INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA INFORMÁTICA




Sistema Help Desk Inteligente - Integração de Chatbot e Sistema de Gestão Automatizada de Pedidos

DAVID AZEVEDO INÁCIO
setembro de 2025

POLITÉCNICO
DO PORTO

ISEP INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO

Sistema Help Desk Inteligente

Integração de Chatbot e Sistema de Gestão Automatizada de Pedidos

David Azevedo Inácio

Dissertação para obtenção do Grau de Mestre em

Engenharia Informática, Área de Especialização em

Engenharia de Software

Orientador: Luís Gomes

Co-orientadora: Goreti Marreiros

Supervisor: Ruan Carvalho

Porto, setembro de 2025

Declaração de Integridade

Declaro ter conduzido este trabalho académico com integridade.

Não plagiei ou apliquei qualquer forma de uso indevido de informações ou falsificação de resultados ao longo do processo que levou à sua elaboração.

Portanto, o trabalho apresentado neste documento é original e de minha autoria, não tendo sido utilizado anteriormente para nenhum outro fim. As exceções estão explicitamente reconhecidas na secção “Considerações éticas” do primeiro capítulo. Esta secção também declara como as ferramentas de IA foram utilizadas e para que finalidade.

Declaro ainda que tenho pleno conhecimento do Código de Conduta Ética do P.PORTO.

ISEP, Porto, 21 de setembro de 2025

Dedicatória

Dedico esta dissertação aos meus avós António e Celeste, que infelizmente partiram no decurso da sua escrita. Ambos fazem parte de mim e da pessoa que sou hoje, e por isso, esta conquista também é vossa. Recordo com carinho o vosso orgulho em dizer que tinham um neto engenheiro. Espero deixar-vos ainda mais orgulhosos ao me formar mestre, e que olhem sempre por mim aí de cima. Amo-vos eternamente.

Resumo

Nos últimos anos, a crescente digitalização acentuou a dependência de software em praticamente todos os setores. Contudo, esta evolução não foi acompanhada por um aumento proporcional da literacia digital, criando uma lacuna entre a complexidade das aplicações e a capacidade dos utilizadores em interagir eficazmente com as mesmas. Este cenário reforça a importância da acessibilidade e do suporte como atributos centrais da qualidade de software. Em contextos empresariais, onde a pressão pelo desenvolvimento rápido frequentemente se sobrepõe a preocupações com a usabilidade, a ausência de mecanismos de suporte adequados traduz-se em frustração dos utilizadores, ineficiências e sobrecarga das equipas de suporte.

Para responder a este desafio, esta dissertação propõe uma aplicação *help desk* inovadora, concebida como sistema facilmente integrável em qualquer aplicação e capaz de recorrer ao sistema de gestão de *tickets* já utilizado pela organização. A proposta assenta em dois elementos centrais: um *chatbot* com base de conhecimento atualizável e uma API que cria e gere pedidos diretamente em plataformas de gestão de *tickets*. O caso de estudo foi a aplicação Capla, da empresa Natixis, onde se verificou a necessidade de otimizar o suporte sem introduzir um novo sistema autónomo. A integração foi realizada com o Jira, sistema de gestão de *tickets* já consolidado na empresa, demonstrando que a solução pode ser aplicada em contextos reais sem exigir substituição das infraestruturas existentes.

O desenvolvimento da solução foi sustentado por uma revisão aprofundada da literatura e do estado da arte em três eixos principais: *help desk*, *chatbots* e *prompt engineering*. Foram comparadas plataformas de construção de *chatbots*, analisados modelos de linguagem de grande escala (LLMs) adequados ao suporte e discutidas *frameworks* de *back-end* para o desenvolvimento da API. A arquitetura final resultou num protótipo modular, escalável e replicável em diferentes contextos empresariais, equilibrando inovação com viabilidade prática.

O protótipo foi validado em cenários reais, revelando-se altamente eficaz. Os pedidos foram resolvidos com uma taxa de sucesso de 99%, registando apenas 1% de erros críticos. Recorrendo à solução desenvolvida, o tempo de resposta poderá passar de 24 horas para apenas alguns segundos, como demonstrado nos casos de estudo. O sistema assegurou ainda consistência na criação de *tickets* e capacidade de identificar *tickets* duplicados, reforçando a fiabilidade do processo. Estes resultados confirmaram a viabilidade da solução e o seu potencial para transformar a forma como aplicações empresariais oferecem suporte.

Palavras-chave: Software *Help Desk*, *Chatbot*, *Prompt Engineering*, Suporte ao Utilizador

Abstract

In recent years, the growing digitalization of society has intensified the dependence on software across virtually all sectors. However, this evolution has not been matched by a proportional increase in digital literacy, creating a gap between the complexity of applications and users' ability to interact effectively with them. This context highlights accessibility and user support as central attributes of software quality. In business environments, where the pressure for rapid development often outweighs usability concerns, the absence of adequate support mechanisms results in user frustration, inefficiencies, and increased workload for support teams.

To address this challenge, this dissertation proposes an innovative help desk application, designed as a system that can be easily integrated into any software and capable of leveraging the ticket management system already in use by the organization. The proposal is based on two core elements: a chatbot with an updatable knowledge base and an API capable of creating and managing tickets directly within existing ticketing platforms. The case study focused on the Capla application at Natixis, where there was a clear need to optimize user support without introducing a new standalone system. Integration was carried out with Jira, the company's established ticket management platform, demonstrating that the solution can be applied in real contexts without requiring the replacement of existing support infrastructures.

The development of the solution was supported by an in-depth review of the literature and state of the art in three key domains: help desk systems, chatbots, and prompt engineering. Different chatbot development platforms were compared, large language models (LLMs) suitable for support were analyzed, and back-end frameworks for API development were discussed. The final architecture resulted in a modular, scalable, and replicable prototype, balancing innovation with practical feasibility.

The prototype was validated in real scenarios, proving highly effective. User requests were resolved with a 99% success rate, with only 1% of critical errors recorded. By relying on the proposed solution, response times could be reduced from an average of 24 hours to just a few seconds, as confirmed by the case studies. The system also ensured consistency in ticket creation and the detection of duplicates, reinforcing the reliability of the process. These results confirmed the viability of the solution and its potential to transform how enterprise applications deliver user support.

Keywords: Help Desk Software, Chatbot, Prompt Engineering, User Support

Agradecimentos

Gostaria de começar por agradecer à minha família e amigos por todo o apoio e motivação que me deram ao longo destes cinco anos de faculdade. Um agradecimento especial aos meus pais, que sempre colocaram o meu sucesso académico como prioridade e me guiaram pelos melhores caminhos, permitindo-me alcançar todos os objetivos que tracei até então. E um obrigado sentido ao meu irmão Gabriel, por seres o meu companheiro de vida, e por toda a ajuda que me deste para que eu pudesse trabalhar nesta dissertação.

Quero também agradecer aos meus amigos Tiago Bento e Bruno Cunha, por me aturarem durante estes cinco anos. Foram um suporte inestimável, tanto a nível académico como pessoal. Sem vocês, este percurso teria sido muito mais difícil. A vossa amizade foi a melhor coisa que a faculdade me trouxe.

Deixo igualmente o meu agradecimento a todos os membros da minha equipa de trabalho da Natixis, que sempre me apoiaram em todo este processo. Um agradecimento especial à Ana Laura e ao Vasco Marinho, pelo contributo fundamental na conceção da ideia base da dissertação e na realização dos casos de estudo. E um agradecimento igualmente especial ao meu supervisor Engenheiro Ruan Carvalho, por se ter disponibilizado a ajudar-me em todas as fases desta dissertação, por todos os ensinamentos e orientação que me deu ao longo destes anos, e pela preocupação genuína com o meu sucesso académico e profissional.

Quero ainda agradecer aos meus orientadores, Professor Doutor Luís Gomes e Professora Doutora Goreti Marreiros, pela disponibilidade demonstrada e apoio na concretização desta dissertação, e por terem acreditado em mim, mesmo perante as dificuldades e problemas que surgiram ao longo do percurso.

Por fim, mas de forma muito especial, quero expressar a minha gratidão à minha Renata. Foste a razão pela qual nunca perdi a esperança, mesmo nos momentos em que tudo parecia desabar e a vontade de desistir era grande. Obrigado por seres incansável, por me inspirares diariamente e por me dares forças para ser sempre melhor por ti. És tudo para mim.

Índice

Lista de Figuras

Lista de Tabelas

Acrónimos e Símbolos

Lista de Acrónimos

API	*Application Programming Interface*
CI/CD	*Continuous Integration / Continuous Deployment*
DTO	*Data Transfer Object*
HTTP	*HyperText Transfer Protocol*
HTTPS	*HyperText Transfer Protocol Secure*
JQL	*Jira Query Language*
KPI	*Key Performance Indicator*
LLM	*Large Language Model*
MVP	*Minimum Viable Product*
PLN	Processamento de Linguagem Natural
RGPD	Regulamento Geral de Proteção de Dados
TI	Tecnologias da Informação
WBS	*Work Breakdown Structure*

1 Introdução

No presente capítulo será feita uma contextualização do problema que motivou a realização desta dissertação, sendo abordadas as razões que levaram o autor a dedicar-se ao tema escolhido. De seguida, será descrita a motivação pessoal e profissional para o desenvolvimento deste trabalho e apresentada a solução proposta para resolver o problema identificado.

Posteriormente, serão definidos os objetivos do projeto e formuladas as questões de investigação que orientam a dissertação. Será também explicada a metodologia de investigação adotada.

Na sequência, serão identificados os principais stakeholders envolvidos no projeto e apresentadas as contribuições esperadas com a realização deste trabalho. Por fim, serão discutidas as considerações éticas relevantes para a execução da dissertação.

1.1 Contextualização

Nos últimos anos, evidenciou-se um crescimento exponencial na utilização de software e aplicações em todo o mundo. De acordo com a Eurostat, entre 2015 e 2024 houve um aumento de 13,06% na percentagem de indivíduos da União Europeia que utilizam a internet, passando de 80,16% para 93,22%, sendo este o valor mais alto alguma vez registado (Eurostat, 2024). Além disso, segundo uma notícia do Jornal de Negócios, a Grand View Research avaliou o tamanho do mercado global de software em 583,47 mil milhões de dólares em 2022, prevendo-se que este cresça a uma taxa composta de crescimento anual (CAGR) de 11,5% entre 2023 e 2030 (Marvão, 2024), o que reflete nitidamente o aumento da procura por soluções digitais e sistemas de software. No entanto, apesar deste crescimento acelerado no uso de aplicações e software, a evolução do conhecimento digital por parte da população nem sempre acompanhou este ritmo. Como demonstrado num outro estudo da Eurostat, embora a utilização da internet tenha aumentado substancialmente, o nível de literacia digital de muitos cidadãos ainda não é tão elevado quanto o expectável. Em 2023, apenas 55,56% das pessoas na União Europeia, com idades entre os 16 e os 74 anos, possuíam, pelo menos, competências

digitais globais básicas (Eurostat, 2024). Esta desproporção entre a expansão do uso de tecnologia e o conhecimento técnico dos utilizadores pode gerar desafios significativos, principalmente no que diz respeito à usabilidade das aplicações e ao suporte que elas oferecem aos seus utilizadores. Embora a digitalização continue a avançar, muitos indivíduos ainda enfrentam complicações em interagir com as plataformas digitais de maneira eficaz, o que resulta numa má e frustrante experiência para os utilizadores.

Este cenário é particularmente significativo quando se considera que, de acordo com um estudo publicado em 2011, 70% dos clientes abandonam um fornecedor de serviços não por questões de preço ou qualidade do produto, mas devido à insatisfação com a qualidade do atendimento ao cliente (Dingding , et al., 2011). Essa estatística sublinha a importância crítica do suporte ao cliente e da experiência do utilizador na manutenção da fidelidade dos clientes e na longevidade das aplicações.

A crescente demanda por soluções que sejam eficientes e acessíveis realça a necessidade de assegurar que as aplicações forneçam uma boa experiência ao utilizador, incluindo um sistema de suporte robusto e eficaz. As empresas precisam de sistemas que não só permitam uma navegação intuitiva no software, mas também ofereçam soluções rápidas e acessíveis de assistência ao utilizador. A ausência de suporte eficiente pode prejudicar a experiência do utilizador e, consequentemente, a sua confiança e satisfação com o software.

E à medida que a pressão por soluções rápidas e eficientes se intensifica, as empresas enfrentam o desafio de equilibrar a entrega de produtos com a qualidade do suporte prestado aos seus utilizadores. Em cenários em que a rapidez na entrega de soluções se torna cada vez mais uma prioridade para as empresas, é comum que aspetos como a acessibilidade e o suporte ao utilizador sejam relegados para segundo plano, o que pode resultar em experiências frustrantes para os utilizadores e aumentar a carga de trabalho das equipas de suporte.

Este contexto enfatiza ainda mais a necessidade de uma solução que seja facilmente aplicável a qualquer aplicação, capaz de mitigar as lacunas de acessibilidade e suporte ao cliente de forma eficaz e eficiente, sem exigir das equipas de suporte o esforço adicional de adaptação a novas ferramentas ou tecnologias. A procura por soluções que assegurem a satisfação do utilizador, enquanto reduzem a carga sobre as equipas de suporte, torna-se cada vez mais evidente à medida que a digitalização continua a expandir-se.

É perante esta necessidade, e motivada por um caso prático vivenciado pelo autor desta tese, que a mesma surge com uma proposta de uma solução universal para o problema apresentado. A proposta centra-se no desenvolvimento de uma aplicação capaz de fornecer o suporte necessário aos utilizadores de qualquer aplicação, através de funcionalidades como um *chatbot* [1] capaz de auxiliar na resolução de problemas frequentes e dúvidas recorrentes. Simultaneamente, a solução oferece suporte às equipas de atendimento ao utilizador, recorrendo a um sistema de gestão de pedidos que permite priorizar e encaminhar

[1] Programa de software concebido para interagir com utilizadores através de linguagem natural, fornecendo respostas automáticas e assistência em tempo real.

automaticamente os pedidos de suporte, otimizando os fluxos de trabalho. Para além do desenvolvimento, esta tese tem como objetivo fornecer um estudo detalhado do impacto desta solução num caso real, bem como contribuir para o avanço do conhecimento geral sobre aplicações *help desk*[2], categoria na qual a aplicação proposta se insere.

1.2 Motivação

Como descrito no tópico anterior, a motivação para a elaboração desta tese encontra-se diretamente ligada à experiência profissional do autor, mais especificamente às dificuldades observadas na utilização e suporte da aplicação Capla.

O Capla é a principal ferramenta utilizada pelos gestores da empresa Natixis para fazer o planeamento e gestão de recursos humanos, tendo um papel crucial no desempenho e produtividade da organização. No entanto, o grau elevado de complexidade inerente à aplicação resulta frequentemente na necessidade de assistência por parte dos utilizadores para compreender ou executar determinadas funcionalidades.

A ausência de um sistema de suporte integrado na própria aplicação agrava ainda mais este problema, o que gera uma dependência excessiva da equipa de suporte técnico. Atualmente, a equipa enfrenta um volume elevado de solicitações diárias, que são geridas principalmente por e-mail. Embora funcional, este método revela-se inadequado para as exigências de agilidade e eficiência das operações modernas.

Este contexto gera um ciclo problemático: os utilizadores enfrentam longos tempos de espera para obter respostas às suas questões, levando à frustração e à redução da sua produtividade. Simultaneamente, a equipa de suporte encontra-se em sobrecarga, o que dificulta a sua capacidade de dar respostas rápidas e eficazes.

Para além disso, importa salientar que a introdução de uma nova ferramenta autónoma poderia agravar ainda mais este problema, uma vez que exigiria tempo e esforço de adaptação por parte da equipa, que já se encontra limitada em termos de disponibilidade.

Perante este cenário, torna-se evidente a necessidade de realizar uma mudança significativa no modelo atual de gestão do suporte ao utilizador. É fundamental identificar e desenvolver uma solução inovadora, que seja capaz de melhorar a experiência dos utilizadores e aliviar a pressão sobre a equipa de suporte, para superar os desafios enfrentados e assegurar uma operação mais eficiente e satisfatória.

[2] Sistema de suporte ao utilizador que centraliza, organiza e acompanha pedidos de assistência técnica ou funcional.

1.3 Solução Proposta

Perante o problema identificado, a solução proposta baseia-se no desenvolvimento de um sistema universal de suporte ao utilizador, desenhado para colmatar lacunas de acessibilidade e eficiência em qualquer tipo de aplicações. Este sistema insere-se na categoria das aplicações help desk, sendo projetado para disponibilizar um meio centralizado e automatizado de suporte ao cliente, que pode ser facilmente ajustado a diferentes contextos e aplicações.

O principal objetivo desta solução é responder à necessidade generalizada de melhorar a experiência dos utilizadores em aplicações digitais, ao reduzir simultaneamente a sobrecarga das equipas de suporte. Para alcançar este objetivo, o sistema integrará duas funcionalidades principais:

- **Chatbot baseado em inteligência artificial**: Este chatbot será responsável por oferecer respostas rápidas e precisas às questões e problemas mais frequentes dos utilizadores, utilizando uma base de conhecimento dinâmica e atualizável.
- **Gestão automatizada de consultas**: nos casos em que a intervenção humana for necessária, o sistema incluirá um módulo de gestão de consultas capaz de organizar, priorizar e encaminhar automaticamente os pedidos de suporte. Este mecanismo será concebido para integrar-se diretamente com sistemas de gestão de tarefas já existentes, evitando a introdução de novas ferramentas e reduzindo o esforço de adaptação das equipas de suporte. Além disso, irá permitir que as equipas de suporte se concentrem nas solicitações mais complexas, reduzindo os tempos de resposta e otimizando os recursos disponíveis.

Embora a proposta seja desenhada como uma solução para o problema geral da falta de acessibilidade e suporte eficiente em aplicações, o Capla, já descrito anteriormente, será utilizado como caso de estudo para validar a eficácia e impacto do sistema desenvolvido. Como uma aplicação crítica no planeamento de recursos humanos da Natixis, o Capla representa um exemplo concreto das dificuldades enfrentadas por utilizadores e equipas de suporte. Assim, o desenvolvimento deste sistema visa não apenas abordar os desafios enfrentados no contexto específico do Capla, mas também criar uma solução escalável e modular, capaz de ser adaptada a outras aplicações empresariais com necessidades semelhantes.

1.4 Objetivos

O objetivo geral desta dissertação é desenhar e desenvolver uma proposta de solução de suporte ao utilizador, utilizando a aplicação Capla como caso de estudo, e realizar testes práticos que comprovem o impacto desta solução no contexto real.

A seguir, são apresentados os subobjetivos do projeto:

- **Realizar um estudo aprofundado sobre o estado da arte de aplicações help desk/suporte ao cliente:**
  Investigar as principais soluções help desk disponíveis no mercado e entender a

atualidade desse tipo de sistemas. Tentar perceber quais as principais funcionalidades destes sistemas, e contribuir com um estudo significativo de quais as tendências tecnológicas relacionadas com suporte ao cliente.

- **Desenvolver uma proposta de solução de suporte ao utilizador**:
  Desenhar uma solução que inclua um sistema centralizado de suporte ao utilizador, com funcionalidades como um chatbot baseado em inteligência artificial capaz de responder a pedidos frequentes em tempo real, e um sistema de gestão de pedidos para priorizar e encaminhar automaticamente estas solicitações de suporte. Esta solução deverá ser concebida com o intuito de ser adaptável ao contexto da aplicação Capla, mas construída de forma a ser facilmente escalável para outras aplicações no futuro.

- **Implementar um MVP[3] (Produto Mínimo Viável) da solução proposta:**
  Desenvolver um MVP que contenha as funcionalidades essenciais do sistema, incluindo a integração do chatbot e o sistema de gestão de pedidos. O MVP será uma versão inicial e funcional da solução, que irá permitir realizar os primeiros testes práticos, assim como validar a viabilidade da solução proposta.

- **Realizar testes práticos para avaliar o impacto da solução:**
  A fase de testes será focada na avaliação prática do sucesso da solução no ambiente real da aplicação Capla. Os testes serão divididos em categorias diferentes para analisar a eficácia da solução em diversas dimensões:

    - **Teste de Eficiência Operacional:** O objetivo deste teste é avaliar a redução do tempo de resposta aos pedidos dos utilizadores. Para isso, será comparado o tempo médio de resposta da equipa de suporte antes e depois da implementação da solução, para medir a eficiência operacional conseguida com a utilização do chatbot e do sistema automatizado de gestão de consultas.
    - **Teste de Satisfação do Utilizador:** O objetivo deste teste é avaliar em que medida a solução desenvolvida irá contribuir para a satisfação dos utilizadores, através da análise da qualidade e relevância das respostas fornecidas pelo chatbot. A avaliação será focada em verificar se os pedidos são corretamente interpretados e resolvidos, bem como na consistência do desempenho em diferentes cenários de interação.
    - **Teste de Eficiência da Equipa de Suporte:** O objetivo deste teste é avaliar em que medida a solução poderá reduzir a carga de trabalho da equipa de suporte, através da automação de tarefas repetitivas e da correta categorização e priorização dos pedidos. A análise será realizada com base no desempenho do sistema e encaminhamento automático das solicitações, permitindo verificar o

[3] Versão inicial de um sistema ou aplicação que contém apenas as funcionalidades essenciais, permitindo validar a sua viabilidade e recolher feedback antes de investimentos adicionais.

potencial impacto da solução na diminuição do esforço manual exigido à equipa de suporte.

- **Analisar os resultados dos testes e planear ajustes da solução conforme necessário:**

Após a realização dos testes, irá ser realizada uma análise pormenorizada dos resultados obtidos. Serão comparados os dados antes e depois da implementação da solução com o intuito identificar melhorias, pontos fortes e áreas a serem ajustadas. Com base nessa análise, serão planeadas melhorias na solução, caso necessário, para otimizar ao máximo a experiência dos utilizadores e a eficiência da equipa de suporte.

