para o *Agent* em um primeiro momento. No contexto da aplicação, este conjunto foi elaborado e armazenado na variável *few\_shot\_prompt* passada para o método. Os detalhes da implementação desse conjunto serão discutidos a seguir.

**5.9.2.1 CRIAÇÃO DO CONJUNTO DE INSTRUÇÕES DO *PROMPT***

Considerando a necessidade do *Agent* de realizar consultas mais complexas referentes ao setor de recursos humanos, como por exemplo a interpretação do contexto das perguntas e a realização de cálculos específicos com os dados obtidos, e ainda, tomando como base o projeto de Izacardi *et al.* (2022), a técnica de implementação do conjunto de instruções escolhida foi a denominada *Few-Shot Prompt.*

A técnica *Few-Shot Prompt* caracteriza-se pela inserção de exemplos junto ao *prompt* que será passado para o modelo. Esses exemplos direcionam de forma objetiva de que modo a resposta deve ser formada ou ainda como interpretar o contexto de uma requisição de maneira mais coerente e concisa com o que se espera. A figura 17 a seguir exemplifica o fluxo de um *prompt* baseado em *few-shot*.

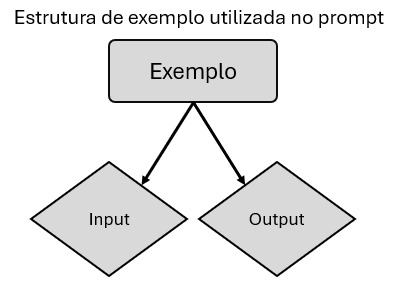
Figura 17 - Fluxo do *prompt* baseado na técnica *few-shot*

Fonte: elaborada pelo autor.

O método da biblioteca *LangChain* que possibilita a implementação da técnica é o *FewShotPromptTemplate*. Para a utilização desse método na aplicação, foram passados 5 parâmetros diferentes, sendo eles: o seletor de exemplos (ou *exemple\_selector*), o *prompt* específico dos exemplos (ou *exemple\_prompt*), uma lista contendo as variáveis de entrada, um prefixo e um sufixo contendo instruções para o *prompt.*

O parâmetro do *prompt* de exemplos consiste em informar a estrutura a qual os exemplos serão passados para o método. A figura 18 a seguir exemplifica a estrutura utilizada no projeto.

Figura 18 - Estrutura de exemplo utilizada no *prompt*



Fonte: elaborada pelo autor.

As variáveis de entrada do modelo são definidas diretamente na lista passada para o parâmetro de variáveis. No contexto da aplicação, as variáveis definidas foram a entrada do usuário (*input*) e a quantidade de registros a serem retornados do banco de dados (*top\_k*).

Além disso, os parâmetros de prefixo e sufixo consistem num conjunto de instruções fornecidas ao modelo que serão seguidas antes da resolução da requisição e depois da resolução, respectivamente. Dentre as principais instruções definidas no *prompt*, destacam-se:

* A definição do papel do modelo como um *Agent* que comunica-se com uma base de dados SQL;
* A necessidade de otimização de cada uma das consultas realizadas;
* A utilização de apenas ferramentas e dados relevantes;
* O tratamento de erros e múltiplas execuções, caso necessário;
* Nunca executar comandos de DML que envolvam inserção, atualização ou exclusão de dados ou objetos do banco de dados;
* Caso não chegue em uma resposta baseada na base de dados, retornar que a resposta não pôde ser elaborada;
* Utilizar o conjunto de exemplos como base na geração das respostas.

Um ponto importante a ser mencionado é a presença das instruções que controlam a manipulação de dados do banco e combatem a injeção de *scripts* SQL, contribuindo com a segurança e o controle das requisições recebidas e executadas pelo *Agent.* A eficiência de tais instruções será melhor abordada no Capítulo 6.

Por fim, o parâmetro do seletor de exemplos configura a peça principal da aplicação da técnica, portanto, será abordado de forma mais detalhada a seguir.

**5.9.2.2 CRIAÇÃO DO SELETOR DE EXEMPLOS**

Primeiramente, para possibilitar a criação do seletor de exemplos, foi necessária a criação de um dicionário de exemplos baseado na estrutura de exemplos apresentada anteriormente na seção 5.9.2.1. Os exemplos foram criados levando em consideração os principais cálculos que poderiam ser solicitados para o *Agent*, considerando seu grau de complexidade.

Partindo do dicionário de exemplos definido, a implementação do seletor de exemplos se deu por meio do método *SemanticSimilarityExampleSelector,* o qual recebe como parâmetro o conjunto de exemplos, um método de comparação por meio de *embeddings*, um vetor de *embeddings* e uma quantidade *k* de exemplos a serem utilizados, bem como a entrada do usuário.

No processo de criação do seletor vale ressaltar o trabalho em conjunto realizado pelo método de comparação de *embeddings* com o vetor de *embeddings*. O modelo comparador utilizado foi o da *OpenAI* por meio do consumo de sua *API*. Esse modelo utiliza um vetor previamente montado da empresa Meta, denominado *FAISS*. O conjunto com o vetor permite o processo de comparação da solicitação do usuário com o conjunto de exemplos fornecidos, de modo a buscar aquele que mais se aproxima da requisição recebida, com o intuito de gerar a resposta mais coerente possível.

**5.9.3 DEFINIÇÃO DO *AGENT TYPE***

Por fim, o último parâmetro utilizado na criação do *Agent* foi o *agent\_type*, que consiste no tipo de agente que será implementado, de forma objetiva, esse parâmetro define o caminho lógico que será tomado durante toda tomada de decisão do *Agent.*

Considerando a abordagem do projeto, o tipo definido foi o *tool-calling*, o qual permite que o modelo decida por meio de seu discernimento qual a melhor ferramenta utilizar, dentro de conjunto pré-determinado de ferramentas, em situações específicas as quais for apresentado.

**5.9.3.1 UTILIZAÇÃO DE *TOOLS***

*Tools* dentro da biblioteca *LangChain* representam funcionalidades poderosas que podem acrescentar e potencializar a execução dos *agents*, elas influenciam na tomada de decisão e são caracterizadas por um conjunto pré-definido de instruções que são interpretadas pelo *Agent* e que serão utilizadas caso ele as considere úteis em determinado contexto.

**5.10 CONTROLE DE ACESSO E SEGURANÇA NA EXECUÇÃO DO *AGENT***

Outro ponto de suma importância no processo de desenvolvimento foi o estabelecimento de medidas que pudessem controlar o acesso do usuário ao sistema e consequentemente às informações contidas nele, a seguir essas medidas serão abordadas de maneira mais detalhada.

**5.10.1 IMPLEMENTAÇÃO DE TOKEN DE USUÁRIO**

O *token* mencionado anteriormente em outras seções foi implementado de modo a ser utilizado em todo primeiro contato do usuário com a aplicação. Ele funciona de maneira simples, após a realização do cadastro, o *token* é criptografado pelos métodos do *libsodium*, já abordado no Capítulo 4, e armazenado no banco de dados. Em outro momento, quando o usuário faz o *login,* o *token* criptografado presente no banco é armazenado no contexto da aplicação e, no momento que é feita uma pergunta ao assistente, esse *token* é enviado junto à requisição, onde é descriptografado e utilizado no momento que a aplicação define se a resposta poderá ser obtida ou não.