1.5 Questões de Investigação

As questões de investigação são fundamentais para guiar o desenvolvimento deste trabalho, ao permitir a exploração e avaliação dos impactos, benefícios e a eficácia da solução proposta.

Posto isto, as questões de investigação que guiarão o rumo do trabalho são as seguintes:

1. Qual é o impacto da implementação de uma aplicação Help Desk no suporte ao cliente de uma aplicação?
2. De que forma a utilização de um chatbot pode melhorar a eficácia na resolução de problemas frequentes de utilizadores?
3. Como pode um sistema automatizado de gestão de pedidos contribuir para o aumento da produtividade e eficiência de uma equipa de suporte?

Estas questões de investigação refletem os principais desafios identificados no enquadramento inicial do trabalho. A primeira centra-se na avaliação do impacto global da solução proposta, verificando se a sua implementação melhora efetivamente o suporte prestado aos utilizadores. A segunda direciona a análise para o papel do chatbot, procurando determinar em que medida a automatização das respostas pode reduzir tempos de espera e aumentar a satisfação dos utilizadores. Por fim, a terceira aborda a vertente operacional, explorando se a gestão automatizada de consultas é capaz de aliviar a sobrecarga das equipas de suporte e de otimizar os seus fluxos de trabalho.

Através da resposta a estas questões, será possível comprovar não só a viabilidade técnica da solução, mas também o seu valor acrescentado em termos de eficiência, experiência do utilizador e impacto organizacional.

1.6 Metodologia de Investigação

A metodologia *Design and Create* (Oates, 2005), que se revelou como a abordagem mais alinhada com os objetivos definidos, será adotada para conduzir a investigação e o desenvolvimento deste projeto. De modo geral, estes objetivos incluem a definição de um MVP (Produto Mínimo Viável), a sua implementação prática e, posteriormente, a realização de testes para avaliar o sucesso da solução desenvolvida. Esta metodologia caracteriza-se precisamente por este enquadramento, ao propor um processo estruturado que articula a conceção, implementação e avaliação de soluções inovadoras. Deste modo, esta foi a metodologia escolhida por se alinhar naturalmente com os objetivos traçados, o que garante uma integração consistente entre as fases de design e os resultados alcançados na prática.

O processo começa com a fase de desenho da solução (*design*), onde será elaborado um protótipo detalhado do MVP (Produto Mínimo Viável), que irá incluir as funcionalidades vitais para responder ao problema identificado. Esta etapa tem como foco definir a estrutura e o funcionamento da solução, garantindo que o design reflete as necessidades reais dos stakeholders.

De seguida, será realizada a fase de implementação, onde o MVP será desenvolvido na prática, com base no design anteriormente elaborado. Esta implementação permitirá materializar a solução, tornando-a funcional e pronta para ser testada.

Por fim, será conduzida a fase de testes com base em interações reais entre os utilizadores e a aplicação. No decorrer desta etapa, serão recolhidos dados qualitativos e quantitativos para avaliar a eficácia da solução, ao analisar indicadores como a diminuição do tempo de resposta, a satisfação dos utilizadores e a redução da carga de trabalho da equipa de suporte.

Esta abordagem iterativa e estruturada assegura não só a adaptação da solução às necessidades identificadas, mas também fornece uma base sólida para avaliar o impacto do projeto, permitindo ajustes e melhorias futuras, em caso de necessidade.

1.7 Stakeholders

Os principais stakeholders deste projeto são aqueles que, direta ou indiretamente, serão mais impactados pela solução proposta e que mais interesse têm em ver o projeto desenvolvido com sucesso. Estudar quem são e como são beneficiados pelo projeto ao longo das suas fases de desenvolvimento é crucial, uma vez que a finalidade é melhorar processos e resultados obtidos para todos os envolvidos. De seguida serão apresentados os stakeholders com mais relevância para este projeto, ordenados por ordem de impacto e interesse:

1. **Equipa de Suporte do Capla**

   A equipa de suporte é, sem dúvida, o stakeholder que mais irá ser impactado pela solução proposta. São eles que enfrentam diariamente o problema da sobrecarga de pedidos dos utilizadores, o que afeta diretamente a sua produtividade e tranquilidade. A solução proposta, integrando um chatbot e automatizando a gestão de pedidos de suporte, procura reduzir significativamente a quantidade de trabalho manual da equipa. Como resultado, são esperados tempos de resposta mais curtos e uma carga de trabalho mais equilibrada, permitindo à equipa focar-se em questões mais complicadas e melhorar a qualidade do atendimento aos utilizadores. Sendo diretamente responsáveis pela gestão das consultas de suporte, a equipa de suporte tem o maior interesse na implementação de uma solução eficiente e eficaz.

2. **Utilizadores do Capla**

   Os utilizadores do Capla, que utilizam a plataforma nas suas previsões e gestão de recursos humanos, também serão impactados significativamente pela solução. Ao receberem respostas mais rápidas para os seus problemas, sem a necessidade de esperar por uma resposta da equipa de suporte, terão uma experiência de utilização mais eficiente. Isto irá permitir que os gestores de equipas usem o Capla de forma mais eficaz nas suas tarefas do dia a dia, o que aumentará a sua produtividade e reduzirá o tempo gasto em questões técnicas. Com menos necessidade de recorrer à equipa de suporte, os utilizadores poderão focar-se mais nas suas funções de gestão de recursos humanos, beneficiando assim da autonomia e agilidade proporcionadas pela solução.

3. **Gestores das Equipas de Desenvolvimento do Capla**

   Embora mais indiretamente, os gestores das equipas de desenvolvimento do Capla também têm um interesse considerável no sucesso deste projeto. A implementação de um sistema automatizado de suporte pode trazer reduções nas despesas operacionais relativas à equipa de suporte, uma vez que problemas repetitivos serão resolvidos mais rapidamente. Além disso, esta melhoria na eficiência e na produtividade com o uso da aplicação irá impactar positivamente nos resultados da empresa. Ao investir numa solução que beneficia a experiência do utilizador e otimiza o trabalho da equipa de suporte, os gestores das equipas de desenvolvimento ganham em termos de satisfação dos clientes e eficiência operacional.

4. **Equipa de Desenvolvimento do Capla**

   A equipa de desenvolvimento do Capla será também beneficiada, uma vez que terá o seu fluxo de trabalho mais organizado e eficiente, com feedback mais rápido sobre bugs e outras questões técnicas. Isso permitirá uma resolução mais ágil dos problemas, resultando numa melhoria contínua da aplicação. Embora o impacto sobre esta equipa não seja tão rapidamente sentido quanto para os outros stakeholders, ela terá um interesse substancial no sucesso do projeto, visto que a eficiência do sistema de suporte

contribui para uma utilização mais fluida e satisfatória do Capla, o que reflete diretamente no seu trabalho de desenvolvimento e manutenção da aplicação.

É natural que os principais stakeholders estejam ligados ao Capla, uma vez que será o caso de estudo para este projeto. No entanto, é importante salientar que os stakeholders identificados representam perfis de pessoas ou organizações que podem ter um interesse significativo no desenvolvimento da solução, tanto dentro como fora do contexto específico da Natixis. Existem, certamente, outros indivíduos ou empresas que se encontram em situações semelhantes às dos stakeholders mencionados e que, portanto, podem beneficiar de uma solução de suporte mais eficiente e automatizada. Estes potenciais stakeholders também podem ver neste projeto uma oportunidade de melhorar os seus próprios processos de gestão de suporte, tornando-se, assim, interessados no sucesso da solução desenvolvida.

1.8 Contribuições

Tal como foi referido ao longo da introdução, a principal contribuição desta dissertação reside na proposta de uma solução inovadora que procura melhorar o suporte ao cliente em qualquer tipo de software, resolvendo problemas de falta de acessibilidade e eficiência no atendimento ao utilizador. A solução desenvolvida deverá distinguir-se pela sua facilidade de integração em contextos empresariais já existentes, aproveitando sistemas de gestão de pedidos consolidados, não impondo a adoção de novas plataformas. Ao conjugar funcionalidades como um chatbot baseado em inteligência artificial e um sistema automatizado de gestão de pedidos, pretende aprimorar a forma como o suporte ao cliente é prestado, tornando-o mais ágil, eficiente e eficaz.

Além disso, este trabalho contribui para a expansão do conhecimento relativo a aplicações help desk, ao demonstrar como este tipo de aplicações pode ser a solução para qualquer tipo de problema no suporte ao cliente e nas equipas de suporte. Através do caso de estudo do Capla, será possível fazer uma análise do impacto desta solução num ambiente real, permitindo extrair conclusões que poderão ser aplicadas noutros contextos empresariais e softwares, ampliando assim a aplicabilidade da solução proposta para diversos cenários.

Por fim, caso a empresa opte por adotar a solução desenvolvida, é espectável que haja um impacto notório na vida profissional dos stakeholders envolvidos. A equipa de suporte, os utilizadores e as equipas de desenvolvimento poderão beneficiar de melhorias na qualidade do seu trabalho diário, com processos mais eficientes, redução do tempo de resposta e maior satisfação no atendimento. Isso poderá resultar num ambiente de trabalho mais produtivo e gratificante, impactando positivamente a dinâmica interna da empresa.

1.9 Considerações Éticas

No que diz respeito ao uso de dados pessoais e confidenciais no projeto, foi tido especial cuidado em garantir a conformidade com as regulamentações de proteção de dados, nomeadamente o Regulamento Geral de Proteção de Dados (RGPD). Na eventualidade de a plataforma processar informações pessoais, serão aplicados mecanismos de segurança adequados, como a encriptação, assegurando a confidencialidade, integridade e disponibilidade em conformidade com a legislação em vigor.

Adicionalmente, tendo em conta o enquadramento regulatório emergente na União Europeia, foram consideradas as orientações do AI Act (European Union, 2024), que estabelece requisitos específicos para o desenvolvimento e utilização de sistemas de inteligência artificial, nomeadamente em termos de transparência, robustez técnica, prevenção de riscos e respeito pelos direitos fundamentais.

Considerando ainda que o projeto envolve a aplicação Capla, uma plataforma interna e de carácter restritivo, todo o conteúdo associado foi tratado de forma cuidadosa. Não foram divulgados dados sensíveis ou informações específicas da aplicação, limitando-se a descrição a uma explicação abstrata da lógica geral, de modo a garantir a segurança e a integridade dos dados envolvidos.

Outra dimensão ética relevante prende-se com o uso do chatbot. De acordo com as melhores práticas, sempre que um utilizador interage com o sistema, deve ser informado de forma clara que está a dialogar com uma inteligência artificial, e não com um ser humano. Esta transparência assegura o respeito pelo princípio da honestidade, reforça a confiança dos utilizadores e permite que as suas decisões sejam tomadas de forma informada.

Ao longo do desenvolvimento, foram ainda consideradas boas práticas de ética digital, respeitando os direitos dos utilizadores, prevenindo qualquer forma de discriminação e assegurando que a aplicação não seja utilizada de forma a prejudicar indivíduos ou grupos. A privacidade e a segurança permaneceram prioridades em todas as fases do projeto.

Por fim, para além das questões éticas relacionadas com a aplicação desenvolvida, importa também referir o uso de ferramentas de inteligência artificial no próprio processo de redação desta dissertação. No decurso da escrita desta dissertação, recorreu-se ao ChatGPT[4], como ferramenta de apoio exclusivamente em tarefas de revisão textual e de aperfeiçoamento da expressão escrita. O objetivo desta utilização foi melhorar a clareza, a coerência e a correção gramatical do texto, bem como apoiar a construção frásica de algumas passagens mais complexas. É importante sublinhar que todas as ideias, análises, metodologias, resultados e conclusões apresentadas são da autoria do autor, concebidas e estruturadas de forma autónoma. O uso da inteligência artificial limitou-se a uma função comparável à de um revisor

[4] Modelo de linguagem desenvolvido pela OpenAI, concebido para compreender e gerar texto em linguagem natural.

ou corretor linguístico, não substituindo em momento algum o trabalho intelectual desenvolvido.

Assim, o recurso a inteligência artificial encontra-se devidamente identificado e explicado, em alinhamento com os princípios de integridade académica do P.PORTO e em conformidade com a Declaração de Integridade assinada, que reconhece explicitamente esta utilização como exceção.

2 Estado da Arte

Neste capítulo será feita a revisão do estado da arte, estabelecendo o enquadramento teórico e tecnológico do trabalho desenvolvido. Será descrita a metodologia de pesquisa adotada, com destaque para as fontes e critérios de seleção utilizados. Em seguida, serão explorados os principais conceitos relacionados com software help desk, chatbots e *prompt engineering*, procurando contextualizar os avanços recentes e os desafios ainda presentes. Posteriormente, serão analisados trabalhos académicos e soluções de mercado que permitirão identificar boas práticas e lacunas relevantes para o desenvolvimento deste projeto. Por fim, será apresentada uma revisão das tecnologias consideradas para o protótipo, culminando numa síntese que servirá de base às opções de desenho e implementação apresentadas nos capítulos seguintes.

2.1 Metodologia de Pesquisa

No processo de pesquisa por artigos científicos para realização do estado da arte, foi adotada uma metodologia simples de pesquisa. Esta metodologia consistia em realizar pesquisas utilizando o motor de pesquisa Google Scholar, utilizando queries[5] com o objetivo de limitar o número de resultados obtidos para corresponder ao que realmente se tencionava encontrar. Para além disso foram aplicados filtros para apenas encontrar artigos escritos em português ou inglês, e com não mais de 10 anos.

[5] Consulta ou pesquisa

Na seguinte tabela estão representadas as queries utilizadas na busca por artigos.

Tabela 1 – Queries utilizadas na pesquisa por artigos no Google Scholar

Query	Número de Artigos
{“helpdesk” OR "help desk"} AND {software OR application OR system}	20,000
{"chatbot" OR "conversational agent"} AND {state of the art OR overview OR survey} AND {development OR implementation OR design}	23,900
{"chatbot" OR "conversational agent"} AND {"helpdesk" OR "help desk"} AND {"application" OR "software" OR "system"}	2,510
{"prompt engineering" OR "prompt design"} AND {"chatbot" OR "LLM" OR "conversational AI" OR "conversational agent" OR "support agent"}	16 900

Para fazer uma maior filtragem dos artigos, com o objetivo de manter apenas os de maior relevância, foram consideradas apenas as duas primeiras páginas de artigos. E aos artigos resultantes foi aplicado a estratégia *3-Pass-Aproach*, que permite fazer uma pré-análise do conteúdo dos artigos e com base nas primeiras leituras, determinar o valor científico destes para o trabalho (Keshav, 2007). Com a aplicação desta filtragem e da estratégia *3-Pass Approach*, identificaram-se 13 documentos principais, que serviram de base para a elaboração do estado da arte.

2.2 Conhecimento Atual

Neste tópico será realizado um estudo abrangente sobre os principais conceitos abordados ao longo da dissertação, com o objetivo de apresentar a atualidade dessas tecnologias e abordagens. A análise procura contextualizar os conceitos dentro do panorama atual, proporcionando uma base sólida para o desenvolvimento do trabalho.

Através de uma análise geral do conteúdo da dissertação, é possível destacar dois conceitos macro que servem como base para o projeto em questão: Software Help Desk e Chatbots. O primeiro está associado à gestão e organização de tickets de suporte, enquanto o segundo representa a camada de interação conversacional da solução.

No entanto, dado que o processo de ensinamento do chatbot tem um peso determinante na construção da solução desenvolvida, optou-se por incluir ainda um terceiro subtópico dedicado ao estudo do Prompt Engineering. Este tema, embora intrinsecamente ligado aos chatbots, será explorado de forma autónoma, permitindo um olhar aprofundado sobre as melhores

estratégias para estruturar prompts[6] eficazes, de modo a orientar o comportamento do agente conversacional no contexto do suporte ao cliente.

2.2.1 Software Help Desk

O conceito de help desk está frequentemente associado a serviços que fornecem assistência e informações aos utilizadores, com especial ênfase nos contextos em que redes de computadores são utilizadas. Segundo o Cambridge Dictionary, help desk é definido como "um serviço que fornece informações e ajuda às pessoas, especialmente aquelas que utilizam uma rede de computadores" (Cambridge, 2024). Esta definição sugere que o help desk desempenha um papel central na resolução de problemas e no fornecimento de assistência técnica, com especial relevância no ambiente corporativo e organizacional.

Para entender plenamente o conceito de software help desk, é importante analisar a sua função e evolução. Em muitas organizações, o help desk é um componente vital da infraestrutura de suporte ao cliente e aos utilizadores internos. Ele oferece não só apoio técnico, mas também suporte administrativo, com o objetivo de resolver os problemas que surgem enquanto os utilizadores interagem com os recursos e sistemas da organização. De acordo com Masongsong e Damian (2016), o help desk funciona como um centro de atendimento ao cliente, com o objetivo de fornecer soluções rápidas e eficazes. Dependendo do porte da organização, um sistema help desk pode ser operado por um único profissional ou por uma equipa inteira de suporte. (Masongsong & Damian, 2016)

A principal função do software help desk é permitir que as organizações centralizem e automatizem o processo de resolução de problemas reportados pelos utilizadores. Estes problemas podem ser submetidos através de diversos canais de comunicação, como e-mail, telefone ou chat. O software ajuda a gerir e rastrear solicitações (também conhecidas como "*tickets*"), a classificar os problemas conforme a sua urgência e complexidade, a direcionar os tickets para os agentes adequados e a fornecer uma base de dados com informações úteis para a resolução de problemas (Masongsong & Damian, 2016). A utilização deste tipo de software assegura que todos os problemas sejam tratados de forma organizada e eficiente, além de proporcionar aos gestores uma visão clara sobre o desempenho da equipa de suporte.

A implementação de sistemas automatizados de help desk tem sido uma tendência crescente nas empresas, pois estes sistemas oferecem uma série de vantagens. Eles permitem reduzir os custos operacionais, melhorar a produtividade e aumentar a satisfação do cliente. Segundo Masongsong e Damian (2017), sistemas baseados em inteligência artificial (IA) estão a ser cada vez mais utilizados em help desks. Estes sistemas utilizam técnicas como o raciocínio baseado em casos e sistemas especialistas para fornecer soluções automatizadas aos problemas mais

[6] Instrução ou conjunto de instruções fornecidas a um modelo de inteligência artificial, com o objetivo de orientar a sua resposta.

comuns, sem a necessidade de intervenção humana. Além disso, a IA e o *machine learning*[7] têm sido incorporados aos sistemas de help desk para otimizar a classificação e priorização dos tickets, bem como melhorar a comunicação entre os agentes de suporte e os utilizadores (Masongsong & Damian, 2017).

Outro benefício significativo dos softwares help desk é a capacidade de análise de desempenho. Muitas dessas ferramentas permitem gerar relatórios que ajudam a medir a eficácia do atendimento, avaliando o desempenho da equipa de suporte com base em indicadores-chave de desempenho (KPIs). A análise contínua do desempenho da equipa de suporte contribui para a melhoria da qualidade do serviço prestado. Ferramentas como estas permitem às organizações recolher dados essenciais, como o tempo médio de resolução dos problemas, o número de incidentes não resolvidos e outras métricas, que podem ser usadas para aprimorar os processos de atendimento e melhorar a satisfação do cliente (Al-Hawari & Barham, 2021)

No entanto, os sistemas de help desk não estão isentos de desafios. A dependência de tecnologia pode ser uma barreira significativa, especialmente quando o sistema falha ou não funciona corretamente, o que pode comprometer a capacidade de resposta da equipa de suporte (Al-Hawari & Barham, 2021). Já quanto a automação, embora esta ajude a reduzir custos e aumentar a eficiência, ela pode também prejudicar a personalização do atendimento ao cliente, uma vez que muitos utilizadores podem preferir interagir com um atendente humano em vez de receber respostas automáticas (Masongsong & Damian, 2016). Outro ponto importante é que os sistemas de help desk exigem manutenção constante e atualizações periódicas para garantir que continuem a funcionar de maneira eficiente, o que pode representar custos adicionais para as organizações (Masongsong & Damian, 2016).

No geral, os softwares de help desk são ferramentas essenciais para organizações que procuram garantir um atendimento de alta qualidade a clientes e utilizadores internos. Esses sistemas não só aumentam a eficiência operacional, mas também contribuem para a melhoria da satisfação do cliente, proporcionando um atendimento mais rápido e organizado. A integração de tecnologias emergentes, como a inteligência artificial, continua a evoluir nos sistemas de help desk, tornando-os mais ágeis, precisos e capazes de oferecer um atendimento mais personalizado (Masongsong & Damian, 2016).

2.2.2 Chatbots

Origem e Definição:
A origem e a definição dos chatbots estão intrinsecamente ligadas aos avanços da inteligência artificial (IA) e da interação homem-máquina. O termo chatbot deriva de "chatterbot", uma expressão criada por Michael Mauldin em 1997 para descrever programas que possibilitavam conversas entre humanos e máquinas (Luo, et al., 2021). Os chatbots são, essencialmente, agentes conversacionais que utilizam técnicas de *Natural Language Processing* (Processamento

[7] Ramo da inteligência artificial que permite aos sistemas aprenderem a partir de dados e melhorarem o seu desempenho sem serem explicitamente programados para cada tarefa.

de Linguagem Natural) para interpretar e responder em linguagem humana, criando interações que tentam simular um diálogo natural (Adamopoulou & Moussiades, 2020).