**5.10.2 IMPLEMENTAÇÃO DE *AGENT* INTERPRETADOR PARA CONTROLE DE ACESSO**

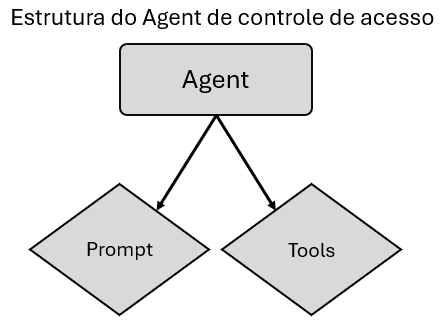
De modo a complementar a utilização do *token* e dinamizar o controle de acesso baseado no tipo de perfil, visto que diferente de gestores, perfis com acesso de funcionário não possuem acesso direto aos dados restritos, foi desenvolvido um segundo *Agent* complementar*,* o qual possui a única tarefa de interpretar a requisição feita pelo usuário e determinar se a resposta pode ser fornecida ou não baseando-se na análise do contexto da pergunta, ou seja, caso se faça necessário o fornecimento de informações sensíveis as quais um usuário com acesso de funcionário não pode acessar, o papel desse *Agent* é justamente impedir esse acesso.

De maneira simples, após a análise da requisição, esse *Agent* retorna uma *string* que define para a aplicação se é permitida ou não a execução do *Agent* que consulta a base de dados, abordado anteriormente.

Vale ainda ressaltar que esse *Agent* complementar é apenas executado quando um funcionário realiza uma requisição, visto que o controle de acesso não se aplica ao perfil de gestor, o qual possui acesso direto aos dados.

Comparado ao *Agent* anterior, este se apresenta com implementação um pouco mais simples, visto que possui funções mais diretas. A figura 19 ilustra a estrutura definida. Os detalhes de seu desenvolvimento serão abordados a seguir.

Figura 19 - Modelo de estrutura do *Agent* de controle



Fonte: elaborada pelo autor.

**5.10.2.1 CRIAÇÃO DE *TOOL* DE TRATAMENTO DE TEXTO**

O primeiro passo foi a criação de uma *tool* com a função de tratar um texto de entrada de modo a facilitar a interpretação do *Agent*.

Para isso, foi utilizado o método *tool* da biblioteca *Langchain*, o qual permite a definição customizada de ferramentas que podem ser utilizadas por modelos.

Sua implementação assemelha-se à criação de um método em python, porém com a tag *tool* acompanhando o método. Uma *tool* consiste em um conjunto de direcionamentos que são fornecidos para o A*gent*. Todas as instruções passadas para a ferramenta consistem em diferentes tipos de tratamento de texto, seguidas de um processo de separação de palavras, que são posteriormente utilizadas pelo modelo para facilitar a análise do contexto por meio das palavras isoladas.

**5.10.2.2 CRIAÇÃO DO *PROMPT* DE CONTROLE**

A criação do *prompt* do *Agent* de controle de acesso utilizou métodos semelhantes aos já mencionados na implementação do primeiro *Agent*, como o *ChatPromptTemplate* e o *MessagesPlaceholder*, entretanto, o conjunto de instruções foi definido com o intuito de instruir o modelo a interpretar a requisição do usuário e definir uma resposta direta. Dentre as principais instruções definidas, podem ser citadas:

* A definição do papel do *Agent*;
* A função de determinar se uma requisição possui relação com temas pré-definidos como salário, pagamentos, entre outras informações com acesso restrito.
* O fornecimento de uma resposta simples, permitindo ou bloqueando o acesso baseado na requisição.

Como pode ser inferido, as instruções são diretas e, além disso, o modelo também é instruído a responder de forma simples, apenas informando se permite o acesso ou não àquela informação. Vale destacar também que assim como o *Agent* de dados abordado anteriormente, o *Agent* de controle também possui uma instrução como medida para combater a injeção de *scripts* e sua possível execução pela aplicação.

**5.10.2.3 CRIAÇÃO DO *AGENT***

A estrutura final do *Agent* de controle foi definida por meio de uma estrutura construída de modo a combinar todos os objetos já mencionados. Por meio dela são definidos como serão processados os dados de entrada e as etapas intermediárias do *Agent*.

Para isso, foram definidos o *input,* que consiste na entrada do usuário que foi recebida, e foi utilizada a propriedade *agent\_scratchpad* a qual armazena informações sobre os processos e interações intermediários realizados pelo *Agent* durante o processo de aquisição da resposta.

Ademais, também foi utilizada a propriedade *llm\_with\_tools,* queconsiste na união entre as *tools* nativas presentes nos modelos da *OpenAI* em conjunto com a *tool* de tratamento de texto definida previamente, de modo que todas essas ferramentas possam ser utilizadas pelo modelo.

Além disso, foi utilizado o método *OpenAIToolsAgentOutputParser*, para garantir que as respostas fornecidas pelo modelo sejam devidamente interpretadas e processadas pelo *Agent*.

Finalmente, para realizar a execução do *Agent* foi utilizado o método *AgentExecutor*, o qual age como uma objeto de abstração de um *Agent*. Esse método recebe como parâmetro o *Agent* criado e as ferramentas que poderá utilizar.

**5.10.3 IMPLEMENTAÇÃO DE *TAGS* E PARÂMETROS *HEADER* DA REQUISIÇÃO**

Com o intuito de aumentar a segurança da aplicação contra vazamento ou má utilização dos dados do banco ou informações do próprio projeto, e utilizando como base a pesquisa de Yoshioka e Silva (2023), foram implementados no ambiente de *back-end* meios de reduzir possíveis vulnerabilidades presentes na execução dos algoritmos.

De maneira prática, os componentes utilizados foram:

* *Content-Type*: Este cabeçalho indica basicamente o tipo de conteúdo da resposta. No contexto do projeto, *application/JSON* informa ao cliente que a resposta contém dados no formato *JSON*. Isso é essencial para que o cliente saiba como processar os dados recebidos, principalmente em APIs.
* *Content-Security-Policy*: utilizado para implementar políticas de segurança relacionadas ao carregamento de recursos da aplicação, permitindo que apenas recursos do mesmo domínio sejam carregados, e impedindo que sejam utilizados *frames* no carregamento do *site*, combatendo ataques que utilizam o carregamento de recursos na aplicação.
* X-*Frame-Options:* de modo complementar ao anterior, impede que haja o carregamento de conteúdo sobreposto à aplicação.
* X-*Content-Type-Options*: Impede que navegadores tentem adivinhar o tipo de mídia utilizada como recurso na aplicação, respeitando o tipo definido no *Content-Type* e combatendo injeções de código*.*

Ademais, com o intuito de remover informações que pudessem acabar expostas por meio do servidor, foi utilizado o *decorator* *after\_request* na definição da rota, de modo que qualquer dado presente no servidor seja removido antes do fornecimento da resposta pelos métodos, impedindo a obtenção de dados sobre as tecnologias utilizadas no processo de desenvolvimento.

**6 RESULTADOS**

O capítulo a seguir abordará a apresentação e análise dos resultados obtidos no decorrer do desenvolvimento do projeto.