A evolução dos chatbots teve início com sistemas simples, como o ELIZA e o ALICE, que empregavam técnicas de reconhecimento de padrões. Embora esses sistemas fossem pioneiros, a sua capacidade de gerar respostas era limitada e sem memória contextual, o que dificultava conversas mais fluídas e adaptativas. Com o avanço da tecnologia, novas abordagens foram sendo desenvolvidas, incorporando técnicas como redes neurais, aprendizado por reforço e sistemas híbridos. Esses avanços permitiram criar chatbots mais sofisticados, capazes de fornecer respostas mais personalizadas e adaptativas às necessidades dos utilizadores (Luo, et al., 2021).

Além disso, a crescente evolução das interfaces multimodais ampliou as possibilidades de interação com os chatbots. Ao integrar texto, voz e até sinais visuais, esses sistemas conseguiram expandir seu alcance e flexibilidade, permitindo interações mais dinâmicas e imersivas (Adamopoulou & Moussiades, 2020). O crescimento no uso de chatbots é atribuído à sua capacidade de realizar uma vasta gama de tarefas, desde o atendimento ao cliente até serviços de saúde e educação, proporcionando soluções rápidas, acessíveis e eficientes para diversos setores.

Com o tempo, os chatbots tornaram-se uma ferramenta essencial em diversos campos, sendo amplamente utilizados em empresas de *e-commerce*[8], como assistentes virtuais, sistemas de suporte ao cliente, e até mesmo em contextos educacionais e de saúde, devido à sua capacidade de automatizar tarefas repetitivas e interagir com utilizadores de forma inteligente e eficiente (Adamopoulou & Moussiades, 2020).

NLP - *Natural Language Processing*:

Natural Language Processing (NLP) é uma área da inteligência artificial (IA) dedicada à interação entre computadores e seres humanos utilizando a linguagem natural. Essencialmente, o NLP permite que os chatbots compreendam, processem e respondam adequadamente a entradas em linguagem humana, seja por texto ou voz (Adamopoulou & Moussiades, 2020).

No contexto dos chatbots, o NLP é crucial para a construção de interfaces conversacionais eficazes. Ele possibilita que o chatbot "entenda" a intenção do utilizador, extraia entidades relevantes e forneça respostas contextualmente apropriadas. O processo de NLP nos chatbots envolve diversas técnicas, que vão desde a tokenização[9] e análise sintática até métodos mais avançados, como *embeddings*[10] de palavras e análise semântica (Luo, et al., 2021).

[8] Compra e venda de produtos ou serviços através da internet.

[9] Processo de dividir um texto em unidades menores, chamadas *tokens* (como palavras ou frases), para facilitar a sua análise computacional.

[10] Representações numéricas de palavras ou frases em vetores, que captam relações de significado e contexto entre elas.

Uma das técnicas mais populares em NLP é o *word embedding*, que transforma palavras em vetores de alta dimensão, permitindo que as palavras com significados semelhantes fiquem próximas no espaço vetorial. Modelos como o Word2Vec (Mikolov, et al., 2013) e o BERT (Devlin, et al., 2019) são usados para representar palavras ou frases de maneira que o chatbot consiga compreender o seu significado em diferentes contextos (Luo, et al., 2021). A utilização de *sentence embedding*, que gera representações vetoriais de frases inteiras, é outra abordagem avançada que permite a compreensão do contexto completo da comunicação do utilizador, superando algumas limitações dos métodos de *word embedding*.

Além disso, técnicas de Natural Language Understanding (NLU), que são fundamentais para a interpretação dos *inputs*[11] dos utilizadores, também desempenham um papel vital. O NLU envolve a identificação da intenção do utilizador (*intent*) e a extração de entidades relevantes (*slot-filling*). Através dessa análise, o chatbot consegue compreender o que o utilizador deseja, como, por exemplo, se ele está a procurar informações, fazendo uma solicitação ou efetuando uma compra. A deteção de intenções é frequentemente abordada como uma tarefa de classificação de sentenças, onde o modelo prevê um ou mais rótulos de intenção com base no texto fornecido (Adamopoulou & Moussiades, 2020).

Outro aspeto importante é o uso de *topic modeling*, como o modelo Latent Dirichlet Allocation (LDA), para identificar e extrair tópicos-chave das conversas. Isso ajuda os chatbots a entenderem melhor o conteúdo subjacente das interações, o que, por sua vez, melhora a geração de respostas mais precisas e relevantes (Luo, et al., 2021).

Com o avanço das tecnologias de NLP, os chatbots estão a tornar-se cada vez mais sofisticados, não apenas nas suas habilidades de diálogo, mas também na sua capacidade de realizar tarefas complexas. A evolução do NLP permite que os chatbots operem numa gama de domínios e cenários, desde o atendimento ao cliente até assistentes pessoais e educacionais. As futuras inovações nessa área continuarão a refinar a capacidade dos chatbots de interagir de forma mais natural e intuitiva com os utilizadores, aumentando a sua eficiência e efetividade em diversas aplicações (Adamopoulou & Moussiades, 2020) (Luo, et al., 2021).

Machine Learning:

O conceito de *Machine Learning* (ML) é fundamental na evolução e desenvolvimento de chatbots, permitindo que estas ferramentas automatizadas melhorem a sua capacidade de interagir com os utilizadores e de fornecer respostas mais precisas e adaptadas ao contexto. Os chatbots modernos têm vindo a integrar cada vez mais algoritmos de aprendizado de máquina, o que lhes confere uma maior flexibilidade e eficiência na interação com os utilizadores. Esses sistemas são construídos sobre técnicas como redes neurais[12] profundas, aprendizado por reforço e redes neurais recorrentes (RNN), que são essenciais para o seu desenvolvimento, uma vez que permitem gerar respostas a partir de grandes volumes de dados e interações passadas,

[11] Entradas de dados.

[12] Modelos computacionais inspirados no cérebro humano, formados por unidades interligadas ("neurónios artificiais") capazes de aprender padrões a partir de dados.

em vez de simplesmente recuperar respostas pré-definidas de bases de dados estáticas (Luo, et al., 2021).

As redes neurais, como as RNNs, têm sido amplamente utilizadas na criação de chatbots, especialmente para sistemas de resposta generativa, onde o modelo pode gerar respostas novas com base nas entradas dos utilizadores. Essas abordagens são favorecidas pela sua capacidade de lidar com sequências de dados, como textos e discursos, o que as torna adequadas para aplicações como assistentes virtuais e serviços de atendimento ao cliente (Adamopoulou & Moussiades, 2020). A utilização de *Reinforcement Learning* [13](RL) também tem ganho destaque, permitindo que os chatbots aprendam com as interações e ajustem as suas respostas com base no *feedback* recebido, de modo a melhorar a experiência do utilizador ao longo do tempo (Luo, et al., 2021).

Em termos de aplicação, a aprendizagem de máquina tem permitido aos chatbots evoluir e se adaptar a diversos contextos, desde o atendimento ao cliente até a assistência pessoal e sistemas de saúde. Com o uso de algoritmos que analisam grandes volumes de dados, os chatbots são capazes de melhorar a personalização das suas respostas e oferecer uma experiência de utilizador mais dinâmica e eficiente (Luo, et al., 2021). Este avanço, no entanto, vem com o desafio de lidar com grandes quantidades de dados e garantir a precisão e a relevância das respostas geradas.

Modelos de Linguagem de Grande Escala (LLMs):

Nos últimos anos, os Modelos de Linguagem de Grande Escala (LLMs) transformaram de forma significativa a forma como os chatbots são concebidos e utilizados. Diferentemente dos sistemas tradicionais baseados em regras ou em reconhecimento de intenções, os LLMs são modelos treinados em vastos volumes de dados textuais, o que lhes permite compreender e gerar linguagem natural com elevado grau de fluidez e contextualização (Daeseung , et al., 2023).

A principal inovação destes modelos reside na capacidade de realizar *in-context learning*, ou seja, aprender padrões e responder adequadamente a partir de exemplos fornecidos diretamente no *prompt*, sem necessidade de treino adicional para cada tarefa. Isto confere aos chatbots uma flexibilidade inédita, permitindo-lhes lidar com uma variedade maior de cenários e adaptar-se a diferentes contextos de forma dinâmica (Liat , et al., 2024).

No entanto, a adoção de LLMs também introduz novos desafios. O seu funcionamento exige uma quantidade significativa de recursos computacionais, o que implica custos elevados de operação e manutenção. Além disso, questões relacionadas com a fiabilidade das respostas, possíveis enviesamentos nos dados de treino e preocupações éticas sobre segurança e privacidade estão no centro dos debates atuais (Christine, et al., 2024).

[13] Técnica de aprendizagem automática em que um agente aprende a tomar decisões por tentativa e erro, recebendo recompensas ou penalizações conforme as suas ações.

Outro fator relevante é a crescente importância do Prompt Engineering na utilização de LLMs. Estudos recentes mostram que a qualidade do *prompt* — incluindo clareza, estrutura e até formatos específicos como JSON[14] ou YAML[15] — influencia diretamente a eficácia das respostas, podendo inclusive compensar limitações de modelos mais pequenos (Pawlik, 2025). Assim, mais do que apenas escolher o modelo, torna-se crucial saber como interagir com ele para extrair o melhor desempenho possível.

A escolha do LLM é, por isso, determinante para o funcionamento do chatbot. Modelos diferentes apresentam variações em termos de custo, velocidade de resposta, capacidade de manter contexto e suporte a diferentes idiomas. De acordo com estudos recentes, LLMs como o GPT-4, o LLaMA e o Mistral têm sido avaliados em múltiplos cenários de utilização, demonstrando diferenças significativas em termos de desempenho e requisitos de recursos (Daeseung , et al., 2023) (Pawlik, 2025). A decisão sobre qual LLM utilizar impacta diretamente a experiência do utilizador e os requisitos de infraestrutura da solução, sendo este um tema que será aprofundado na secção 3.4.2 através de um estudo comparativo dedicado.

Em suma, os LLMs representam um salto qualitativo na evolução dos chatbots, ampliando substancialmente o seu potencial de aplicação. Apesar dos desafios técnicos e éticos que levantam, estes modelos consolidaram-se como a base da nova geração de agentes conversacionais, sendo fundamentais para sistemas de suporte ao cliente, educação, saúde e diversos outros setores.

Vantagens da utilização de Chatbots no Suporte ao Cliente:

A utilização de chatbots no suporte ao cliente tem vindo a crescer devido às suas inúmeras vantagens, tanto para as empresas quanto para os utilizadores. Um dos principais benefícios é a eficiência operacional. Os chatbots são capazes de realizar tarefas repetitivas e de responder a perguntas frequentes de forma automatizada, o que reduz significativamente a carga de trabalho dos atendentes humanos (Luo, et al., 2021). Essa automação não só melhora a produtividade, mas também diminui o tempo de resposta, proporcionando uma experiência mais rápida para os utilizadores.

Além disso, os chatbots operam 24 horas por dia, 7 dias por semana, permitindo que as empresas ofereçam suporte contínuo, independentemente da localização geográfica ou do horário. Este nível de disponibilidade aumenta a satisfação do cliente, especialmente em mercados globais e em situações que exigem suporte imediato (Adamopoulou & Moussiades, 2020). A capacidade de atendimento constante ajuda a criar uma sensação de proximidade e confiança com a marca.

[14] Formato de serialização de dados, baseado em texto simples, muito usado em APIs e comunicação entre sistemas.

[15] Formato de serialização de dados, semelhante ao JSON, mas com sintaxe mais legível, usado sobretudo em ficheiros de configuração.

Outra vantagem importante é a personalização da interação. Utilizando técnicas avançadas de processamento de linguagem natural (PNL), os chatbots conseguem compreender melhor a intenção do utilizador, adaptando as respostas de acordo com o contexto e as necessidades individuais. Esta personalização, aliada à capacidade de aprender com interações passadas, torna a experiência mais relevante e precisa, melhorando a eficiência do suporte ao cliente (Luo, et al., 2021).

Além disso, a integração de chatbots com outras plataformas e serviços oferece uma experiência *omnicanal*[16]. Eles podem ser usados em diversas plataformas de comunicação, como redes sociais, sites e aplicações de mensagens, permitindo uma experiência fluida e sem interrupções para os utilizadores (Adamopoulou & Moussiades, 2020).

Por fim, os chatbots contribuem para a redução de custos operacionais. Com a automação de tarefas simples e repetitivas, as empresas podem redirecionar recursos humanos para tarefas mais complexas, além de reduzir os custos com pessoal dedicado ao atendimento (Luo, et al., 2021). A implementação de chatbots, portanto, resulta numa solução economicamente vantajosa, sem comprometer a qualidade do atendimento ao cliente.

Em resumo, a utilização de chatbots no suporte ao cliente proporciona uma série de vantagens, incluindo a melhoria na eficiência operacional, disponibilidade contínua, personalização, integração *omnicanal* e redução de custos, fatores que fazem desta tecnologia uma ferramenta essencial para empresas que procuram oferecer um serviço de atendimento ágil e eficaz.

Desvantagens da Utilização de Chatbots no Suporte ao Cliente:

Apesar das vantagens dos chatbots, como a automação de tarefas repetitivas e a redução de custos operacionais, a sua utilização no suporte ao cliente também apresenta várias desvantagens que devem ser consideradas.

Primeiramente, a qualidade das respostas pode ser comprometida, especialmente em sistemas baseados em *pattern matching* ou reconhecimento de padrões. Esses chatbots geram respostas fixas e previsíveis que podem não ser adequadas em situações complexas ou inesperadas, resultando em interações que parecem robotizadas e que carecem de um toque humano (Adamopoulou & Moussiades, 2020). Isso é particularmente problemático em contextos de suporte, onde a empatia e a compreensão das nuances do problema são cruciais para uma resolução eficaz.

Além disso, muitos chatbots não possuem memória de longo prazo, o que significa que cada interação é tratada como uma nova conversa. Isso pode gerar frustração entre os utilizadores, que terão de repetir informações ou problemas já abordados em sessões anteriores (Luo, et al., 2021). A falta de continuidade nas interações pode impactar negativamente a experiência do

[16] Estratégia de comunicação e atendimento que integra vários canais, permitindo ao utilizador uma experiência contínua e consistente, independentemente do meio escolhido.

cliente, que pode sentir que está a falar com uma "máquina" e não com um sistema que entende as suas necessidades.

Outro ponto importante é a limitação das capacidades dos chatbots em lidar com questões mais complexas ou emocionais. Embora os chatbots possam ser eficientes para resolver questões simples, como consultas de status ou informações de produtos, eles frequentemente falham em responder adequadamente a problemas mais complicados ou em entender o contexto emocional do cliente, o que é uma falha crítica em ambientes de suporte onde a empatia é muitas vezes fundamental (Adamopoulou & Moussiades, 2020).

Por fim, a dependência de tecnologias de Processamento de Linguagem Natural (NLP) pode levar a erros de interpretação. O reconhecimento de intenções e entidades em entradas de linguagem natural, especialmente quando estas são ambíguas ou informais, pode resultar em respostas inadequadas ou até em interações incompletas, prejudicando a eficácia do atendimento ao cliente (Luo, et al., 2021).

Em suma, enquanto os chatbots oferecem inúmeras vantagens em termos de eficiência e custo, as suas limitações em termos de flexibilidade, empatia e capacidade de resolução de problemas complexos devem ser cuidadosamente avaliadas ao considerar a sua utilização no suporte ao cliente.

2.2.3 Prompt Engineering

O conceito de prompt engineering surgiu como uma prática essencial para o uso eficaz de modelos de linguagem de grande escala. Ao contrário dos sistemas tradicionais de processamento de linguagem natural, onde o desempenho dependia fortemente de treino supervisionado e de grandes volumes de dados rotulados, os modelos recentes podem ser orientados para desempenhar tarefas distintas através da simples formulação de instruções adequadas. Isto torna o prompt a principal interface entre o utilizador e o modelo, e o seu desenho influencia de forma direta a qualidade das respostas obtidas (Daeseung , et al., 2023).

O interesse académico e industrial por esta área deve-se ao facto de que pequenas alterações no prompt podem produzir variações significativas nos resultados gerados. A clareza, a forma e até a ordem da informação têm impacto no desempenho. Estudos mostram que técnicas de prompt engineering podem melhorar a precisão de um chatbot em mais de 10% quando comparado a instruções mal estruturadas (Daeseung , et al., 2023). Mais do que apenas obter boas respostas, a prática contribui para a consistência, reduz ambiguidades e ajuda a alinhar o comportamento do modelo com o objetivo pretendido pelo sistema.

Um dos pontos mais relevantes discutidos na literatura é que prompt engineering não é apenas um meio de melhorar o desempenho, mas também uma estratégia de otimização de recursos. Há evidência de que modelos mais pequenos, quando acompanhados de prompts bem construídos, podem alcançar resultados comparáveis a modelos muito maiores, representando

uma solução mais económica e eficiente (Liat , et al., 2024). Isto é particularmente importante no contexto empresarial, onde os custos de operação são um fator determinante para a viabilidade de soluções baseadas em inteligência artificial.

Diversas técnicas foram propostas e estudadas para estruturar *prompts* de forma eficaz. Entre as mais conhecidas está o *few-shot prompting*, que consiste em fornecer ao modelo alguns exemplos de entradas e saídas desejadas, ajudando-o a compreender o padrão da tarefa. Há também o *zero-shot prompting*, onde o modelo recebe apenas a descrição da tarefa, e o *chain-of-thought prompting*, que encoraja o modelo a detalhar o raciocínio passo a passo, melhorando a precisão em problemas complexos (Daeseung , et al., 2023).

Além destas abordagens gerais, foram desenvolvidas técnicas mais específicas que procuram sistematizar a construção dos prompts. Um exemplo é o *Query Transformation Module* (QTM), que transforma o prompt em partes distintas, identificando objetivos e pontos-chave para torná-los mais compreensíveis para o modelo. Essa técnica mostrou melhorias claras na qualidade das respostas, especialmente em contextos conversacionais (Daeseung , et al., 2023). Outro contributo relevante é o *Conversational Prompt Engineering* (CPE), que utiliza interações curtas entre o utilizador e um modelo auxiliar para refinar o prompt. Neste processo, o utilizador fornece dados não rotulados e vai ajustando as instruções com base no feedback do próprio modelo, criando instruções mais precisas e adaptadas ao caso específico (Liat , et al., 2024).

Outro aspeto que tem ganho destaque é a utilização de formatos estruturados, como JSON ou YAML, na definição de saídas. Estes formatos permitem que o modelo produza respostas organizadas e fáceis de interpretar por outros sistemas, o que se torna essencial em soluções integradas, como é o caso de chatbots conectados a plataformas de help desk. Estudos mostram que a adoção destes formatos pode aumentar significativamente a precisão das respostas e reduzir erros de interpretação (Pawlik, 2025).

No contexto dos chatbots de suporte ao cliente, a relevância do prompt engineering é ainda maior. Não basta que o modelo produza uma resposta correta em termos linguísticos, é necessário que ela seja consistente, adequada ao tom da empresa e funcional para o processo de suporte. A literatura reforça que prompts bem desenhados são fundamentais para manter a coerência do agente, gerir interações com utilizadores insatisfeitos e evitar respostas vagas ou desalinhadas com o objetivo do atendimento (Pawlik, 2025).

Com base nesta revisão, a estrutura de prompt a ser adotada nesta dissertação foi desenhada para refletir as melhores práticas identificadas na literatura, enquanto responde às necessidades específicas de um chatbot de suporte. A sua constituição baseia-se em sete componentes:

- Identidade e propósito: corresponde à definição clara do papel do agente no sistema e à responsabilidade principal que lhe cabe. A clareza de instruções é apontada como fator determinante para o desempenho dos modelos (Daeseung , et al., 2023).

- Voz e persona: define o tom de comunicação, estilo e postura do agente. A consistência estilística é destacada em estudos sobre Conversational Prompt Engineering, que mostram como o ajuste iterativo de preferências de escrita contribui para respostas mais satisfatórias (Liat , et al., 2024).

- Fluxo conversacional: organiza a sequência de passos de cada interação (início, identificação do pedido, tratamento e encerramento). Esta abordagem aproxima-se de técnicas como o QTM, que sugerem a decomposição de instruções em objetivos e passos claros (Daeseung , et al., 2023).

- Diretrizes de resposta: apresentam regras para garantir consistência e clareza, e orientar o comportamento perante situações sensíveis, como lidar com utilizadores insatisfeitos. O uso de regras explícitas tem paralelo na literatura, que mostra ganhos quando comandos são formulados de forma precisa (Pawlik, 2025).

- Funcionalidade específica: descreve a tarefa concreta do agente, tal como responder a dúvidas ou registar pedidos. Estudos indicam que a definição clara da tarefa é essencial para que o modelo compreenda corretamente a intenção (Liat , et al., 2024).

- Cenários de interação: fornecem exemplos típicos que o agente deve saber gerir. Isto remete diretamente à técnica de few-shot prompting, que utiliza exemplos para guiar o comportamento do modelo (Daeseung , et al., 2023) (Liat , et al., 2024).

- Ações de encaminhamento: definem as variáveis a preencher e os caminhos a seguir quando é necessário redirecionar a conversa ou criar um ticket. A literatura mostra que instruções estruturadas, incluindo formatos como JSON, ajudam a reduzir erros nestes processos (Pawlik, 2025).

Esta estrutura garante uma base comum a todos os agentes de suporte desenvolvidos, promovendo previsibilidade e consistência, mas ao mesmo tempo permite a personalização para cada módulo específico. A sua adoção responde tanto às orientações presentes na literatura estudada como às exigências práticas de um sistema de suporte ao cliente, que requer clareza, fiabilidade e eficiência na interação.