**6.1 CARACTERIZAÇÃO DA INTERFACE**

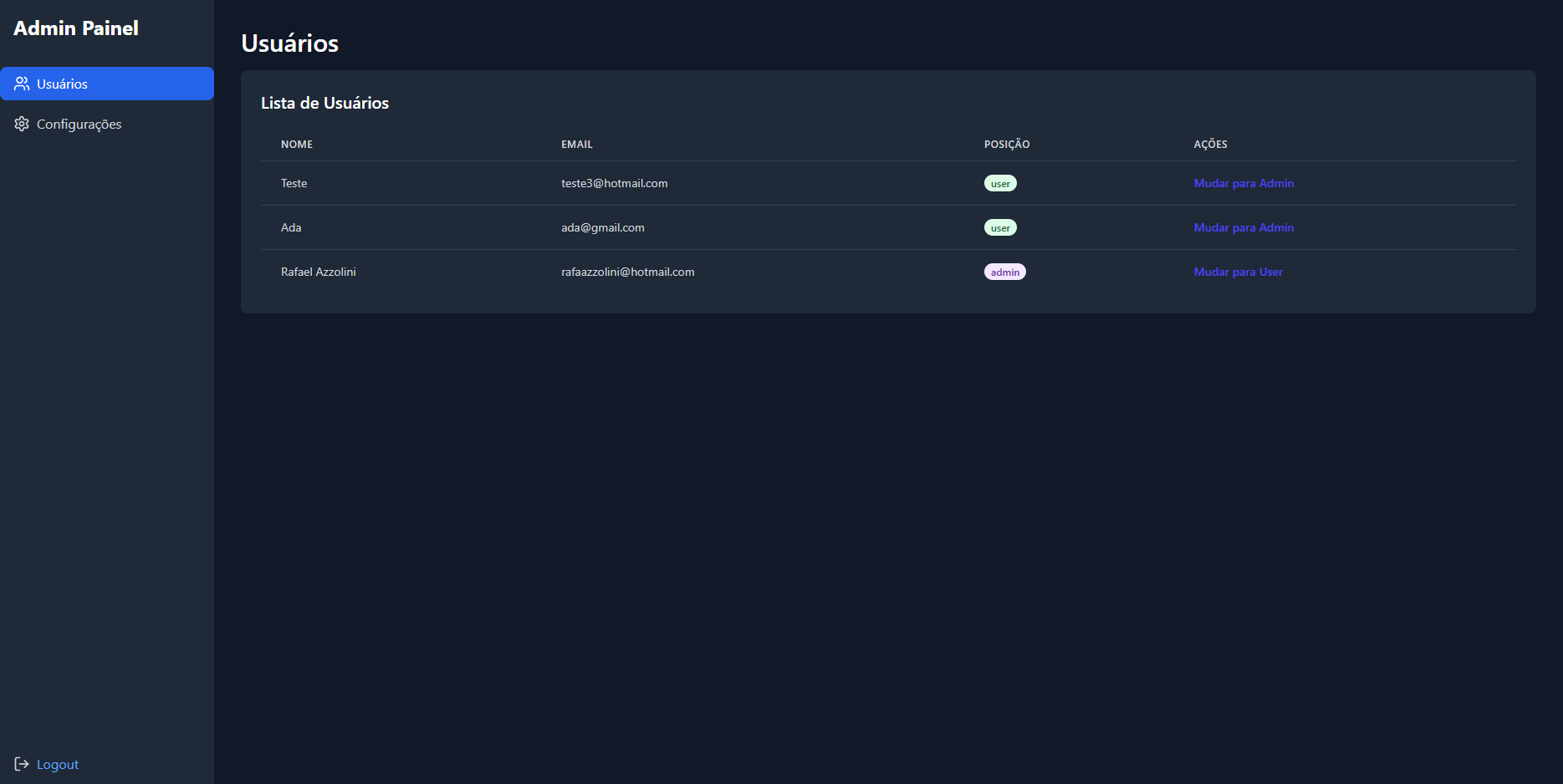
Durante o desenvolvimento das interfaces que compõem o núcleo do projeto, buscou-se seguir um padrão semelhante ao de modelos já estabelecidos no mercado, como o *ChatGPT*, devido à sua interface intuitiva e à familiaridade do público em geral com esse formato. A escolha desse padrão reflete a necessidade de criar uma experiência de uso acessível e eficiente, especialmente considerando que o projeto visa ser um assistente para o setor de Recursos Humanos, destinado a funcionários e gestores.

A interface foi projetada com foco em formatos de tela, paleta de cores e disposição das informações, buscando manter uma organização clara e funcional.

Buscando a familiaridade de elementos, como o *prompt* para inserção de perguntas, o espaço destinado às respostas, e uma lista com os *chats* anteriores, além da opção para iniciar uma nova conversa.

O funcionamento da interface varia conforme o perfil do usuário que efetua o acesso, podendo ser um funcionário comum, um gestor ou um administrador. Essa configuração foi adotada para estabelecer diferentes níveis de permissão e controle de informações de acordo com a hierarquia da organização. A configuração de perfis é definida inicialmente por um administrador e é automatizada para novos usuários, garantindo o controle adequado de acessos.

Na Figura 20, é possível visualizar o painel de administração, que permite o gerenciamento de acessos e permissões no sistema. A interface deste painel possibilita alternar entre diferentes funções e níveis de acesso para cada usuário.

Figura 20 - Interface de administração de usuários

Fonte: elaborada pelo autor.

Além disso, com o progresso no desenvolvimento das interfaces, optou-se por adotar o *framework Tailwind CSS*, que fornece classes utilitárias para estilização. O uso do *Tailwind* otimizou o processo de desenvolvimento, facilitando a criação de componentes com design intuitivo e ágil. Assim, foi possível aplicar combinações eficazes de estilos que melhoraram a navegação, tornando-a mais clara e informativa. Esse padrão foi aplicado também às telas de login e registro de usuários, que passaram a seguir o mesmo padrão de usabilidade e rapidez no desenvolvimento.

**6.2 AVALIAÇÃO DOS MODELOS**

A avaliação dos modelos de *Agent* implementados foi realizada diretamente por meio de uma ferramenta nativa da biblioteca *LangChain*, denominada *LangSmith*. Essa solução facilita os processos de teste e validação de aplicações baseadas em LLMs de maneira clara e eficiente, contando também com uma plataforma de monitoramento online.

O principal parâmetro definido para a tarefa de avaliação foi a acurácia de respostas do modelo. De maneira simples, essa métrica é calculada seguindo 3 passos:

* Criação de *datasets* com valores de *input* e *output;*
* Realização de requisições para o modelo;
* Utilização de métodos de avaliação das respostas.

O processo de criação de *datasets* consiste na definição de uma coleção de dicionários representados pelas chaves *input* e output. As chaves *input* são vinculadas às diversas perguntas e requisições que podem ser feitas para o *Agent*, ou seja, as requisições enviadas pelo usuário, enquanto as chaves *output* contém as respectivas respostas que espera-se que sejam geradas pelo assistente. A figura 21 a seguir ilustra um exemplo reduzido de um *dataset* de validação.

Figura 21 - Exemplo de *dataset* utilizado

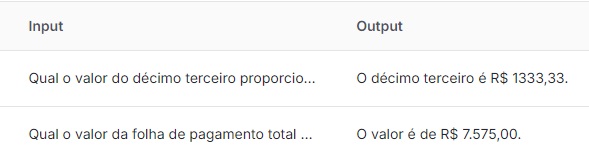


Fonte: elaborada pelo autor.

A segunda etapa consiste na realização de requisições baseadas no *dataset*. Na execução dessa etapa, todos os valores de *input* definidos no *dataset* pré-selecionado são coletados e enviados na forma de requisição para o LLM (no caso deste projeto, para o *Agent*). Posteriormente, todas as respostas geradas de acordo com cada requisição são armazenadas para utilização na fase final do processo.