2.3 Trabalhos Existentes

Nesta secção serão analisados estudos académicos e as principais aplicações de help desk disponíveis no mercado, com o objetivo de identificar práticas, características e desafios relevantes para o desenvolvimento da solução proposta. A primeira parte irá incidir sobre a análise de uma tese de mestrado e de um artigo científico que descrevem o desenvolvimento e a avaliação de sistemas de help desk. A segunda parte irá examinar aplicações de help desk disponíveis no mercado, realçando funcionalidades principais, usabilidade, integrações e aspetos de inovação. Esta estrutura permitirá articular a perspetiva académica com a prática do mercado e fornecer uma base sólida para as recomendações apresentadas nesta dissertação.

2.3.1 Artigos Científicos Semelhantes

Nesta secção, serão analisados dois artigos científicos selecionados devido à sua relevância e afinidade com o tema da dissertação, ambos focados no desenvolvimento de sistemas de help desk. O objetivo é compreender as abordagens utilizadas, as soluções propostas e os desafios enfrentados no processo de criação e implementação desses sistemas. A análise de cada artigo será estruturada nos seguintes pontos:

- Resumo do Trabalho
- Tecnologias e Ferramentas Utilizadas no Desenvolvimento
- Funcionalidades Help Desk Desenvolvidas
- Resultados Obtidos
- Desafios e Limitações
- Conclusões Obtidas

Esta análise permitirá identificar as melhores práticas, desafios comuns e soluções inovadoras, oferecendo uma base sólida para o desenvolvimento do sistema de help desk proposto nesta dissertação.

2.3.1.1 Tese de Mestrado

O primeiro documento a ser analisado será uma tese de mestrado intitulada “Desenvolvimento de uma Aplicação Help Desk para o Redmine”, escrita por Bc. Jakub Lukačín. A tese foi elaborada sob a supervisão de Ing. Oldřich Malec e foi aprovada eletronicamente em 2023.

Resumo do Trabalho:

O documento aborda a criação de uma aplicação help desk que opera em conjunto com o Redmine, um software amplamente utilizado para a gestão de projetos. A tese inicia com uma introdução ao Redmine e ao conceito de help desk, seguida pela análise das necessidades dos utilizadores e das tecnologias que serão utilizadas na implementação. O resumo do trabalho é apresentado nas páginas iniciais, onde se destaca que "a aplicação visa não apenas facilitar a gestão de solicitações, mas também melhorar a comunicação entre os utilizadores e a equipa

de suporte". O documento detalha as fases de design, implementação, testes de utilizadores e as perspetivas futuras do projeto, destacando a importância da internacionalização e da usabilidade para utilizadores sem conhecimento técnico.

Tecnologias e Ferramentas Utilizadas no Desenvolvimento:

O desenvolvimento da aplicação help desk utilizou uma combinação de tecnologias modernas que garantem eficiência e escalabilidade. As principais ferramentas incluem:

- **Vue.js**: Uma *framework*[17] progressiva para construção de interfaces de utilizador, escolhida pela sua flexibilidade e facilidade de integração. O autor menciona que "o Vue se destacou entre outras *frameworks* pela sua simplicidade e capacidade de criar interfaces reativas".
- **Vuetify**: Uma biblioteca de componentes que fornece uma vasta gama de elementos de interface prontos para uso, permitindo um design responsivo e atraente. "A utilização do Vuetify permitiu acelerar o desenvolvimento da interface, garantindo um visual moderno e funcional".
- **Express.JS**: Uma framework para Node.js que facilita a criação de aplicações web robustas e escaláveis, utilizado para desenvolver a API do back-end[18]. O autor destaca que "o Express foi fundamental para a construção de uma API que atende às necessidades do front-end[19] de forma eficiente".
- **PostgreSQL**: Um sistema de gestão de base de dados relacional, escolhido pela sua robustez e suporte a operações complexas. "A escolha do PostgreSQL deu-se pela sua capacidade de lidar com grandes volumes de dados e pela sua fiabilidade".
- **Docker**: Utilizado para *containerização*[20], permitindo que a aplicação seja facilmente implantada e escalada em diferentes ambientes. "A *containerização* com Docker simplificou o processo de implantação e garantiu que a aplicação funcionasse de forma consistente em diferentes ambientes".

Funcionalidades Desenvolvidas e Qualidades de Destaque:

A aplicação help desk desenvolvida possui diversas funcionalidades e qualidades a destacar que visam melhorar a experiência do utilizador e a eficiência do suporte. Entre elas estão:

- **Autenticação de Utilizadores**: Sistema de login seguro que permite o acesso a funcionalidades específicas do help desk. "A autenticação foi implementada utilizando tokens JWT, garantindo segurança e escalabilidade".

[17] Conjunto estruturado de ferramentas, bibliotecas e boas práticas que facilita e acelera o desenvolvimento de software, fornecendo uma base já preparada sobre a qual os programadores podem construir aplicações.
[18] Parte de um sistema ou aplicação responsável pelo processamento interno, lógica de negócio e gestão de dados, geralmente não visível para o utilizador final.
[19] Parte de um sistema ou aplicação responsável pela interface e interação direta com o utilizador, ou seja, tudo aquilo que é visível e manipulável no ecrã.
[20] Processo de empacotar uma aplicação e as suas dependências em "contentores" de software, garantindo que esta seja executada de forma consistente em diferentes ambientes.

- **Criação e Gestão de Pedidos**: Os utilizadores podem abrir, visualizar e gerir as suas solicitações de suporte de forma intuitiva. "A interface permite que os utilizadores acompanhem o status das suas solicitações em tempo real".
- **Internacionalização**: Suporte a múltiplos idiomas, permitindo que utilizadores de diferentes regiões utilizem a aplicação na sua língua nativa. "A internacionalização foi uma prioridade desde o início do projeto, visando atender a um público diversificado".
- **Integração com Redmine**: Sincronização de dados entre o help desk e o Redmine, facilitando o acompanhamento de problemas e solicitações. "A integração com o Redmine foi um dos principais diferenciais da aplicação, permitindo uma gestão mais eficiente".
- **Interface Amigável**: Design focado na usabilidade, permitindo que utilizadores sem conhecimento técnico naveguem facilmente pela aplicação. "O design foi testado com utilizadores reais, resultando num feedback positivo sobre a facilidade de uso".

Resultados Obtidos:

Os resultados obtidos com a implementação do help desk foram positivos. O protótipo foi implantado com sucesso, permitindo a realização de testes de utilizador que forneceram feedback valioso. Os utilizadores relataram uma melhoria significativa na facilidade de uso e na eficiência do processo de suporte. "Os testes de utilizador mostraram que 85% dos participantes consideraram a aplicação fácil de usar". Além disso, a integração com o Redmine foi bem-sucedida, permitindo que as equipas de suporte gerissem solicitações de forma mais eficaz. O projeto atendeu aos objetivos propostos, demonstrando a viabilidade da solução desenvolvida. "A aplicação não só atendeu às expectativas, mas também abriu portas para futuras melhorias e funcionalidades".

Desafios e Limitações:

Durante o desenvolvimento da aplicação, diversos desafios e limitações foram enfrentados. Entre os principais desafios estão:

- **Complexidade da Integração**: A integração com o Redmine apresentou dificuldades técnicas, exigindo um entendimento profundo da API do Redmine e das suas funcionalidades. "A documentação da API do Redmine foi um desafio, pois nem sempre estava clara".
- **Gestão de Expectativas dos Utilizadores**: Alinhar as expectativas dos utilizadores com as funcionalidades disponíveis foi um desafio, especialmente num ambiente onde as necessidades podem variar amplamente. "Foi necessário realizar várias reuniões para garantir que todos estivessem na mesma página".
- **Testes de Utilizador**: A realização de testes de utilizador foi limitada pelo número de participantes disponíveis, o que pode ter impactado a abrangência do feedback recolhido. "Embora tenhamos realizado testes, o número limitado de participantes pode não refletir a totalidade das necessidades dos utilizadores".

Conclusões Obtidas:

A análise desta tese permite concluir que a implementação de uma aplicação help desk integrada no Redmine teve um impacto claramente positivo na gestão de pedidos e na experiência dos utilizadores. O sistema não só facilitou a comunicação entre utilizadores e equipas de suporte, como também melhorou a organização e o acompanhamento das solicitações. Destaca-se, em particular, a valorização da usabilidade e da internacionalização, fatores que aumentaram a acessibilidade da aplicação e a sua aceitação pelos utilizadores.

Do ponto de vista desta dissertação, importa ainda salientar as preocupações a ter em conta no desenvolvimento de uma solução semelhante: a complexidade da integração com plataformas existentes, que exige uma boa compreensão das APIs disponíveis, e a necessidade de alinhar as funcionalidades com as expectativas reais dos utilizadores. Estes pontos reforçam que a eficácia de uma aplicação help desk não depende apenas da tecnologia utilizada, mas também da sua capacidade de se adaptar ao contexto e às necessidades concretas da organização onde é implementada.

2.3.1.2 Artigo Científico

O segundo documento a ser analisado será o artigo científico intitulado "A machine learning based help desk system for IT service management", escrito por Feras Al-Hawari e Hala Barham, ambos da Universidade Germano-Jordaniana (GJU), localizada em Amã, Jordânia. O artigo foi submetido para revisão em 10 de janeiro de 2019, passou por um processo de revisão até 12 de março de 2019, e foi finalmente aceito em 1 de abril de 2019, com a sua disponibilização online a partir de 2 de abril de 2019.

Resumo do Trabalho:

O trabalho apresenta um sistema de help desk que serve como um ponto único de contacto entre os utilizadores e a equipa de TI, abordando a necessidade crescente de soluções eficazes na gestão de serviços de TI. Os autores explicam que "o sistema foi desenvolvido internamente para atender às necessidades contínuas de serviços de TI da GJU". O foco principal do estudo reside na metodologia utilizada para gerar um modelo de aprendizagem de máquina que classifica os tickets de help desk de forma precisa, o que resulta na redução do tempo de resolução e no aumento da satisfação do utilizador. Além disso, o sistema facilita a definição de processos de negócios claros e a medição de KPIs necessários para avaliar o desempenho da equipa de TI e dos processos. Como mencionado, "a utilização de um sistema de help desk bem estruturado pode levar a um aumento significativo na produtividade e na qualidade do serviço prestado".

Tecnologias e Ferramentas Utilizadas no Desenvolvimento:

O desenvolvimento do sistema de help desk envolveu várias tecnologias e ferramentas que foram fundamentais para a sua implementação. Os autores destacam que "o uso de algoritmos de aprendizagem de máquina, especificamente um modelo baseado em SVM (Support Vector Machine), foi crucial para a classificação automática dos tickets". Além disso, o sistema foi

projetado para ser integrado ao portal de funcionários da GJU, permitindo que todos os colaboradores acedessem aos serviços online com as mesmas credenciais de login. A automação de notificações por e-mail e a capacidade de gerar relatórios para medir KPIs foram implementadas como parte das funcionalidades do sistema. Como afirmam os autores, "a integração de tecnologias modernas não só melhora a eficiência, mas também proporciona uma melhor experiência ao utilizador".

Funcionalidades Desenvolvidas e Qualidades de Destaque:

O sistema de help desk desenvolvido possui várias funcionalidades essenciais que visam melhorar a gestão de serviços de TI. Entre as principais funcionalidades, destacam-se:

- **Gestão de Tickets**: O sistema permite a abertura, classificação, priorização, atribuição, resolução, documentação, arquivamento e fechamento de incidentes de TI. Como mencionado, "a gestão eficaz dos tickets é fundamental para garantir que os problemas sejam resolvidos de forma rápida e eficiente".

- **Interação entre Utilizadores e TI**: O sistema possibilita a troca de comentários e arquivos entre colaboradores, permitindo uma interação fluida entre os utilizadores e os agentes de TI. "A comunicação clara e eficiente é um dos pilares para a resolução de problemas".

- **Notificações Automáticas**: O sistema envia notificações por e-mail automáticas para manter todas as partes informadas sobre atualizações de tickets. "As notificações automáticas garantem que todos os envolvidos estejam sempre atualizados sobre o estado dos tickets".

- **Classificação e Roteamento Automáticos**: O sistema classifica e roteia automaticamente os tickets para os agentes de TI responsáveis, minimizando o tempo de resolução e aumentando a satisfação do utilizador. "A automação deste processo é crucial para a eficiência do help desk".

- **Geração de Relatórios**: O sistema permite a geração de relatórios que medem os KPIs necessários para avaliar a saúde dos processos de TI. "A análise de dados é essencial para a melhoria contínua dos serviços prestados".

Resultados Obtidos:

Os resultados obtidos com a implementação do sistema de help desk foram significativos e demonstraram a eficácia da abordagem adotada. Os autores relatam que "a metodologia de aprendizagem de máquina aplicada permitiu uma classificação precisa dos tickets, resultando numa redução do tempo de resolução e num aumento da satisfação do utilizador". A capacidade de medir KPIs e definir processos de negócios claros também contribuiu para uma avaliação mais eficaz do desempenho da equipa de TI. Como afirmam, "os resultados

demonstram que a implementação de um sistema de help desk baseado em aprendizagem de máquina pode transformar a forma como os serviços de TI são geridos".

Desafios e Limitações:

Apesar dos resultados positivos, o desenvolvimento e a implementação do sistema enfrentaram vários desafios e limitações. Um dos principais desafios foi a necessidade de regenerar o modelo de classificação diariamente para incorporar novas informações de tickets. Os autores mencionam que "este processo contínuo de atualização e validação do modelo é crucial para manter a precisão do sistema". Além disso, a integração com sistemas existentes e a adaptação dos utilizadores ao novo sistema também apresentaram dificuldades. "A resistência à mudança é um desafio comum em qualquer implementação de tecnologia".

Conclusões Obtidas:

Em conclusão, o sistema de help desk baseado em aprendizagem de máquina desenvolvido para a GJU demonstrou ser uma solução eficaz para a gestão de serviços de TI. Os autores afirmam que "a implementação de um modelo de classificação de tickets não só melhorou a eficiência operacional, mas também aumentou a satisfação do utilizador". O trabalho destaca a importância de um sistema de help desk bem estruturado e a aplicação de tecnologias de aprendizagem de máquina para otimizar processos de negócios e serviços de TI. Como conclusão, "a adoção de soluções tecnológicas inovadoras é fundamental para o sucesso na gestão de serviços de TI".

2.3.1.3 Conclusões da Análise dos Artigos

A análise dos dois artigos científicos trouxe lições valiosas e identificou boas práticas que servirão como fundamento para o desenvolvimento do sistema de help desk proposto nesta dissertação.

Em primeiro lugar, a integração com sistemas existentes destacou-se como um fator essencial para o sucesso das soluções analisadas. A tese de Jakub Lukačín demonstrou como a sincronização de dados com o Redmine melhorou a eficiência na gestão de tickets e na comunicação entre equipas, enquanto o sistema desenvolvido para a GJU mostrou que a integração com o portal de funcionários aumentou a acessibilidade e garantiu consistência nos processos. Assim, é importante considerar soluções que integrem o sistema de help desk com ferramentas já utilizadas pela organização-alvo, promovendo uma transição suave e evitando redundância.

Outro ponto relevante é o impacto positivo da automação. No sistema da GJU, o uso de algoritmos de aprendizagem de máquina para a classificação de tickets resultou em ganhos de eficiência, ao reduzir o tempo de resolução e melhorar a satisfação dos utilizadores. De forma semelhante, na tese de Lukačín, a automação de notificações e a sincronização com o Redmine foram destacadas como essenciais para a melhoria da comunicação e do acompanhamento de solicitações. Dessa forma, a automação deve ser incorporada ao sistema para processos como

notificações, classificação de tickets e geração de relatórios, elementos essenciais para maximizar a eficiência.

A usabilidade e a internacionalização também foram enfatizadas como aspetos críticos para o sucesso de um sistema de help desk. A tese de Lukačín apresentou uma interface amigável e focada na experiência do utilizador, validada por feedback positivo em testes reais, e priorizou o suporte a múltiplos idiomas para atender utilizadores de diferentes regiões. O sistema da GJU também mostrou como uma interação simples e direta entre utilizadores e agentes de TI facilita a resolução de problemas. Assim, o desenvolvimento do sistema deve priorizar uma interface intuitiva e suporte a múltiplos idiomas, garantindo acessibilidade e inclusão para um público diversificado.

Além disso, a monitorização e avaliação contínua dos serviços revelaram-se cruciais nos dois estudos analisados. A funcionalidade de geração de relatórios para medir indicadores de desempenho (KPIs) permitiu aos sistemas avaliarem a eficiência dos processos e identificarem áreas de melhoria. No contexto do sistema proposto, incluir uma funcionalidade analítica permitirá acompanhar a eficácia do sistema, promover melhorias constantes e fornecer dados estratégicos para a tomada de decisões.

Por outro lado, os desafios enfrentados durante a implementação desses sistemas também oferecem lições importantes. Ambos os trabalhos discutiram dificuldades como a resistência à mudança por parte dos utilizadores e a complexidade técnica da integração com sistemas existentes. No caso da GJU, foi necessário atualizar constantemente o modelo de aprendizagem de máquina para manter a precisão, o que exigiu uma abordagem iterativa. Esses desafios ressaltam a importância de estratégias para minimizar a resistência dos utilizadores, como capacitações e suporte contínuo, bem como a necessidade de um período de testes e validação extensivos para garantir o sucesso da implementação.

Por fim, a escolha de tecnologias modernas, como Vue.js, Vuetify, Docker, PostgreSQL e modelos de aprendizagem de máquina, destacou-se nos dois trabalhos como essencial para a criação de sistemas escaláveis, flexíveis e robustos. A utilização dessas ferramentas demonstra o valor de adotar soluções tecnológicas atuais que ofereçam alto desempenho e facilidade de manutenção.

Em síntese, as conclusões extraídas reforçam a relevância de um sistema de help desk bem estruturado, integrando automação, usabilidade, monitorização e tecnologias modernas, enquanto aborda os desafios inerentes ao processo de implementação. Essas lições constituem uma base sólida para guiar o desenvolvimento do sistema proposto nesta dissertação, alinhando-o às melhores práticas identificadas nos estudos analisados.

2.3.2 Aplicações Help Desk

Neste tópico, serão analisadas as funcionalidades disponibilizadas pelas principais aplicações de help desk disponíveis no mercado. O objetivo é compreender a atualidade dessas ferramentas, entender as soluções oferecidas e identificando as principais características que tornam essas aplicações úteis no suporte ao cliente.

Na pesquisa pelas melhores aplicações de *help desk*, foram identificadas diversas soluções amplamente reconhecidas no mercado. Entre estas, destacam-se o Zendesk, o Freshdesk e o Zoho Desk, que se encontram de forma consistente nos principais relatórios de avaliação e comparação de software da área. Segundo a plataforma G2, estas três ferramentas surgem como líderes de mercado, evidenciando elevados índices de satisfação dos utilizadores, diversidade de funcionalidades e forte presença empresarial (Lin, 2024). Assim, apresentam-se de seguida as principais características que distinguem cada uma destas soluções como referências no domínio das aplicações de suporte ao cliente:

1. **Zendesk (zendesk.com)**
    - **Sistema de tickets:** A aplicação possui um sistema centralizado para rastrear, categorizar e priorizar solicitações de suporte.
    - **Chatbots**: A aplicação fornece automação de respostas para questões comuns.
    - ***Omnichanel* (omicanal):** A aplicação permite o atendimento via e-mail, chat, voz e redes sociais, centralizando todos estes canais de comunicação num só.
    - **Central de ajuda**: A aplicação possui uma base de conhecimento para autoatendimento.
    - **Painéis analíticos nativos:** A aplicação disponibiliza monitorização e análise de métricas de desempenho.
    - **Aplicações internas e integrações**: A aplicação fornece várias opções para expandir as funcionalidades já existentes.
    - **Regras de negócios personalizadas:** A aplicação disponibiliza configuração de gatilhos e automações específicas.
    - **Suporte multilíngue**: A aplicação permite atendimento em diversos idiomas.
    - **Encaminhamento de tickets baseado em habilidades**: A aplicação direciona tickets para agentes com competências específicas.

2. **Freshdesk (freshworks.com)**
    - **Sistema de Tickets**: A aplicação possui um sistema centralizado para rastrear, categorizar e priorizar solicitações de suporte.
    - **Automação de Fluxo de Trabalho**: Automatiza tarefas repetitivas, como atribuição de tickets, priorização e envio de notificações.

- **Chatbots**: A aplicação fornece um assistente virtual para resolver dúvidas comuns e reduzir a carga de trabalho da equipa.
- **Omnichanel**: A aplicação permite o atendimento via e-mail, telefone, chat em tempo real, redes sociais e portal de clientes, centralizando todos estes canais de comunicação num só.
- **Geração de Base de Conhecimento**: A aplicação permite a criação de artigos, FAQs e tutoriais para autoatendimento dos clientes.
- **Relatórios e Análises Personalizáveis**: A aplicação permite a monitorização de métricas como tempos de resposta, satisfação do cliente e desempenho da equipa.
- **Colaboração de Equipa**: A aplicação permite que os agentes discutam tickets com outros membros da equipa para resoluções mais rápidas.
- **Categorização e Prioritização de Tickets**: A aplicação possui um sistema inteligente que categoriza e prioriza automaticamente as solicitações de suporte.
- **Funcionalidades de Gamificação**: A aplicação incentiva a produtividade e desempenho dos integrantes da equipa de suporte com a atribuição de pontos e recompensas.
- **Integração com Aplicações de Terceiros**: A aplicação é compatível com centenas de ferramentas como CRM, sistemas de faturação e gestão de projetos.
- **Suporte multilíngue**: A aplicação permite atendimento em diversos idiomas.