O último passo é a aplicação de métodos de validação em conjunto com um LLM para calcular a acurácia baseada nos dados obtidos. Nessa etapa, os dados de resposta armazenados no processo anterior e os dados de *input* e *output* do *dataset* são passados diretamente para um LLM, para que este gere um valor de acurácia ao comparar a resposta esperada (*output* no *dataset*) com a resposta gerada pela requisição. A figura 22 a seguir ilustra como são representados os valores do *dataset.*

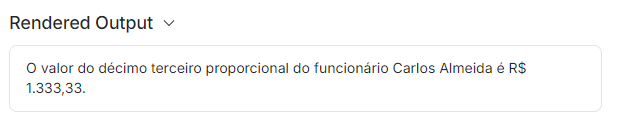
Figura 22 - Valores do *dataset* na plataforma



Fonte: elaborada pelo autor.

De maneira complementar à figura acima, a figura 23 abaixo ilustra o valor de resposta da requisição.

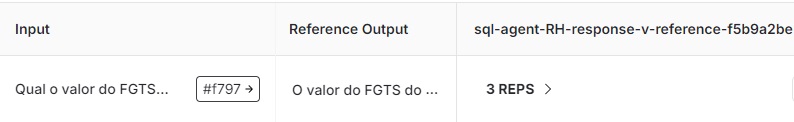
Figura 23 - Valor de resposta da requisição



Fonte: elaborada pelo autor.

Simplificando o processo, na prática, o *Langsmith* pode ser utilizado via codificação ou diretamente *online*. O processo consiste na vinculação de três informações na plataforma, primeiro os exemplos de *input,* e em seguida as respostas esperadas contidas nos respectivos *outputs* relacionados, por fim, após a chamada do *Agent,* é armazenada a resposta gerada por ele, resultando em um conjunto contendo as três informações. A figura 24 ilustra como as informações são armazenadas e relacionadas na plataforma, sendo elas o *input*, o *output* e a resposta do *Agent*. Nesse exemplo, as respostas para cada *input* foram coletadas três vezes para efeito de exemplificação, por isso eles são armazenadas no dicionário 3 REPS, porém a execução continua a mesma.

Figura 24 - Exemplo de dados relacionados na plataforma



Fonte: *Langsmith,* 2024*.*

Resumidamente, por meio da utilização do *Langsmith* buscou-se abordar uma metodologia moderna por meio da utilização de um modelo de linguagem como complemento na geração de resultados e avaliação dos *Agents.*

Vale ressaltar que o *Langsmith* se configura como uma ferramenta extremamente pioneira, com poucas referências de sua utilização principalmente ao considerar o contexto da academia e da literatura, portanto, para fins de orientação e parametrização, a referência de acurácia utilizado será avaliado com base na mesma métrica adotada no projeto de McAvinue e Dev (2024), a qual, de maneira simples, calcula o valor médio de acurácia obtido com cada um das requisições presentes em um *dataset*, sendo esse valor contido no intervalo entre 0 e 1, de modo que o valor 1 seja o melhor cado, e 0 o pior.

Em seguida, serão abordados de maneira mais detalhada os dados obtidos durante o procedimento de teste da aplicação.

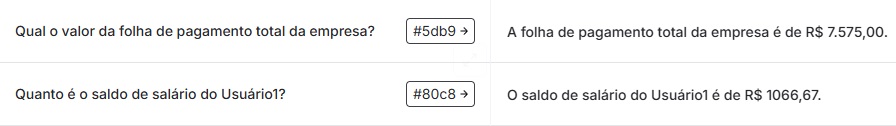
**6.3 TESTES E COLETA DE DADOS**

A seguir serão apresentados e discutidos todos os conjuntos de testes e validações realizados no decorrer do desenvolvimento do projeto. Vale ressaltar que para os testes com requisições, foram utilizados 3 conjuntos de *datasets* contendo em média 10 requisições cada, de forma padronizada.

**6.3.1 TESTE COM REQUISIÇÕES PREVIAMENTE CONHECIDAS**

O primeiro teste realizado foi o teste de validação das respostas fornecidas pelo modelo levando em consideração apenas perguntas às quais ele foi previamente apresentado ou orientado. A Figura 25 a seguir mostra o modelo utilizado em todos os 3 *datasets* aplicados na validação, contendo requisições já ensinadas ao modelo e suas respectivas respostas.

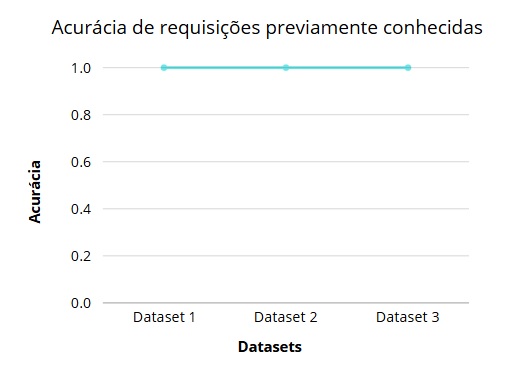
Figura 25 - Exemplo de *dataset* contendo requisições conhecidas



Fonte: elaborada pelo autor.

Nesse sentido, esperava-se que dentre todos os testes, este fornecesse os melhores resultados, considerando a afinidade do modelo com as requisições realizadas. O gráfico da Figura 26 a seguir ilustra os resultados de acurácia obtidos ao utilizar 3 diferentes conjuntos de *dataset* contendo solicitações conhecidas pelo modelo.

Figura 26 - Acurácia de resultados obtidos com requisições previamente conhecidas

****

Fonte: elaborada pelo autor.

Como é possível observar, os resultados demonstram de forma clara o desempenho satisfatório do modelo, visto que a acurácia se manteve estável em 1 para todos os *datasets* utilizados.

**6.3.2 TESTE COM REQUISIÇÕES SIMILARES ÀS PREVIAMENTE CONHECIDAS**

O segundo teste realizado foi o de validação das respostas geradas por requisições semelhantes às requisições já apresentadas, ou seja, os *datasets* utilizados foram preenchidos com perguntas com contextos muito próximos das utilizadas no primeiro teste, porém com alterações ou mesmo parafraseadas. A figura 27 a seguir ilustra exemplos de requisições utilizadas nesse teste.

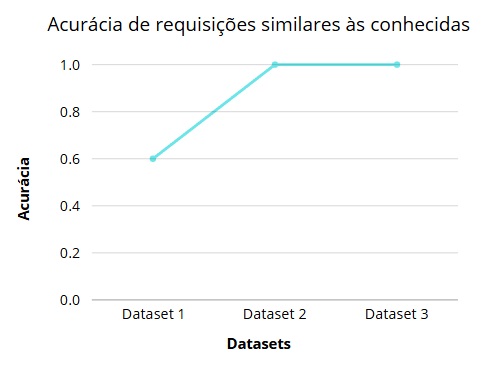
Figura 27 - Exemplo de *dataset* contendo requisições similares



Fonte: elaborada pelo autor.

Considerando a capacidade de interpretação do modelo e as instruções as quais foi orientado, esperava-se que nesse teste os resultados ainda fossem positivos. O gráfico da figura 28 exibe os dados obtidos.

Figura 28 - Acurácia de resultados obtidos com requisições similares



Fonte: elaborada pelo autor.

Por meio da análise do gráfico, destaca-se uma diferença em relação ao teste anterior, visto que houve uma queda de acurácia considerável na avaliação do primeiro conjunto de requisições quando comparado aos demais. Além disso, essa diferença desperta uma atenção maior no entendimento de como a eficácia do modelo pode ser reduzida quando ele é submetido a diferentes requisições, mesmo que similares àquelas as quais ele já conhece e, considerando um cenário real, pode impactar diretamente na adaptação do modelo a distintos cenários.