3. **Zoho Desk (zoho.com)**
    - **Sistema de Tickets**: A aplicação possui um sistema centralizado para rastrear, categorizar e priorizar solicitações de suporte.
    - **Automação de Fluxo de Trabalho**: Automatiza tarefas repetitivas, como atribuição de tickets, priorização e envio de notificações.
    - **Chatbots**: A aplicação fornece um assistente virtual para resolver dúvidas comuns e reduzir a carga de trabalho da equipa.
    - **Omnichanel**: A aplicação permite o atendimento via e-mail, telefone, chat em tempo real, redes sociais e portal de clientes, centralizando todos estes canais de comunicação num só.
    - **Geração de Base de Conhecimento**: A aplicação permite a criação de artigos, FAQs e tutoriais para autoatendimento dos clientes.
    - **Relatórios e Análises Personalizáveis**: A aplicação permite a monitorização de métricas como tempos de resposta, satisfação do cliente e desempenho da equipa.
    - **Colaboração de Equipa**: A aplicação fornece ferramentas como comentários internos nos tickets e partilha de informações entre equipas.

- **Módulo de Gestão de Tarefas:** A aplicação faz a organização e rastreamento das tarefas relacionadas a tickets ou projetos.
- **Personalização Avançada:** A aplicação permite a personalização de layouts de tickets, campos, fluxos de trabalho e painéis de controlo.
- **Integração com Aplicações de Terceiros**: A aplicação permite integração com ferramentas como Slack, Microsoft Teams e Google Workspace.
- **Suporte multilíngue**: A aplicação permite atendimento em diversos idiomas.
- **Portal de Clientes Personalizável**: A aplicação oferece aos clientes um ambiente onde podem rastrear os seus tickets e encontrar soluções em bases de conhecimento.
- **Aplicações Móveis**: A aplicação tem suporte a dispositivos móveis para equipas em movimento.

Conclusões:

Com a análise das principais funcionalidades e qualidades das aplicações help desk que atualmente lideram o mercado de suporte ao cliente, foi possível tirar as seguintes conclusões:

- Existem algumas funcionalidades que são imprescindíveis nas aplicações help desk e que são responsáveis pelo sucesso deste tipo das aplicações. Sendo essas o sistema automático de gestão de tickets, os chatbots, os omnichanels, uma base de conhecimento para autoatendimento, os relatórios e análises, e o suporte multilíngue.
- Outras funcionalidades secundárias que também elevam o valor da aplicação help desk são o encaminhamento automático de tickets, integrações com outras aplicações, um portal onde os utilizadores acompanhem os seus tickets, e o suporte a dispositivos móveis.

Com esta análise foi possível estabelecer quais as funcionalidades importantes de serem desenvolvidas para o projeto em mãos. No entanto, apenas no momento do design da aplicação serão definidas quais funcionalidades integrarão a aplicação final, uma vez que, dadas as limitações temporais da dissertação, nem todas poderão ser desenvolvidas.

2.4 Conhecimento das Tecnologias

Neste capítulo, serão abordadas as ferramentas e tecnologias que serão utilizadas no desenvolvimento dos componentes do projeto. O foco será nas opções escolhidas para os dois grandes componentes do sistema a ser desenvolvido: o Chatbot e a Aplicação Help Desk. O capítulo está, portanto, dividido nestes dois tópicos principais, que explicam as ferramentas e tecnologias específicas disponíveis para cada um desses componentes, e tomam uma decisão sobre quais escolher.

2.4.1 Chatbot

O chatbot será a base do sistema e representará o primeiro contacto que o utilizador final terá com a aplicação, exceto no caso de membros da equipa de suporte. Devido à sua importância, é fundamental que o chatbot seja bem desenvolvido e capaz de se comunicar eficazmente com qualquer utilizador. Considerando a relevância deste componente e o tempo limitado disponível para o desenvolvimento da dissertação, a opção mais viável será recorrer a ferramentas *no-code* (sem código) amplamente utilizadas na criação de chatbots. Deste modo, será realizada uma análise das principais ferramentas atualmente disponíveis para a criação de chatbots, escolhendo-se aquela que melhor se adequar aos requisitos do projeto.

Visto que não foram encontrados artigos ou estudos científicos que analisassem comparativamente as melhores plataformas de desenvolvimento de chatbots, recorreu-se a diferentes relatórios e plataformas de avaliação de software, como G2, Capterra e PeerSpot. Estes portais apresentam rankings e análises baseadas em critérios técnicos e no feedback de utilizadores reais. Para efeitos desta dissertação, foi utilizado como referência principal o ranking do PeerSpot, no qual as plataformas que se destacam neste setor são: Google Dialogflow, Amazon Lex e IBM Watsonx Assistant (Anon., 2025). Para além destas plataformas líderes, incluiu-se também o Voiceflow, que embora não figure nos primeiros lugares destes relatórios, foi considerado relevante para esta análise devido à experiência prévia do autor com a plataforma, o que permitiu avaliar o seu potencial em comparação com alternativas de maior adoção no mercado.

Google Dialogflow (dialogflow.cloud.google.com)

Descrição:
O Dialogflow é uma das soluções mais populares do mercado para a criação de chatbots e assistentes virtuais. Desenvolvido pela Google, está fortemente integrado no ecossistema Google Cloud Platform e oferece recursos avançados de Processamento de Linguagem Natural (PLN).

Características:

- Suporte multilíngue: suporta mais de 20 idiomas de forma nativa.
- Facilidade de uso: exige algum conhecimento técnico na definição de *intents*, entidades e integração com APIs externas.
- Integração: permite ligação direta com Google Assistant, Facebook Messenger, Slack, Telegram, entre outros.
- Personalização: suporta fluxos de conversação complexos e integração com serviços externos.

- Preço: modelo freemium; custo típico de US$ 0,0025 por pedido de texto no plano ES e entre US$ 0,0050 e US$ 0,0065 por request no plano CX. Planos empresariais começam em cerca de US$ 10 000/mês.

Prós:

- Suporte robusto a múltiplos idiomas.
- Recursos poderosos de IA e PLN.
- Elevada capacidade de integração com APIs externas.

Contras:

- Requer conhecimento técnico mais avançado.
- Curva de aprendizagem acentuada.

Amazon Lex (aws.amazon.com/pt/lex)

Descrição:
O Amazon Lex é a solução da AWS para criação de chatbots e assistentes virtuais. Utiliza a mesma tecnologia do Alexa, oferecendo capacidades robustas de PLN e integração direta no ecossistema Amazon.

Características:

- Suporte multilíngue: oferece suporte a vários idiomas através de serviços de tradução da AWS.
- Facilidade de uso: exige configuração em AWS, o que requer algum conhecimento técnico.
- Integração: integração nativa com outros serviços Amazon, incluindo S3, DynamoDB, Lambda, e aplicações externas via API.
- Personalização: oferece flexibilidade no desenho de fluxos, mas com curva de aprendizagem associada ao ecossistema AWS.
- Preço: US$ 0,004 por request de texto e US$ 0,0065 por request de voz, com descontos para grandes volumes.

Prós:

- Integração total no ecossistema AWS.
- Tecnologia robusta baseada no Alexa.
- Escalabilidade para grandes volumes.

Contras:

- Requer conhecimentos técnicos e familiaridade com AWS.
- Custos podem escalar rapidamente em grandes implementações.

IBM Watsonx Assistant (ibm.com/products/watsonx-assistant)

Descrição:
O IBM Watsonx Assistant é a solução corporativa da IBM para chatbots. Destina-se a contextos empresariais que exigem elevados níveis de segurança, escalabilidade e integração com sistemas críticos.

Características:

- Suporte multilíngue: oferece suporte nativo a vários idiomas.
- Facilidade de uso: interface acessível, mas com curva de aprendizagem mais elevada para configuração avançada.
- Integração: integra-se com sistemas empresariais, CRM e plataformas de contacto.
- Personalização: elevado nível de controlo sobre fluxos e integração com outros serviços Watson.
- Preço: licenciamento entre US$ 120 e US$ 300 por utilizador/mês, após oferta gratuita de até 2 500 mensagens/mês.

Prós:

- Forte reputação em contexto empresarial.
- Suporte avançado a segurança e *compliance*.
- Recursos de IA integrados com outros serviços IBM.

Contras:

- Custo elevado.

- Mais indicado para organizações de grande dimensão.

Voiceflow (voiceflow.com)

Descrição:
O Voiceflow é uma plataforma *no-code* que permite criar chatbots e assistentes virtuais através de uma interface visual *drag-and-drop*. É especialmente orientada para prototipagem rápida e design de fluxos conversacionais.

Características:

- Suporte multilíngue: suporta diversos idiomas, mas de forma menos robusta que soluções como Dialogflow ou Lex.
- Facilidade de uso: interface intuitiva e visual, adequada a utilizadores sem conhecimentos técnicos avançados.
- Integração: permite integração com várias APIs externas.
- Personalização: adequada a fluxos simples e intermédios.
- Preço: planos gratuitos disponíveis; planos pagos a partir de cerca de US$ 20/mês.

Prós:

- Interface visual simples e rápida.
- Excelente para prototipagem e MVPs.
- Documentação clara e boa comunidade de apoio.

Contras:

- Menos avançado em PLN.
- Integrações mais limitadas em ambientes complexos.

Análise Comparativa

Tabela 2 – Análise comparativa entre ferramentas de criação de chatbots.

CARACTERÍSTICA	DIALOGFLOW	AMAZON LEX	IBM WATSONX ASSISTANT	VOICEFLOW
SUPORTE MULTILÍNGUE	Robusto (>20 idiomas)	Robusto (vários idiomas AWS)	Robusto (nativo)	Básico/Médio
FACILIDADE DE USO	Média (exige configuração)	Média/Baixa (AWS)	Média (curva de aprendizagem)	Alta (intuitivo)
INTEGRAÇÃO COM APIS	Elevada (Google, APIs externas)	Elevada (AWS)	Elevada (sistemas corporativos)	Boa (APIs externas)
PERSONALIZAÇÃO DE FLUXO	Elevada	Elevada	Elevada	Média
CAPACIDADE DE IA/PLN	Elevada	Elevada	Elevada	Média
PREÇO	~0,0023€ – 0,0059€ por *request*	~0,0036€ (texto) / 0,0055€ (voz)	~110€ – 273€ por utilizador/mês	A partir de ~18€/mês
FACILIDADE DE PROTOTIPAGEM	Média	Média/Baixa	Baixa	Elevada

Após a comparação entre as quatro plataformas, observa-se que o Dialogflow se destaca como a solução mais completa e robusta, com suporte multilíngue avançado, forte capacidade de integração e recursos poderosos de PLN. O Amazon Lex e o IBM Watsonx Assistant são igualmente soluções relevantes, mas apresentam custos ou exigências técnicas que não se adequam ao contexto e calendário desta dissertação.

O Voiceflow, embora menos avançado em termos de PLN e integração, revelou-se a opção mais adequada para este projeto. A sua interface intuitiva e facilidade de utilização permitiram acelerar o desenvolvimento do protótipo, enquanto o facto de o autor já possuir experiência prévia com a plataforma representou uma vantagem prática decisiva. Assim, a escolha recaiu no Voiceflow, não por ser a plataforma tecnicamente mais avançada, mas por oferecer o melhor equilíbrio entre viabilidade, rapidez de implementação e contexto específico deste trabalho académico.

2.4.2 LLM

Como referido anteriormente, a escolha da LLM constitui um elemento central para o desempenho de um chatbot de suporte ao cliente. A LLM é responsável por interpretar corretamente os pedidos, manter o contexto ao longo da conversa e gerar respostas consistentes, claras e úteis. Assim, a seleção do modelo a adotar influencia diretamente a qualidade do atendimento, o custo da operação e até a perceção do utilizador sobre a fiabilidade do sistema. Dada a diversidade de modelos atualmente disponíveis, torna-se fundamental realizar uma breve análise comparativa das principais opções no mercado, de forma a identificar qual ou quais se adequam melhor ao contexto deste projeto.

Segundo um estudo recente, onde foi realizada uma comparação entre diferentes modelos de linguagem aplicados a tarefas de conversação, os modelos que mais se destacaram foram o OpenAI GPT-4, o Anthropic Claude e o Google Gemini (Rangapur & Rangapur, 2024). Considerando que, entretanto, a OpenAI lançou o GPT-5 como sucessor direto do GPT-4, optou-se por incluir este modelo na presente análise, de modo a refletir o estado da arte mais atualizado.

OpenAI GPT-5 (openai.com/gpt-5)

Descrição:
O GPT-5 é o modelo mais avançado da OpenAI, desenvolvido para casos de uso complexos de conversação, agentes autónomos e apoio a tarefas de programação. Disponibilizado através da API oficial da OpenAI, é reconhecido pela elevada qualidade de geração de texto e pelo ecossistema de ferramentas associado.

Características:

- *Tool use*[21] nativo, permitindo ligação a serviços externos.
- Suporte a janelas de contexto alargadas.
- Integração com a plataforma OpenAI e SDKs[22] oficiais.
- Preço de utilização por milhão de tokens: ~ 1,15 €/Milhão (entrada) e ~ 9,20 €/Milhão (saída).

Prós:

- Respostas de elevada qualidade.

[21] Capacidade de o modelo de linguagem chamar ou interagir com ferramentas externas, como bases de dados, sistemas de tickets ou APIs de terceiros, durante uma conversa.
[22] Conjunto de bibliotecas, ferramentas e documentação disponibilizados por um fornecedor para facilitar a integração de uma aplicação ou serviço.

- Forte integração com ferramentas e ecossistema OpenAI.
- Documentação extensa e comunidade ativa.

Contras:

- Pode ter custos elevados em casos de alto volume.
- Limitações no acesso a determinadas funcionalidades avançadas.

Anthropic Claude Sonnet 4 (anthropic.com/claude)

Descrição:
O Claude Sonnet 4 é o modelo de alto desempenho da Anthropic, criado com foco em segurança, consistência e respostas alinhadas com princípios de *Constitutional AI*[23]. Está otimizado para lidar com janelas de contexto muito amplas, sendo adequado a cenários onde é necessário processar grandes volumes de informação de uma só vez.

Características:

- Janela de contexto de até 200 mil tokens.
- Preço médio por milhão de tokens: ~ 2,76 €/Milhão (entrada) e ~ 13,80 €/Milhão (saída).
- Otimizado para reduzir enviesamentos e respostas incoerentes.
- Suporte a integração em APIs empresariais.

Prós:

- Grande robustez em contextos longos.
- Segurança e consistência elevadas.
- Bom para setores sensíveis (finanças, saúde).

Contras:

- Preço mais elevado do que outros modelos equivalentes.
- Algumas funcionalidades avançadas ainda em evolução.

[23] Abordagem de treino utilizada pela Anthropic, que consiste em aplicar princípios explícitos (como segurança e não-discriminação) para guiar o comportamento do modelo.

Google Gemini 2.5 Pro (ai.google.dev/gemini-api)

Descrição:
O Gemini 2.5 Pro é o modelo de linguagem da Google, concebido para fornecer uma combinação entre qualidade de raciocínio, integração com o ecossistema Google e custo competitivo. Está disponível através da Google AI Studio e Vertex AI.

Características:

- Janela de contexto até 200 mil tokens por *prompt*.
- Funções de *grounding*[24] integradas com Google Search.
- Preço médio por milhão de tokens: ~ 1,15 €/Milhão (entrada) e ~ 9,20 €/Milhão (saída).
- Modo *batch*[25] com custos reduzidos (cerca de metade).

Prós:

- Excelente relação custo-benefício.
- Suporte a integração direta com serviços Google.
- *Grounding* nativo que melhora precisão em FAQs.

Contras:

- Algumas funcionalidades (como *grounding* avançado) podem ter custos adicionais.
- Dependência do ecossistema Google.

[24] Técnica que consiste em ligar as respostas do modelo a fontes externas fiáveis de forma a aumentar a precisão e reduzir invenções (*hallucinations*).
[25] Modo de processamento que permite enviar várias solicitações de uma só vez para o modelo, reduzindo o custo de execução em comparação com o processamento em tempo real.

Análise Comparativa:

Tabela 3 - Análise comparativa entre LLMs para chatbot de suporte.

CARACTERÍSTICA	GPT-5 (OPENAI)	CLAUDE SONNET 4 (ANTHROPIC)	GEMINI 2.5 PRO (GOOGLE)
SUPORTE DE CONTEXTO	Elevado	Muito elevado (200m)	Elevado (200m)
PREÇO (ENTRADA)	~ 1,15 €/Milhão	~ 2,76 €/Milhão	1,15 €/Milhão
PREÇO (SAÍDA)	~ 9,20 €/Milhão	~ 13,80 €/Milhão	~ 9,20 €/Milhão
QUALIDADE DA RESPOSTA	Muito elevada	Elevada e consistente	Elevada
FOCO PRINCIPAL	Versatilidade	Segurança e consistência	Custo/benefício e integração
ADEQUAÇÃO A SUPORTE	Muito boa	Boa (mais cara)	Muito boa (eficiente)

A análise mostra que os três modelos avaliados são adequados para a construção de um chatbot de suporte ao cliente, mas com diferentes pontos fortes:

- O GPT-5 destaca-se pela versatilidade e pelo ecossistema maduro, sendo uma opção equilibrada entre qualidade e custo.
- O Claude Sonnet 4 apresenta vantagens em contextos que exigem segurança acrescida e processamento de grandes contextos, embora a um preço mais elevado.
- O Gemini 2.5 Pro surge como a solução com melhor relação custo-benefício, particularmente em cenários de grande volume de interações e integração com ferramentas Google.

No contexto desta dissertação, onde é necessário equilibrar qualidade de resposta, custo e viabilidade prática, os modelos GPT-5 e Gemini 2.5 Pro surgem como as escolhas mais adequadas. O Claude Sonnet 4, embora tecnicamente robusto, é menos competitivo em termos de custo para um caso de uso orientado a helpdesk.

2.4.3 Sistema de Gestão de Pedidos

Os sistemas de help desk analisados no estado da arte mostram que a sua eficácia depende de um conjunto de funcionalidades fundamentais, entre as quais se incluem a gestão automática de tickets, os chatbots, a comunicação *omnicanal*, as bases de conhecimento para autoatendimento, os relatórios e análises e o suporte multilíngue.

No entanto, devido às limitações temporais desta dissertação e ao objetivo de apresentar um protótipo funcional, o trabalho irá concentrar-se apenas em duas funcionalidades basilares num sistema helpdesk: o chatbot e o sistema de gestão automática de pedidos. Esta escolha procura refletir a prática comum neste tipo de aplicações, em que os pedidos que não são resolvidos pelo chatbot são encaminhados para as equipas de suporte humano na forma de tickets[26]. Estes tickets são depois classificados e priorizados, garantindo que o fluxo de trabalho se mantém organizado e consistente.

Uma possibilidade seria criar um sistema completo de raiz, incluindo interface de utilizador, camada de back-end e base de dados própria. Esta solução permitiria um controlo total sobre a aplicação e flexibilidade máxima para personalização, permitindo responder de forma precisa às necessidades do projeto. Contudo, esta abordagem implicaria custos de transição mais elevados, uma curva de aprendizagem adicional para as equipas e a possível duplicação de ferramentas já em uso, fatores que poderiam dificultar a adoção da solução em ambiente empresarial.

A segunda possibilidade consistiria em criar uma API capaz de interligar o chatbot com sistemas de gestão de tickets já implementados, como o Jira, que é o sistema utilizado na Natixis. Esta opção apresentaria vantagens significativas, uma vez que aproveitaria ferramentas já familiares às equipas e reduziria a curva de adaptação. Ao manter todos os pedidos centralizados no Jira, evitar-se-ia a duplicação de sistemas de suporte e assegurava-se a continuidade organizacional. A principal limitação desta solução estaria no facto de a integração depender das funcionalidades disponibilizadas pela API oficial do Jira, o que poderia restringir a implementação de determinadas personalizações.

Embora ambas as opções fossem tecnicamente viáveis, tal como já referido, no caso específico da Natixis já existia o Jira como sistema consolidado de gestão de tickets. A criação de uma aplicação independente revelar-se-ia redundante e traria custos de transição desnecessários. Por essa razão, a solução mais adequada para este projeto passaria pelo desenvolvimento de uma API modular que permitisse ao chatbot criar automaticamente tickets no Jira.

A API deveria ser concebida de forma extensível, possibilitando futuras adaptações a outras plataformas de gestão de tickets sem alterações estruturais profundas. A inovação desta abordagem resultaria do facto de não procurar substituir sistemas existentes, mas antes potenciá-los, garantindo que os pedidos tratados pelo chatbot se integrariam diretamente no ecossistema já utilizado pela empresa. Este contributo revelar-se-ia especialmente relevante,

[26] Registo formal de um pedido ou problema feito por um utilizador.

uma vez que muitas organizações modernas utilizam plataformas como o Jira e poderiam beneficiar de soluções de integração em vez de substituição.

É neste ponto que reside o contributo diferenciador desta dissertação. Após uma pesquisa extensiva sobre as soluções help desk atualmente disponíveis, não foi identificado nenhum sistema que apresente esta característica de oferecer um suporte inteligente totalmente integrado em plataformas já consolidadas, sem obrigar as equipas a adotar novas ferramentas. Assim, a proposta distingue-se pela sua facilidade de integração, simplicidade de adoção e pelo facto de se alinhar diretamente com a realidade de empresas sobrecarregadas, que procuram reduzir custos de transição e evitar curvas de aprendizagem adicionais.

Para desenvolver a API de integração com o Jira, será necessário recorrer a uma framework de *back-end* que garanta a construção de um serviço REST fiável, escalável e de fácil manutenção. Entre as várias opções existentes no mercado, três frameworks destacaram-se num estudo comparativo recente sobre eficiência e fiabilidade de tecnologias de *back-end*: Spring Boot, Express.js e Django REST Framework (Dominik, et al., 2023).

Segundo o estudo, o Spring Boot, baseado em Java, demonstrou um desempenho superior em cenários de elevada carga e destacou-se pela fiabilidade e robustez. Estas características fizeram dele uma das soluções mais utilizadas em ambientes empresariais, beneficiando ainda de uma comunidade extensa e de documentação autualizada. Contudo, apresentou uma curva de aprendizagem mais acentuada e uma maior complexidade inicial quando comparado com alternativas mais leves (Dominik, et al., 2023).

O Express.js, desenvolvido sobre Node.js, revelou-se bastante competitivo em termos de rapidez de desenvolvimento e simplicidade. Foi identificado como uma framework minimalista e flexível, ideal para prototipagem de APIs, apoiada por um ecossistema rico de bibliotecas. Apesar das suas vantagens em termos de agilidade, ofereceu menor estruturação em projetos de grande escala, o que representou desafios de manutenção a longo prazo (Dominik, et al., 2023).

Já o Django REST Framework, baseado em Python, destacou-se pela produtividade elevada e pela simplicidade na criação de APIs REST. O suporte nativo a autenticação, serialização e gestão de permissões facilitou o desenvolvimento rápido de aplicações seguras. No entanto, os estudos apontaram um desempenho inferior quando comparado ao Spring Boot e até mesmo ao Express.js em cenários de alto volume de pedidos, apresentando maior latência e menor estabilidade sob carga intensiva (Dominik, et al., 2023).

Em síntese, a análise demonstrou que cada framework possuía pontos fortes distintos. O Spring Boot destacou-se em termos de robustez e fiabilidade, o Express.js em rapidez e flexibilidade, e o Django REST Framework em produtividade e facilidade de implementação. Todas foram consideradas tecnicamente viáveis para o desenvolvimento da API em questão. No entanto, a escolha final recairá sobre o Spring Boot, não apenas pelo seu desempenho comprovado em ambientes empresariais, mas também pela experiência prévia do autor com esta tecnologia, o que permitirá reduzir a curva de aprendizagem e aumentar a eficiência no desenvolvimento do

protótipo. Embora o Express.js e o Django REST Framework possam oferecer vantagens em contextos específicos, a prioridade deste projeto será a entrega de um MVP estável e escalável dentro das restrições temporais da dissertação.

Tabela 4 - Análise comparativa entre frameworks de back-end para a API de integração

CRITÉRIO	SPRING BOOT	EXPRESS.JS	DJANGO REST FRAMEWORK
DESEMPENHO / LATÊNCIA	Elevado	Médio	Baixo
CONFIABILIDADE	Muito alta	Média-alta	Média-baixa
FACILIDADE DE USO	Média-baixa	Alta	Alta
ESCALABILIDADE	Elevada	Média	Média
COMUNIDADE & DOCUMENTAÇÃO	Muito ampla	Ampla	Ampla
FAMILIARIDADE DO AUTOR	Alta	Média	Baixa

Assim, a arquitetura proposta baseia-se na combinação entre o chatbot e a API de integração com o Jira, constituindo a fundação do software a ser desenvolvido nesta dissertação.

2.5 Conclusões de Capítulo

Este capítulo consolidou a base conceptual e tecnológica necessária para o desenvolvimento do projeto. Partiu-se de uma metodologia de pesquisa clara, suportada por consultas estruturadas no Google Scholar e filtragem sistemática dos resultados, garantindo que a revisão incidisse sobre literatura recente e relevante. A partir daí, foram aprofundados três eixos centrais: software help desk, chatbots e prompt engineering, compondo o quadro de referência que sustenta as opções técnicas e de desenho do sistema.

No domínio do help desk, identificaram-se as funcionalidades que caracterizam as soluções consolidadas do mercado. Sendo elas a gestão automática de tickets, presença de chatbots, comunicação *omnicanal*, base de conhecimento para autoatendimento, relatórios e métricas operacionais, e suporte multilíngue. Esta leitura evidenciou simultaneamente oportunidades e desafios: ganhos claros de eficiência e qualidade de serviço, mas também riscos de sobrecarga tecnológica, resistência à mudança e necessidade de manutenção contínua.

A análise dos chatbots mostrou a sua evolução desde abordagens baseadas em padrões para modelos orientados por técnicas modernas de processamento de linguagem natural e, mais recentemente, por modelos de linguagem de grande escala. Reforçou-se que a adoção de LLMs altera substancialmente o alcance e a qualidade das interações, mas traz novas decisões

técnicas (custo, latência, contexto, idioma) e exige especial atenção à forma como o modelo é instruído. Isto demonstrou também a relevância do *prompt engineering* que para além de aumentar a precisão, permite alinhar o comportamento do agente com os objetivos do suporte, garantindo consistência de tom, rigor informativo e operacionalização de encaminhamentos.

O estudo do prompt engineering foi aprofundado ao ponto de identificar técnicas como *zero-shot prompting*, *few-shot prompting* e *chain-of-thought prompting*, que influenciam diretamente a eficácia e a consistência das respostas. Da análise resultou uma proposta estruturada para os prompts a serem usados nos agentes a serem desenvolvidos no âmbito desta dissertação, composta por sete componentes fundamentais: identidade e propósito, voz e persona, fluxo conversacional, diretrizes de resposta, funcionalidade específica, cenários de interação e ações de encaminhamento. Esta estrutura garante não apenas clareza e previsibilidade na interação, mas também a flexibilidade necessária para adaptar os agentes a diferentes contextos de utilização, reforçando o papel do prompt engineering como elemento de design do sistema.

Com base nesse enquadramento, procedeu-se ao estudo de ferramentas para construção do chatbot, comparando soluções de referência e incluindo o Voiceflow devido à experiência prévia do autor. A comparação apontou para um compromisso entre capacidade técnica e viabilidade de um MVP dentro das restrições do projeto, justificando a opção por uma plataforma que acelera a prototipagem sem perder a integração necessária com serviços externos.

No que respeita aos modelos de linguagem, foi discutida a importância de selecionar LLMs adequados ao contexto de suporte. A análise comparativa evidenciou alternativas robustas e atuais, destacando diferentes forças (versatilidade, segurança/consistência, custo/benefício e integração), e apontou o caminho para escolhas pragmáticas que equilibrem qualidade de resposta e custos operacionais.

Por fim, apresentou-se a decisão arquitetural para o componente de gestão de pedidos. Em vez de desenvolver um sistema completo de raiz, o projeto adotará uma integração direta com o Jira através de uma API própria. Esta abordagem permite aproveitar processos e ferramentas já consolidados na organização, reduz a curva de adoção e preserva a governança operacional existente, enquanto abre espaço para inovação na camada conversacional e no desenho do fluxo de tickets. Em termos práticos, a solução proposta combina um chatbot orientado por boas práticas de *prompt engineering* com uma API modular capaz de criar, classificar e priorizar pedidos no sistema nativo da empresa.

Em síntese, o capítulo demonstrou que é possível atacar um problema clássico do ecossistema help desk por uma via distinta: não substituindo plataformas estabelecidas, mas ampliando-as com uma interface conversacional inteligente e uma integração cuidadosa. Este posicionamento confere ao projeto um carácter inovador e transferível pois responde ao caso específico em estudo e, pela natureza modular da integração, pode ser replicado noutras organizações com sistemas de gestão de tickets semelhantes. Nos capítulos seguintes, esta base será traduzida em decisões de desenho, implementação e avaliação do protótipo.

3 Desenho da Solução

No seguinte capítulo será apresentada a proposta de solução para o problema basilar desta dissertação, desde a análise e definição dos requisitos até ao desenho detalhado da arquitetura e componentes do sistema. Será iniciada com a caracterização do problema e a definição clara dos objetivos da solução, seguida da especificação dos requisitos funcionais e não funcionais. Posteriormente, serão exploradas as opções de design e a arquitetura definida, recorrendo a diferentes vistas do modelo C4 e 4+1 para representar de forma estruturada a solução. Cada componente do sistema será descrito em detalhe, evidenciando o seu papel, principais funcionalidades e fluxos de informação, assim como as alternativas de design analisadas. Por fim, serão apresentados os padrões e princípios adotados no desenvolvimento do código, assegurando que a solução cumpra os objetivos definidos e promova qualidade, escalabilidade e manutenção futura.

3.1 Análise do Problema

A criação de uma solução eficaz de suporte ao utilizador deve basear-se numa compreensão clara das necessidades reais dos seus intervenientes. Essa compreensão foi construída com base em dois pilares complementares: a análise qualitativa de cinquenta emails trocados entre utilizadores da aplicação Capla e a respetiva equipa de suporte, e a experiência profissional do autor desta dissertação, adquirida ao longo de dois anos de colaboração direta com essa mesma equipa.

Os emails foram disponibilizados pela empresa exclusivamente para fins académicos, permitindo uma observação sistemática e representativa dos tipos de interações existentes. Esta análise, aliada ao conhecimento prático do funcionamento da equipa, permitiu identificar os principais padrões de contacto, dificuldades recorrentes e oportunidades de melhoria no processo de suporte.

3.1.1 Observações e Categorias Identificadas

Após uma análise qualitativa das interações recolhidas, verificou-se que os pedidos realizados à equipa de suporte podiam ser agrupados em quatro categorias principais, de acordo com a sua natureza. Para além da categorização, foi também analisada a gravidade associada a cada tipo de pedido, permitindo estabelecer uma lógica de priorização automática que fundamentará o sistema a desenvolver.

1. **Reportes de Problemas / Bugs — Prioridade Alta**
   Situações em que os utilizadores detetaram comportamentos inesperados ou erros no sistema. Estes casos assumem maior criticidade, uma vez que podem impedir totalmente a utilização de determinadas funcionalidades. Além disso, quando um erro é reportado, é expectável que outros utilizadores enfrentem o mesmo problema em pouco tempo, aumentando o impacto sobre a operação.
   Exemplo: "A página de histórico não está a carregar corretamente quando se seleciona um filtro de mais de 90 dias."

2. **Pedidos de Acesso ou Permissão — Prioridade Média**
   Solicitações relacionadas com a atribuição de permissões ou acessos a plataformas integradas. Estes pedidos, embora não configurem falhas técnicas, bloqueiam o progresso do utilizador nas suas tarefas diárias. A resolução célere é, portanto, essencial para garantir continuidade operacional, justificando a atribuição de uma prioridade intermédia.
   Exemplo: "Preciso de permissões de Gestor Financeiro para gerir os gastos com a equipa X."

3. **Perguntas Abertas (FAQs) — Prioridade Baixa**
   Questões levantadas pelos utilizadores acerca do funcionamento da aplicação. Embora importantes para a usabilidade, estes pedidos raramente comprometem a continuidade do trabalho e, na maioria dos casos, refletem dúvidas de utilizadores menos experientes ou situações pontuais.
   Exemplo: "Como posso criar uma linha de previsão para o recurso X?"

4. **Sugestões de Melhoria — Prioridade Muito Baixa**
   Recomendações feitas pelos utilizadores para otimizar funcionalidades ou simplificar processos. Apesar de representarem uma fonte valiosa de feedback, não têm caráter urgente e podem ser analisadas e implementadas gradualmente.
   Exemplo: "Seria útil poder exportar as previsões diretamente para Excel."

As quatro categorias identificadas revelaram-se suficientemente abrangentes para acomodar praticamente todos os pedidos analisados, o que justifica a sua utilização como base estrutural da lógica conversacional do chatbot e da gestão de tickets. Para além disso, a atribuição de prioridades específicas a cada tipo de pedido garante que os problemas mais críticos são resolvidos em primeiro lugar, assegurando um equilíbrio entre eficiência operacional e qualidade do suporte prestado.

3.1.2 Necessidades Identificadas

A análise detalhada das interações permitiu identificar um conjunto de limitações estruturais e operacionais no processo de suporte da aplicação Capla. Estes problemas, recorrentes nas trocas de comunicação entre utilizadores e equipa de suporte, impactam diretamente a eficiência do serviço prestado e a experiência do utilizador. As principais dificuldades observadas foram:

Volume elevado de pedidos repetitivos:
Verificou-se uma elevada incidência de dúvidas recorrentes e de baixa complexidade, como questões sobre a utilização de funcionalidades básicas da aplicação ou a interpretação de certos elementos da interface. Estas perguntas, muitas vezes semelhantes entre si, exigem tempo da equipa de suporte, mas poderiam, potencialmente, ser resolvidas através de um mecanismo de resposta automatizada ou autoatendimento. Por exemplo, pedidos como "Como posso utilizar o sistema de criação de previsões para fazer X?" ou "Como posso modificar os valores gastos por recurso humano?" surgiram com frequência nas conversas analisadas.

Demora nas respostas:
Durante períodos de maior carga de trabalho, ou em situações em que a equipa de suporte se encontrava reduzida, os tempos de resposta aumentavam significativamente. Isto era especialmente evidente em pedidos menos urgentes, que acabavam por ficar pendentes por longos períodos. A ausência de ferramentas que permitissem respostas automáticas ou algum tipo de triagem inicial dificultava a priorização e escalonamento eficiente dos pedidos, contribuindo para a frustração dos utilizadores e a sobrecarga dos técnicos.

Falta de triagem automatizada:
Todos os pedidos recebidos eram tratados de forma linear, sem qualquer categorização ou avaliação prévia quanto à sua urgência ou tipo. Isso obrigava a equipa a ler, interpretar e encaminhar manualmente cada solicitação, o que não só é moroso como propenso a falhas ou esquecimentos. Problemas críticos, como bugs em produção, podiam ser tratados com o mesmo tempo de resposta que simples pedidos de esclarecimento, por falta de mecanismos que distinguissem automaticamente os diferentes tipos de pedidos.

Dificuldade na gestão e rastreamento dos pedidos:
O uso do email como principal canal de suporte não proporcionava uma visão estruturada dos pedidos nem garantia o seu acompanhamento eficiente ao longo do tempo. Muitos casos não tinham histórico acessível, não permitiam a consulta rápida do estado atual ou da resposta mais recente, e estavam sujeitos a duplicação ou esquecimento. Esta limitação dificultava tanto o trabalho da equipa como a experiência do utilizador, ao não oferecer visibilidade nem previsibilidade na resolução.

Utilizadores sem apoio imediato:
Verificou-se também a ausência de qualquer mecanismo que permitisse aos utilizadores obter respostas imediatas às suas questões, sobretudo fora do horário de funcionamento da equipa de suporte. Mesmo para dúvidas simples ou repetitivas, os utilizadores ficavam dependentes

da resposta humana. Esta limitação compromete a autonomia do utilizador e a sua capacidade de resolver problemas de forma autónoma em momentos críticos.

Estas fragilidades evidenciam a necessidade do desenvolvimento de uma solução que permita automatizar a resposta a pedidos simples, realizar uma triagem inicial dos pedidos recebidos, estruturar a sua gestão e disponibilizar canais de apoio acessíveis, mesmo fora do horário de expediente. Este levantamento serviu de base à definição dos requisitos que serão apresentados nas secções seguintes, que procuram responder de forma estruturada aos desafios identificados nesta análise.

3.1.3 Modelo de Domínio

O modelo de domínio representa, de forma abstrata, os principais conceitos envolvidos no sistema de suporte ao utilizador, bem como as relações entre eles. Este modelo constitui a base conceptual sobre a qual irá assentar o desenho da arquitetura da solução e orientará o desenvolvimento técnico, servindo de ponto de partida para a criação dos modelos de dados, regras de negócio e fluxos de interação.

A sua construção resultou da análise qualitativa das interações entre utilizadores e equipa de suporte, bem como da identificação das categorias de pedidos e necessidades operacionais descritas nas secções anteriores. As entidades que o compõem refletem os elementos essenciais ao funcionamento do sistema, independentemente das tecnologias utilizadas para os implementar.

Segue-se a descrição das principais entidades identificadas:

- **Utilizador**
  Representa o indivíduo que solicita apoio através do sistema. Pode submeter diferentes tipos de pedidos, como dúvidas, reportes de problemas, sugestões ou pedidos de acesso. Cada utilizador está associado a um conjunto de consultas ou tickets.

- **Consulta**
  Corresponde ao pedido de suporte efetuado por um utilizador. Pode ser resolvida automaticamente, com base numa base de conhecimento, ou encaminhada para acompanhamento humano através da criação de um ticket.

- **Chatbot**
  Canal de entrada entre o utilizador e o sistema de suporte. O chatbot atua como interface conversacional, podendo interagir com a base de conhecimento, recolher dados estruturados e encaminhar pedidos que necessitem de acompanhamento para o sistema de gestão de tickets.

- **Categoria**
  Classifica a consulta segundo os quatro tipos identificados: Pergunta Aberta, Reporte de Problema, Pedido de Acesso ou Sugestão de Melhoria. Esta classificação orienta o tratamento do pedido ao longo do seu ciclo de vida.

- **Ticket**
  Representa o registo formal de uma consulta que necessita de acompanhamento. Um ticket contém dados relevantes como o título, a descrição do problema, a data de submissão, o estado atual e a sua categoria. É criado quando o pedido do utilizador não pode ser resolvido de forma imediata ou automatizada.

- **Estado do Ticket**
  Define a fase em que o ticket se encontra no seu ciclo de vida, como *Aberto*, *Em Análise*, *Em Resolução* ou *Fechado*. Permite acompanhar a evolução do pedido até à sua conclusão.

- **Membro da Equipa de Suporte**
  Representa o técnico responsável por analisar e tratar os tickets criados no sistema. Tem acesso à listagem de tickets, pode alterar os seus estados, adicionar observações e resolver os pedidos de acordo com as prioridades definidas.

- **Base de Conhecimento**
  Repositório de conteúdos e respostas a perguntas frequentes. Este componente pode ser utilizado para responder automaticamente a determinadas consultas, reduzindo o volume de tickets e aumentando a autonomia do utilizador.

- **Sistema de Gestão de Tickets**
  Plataforma responsável por centralizar a criação, organização, rastreamento e resolução dos tickets. No caso específico deste estudo, o sistema adotado será o Jira.

Este modelo de domínio estabelece a estrutura conceitual necessária para o desenvolvimento de uma solução alinhada com as necessidades reais identificadas no processo de análise do problema. A sua representação gráfica será incluída a seguir, sob a forma de diagrama UML.

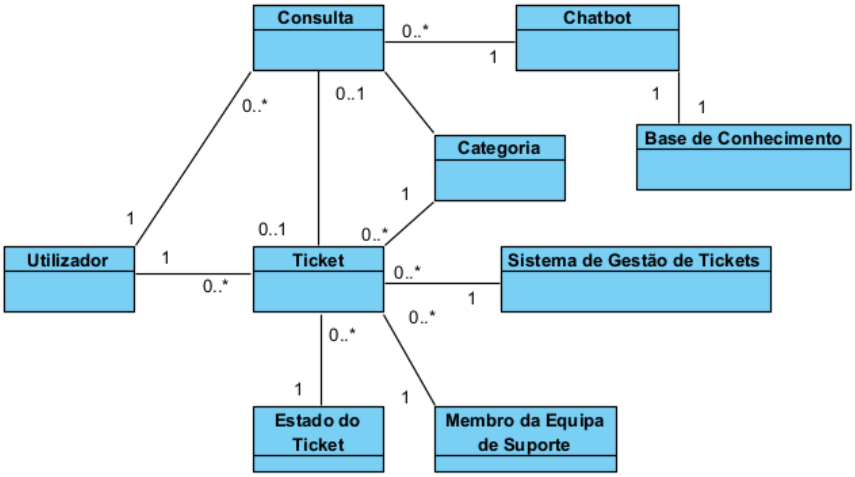


Figura 1 - Modelo de Domínio

3.1.4 Requisitos Funcionais

Com base nas necessidades identificadas na análise do problema, foram definidos os seguintes requisitos funcionais. Estes requisitos representam as funcionalidades essenciais que o sistema deverá implementar para responder de forma eficaz às limitações observadas no processo atual de suporte. São, portanto, a tradução prática dos objetivos da solução proposta e garantem que o sistema terá a capacidade de tratar, encaminhar e acompanhar os pedidos de suporte de forma estruturada, eficiente e automatizada.

RF01 – O sistema deve permitir que os utilizadores submetam pedidos de suporte através do chatbot.
Este requisito garante um ponto de entrada unificado, simples e acessível, permitindo que qualquer utilizador inicie um pedido de forma rápida e intuitiva, sem necessidade de contacto direto com a equipa de suporte.

RF02 – O chatbot deve responder automaticamente a perguntas frequentes com base numa base de conhecimento atualizável.
Este requisito garante que o chatbot tem acesso a uma base de conhecimento capaz de apoiar o utilizador em diferentes contextos, incluindo dúvidas sobre o funcionamento da aplicação, resolução de problemas, esclarecimento de funcionalidades ou comportamento inesperado do sistema. A base de conhecimento deve ser facilmente gerida e atualizada pela equipa de suporte, sem depender de intervenções técnicas, assegurando que o conteúdo se mantém alinhado com a evolução da aplicação e com as necessidades reais dos utilizadores.

RF03 – O sistema deve criar um ticket quando o chatbot não consegue resolver a consulta.
Sempre que uma questão não possa ser tratada automaticamente, o sistema deverá garantir a sua formalização sob a forma de um ticket, assegurando que será analisada e resolvida pela equipa de suporte.

RF04 – Os tickets devem ser automaticamente classificados por categoria e prioridade.
A categorização e priorização automática são essenciais para um tratamento mais eficiente dos pedidos. Este requisito permite que os tickets sejam corretamente encaminhados e tratados consoante a sua natureza e urgência.

RF05 – A equipa de suporte deve poder fazer a gestão dos tickets.
Este requisito garante que a equipa de suporte é capaz de intervir ativamente nos tickets criados, atualizando o seu estado, acrescentando informações relevantes, resolvendo diretamente os pedidos ou encaminhando-os para os profissionais mais adequados à sua resolução. Esta capacidade é essencial para uma gestão eficaz e dinâmica do suporte, permitindo à equipa priorizar, delegar e acompanhar cada caso com clareza, assegurando que nenhum pedido é ignorado ou maltratado.