Entretanto, estabelecendo um paralelo entre os três conjuntos utilizados, e considerando que nos dois últimos a acurácia se manteve estável em 1, pode-se supor que o resultado obtido com o primeiro conjunto sofreu alteração em decorrência de uma maior incidência de requisições embaralhadas no *dataset*, o que não se observa no resultado dos demais.

**6.3.3 TESTES COM REQUISIÇÕES DESCONHECIDAS**

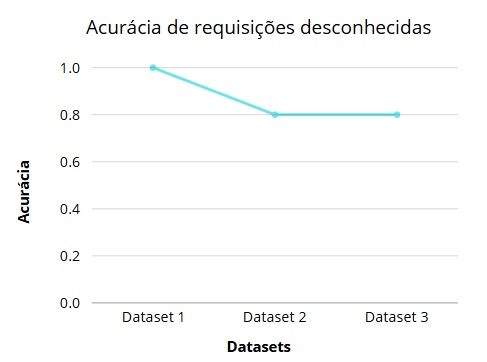
Outro teste realizado em relação ao desempenho do modelo principal abordou a execução de requisições as quais o *Agent* não havia sido instruído ou informado. A figura 29 a seguir exemplifica as requisições utilizadas nos *datasets* para a realização desta validação.

Figura 29 - Exemplo de *dataset* contendo requisições desconhecidas

Fonte: elaborada pelo autor.

Por meio da aplicação deste método, buscou-se entender da melhor forma possível o comportamento de geração de respostas do assistente quando submetido a perguntas e questionamentos de forma mais abrangente e dinâmica. O gráfico da figura 30 exibe os valores de acurácia obtidos de acordo com o *dataset* utilizado.

Figura 30 - Acurácia de resultados obtidos com requisições previamente desconhecidas

****

Fonte: elaborada pelo autor.

Como pode ser interpretado acima, o modelo apresentou resultados relativamente significativos mesmo ao se deparar com solicitações as quais nunca havia sido apresentado. Isso demonstra a capacidade do *Agent* ao se adaptar a diferentes cenários e situações adversas. Além disso, vale destacar que as solicitações utilizadas na validação foram definidas exclusivamente considerando o contexto da base de dados adotada no projeto, em detrimento da abordagem de contextos adversos, os quais não participam ou configuram parte do objeto de estudo.

**6.3.4 TESTE DE ALUCINAÇÃO**

Outro fator importante ao considerar aplicações baseadas em LLMs e a aplicação desenvolvida neste projeto foi a verificação do comportamento do modelo quando submetido a perguntas que não abordam o contexto ao qual ele foi desenvolvido ou às quais ele não possui os dados necessários para a geração de uma resposta coerente. A figura 31 expõe o tipo de requisição utilizado para testar o modelo.

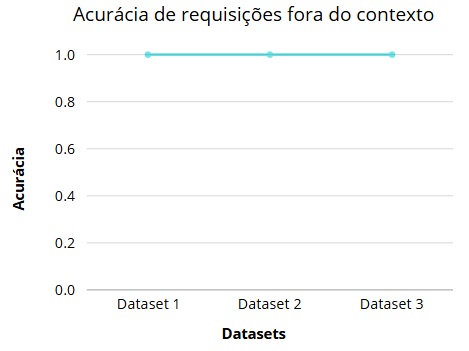
Figura 31 - Exemplo de *dataset* contendo requisições fora de contexto



Fonte: elaborada pelo autor.

Como mencionado em capítulos anteriores, o *Agent* foi instruído a trabalhar apenas com informações as quais forem julgadas em conformidade com a área de estudo abordada em sua implementação. Portanto, o teste de alucinação foi aplicado ao criar *datasets* de validação contendo apenas solicitações que fogem do escopo do projeto, com questionamentos sobre as mais diversas áreas, solicitação de dados os quais ele não possui e até mesmo tentativas de se iniciar uma conversa descontraída com o assistente. O gráfico da figura 32 a seguir ilustra os resultados.

Figura 32 - Acurácia de resultados obtidos com requisições fora do contexto do projeto



Fonte: elaborada pelo autor.

Os resultados exibidos ilustram uma média de acurácia fixa em 1, o que permite concluir que mesmo que de maneira simples, em nenhum dos casos o assistente apresentou qualquer tipo de alucinação ou tentativa equivocada de gerar uma resposta a qual estivesse fora do cenário ao qual foi implementado.

**6.3.5 TESTE DO *AGENT* DE VALIDAÇÃO DE ACESSO À INFORMAÇÃO**

Como visto anteriormente, além do assistente principal, o projeto também conta com um assistente auxiliar cuja função é exclusivamente determinar se o usuário que realiza a pergunta pode ter acesso ao tipo de dado presente na resposta.

Considerando essa abordagem, o teste de *Agent* foi estruturado de modo que os *datasets* fossem carregados com requisições que solicitam dados sensíveis contidos na base de dados, simulando a tentativa de um usuário com acesso restrito, no caso, o de funcionário, estivesse requisitando os dados. A figura 33 ilustra as requisições utilizadas para realização do teste.

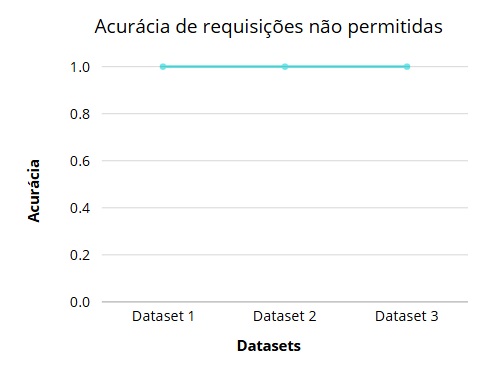
Figura 33 - Exemplo de *dataset* contendo requisições não permitidas



Fonte: elaborada pelo autor.

Dessa forma, esperava-se que todas as respostas fossem bloqueadas indubitavelmente. O gráfico da figura 34 ilustra os dados obtidos.

Figura 34 - Acurácia de resultados com requisições não permitidas



Fonte: elaborada pelo autor.

Ao observar a figura, pode-se concluir que o assistente focalizado no controle de acesso de um determinado perfil a uma informação com maior nível de sensibilidade apresentou resultados de acurácia positivos para os 3 diferentes *datasets* utilizados, demonstrando alta eficácia em sua tarefa e considerável poder de discernimento de modo geral.

**6.3.6 TESTE DE REQUISIÇÕES COM *INJECTION***

Outro teste de grande importância foi de realização de requisições com injeção de *scripts* em linguagem SQL. Como visto anteriormente no Capítulo 5, os *Agents* foram instruídos a nunca consultar o banco ou tentar responder uma requisição a qual fosse detectado qualquer característica de *scripts* de manipulação de dados.

Portanto, para a execução desse teste, os *datasets* foram carregados com requisições contendo *scripts* completos e partes de *scripts* de manipulação de banco ou de tentativas de injeção. A figura 35 exemplifica as requisições utilizadas.

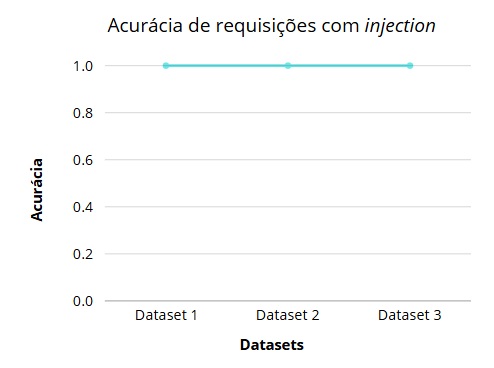
Figura 35 - Exemplo de *dataset* contendo requisições com *injection*



Fonte: elaborada pelo autor.