RF06 – O sistema deve fornecer uma interface para visualização e histórico de tickets por parte dos utilizadores.

Ao permitir que os utilizadores consultem o estado dos seus pedidos e o respetivo histórico, este requisito contribui para a transparência do processo e reduz a submissão de pedidos repetidos.

RF07 – O sistema deve verificar automaticamente se o conteúdo de um novo pedido é semelhante a tickets anteriores.
A verificação de duplicados é essencial para manter a base de tickets organizada, evitar redundâncias e melhorar a eficiência da equipa, garantindo que o mesmo problema não é tratado múltiplas vezes.

3.1.5 Requisitos Não-Funcionais

Os requisitos não funcionais definem as propriedades de qualidade que o sistema deve assegurar para além das suas funcionalidades principais. Estes requisitos influenciam diretamente a experiência de utilização, o desempenho, a segurança e a capacidade de evolução da solução.

Para uma melhor organização e clareza, optou-se por dividir os requisitos não funcionais em dois grupos distintos:

- Os que dizem respeito exclusivamente ao chatbot, enquanto canal de interação com o utilizador;
- E os que se aplicam ao sistema de suporte como um todo, incluindo a API, o backend e as integrações externas.

Esta separação permite refletir de forma mais precisa as exigências específicas de cada componente, tendo em conta que o chatbot possui características de comunicação e usabilidade distintas, enquanto o sistema global exige robustez, segurança, rastreabilidade e

adaptabilidade.
Durante a definição destes requisitos, foram consideradas as principais dimensões de qualidade descritas no modelo FURPS+[27] (Grady, 1992), como funcionalidade, usabilidade, fiabilidade, desempenho e capacidade de suporte.

Requisitos Não Funcionais do Chatbot

RNF01 – O chatbot deve gerar respostas em menos de 3 segundos após cada interação do utilizador.
Este tempo de resposta é fundamental para manter uma interação fluida e evitar a perda de atenção ou frustração por parte do utilizador.

RNF02 – O chatbot deve manter um discurso coerente, claro e orientado à resolução.
A clareza e consistência na linguagem utilizada são essenciais para garantir que o utilizador compreende as instruções e sente confiança no sistema.

RNF03 – O chatbot deve apresentar uma taxa mínima de 90% de compreensão correta dos pedidos dos utilizadores no primeiro contacto.
Este requisito assegura que o sistema interpreta corretamente a intenção do utilizador na maioria dos casos, minimizando erros e repetições.

RNF04 – O número médio de interações por conversa até à resolução ou criação de ticket não deve exceder 5 mensagens.
Este limite promove fluxos de conversa mais objetivos e evita diálogos excessivamente longos ou confusos, melhorando a eficiência da interação.

Requisitos Não Funcionais do Sistema Global

RNF05 – A base de conhecimento do chatbot deve ser facilmente atualizável pela equipa de suporte através de ficheiros estruturados.
Este requisito garante que a equipa pode manter o conteúdo sempre atual e adaptado às novas realidades, sem depender de equipas técnicas.

RNF06 – O chatbot deve poder ser facilmente integrado em aplicações externas.
A facilidade de integração amplia o leque de contextos onde o chatbot pode ser utilizado e garante que a solução se adapta a diferentes ambientes.

[27] O modelo FURPS+ (Functionality, Usability, Reliability, Performance, Supportability + extensões) foi desenvolvido pela Hewlett-Packard para ajudar na categorização e análise de requisitos de qualidade de software. Esta abordagem é amplamente utilizada na engenharia informática como referência para avaliar a completude e equilíbrio de requisitos não funcionais. (Grady, 1992)

RNF07 – A interface do chatbot deve ser visualmente apelativa, destacando-se de forma clara sem interferir com o conteúdo principal da aplicação.
Esta característica permite que o chatbot seja rapidamente identificado pelo utilizador, sem comprometer a usabilidade geral da interface onde está inserido.

RNF08 – A comunicação entre o sistema e o sistema de gestão de tickets deve estar protegida por autenticação segura e encriptação de dados.
Este requisito garante que as integrações com serviços externos são feitas de forma segura, evitando fugas de informação e acessos indevidos a dados sensíveis.

RNF09 – O sistema deve assegurar a proteção de dados sensíveis fornecidos pelos utilizadores.
Este requisito garante que toda a informação partilhada durante as interações, como conteúdos de pedidos ou dados pessoais, é tratada de forma segura, evitando exposições indevidas ou acessos não autorizados.

RNF10 – O sistema deve possuir documentação detalhada quanto à sua constituição, funcionamento e modo de utilização.
Este requisito garante que o sistema pode ser compreendido e mantido por outros elementos da equipa ou por futuros responsáveis, mesmo após a conclusão da dissertação. A documentação deve incluir descrições dos componentes, fluxos de comunicação, endpoints da API, estrutura de dados e instruções para instalação e integração. A presente dissertação contribui diretamente para o cumprimento deste requisito.

O cumprimento destes requisitos não funcionais é essencial para garantir que o sistema desenvolvido não só responde às necessidades funcionais identificadas, mas também assegura um elevado padrão de qualidade ao nível da performance, segurança, fiabilidade, usabilidade e facilidade de manutenção. Esta abordagem permite criar uma solução robusta, escalável e preparada para evoluir com as necessidades da organização, contribuindo para uma experiência de suporte mais eficaz, automatizada e satisfatória para os utilizadores finais.

3.2 Desenho da Solução

O sistema proposto foi concebido tendo como objetivo principal garantir o cumprimento de todos os requisitos funcionais e não funcionais identificados na fase de análise do problema. Para tal, o seu desenho teve em consideração os princípios fundamentais da engenharia informática, como a separação de responsabilidades, baixo acoplamento, elevada coesão, modularidade, e reutilização de componentes.

Neste capítulo será descrito o design da solução implementada, abordando a sua organização estrutural, os principais componentes que a integram e a forma como comunicam entre si. A descrição será apresentada de forma progressiva, iniciando-se com uma visão global do sistema e aprofundando-se posteriormente os seus principais módulos: o chatbot, a API de interligação e o sistema Jira.

Para facilitar a compreensão do sistema como um todo e permitir uma representação clara, visual e escalável dos seus elementos, optou-se por recorrer à utilização de diagramas baseados nos modelos **C4** (Brown, 2025) e **4+1** (Kruchten, 1995). Estes modelos foram concebidos precisamente com o intuito de permitir uma representação completa e multidimensional de sistemas de software, abrangendo tanto a perspetiva estrutural como comportamental, desde o nível mais abstrato até ao detalhe técnico.

3.2.1 Sistema Help Desk

Antes de aprofundar a análise dos componentes internos da solução, é importante compreender como o sistema Help Desk se apresenta externamente e com que entidades interage para cumprir a sua função de suporte ao utilizador.

A figura seguinte representa o sistema de forma abstrata, evidenciando as suas interfaces principais com o mundo exterior:

Figura 2 - Vista Lógica de Nível 1

Este diagrama apresenta o sistema como um componente encapsulado, que disponibiliza uma interface de entrada para os utilizadores e consome um serviço externo:

- **Interface do Utilizador**: representa o ponto de contacto entre o sistema Help Desk e o utilizador final. Esta interface é acessível a partir da aplicação onde o sistema está integrado, permitindo que o utilizador interaja com o chatbot e submeta os seus pedidos de suporte.
- **API[28] do Jira**: corresponde à única dependência externa do sistema. Através desta interface, o sistema Help Desk comunica com o sistema Jira, enviando e consultando informação relativa aos tickets de suporte.

[28] API (Application Programming Interface): Conjunto de definições e protocolos que permite que diferentes aplicações comuniquem entre si. Uma API expõe funcionalidades de um sistema de forma estruturada, possibilitando a sua utilização por outras aplicações sem necessidade de conhecer os detalhes internos da sua implementação. (Fielding, 2000)

Após a apresentação do sistema numa perspetiva abstrata, segue-se uma análise mais detalhada da sua composição interna, destacando os principais componentes responsáveis pelo seu funcionamento e a forma como estes interagem entre si.

A **Vista Lógica de Nível 2**, representada na figura seguinte, permite compreender a estrutura interna do sistema Help Desk e os fluxos de comunicação entre os seus elementos.

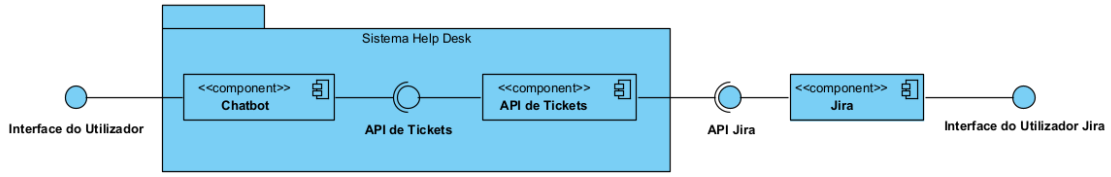


Figura 3 - Vista Lógica de Nível 2

O sistema é composto por **dois componentes internos** e **um componente externo**:

- **Chatbot**: componente interno responsável por fornecer o ponto de entrada para os utilizadores. Recebe os pedidos dos utilizadores e tenta resolvê-los com base numa base de conhecimento interna. Quando não é possível dar resposta direta, o chatbot prepara e encaminha a informação para a API de Tickets.

- **API de Tickets**: componente interno que atua como intermediário entre o chatbot e o sistema Jira. Recebe os dados do pedido a ser transformado em ticket, valida se existe duplicação e, caso necessário, cria um ticket no Jira. Para isso, consome a API do Jira e disponibiliza uma interface interna para ser utilizada pelo chatbot, através da qual recebe a informação para criação dos tickets.

- **Jira**: sistema externo de gestão de tickets onde os pedidos encaminhados pelo sistema Help Desk são efetivamente registados e acompanhados. Disponibiliza uma API, que é consumida pela API de Tickets para criação e consulta de tickets, e uma interface de utilizador, através da qual os utilizadores e os membros da equipa de suporte podem visualizar, atualizar e acompanhar o estado dos tickets criados.

Com base nesta arquitetura, podem ocorrer **dois fluxos distintos de informação**:

- **Fluxo 1 – Resolução direta pelo chatbot**:
  O utilizador submete um pedido através da interface conversacional. O chatbot analisa o conteúdo e, caso consiga responder com base na sua base de conhecimento, apresenta a resposta ao utilizador. A interação termina aqui.

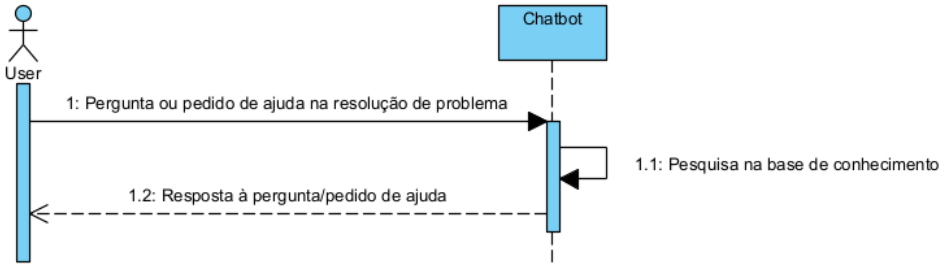


Figura 4 - Vista de Processos de Nível 2 - Fluxo 1

- **Fluxo 2 – Escalada para criação de ticket**:
  Quando o pedido não pode ser resolvido diretamente, o chatbot faz uma sugestão do ticket a ser criado e envia ao utilizador. Após este validar o seu conteúdo, o chatbot envia a solicitação de criação para a API de Tickets. A API de Tickets verifica se já existe um ticket semelhante. Se sim, envia o ticket existente para o utilizador acompanhar a sua resolução. Caso contrário, cria o ticket utilizando a API do Jira. Após a criação, a API de Tickets informa o chatbot, que por sua vez notifica o utilizador. Este pode então consultar o estado do ticket através da interface do Jira.

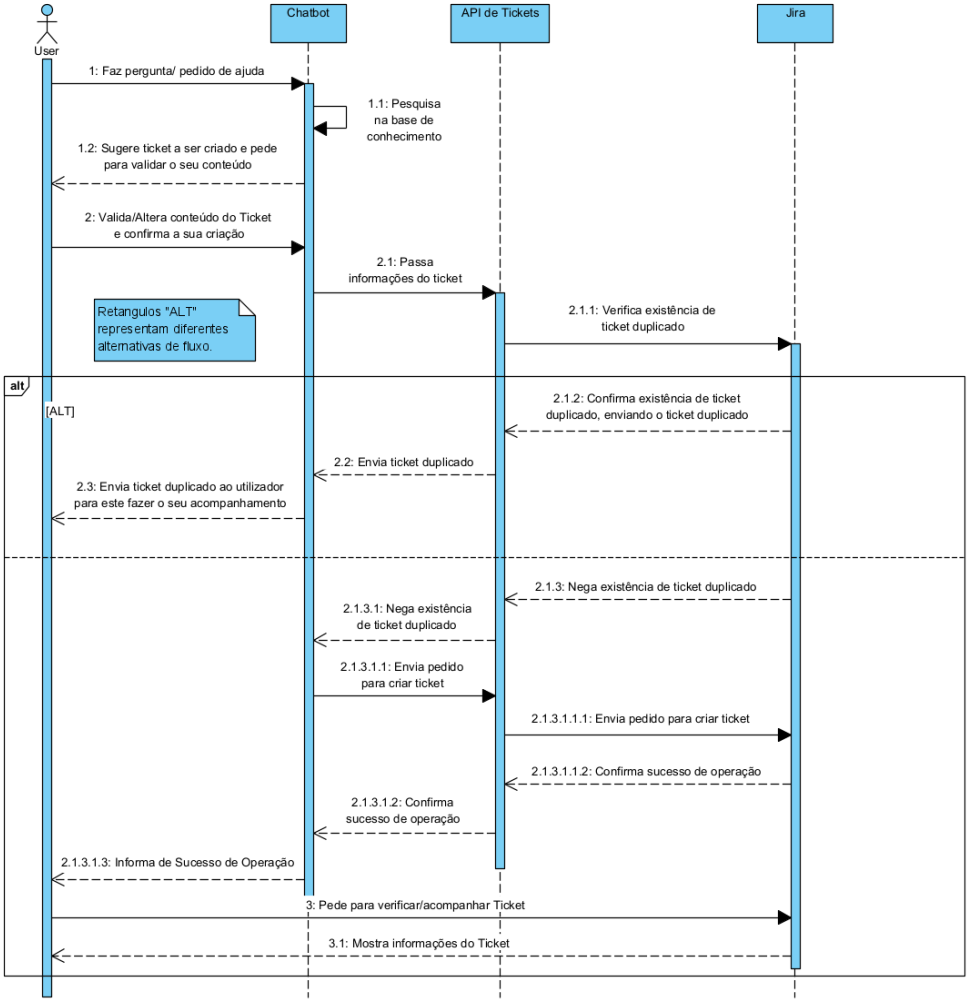


Figura 5 - Vista de Processos de Nível 2 - Fluxo 2

Esta separação clara de responsabilidades, aliada ao encapsulamento de cada componente e à utilização de interfaces bem definidas, contribui para um sistema robusto, modular e facilmente escalável.

Por fim, relativamente à arquitetura do sistema como um todo, é fundamental compreender onde cada componente irá residir fisicamente e de que forma ocorre a comunicação entre os diferentes elementos. A figura seguinte apresenta a **Vista Física de Nível 2**, que descreve a distribuição dos componentes por servidores e os protocolos utilizados para a sua comunicação:

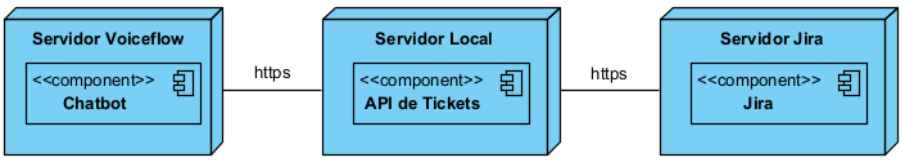


Figura 6 - Vista Física de Nível 2

Nesta vista, observa-se que cada componente principal do sistema está alocado a um servidor dedicado distinto, garantindo uma separação de responsabilidades e facilitando a escalabilidade e manutenção da solução.

A comunicação entre os componentes é realizada exclusivamente através do protocolo HTTPS[29], assegurando a segurança, confidencialidade e integridade dos dados transmitidos. Este protocolo encriptado é essencial para proteger as interações entre sistemas, especialmente quando se lida com dados sensíveis ou informações pessoais dos utilizadores. Desta forma, fica garantido o cumprimento do Requisito Não Funcional 10 (RNF10), relativo à segurança da comunicação entre sistemas.

A adoção desta arquitetura distribuída e segura garante a robustez da solução, permitindo uma integração fluida entre componentes internos e externos, bem como a adaptabilidade do sistema a diferentes contextos de execução.

Alternativa de Design:

Com o intuito de comprovar que a opção de design escolhida foi a mais adequada, foi também considerado um modelo alternativo para o sistema, igualmente viável do ponto de vista funcional.

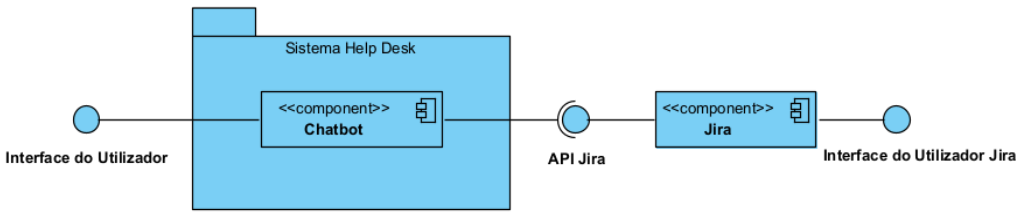


Figura 7 - Vista Lógica de Nível 2 - Alternativa de Design para Sistema Helpdesk

Neste modelo, o sistema passaria a ter apenas dois componentes principais: o Chatbot e o sistema externo Jira. O objetivo seria reduzir a complexidade estrutural e o número de componentes físicos envolvidos, evitando a necessidade de gerir a implantação de um servidor adicional. No entanto, para isso, o Chatbot teria de acumular toda a lógica de comunicação com o utilizador e com o Jira, ficando responsável por construir os pedidos, validar entradas, verificar duplicados e comunicar diretamente com a API externa.

Embora esta abordagem possa parecer vantajosa pela sua simplicidade inicial, a sua adoção traria várias desvantagens:

[29] Protocolo de comunicação que garante que os dados trocados entre sistemas estejam protegidos contra acessos não autorizados ou alterações durante a transmissão.

- **Responsabilidade única violada**: o Chatbot deixaria de ser um componente exclusivamente conversacional, acumulando também responsabilidades de negócio e integração técnica.
- **Baixa coesão**: misturar lógica de interação com lógica de backend prejudicaria a organização interna do componente, tornando-o mais difícil de manter.
- **Menor capacidade de reutilização**: a lógica de criação de tickets ficaria acoplada ao Chatbot, impedindo o reaproveitamento por outras interfaces que no futuro possam interagir com o sistema.
- **Acoplamento elevado**: a ligação direta entre o Chatbot e o Jira dificultaria testes isolados, substituições tecnológicas e introdução de mecanismos de monitorização ou segurança.
- **Escalabilidade limitada**: qualquer alteração na lógica de tickets exigiria mudanças diretas no Chatbot, comprometendo a sua estabilidade e obrigando a atualizações mais frequentes.

Já o design implementado, com um componente dedicado à API de Tickets, permite separar claramente a lógica conversacional da lógica de negócio e integração, tornando o sistema mais modular, reutilizável e escalável.

Esta comparação entre abordagens reforça a validade do design escolhido e demonstra que, apesar de alternativas mais simples existirem, a solução final oferece um compromisso mais sólido entre clareza arquitetónica, manutenibilidade e preparação para o crescimento futuro do sistema.

3.2.2 Chatbot

O chatbot proposto para o sistema Help Desk será desenvolvido com recurso à plataforma Voiceflow, que permite a criação de experiências conversacionais através de uma abordagem *low-code*[30]*/no-code*[31]. Este componente atua como o ponto de entrada do sistema, sendo responsável por interpretar os pedidos dos utilizadores, tentar resolvê-los autonomamente com base numa base de conhecimento, e encaminhá-los para a criação de tickets quando necessário.

Para descrever a arquitetura e o funcionamento do chatbot, optou-se por utilizar igualmente os modelos C4 e 4+1, adaptando-os ao contexto de desenvolvimento *low-code*. Embora tradicionalmente aplicados em projetos de software convencionais, estes modelos continuam

[30] Abordagem de desenvolvimento de software que utiliza interfaces visuais e ferramentas de arrastar e largar para criar aplicações com pouca escrita de código manual, acelerando o desenvolvimento e permitindo a participação de utilizadores com conhecimentos técnicos moderados.

[31] Metodologia que permite criar aplicações completas sem escrever qualquer linha de código, através de interfaces visuais intuitivas, geralmente destinada a utilizadores sem formação em programação.

adequados para representar componentes lógicos e fluxos de informação, mesmo em soluções construídas com plataformas visuais.

A seguinte figura apresenta a Vista Lógica de Nível 3 do Chatbot, onde são identificados os principais módulos e componentes responsáveis pelo tratamento dos diferentes tipos de pedidos.