Como resultado, esperava-se que em cada uma das respostas fosse informado ao usuário a impossibilidade de executar a requisição, pois a mesma encontrava-se inválida. O gráfico da figura 36 ilustra os resultados de acurácia obtidos.

Figura 36 - Acurácia de resultados com requisições contendo injeção



Fonte: elaborada pelo autor.

Como pode ser observado, a média de acurácia sempre em 1 demonstra a capacidade dos *Agents* de detectar a presença de qualquer componente de um *script*, visto que foram utilizadas também partes de *scripts*, e recusar a geração da resposta, seguindo sua definição de instruções.

**6.4 TESTES DE SEGURANÇA E VULNERABILIDADE**

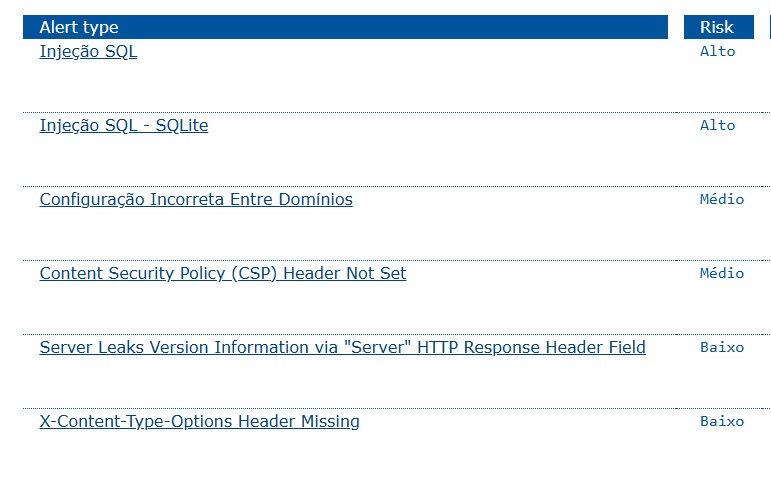
Além dos processos de implementação dos assistentes e suas respectivas validações, outro ponto de suma importância no desenvolvimento do projeto é a validação do sistema quanto ao seu nível de vulnerabilidade e segurança de modo geral.

O método de avaliação de segurança adotado como referência se deu por meio da utilização do software OWASP ZAP (melhor detalhado no Capítulo 4), que consiste, de maneira simples, em uma ferramenta poderosa cuja função é tentar atacar ou invadir diretamente uma aplicação ou sistema, detectando vulnerabilidades e possíveis soluções.

A validação de segurança seguiu a metodologia apresentada no projeto de Alfarizi, Najie, Afif e Ashari (2022), onde são executados os mesmos passos para obtenção dos relatórios de vulnerabilidade e suas respectivas análises.

A figura 37 a seguir representa as vulnerabilidades detectadas de acordo com o risco que representam para uma aplicação.

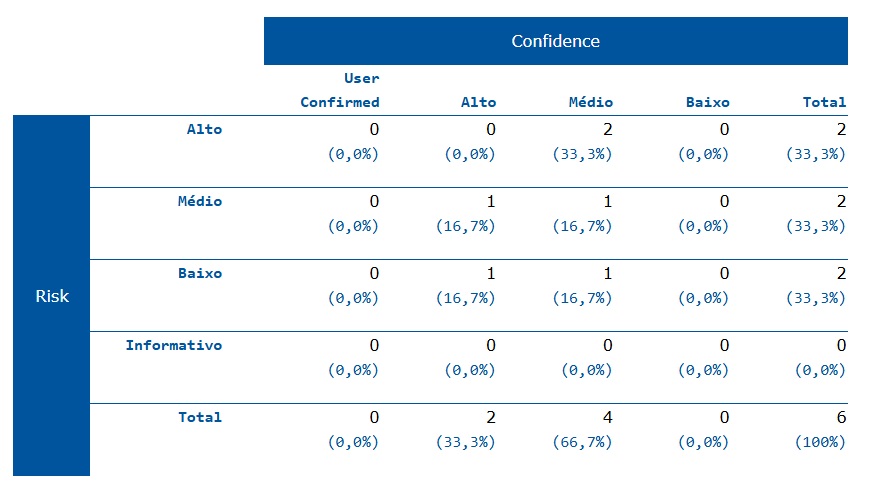
Figura 37 - Relatório geral de vulnerabilidades e riscos



Fonte: elaborada pelo autor.

Em um primeiro momento, ao observar o relatório acima pode-se ter uma ideia de altos riscos de segurança presentes na aplicação, entretanto, o relatório a seguir presente na figura 38 complementa a análise, fornecendo o nível de confiabilidade da definição das vulnerabilidades considerando o contexto e a aplicação testados.

Figura 38 - Vulnerabilidades e confiabilidade da análise conforme contexto



Fonte: elaborada pelo autor.

A análise em conjunto das tabelas permitiu entender de maneira mais clara quais os riscos de fato presentes na aplicação para adoção de medidas para combatê-los.

Os dois riscos de alto nível detectados dizem respeito à possibilidade de existir a vulnerabilidade de SQL *Injection*. Basicamente, essa vulnerabilidade consiste na possibilidade de obtenção de dados confidenciais da base de dados ou mesmo corrupção e alteração das informações por meio da injeção de scripts de linguagem SQL nas requisições. Entretanto, a figura 37 mostra que ambos os riscos detectados foram classificados com nível médio de confiabilidade, isso se deve ao fato de que para a detecção da existência do risco, a ferramenta executa uma requisição simulando um ataque de SQL *Injection* e utilizou 2 parâmetros detectados para considerar a possível existência da vulnerabilidade, sendo eles o *status* da resposta HTTP e o tempo de execução. A figura 39 ilustra o *status* da requisição contendo a injeção.

Figura 39 - *Status* retornado pela requisição de teste de injeção



Fonte: elaborada pelo autor.

De forma complementar, a figura 40 exibe a mensagem utilizada como critério para definição da vulnerabilidade. Na mensagem a ferramenta informa que o tempo necessário para executar a requisição com injeção foi maior do que o tempo para executar a mesma requisição sem injeção.

Figura 40 - Tempo utilizado como critério



Fonte: elaborada pelo autor.

Considerando o resultado das requisições, em nenhuma situação foi detectado o fornecimento de informações ou alteração de dados na resposta da requisição, foram apenas considerados o *status* da resposta e o tempo decorrido, que pode ser facilmente variável dependendo da requisição feita, seja ela com injeção ou não, o que explica a classificação de confiabilidade definida como nível médio para essas vulnerabilidades.

Além disso, deve-se destacar que o *Agent* de dados possui instruções para não realizar *scripts* de alteração na base de dados, e ainda ambos os *Agent*s implementados no projeto possuem instruções para não executarem requisições caso elas contenham qualquer tipo de *script* ou mesmo partes de *scripts* SQL, como já discutido nas seções do Capítulo 5 e, de maneira complementar, os testes anteriores realizados utilizando requisições contendo injeções demonstraram bons resultados de comportamento dos modelos, que detectaram a presença de *scripts* e não retornaram nenhuma resposta com dados sensíveis.

Por outro lado, as vulnerabilidades de *Content Security Policy* (*CSP*) e *X-content-type*, que representam níveis médio e baixo de confiabilidade na detecção, dizem respeito a problemas de configuração de um cabeçalho seguro no servidor de *back-end* da aplicação.

As vulnerabilidades de *CSP* dizem respeito à ausência da configuração de uma política de segurança nos cabeçalhos das requisições HTTP, o que acarreta no risco de ataques de injeção de conteúdo na aplicação. Como medida de combate a essa vulnerabilidade, foi inserido no *header* da aplicação uma lista de endereços considerados seguros para utilização da aplicação, como já discutido nas seções do Capítulo 5.

Outrossim, a vulnerabilidade de ausência de X*-content-type* consiste na falta de uma definição de tipos de mídia que um recurso da aplicação utiliza em sua execução, a não definição desse parâmetro no cabeçalho da aplicação a torna exposta a interpretação de mídias por parte da aplicação, tornando-a vulnerável a ataques de injeção de conteúdo. Para prevenir esse problema, como já abordado no Capítulo 5, foi definido também no *header* da aplicação o tipo de mídia e a impossibilidade de se descobrir o tipo de mídia utilizado arbitrariamente.

A vulnerabilidade de configuração de domínios representa uma possível má configuração da utilização de diferentes domínios. No contexto da aplicação, foi utilizada a configuração padrão do *CORS*, sem a necessidade de definições mais específicas.

Por fim, a vulnerabilidade de vazamento de versão do *server* é caracterizada pelo fornecimento de informações sobre versões e características dos pacotes contidos na aplicação. A solução para essa situação foi a inserção do método de remoção das versões no cabeçalho das requisições, por meio do método *after\_request*, também já discutido no Capítulo 5.

**6.5 ANÁLISE DOS RESULTADOS E CONCLUSÕES**

A seguir os resultados serão abordados de maneira analítica e conclusiva, em conformidade com o conteúdo já exposto.

**6.5.1 RESULTADOS DAS REQUISIÇÕES**

A análise dos resultados das requisições representadas pelos gráficos de acurácia permitiu entender como funciona o processo de construção e instrução de um *Agent* para geração de respostas. Foi possível observar que requisições as quais o modelo foi previamente apresentado forneceram resultados melhores em comparação a requisições desconhecidas, mesmo que aquelas sejam escritas de forma distorcida ou parafraseada. Comprovando a hipótese inicial dos resultados de que a prévia instrução torna o modelo mais preciso e eficiente.

Além disso, outro ponto a ser destacado é a capacidade de contornar o problema de alucinação e geração de dados fictícios, como pode ser observado nos resultados referentes às requisições fora de contexto.

A capacidade do assistente de combater requisições com injeção de modo a proteger os dados também foi de grande impacto no entendimento das possibilidades de utilizar dados restritos.

Ademais, o processo de controle de acesso por meio de um *Agent* se mostrou eficaz na definição e discernimento de interpretação para geração de resposta levando em consideração o nível de acesso do usuário e a requisição que foi realizada.

Finalmente, vale ainda destacar que o *Agent* não possui a capacidade de responder qualquer tipo de pergunta recebida, mesmo que esta esteja presente dentro do contexto ao qual ele foi definido, podendo haver casos de confusão na geração das respostas ou mesmo apresentar dificuldade com requisições complexas que envolvam a realização de cálculos e manipulação de datas, fazendo-se de suma importância o prévio treinamento específico e apresentação do contexto para essas situações em especial.

**6.5.2 RESULTADOS DA AVALIAÇÃO DE SEGURANÇA**

O processo de avaliação de segurança mostrou-se simples e efetivo ao elencar as possíveis vulnerabilidades detectadas e suas respectivas probabilidades de real aparecimento no projeto.

De forma geral, as vulnerabilidades detectadas que possuem maior taxa de confiabilidade representam níveis de risco médios ou inferiores, e são caracterizadas especificamente como falhas referentes à configuração do servidor da aplicação, o que pode ser combatido ao incluir parâmetros e tags no cabeçalho das requisições.

Por outro lado, as vulnerabilidades de alto risco, no contexto do projeto possuem pouco embasamento e nível de confiabilidade em decorrência da falta de parâmetros para classificação mais assertiva do risco e, além disso, o projeto possui mecanismos que contribuem com o controle e combate a esse tipo de fragilidade.

**7 CONSIDERAÇÕES FINAIS**

No decorrer de todo o processo de pesquisa e implementação do trabalho buscou-se entender caminhos que possibilitaram propor uma forma geral de utilizar dados sensíveis em conjunto com aplicações baseadas em modelos de linguagem e inteligência artificial, de modo a contribuir com os estudos na área de forma complementar.

A falta de controle de acesso a informações em aplicações que utilizam modelos de linguagem torna os dados utilizados vulneráveis e passíveis de exposição para qualquer usuário, contribuindo com o aumento da vulnerabilidade das aplicações e com a limitação do tipo de informação que pode ser apresentado aos modelos.

Em um primeiro momento, a hipótese principal era realizar o estudo e procurar entender as possibilidades de utilizar qualquer tipo de dado, inclusive os sensíveis, em conjunto com uma inteligência artificial, visto que o modelo focaliza na geração de respostas, e não no conteúdo presente nelas. Portanto, considerando tais afirmações, buscou-se desenvolver um assistente com controle de acesso que pudesse utilizar os dados e ter um maior discernimento em relação à geração de respostas e a quais informações podem ser fornecidas.

O desenvolvimento do assistente apresentou resultados consideráveis quando submetido a diferentes situações, que incluíram a geração de respostas baseadas em diferentes requisições ou que necessitavam a utilização de diferentes tipos de dados, realizando o bloqueio quando necessário.

Além disso, o assistente apresentou baixa alucinação e grande capacidade de discernimento sobre o que foi requisitado, mesmo em casos onde a requisição envolveu a solicitação de informações que não poderiam ser fornecidas ou mesmo a injeção de *scripts*.

Por outro lado, o desempenho não foi o mesmo quando as perguntas envolviam consultas complexas as quais o modelo não conhecia previamente ou que envolviam um grande conjunto de dados, demonstrando certo grau de dependência nessas situações em específico.

Contudo, de maneira geral, os resultados obtidos neste trabalho apresentam uma contribuição significativa na abordagem do problema da impossibilidade de utilização de informações restritas em conjunto com modelos de inteligência artificial, e ainda contribui com os estudos da área ao explorar meios de implementação e utilização dos recursos.

Destarte, pôde-se concluir por meio deste trabalho que a utilização de tecnologias emergentes pode ser aliada a diferentes métodos que possibilitem a manipulação de dados de maneira controlada, de modo que torne-se possível o enriquecimento do conjunto de informações utilizado por aplicações baseadas em *large language models* e inteligência artificial generativa, principalmente ao considerar aspectos como segurança e controle.

Considerando o contexto de futuras implementações, podem ser mencionados a implementação de outras técnicas de segurança que envolvam variados contextos não abordados no trabalho, em que foi adotado como contexto o ambiente de recursos humanos. Além disso, o desenvolvimento de um sistema de *logs*, que armazena cada uma das requisições e acessos realizados também contribuiria com a robustez e controle do tráfego de dados. Por fim, a adaptação da aplicação do assistente para diferentes setores da sociedade, os quais utilizam outros tipos de dados, se mostraria de grande contribuição com o estudo das possibilidades de utilização em diferentes situações e contextos.

**REFERÊNCIAS**

ALFARIZI, Muhammad. *et al*. Vulnerability Analysis And Proven On The neonime.co Website using OWASP Zap 4 and XSpear. In: Jurnal Teknologi Komputer dan Sistem Informasi. 2022. DOI 10.56327/jtksi.v5i2.1130. Disponível em: http://www.jurnal.stmikpringsewu.ac.id/index.php/jtksi/article/view/1130. Acesso em: 29 set. 2024.

ALVES, Filipe; NUNO Mateus-Coelho ; CRUZ-CUNHA, Manuela. Encryption File System Framework - Proof of Concept. In: Procedia Computer Science, 181. 2021. **Proceedings** [...]. [S.I.]. p. 1237-1246, DOI 10.1016/j.procs.2021.01.322. Disponível em: https://doi.org/10.1016/j.procs.2021.01.322. Acesso em: 29 set. 2024.

BARBOSA, J. N. **Ontologias e Processamento de Linguagem Natural na Publicação e Acesso de Dados para Carta de Serviços do Ceará**. 2022. 64f.Trabalho de Conclusão de Curso (Graduação) - Universidade Federal do Ceará, Quixadá, 2022. Disponível em: https://repositorio.ufc.br/bitstream/riufc/68202/1/2022\_tcc\_jnbarbosa.pdf. Acesso em: 24 abr. 2024.

CHEN, Jiawei; LIN, Hongyu; HAN, Xianpei.; SUN, Le. Benchmarking Large Language Models in Retrieval-Augmented Generation. In: AAAI Conference on Artificial Intelligence. 2024, Vancouver, Canadá. **Proceedings** [...]. [S.I.]. p. 17754-17762, DOI 10.1609/aaai.v38i16.29728. Disponível em:

https://ojs.aaai.org/index.php/AAAI/article/view/29728. Acesso em: 26 ago. 2024.

GANESH, Sai; PURWAR, Anupam ; B, Gautam. **Context-augmented Retrieval: A Novel Framework for Fast Information Retrieval based Response Generation using Large Language Model**. arXiv.org. Disponível em: https://arxiv.org/abs/2406.16383. Acesso em: 25 jul. 2024.

**Get started with LangSmith**. Disponível em: https://docs.smith.langchain.com/. Acesso em: 2 out. 2024.

GRININ, L; GRININ, A; KOROTAYEV, A. A quantitative analysis of worldwide long-term technology growth: From 40,000 BCE to the early 22nd century. **Technological Forecasting and Social Change.** 15 mar. 2020. Disponível em: https://www.sciencedirect.com/science/article/abs/pii/S0040162519314118. Acesso em 25 out. 2024.

HABBAL, A; ALI, M. K; MUSTAFA A. A. Artificial Intelligence Trust, Risk and Security Management (AI TRiSM): Frameworks, applications, challenges and future research directions. **Expert Systems with Applications**. 15 abr. 2024. Disponível em: https://www.sciencedirect.com/science/article/abs/pii/S0957417423029445. Acesso em: 25 out. 2024.

**Introduction | LangChain**. Langchain.com. Disponível em: https://python.langchain.com/docs/introduction/. Acesso em: 2 out. 2024.

IZACARD, Gautier. *et al*. Few-shot Learning with Retrieval Augmented Language Models. In: The Journal of Machine Learning Research. 2023. DOI 10.5555/3648699.3648950 Disponível em: https://dl.acm.org/doi/10.5555/3648699.3648950. Acesso em: 10 set. 2024.

**LangSmith.** Langchain.com. Disponível em: https://www.langchain.com/langsmith. Acesso em: 2 out. 2024.

LARA, D. F; *et al.* Atendente Artificial Humanizada Usando Langchain para Manipulação de Modelos de Linguagem em Larga Escala. In: VI Simpósio de Iniciação Científica, 6, 2023, Itajubá. Proceedings [...]. 6 nov. 2023. p 1-4. Disponível em: https://doi.org/10.29327/1386870.6-80. Acesso em: 27 out. 2024.

LEWIS, P. *et al*. **Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks**. In: 34th Conference on Neural Information Processing Systems, 34. 2020 Vancouver, Canada. **Proceedings** [...]. [S.I.]. p. 9459-9474, DOI 10.5555/3495724.3496517. Disponível em: https://dl.acm.org/doi/abs/10.5555/3495724.3496517. Acesso em: 31 mar. 2024.

LIU, Y. *et al.* Trustworthy LLMs: a Survey and Guideline for Evaluating Large Language Models' Alignment. 2023. DOI 10.48550/arXiv.2308.05374. Disponível em: https://arxiv.org/abs/2308.05374, acesso em: 31 mar. 2024.

MCAVINUE, Sarah ; DEV, Kapal. Comparative evaluation of Large Language Models using key metrics and emerging tools**.** In: Expert Systems, 2024. DOI 10.1111/exsy.13719. Disponível em: https://onlinelibrary.wiley.com/doi/abs/10.1111/exsy.13719. Acesso em: 2 out. 2024.

PEREIRA, K. **Um estudo sobre o uso da Inteligência Artificial nas empresas**. 2021. 26f. Monografia - Instituto de Ciências Exatas e Tecnologia da Universidade Federal do Amazonas, Itacoatiara – Amazonas, 2021. Disponível em: https://riu.ufam.edu.br/bitstream/prefix/5989/2/TCC\_KeithAnnyPereira.pdf. Acesso em: 24 abr. 2024.

PROENÇA, V. L. **Automatização da revisão de literatura científica com geração aumentada por recuperação**. 2024. 59f. Trabalho de conclusão de curso - Universidade Federal do Rio de Janeiro. Disponível em: https://pantheon.ufrj.br/handle/11422/23629. Acesso em: 29 ago. 2024.

SANTOS, F. A lei geral de proteção de dados pessoais (LGPD) e a exposição de dados sensíveis nas relações de trabalho. **Revista do Tribunal Regional do Trabalho da 10ª Região**, Brasília, v. 24, n. 2, p. 145-151, 14 jan. 2021.

SOONG, David. *et al*. **Improving accuracy of GPT-3/4 results on biomedical data using a retrieval-augmented language model**. In: PLOS Digital Health, 2024. DOI 10.1371/journal.pdig.0000568. Disponível em: https://doi.org/10.1371/journal.pdig.0000568. Acesso em: 10 set. 2024.

VIDIVELLI, Soundarajan; RAMACHANDRAN, Manikandan; DHARUNBALAJI, A. **Efficiency-Driven Custom Chatbot Development: Unleashing LangChain, RAG, and Performance-Optimized LLM Fusion**. In: Computers, Materials & Continua, 80. 2024, Thanjavur, Índia. **Proceedings** [...]. [S.I.]. p. 2423-2442, DOI 10.32604/cmc.2024.054360. Disponível em: https://doi.org/10.32604/cmc.2024.054360. Acesso em: 29 set. 2024.

YOSHIOKA, R; SILVA, C. da. **Análise de Vulnerabilidades em domínios**. 2024. 6f. Trabalho de conclusão de curso - Fundação Universidade Federal de Mato Grosso do Sul. Disponível em: https://repositorio.ufms.br/handle/123456789/9135. Acesso em: 29 set. 2024.

**ZAP – Documentation**. Zaproxy.org. Disponível em: https://www.zaproxy.org/docs/. Acesso em: 29 set. 2024.

1. Image taken from: https://doi.org/10.48550/arXiv.2308.05374 [↑](#footnote-ref-1)
2. Image taken from: https://doi.org/10.48550/arXiv.2308.05374 [↑](#footnote-ref-2)