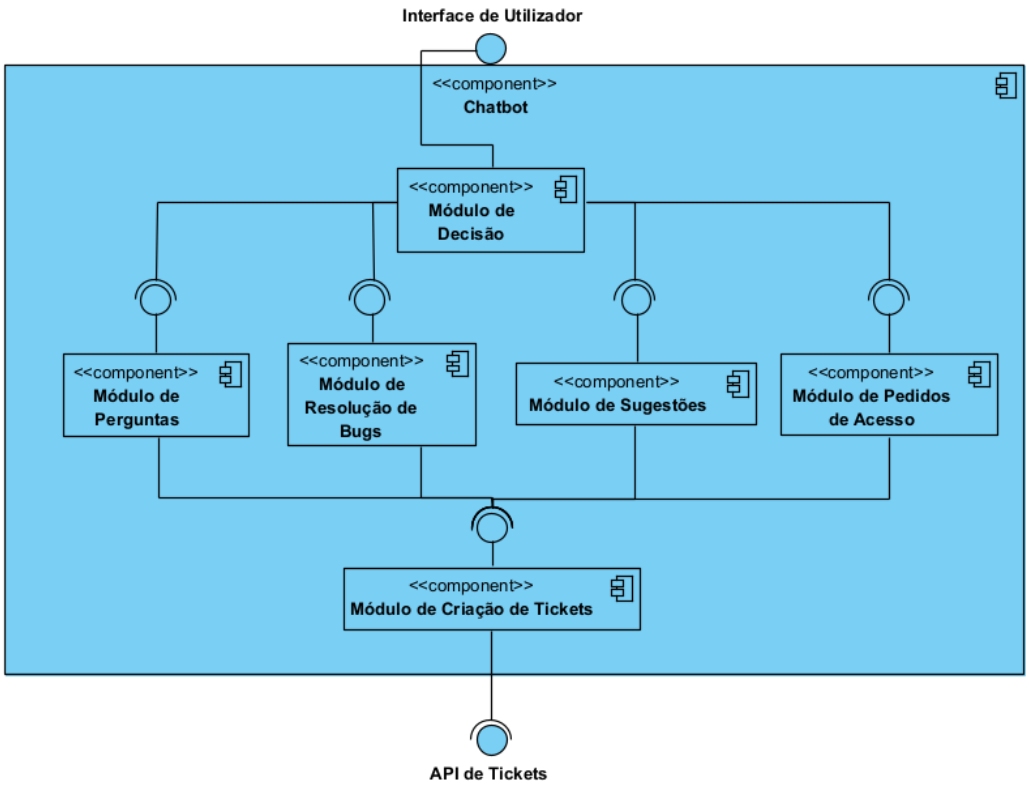


Figura 8 - Vista Lógica de Nível 3 do Chatbot

Para promover uma clara segregação de responsabilidades e facilitar a manutenção e evolução futura do sistema, cada tipo de pedido é tratado num módulo independente. Adicionalmente, o "Módulo de Decisão" é responsável por analisar a intenção do utilizador e encaminhar o pedido para o módulo apropriado, enquanto o "Módulo de Escrita de Tickets" centraliza a comunicação com a API de Tickets, sendo responsável pela criação de Tickets.

Assim, os módulos principais são:

- **Módulo de Decisão**: interpreta o pedido do utilizador e determina a sua natureza, direcionando-o para o módulo de tratamento adequado.
- **Módulo de Perguntas**: responde a dúvidas gerais sobre o funcionamento da aplicação.

- **Módulo de Resolução de Bugs**: auxilia na identificação e resolução de comportamentos inesperados ou erros reportados.
- **Módulo de Sugestões**: recolhe propostas de melhoria feitas pelos utilizadores.
- **Módulo de Pedidos de Acesso**: trata pedidos relacionados com permissões ou alterações de perfil.
- **Módulo de Escrita de Tickets** – recebe informações provenientes dos módulos de tratamento de pedidos dos utilizadores e comunica com a API de Tickets para tratar da lógica de criação e verificação de duplicação de tickets.

O núcleo de cada módulo deverá ser um Agente Conversacional, entidade responsável por conduzir o diálogo com o utilizador de forma orientada e dinâmica. Um agente conversacional pode ser definido como uma unidade lógica especializada em gerir interações dentro de um determinado contexto, seguindo orientações e comportamentos previamente definidos. Esta estrutura permite uma comunicação natural e adaptada ao utilizador, assegurando que cada tipo de pedido é tratado de forma eficaz e consistente.

Adicionalmente, dado que o funcionamento dos agentes conversacionais envolve o consumo de tokens (recursos computacionais utilizados para processar texto), os fluxos conversacionais foram desenhados com esse fator em mente. Procurou-se reduzir o número de interações desnecessárias, promovendo mensagens mais claras e objetivas, de forma a otimizar o uso de tokens sem comprometer a eficácia do suporte prestado.

Para cada módulo e respetivo agente conversacional foi desenhada uma Vista de Processo, com o objetivo de representar de forma clara o fluxo conversacional esperado em cada tipo de interação.

Módulo de Decisão:

O Módulo de Decisão é o ponto de entrada lógico do chatbot e tem como função principal analisar o pedido inicial do utilizador para compreender a sua natureza e determinar qual o módulo especializado que deverá tratá-lo.

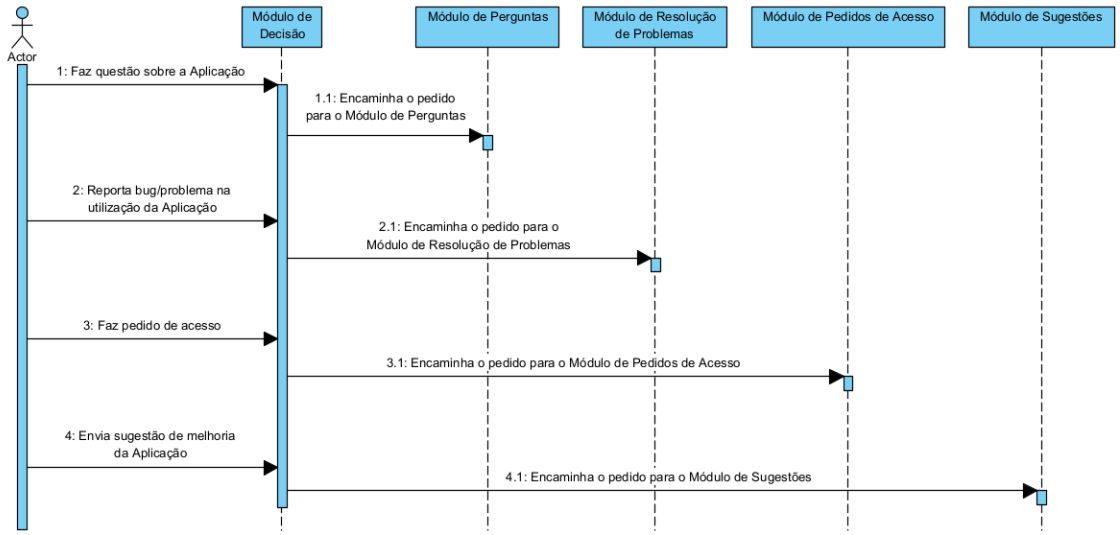


Figura 9 - Vista de Processos de Nível 3 - Módulo de Decisão

A Figura 9 ilustra os diferentes fluxos possíveis de interação com este módulo. Quando o utilizador submete o seu pedido, o agente conversacional associado ao módulo interpreta a intenção da mensagem com base em palavras-chave, padrões de linguagem e contexto. A partir dessa análise, o pedido é encaminhado para um dos seguintes módulos especializados:

- **Módulo de Perguntas** – para questões relacionadas com a utilização geral da aplicação.
- **Módulo de Resolução de Bugs** – quando é reportado um erro, falha ou comportamento inesperado.
- **Módulo de Pedidos de Acesso** – em casos onde o utilizador solicita permissões ou acessos adicionais à plataforma.
- **Módulo de Sugestões** – para quando são submetidas ideias ou propostas de melhoria da aplicação.

Este mecanismo permite isolar logicamente os fluxos de tratamento consoante a natureza do pedido, simplificando a gestão interna do chatbot e preparando o sistema para uma escalabilidade mais facilitada.

Módulo de Perguntas:

O Módulo de Perguntas é responsável por tratar questões relacionadas com a utilização da aplicação, dúvidas frequentes e dificuldades pontuais expressas pelos utilizadores. Recebe os pedidos encaminhados pelo Módulo de Decisão e tenta resolvê-los autonomamente com base na base de conhecimento existente. Caso não consiga resolver, gera uma sugestão de ticket a ser criado e encaminha um pedido de criação para o Módulo de Criação de Tickets.

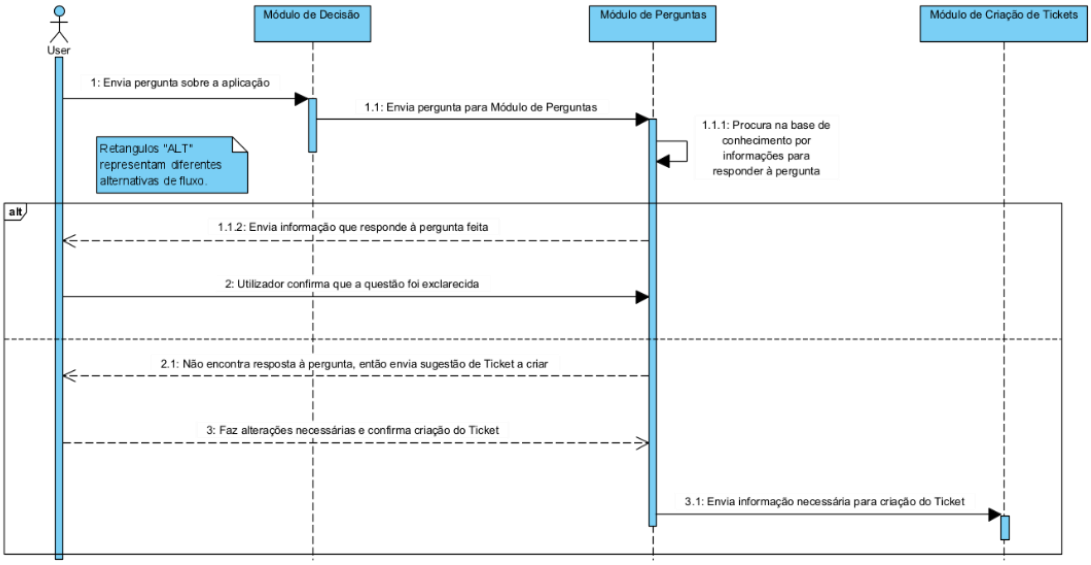


Figura 10 - Vista de Processos de Nível 3 - Módulo de Perguntas

O fluxo esperado de funcionamento, representado na Figura 10, segue os seguintes passos:

1. O utilizador envia uma questão sobre a aplicação, que é encaminhada pelo Módulo de Decisão para o Módulo de Perguntas.
2. O módulo consulta a base de conhecimento em busca de uma resposta adequada.
3. Se for encontrada uma resposta satisfatória, esta é apresentada ao utilizador, que confirma se a dúvida foi esclarecida.
4. Caso não seja encontrada uma resposta ou o utilizador não considere a questão resolvida, é gerada uma sugestão de criação de ticket.
5. O utilizador pode rever e efetuar alterações à sugestão antes de confirmar a criação.
6. O pedido é então encaminhado para o Módulo de Criação de Tickets, juntamente com as informações necessárias para prosseguir com o processo.

Este módulo reforça o objetivo de oferecer uma primeira linha de suporte autónoma, reduzindo a carga da equipa humana e garantindo que apenas os pedidos não resolvidos automaticamente são encaminhados para tratamento manual.

Módulo de Resolução de Bugs:

O Módulo de Bugs é responsável por tratar reportes de erros ou comportamentos inesperados identificados pelos utilizadores na aplicação. Recebe este tipo de pedidos a partir do Módulo de Decisão e avalia, com base na base de conhecimento existente, se o problema descrito corresponde de facto a um bug ou se resulta de uma má interpretação do funcionamento

normal do sistema. Caso seja identificado um erro genuíno, o módulo elabora uma sugestão de ticket e encaminha o pedido para o Módulo de Criação de Tickets.

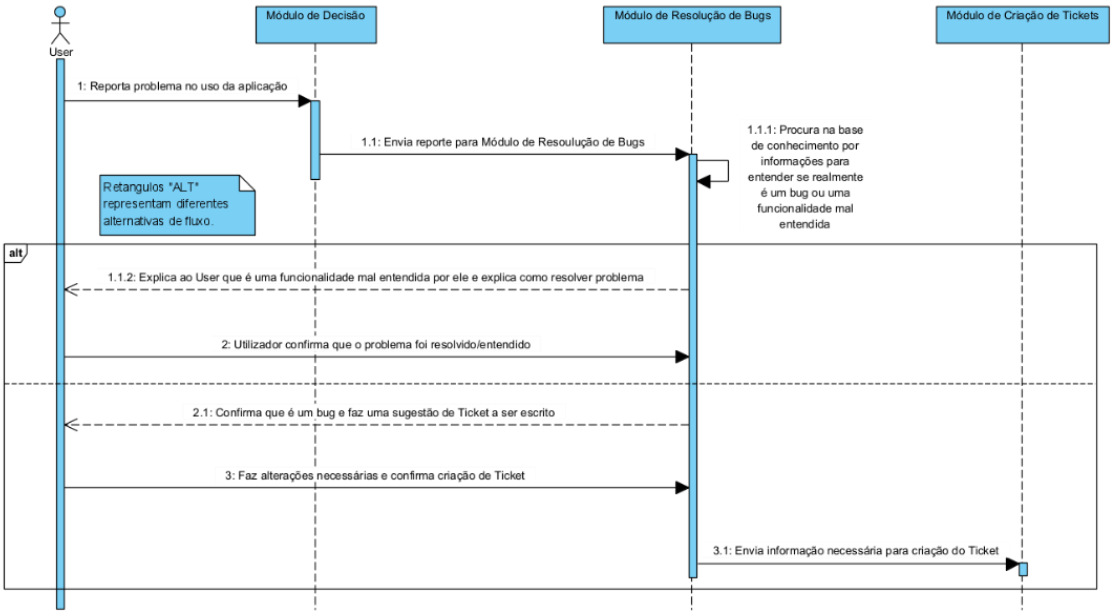


Figura 11 - Vista de Processos de Nível 3 - Módulo de Resolução de Bugs

O fluxo esperado de funcionamento, representado na Figura 11, segue os seguintes passos:

1. O utilizador reporta um problema no uso da aplicação, que é encaminhado pelo Módulo de Decisão para o Módulo de Resolução de Bugs.
2. O módulo consulta a base de conhecimento para verificar se o comportamento descrito é realmente um bug ou uma funcionalidade mal interpretada.
3. Caso seja identificado como comportamento esperado, o sistema apresenta uma explicação ao utilizador e orienta-o na resolução da dificuldade.
4. Caso seja confirmado que se trata de um bug, é gerada uma sugestão de criação de ticket.
5. O utilizador pode rever e efetuar alterações na sugestão antes de confirmar a criação.
6. O pedido é encaminhado para o Módulo de Criação de Tickets, juntamente com as informações necessárias para prosseguir com o processo.

Este módulo assegura que os bugs são corretamente filtrados e formalizados, evitando a criação de tickets redundantes e garantindo que apenas os erros genuínos chegam à equipa de suporte técnico. Ao mesmo tempo, promove a literacia do utilizador ao esclarecer comportamentos legítimos do sistema que possam ser confundidos com falhas.

Módulo de Pedidos de Acesso:

O Módulo de Pedidos de Acesso é responsável por tratar solicitações relacionadas com acessos a sistemas, recursos ou permissões específicas da aplicação. Recebe os pedidos encaminhados pelo Módulo de Decisão e valida se o pedido é permitido, considerando o perfil do utilizador e uma lista de acessos válidos. Quando validado, comunica com o Módulo de Criação de Tickets para criar o ticket correspondente ao pedido de acesso.

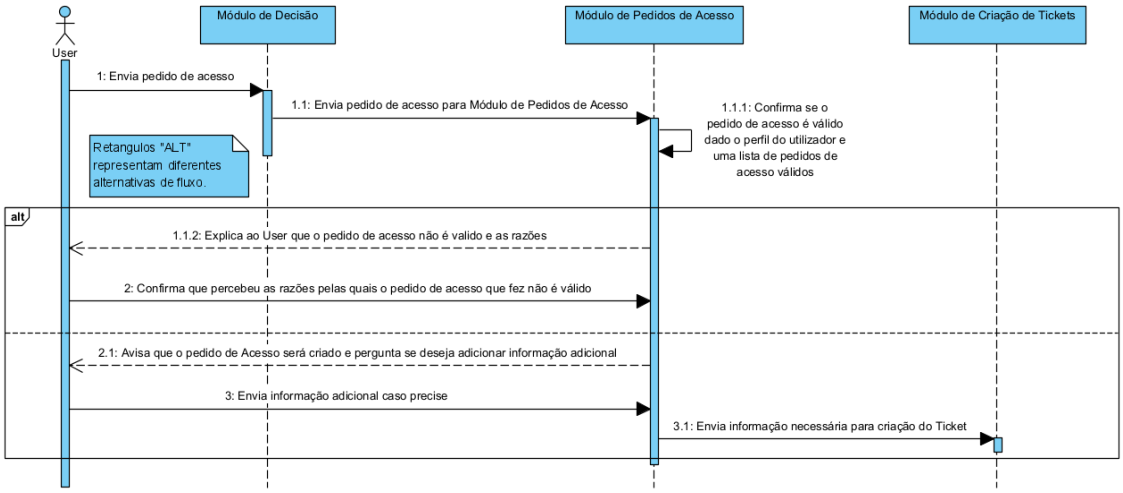


Figura 12 - Vista de Processos de Nível 3 – Módulo de Pedidos de Acesso

O fluxo esperado de funcionamento, representado na Figura 12, segue os seguintes passos:

1. O utilizador envia um pedido de acesso, que é encaminhado pelo Módulo de Decisão para o Módulo de Pedidos de Acesso.
2. O módulo verifica se o pedido é válido com base no perfil do utilizador e numa lista de pedidos permitidos.
3. Caso o pedido não seja válido, o sistema explica ao utilizador as razões e o processo é encerrado.
4. Se o pedido for válido, o módulo informa o utilizador de que o ticket será criado e questiona se deseja acrescentar informação adicional.
5. O utilizador pode enviar informação adicional, se necessário.
6. O pedido, com toda a informação reunida, é encaminhado para o Módulo de Criação de Tickets, que se encarrega de efetuar a comunicação com a API de Tickets.

Este módulo garante que os pedidos de acesso são validados e documentados de forma completa logo na sua origem, evitando trocas de informação desnecessárias e permitindo que a equipa de suporte atue de forma mais rápida e eficaz.

Módulo de Sugestões:

O Módulo de Sugestões é responsável por tratar propostas de melhoria enviadas pelos utilizadores, como ideias para otimizar a aplicação, implementar novas funcionalidades ou alterar comportamentos existentes. Recebe as sugestões encaminhadas pelo Módulo de Decisão e valida a sua relevância, verificando se já existe a funcionalidade proposta ou se se enquadra no contexto atual da aplicação.

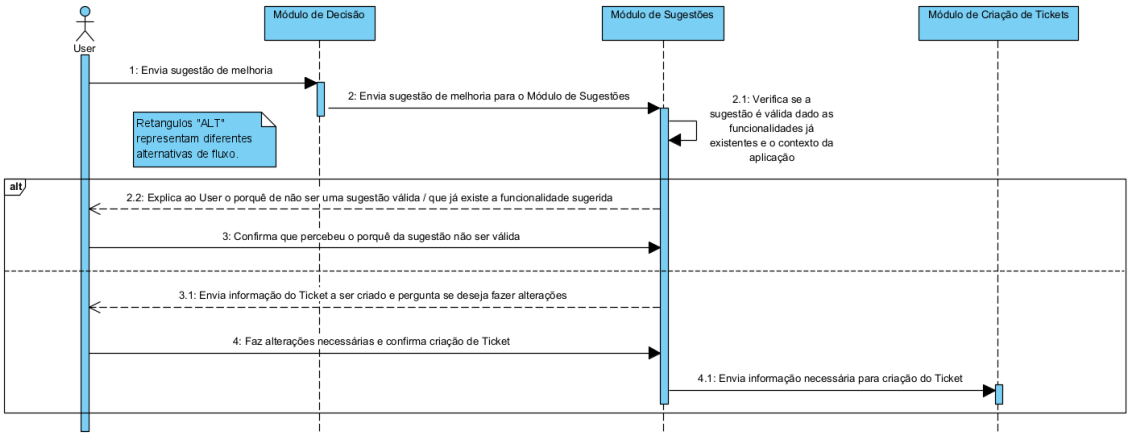


Figura 13 - Vista de Processos de Nível 3 - Módulo de Sugestões

O fluxo esperado de funcionamento, representado na Figura 13, é o seguinte:

1. O utilizador envia uma sugestão de melhoria, que é encaminhada pelo Módulo de Decisão para o Módulo de Sugestões.
2. O módulo verifica se a sugestão é válida, tendo em conta funcionalidades já existentes e o contexto da aplicação.
3. Caso a sugestão não seja considerada válida, o sistema explica ao utilizador o motivo e, após essa explicação, o processo é encerrado.
4. Se for válida, é gerada uma proposta de ticket que é apresentada ao utilizador, podendo este confirmar ou alterar o conteúdo.
5. O pedido é encaminhado para o Módulo de Criação de Tickets, juntamente com as informações necessárias para prosseguir com o processo.

Este módulo garante que apenas sugestões relevantes e viáveis são registadas, promovendo um canal direto de participação dos utilizadores na evolução do produto e reduzindo o esforço manual da equipa de desenvolvimento.

Módulo de Criação de Tickets:

O Módulo de Criação de Tickets é responsável por centralizar a lógica de criação de tickets, garantindo que cada solicitação seja registada de forma consistente e evitando duplicações. Recebe pedidos provenientes dos Módulos de Perguntas, Resolução de Bugs, Pedidos de Acesso e Sugestões, e comunica com a API de Tickets para efetuar as operações necessárias para a criação de Tickets.

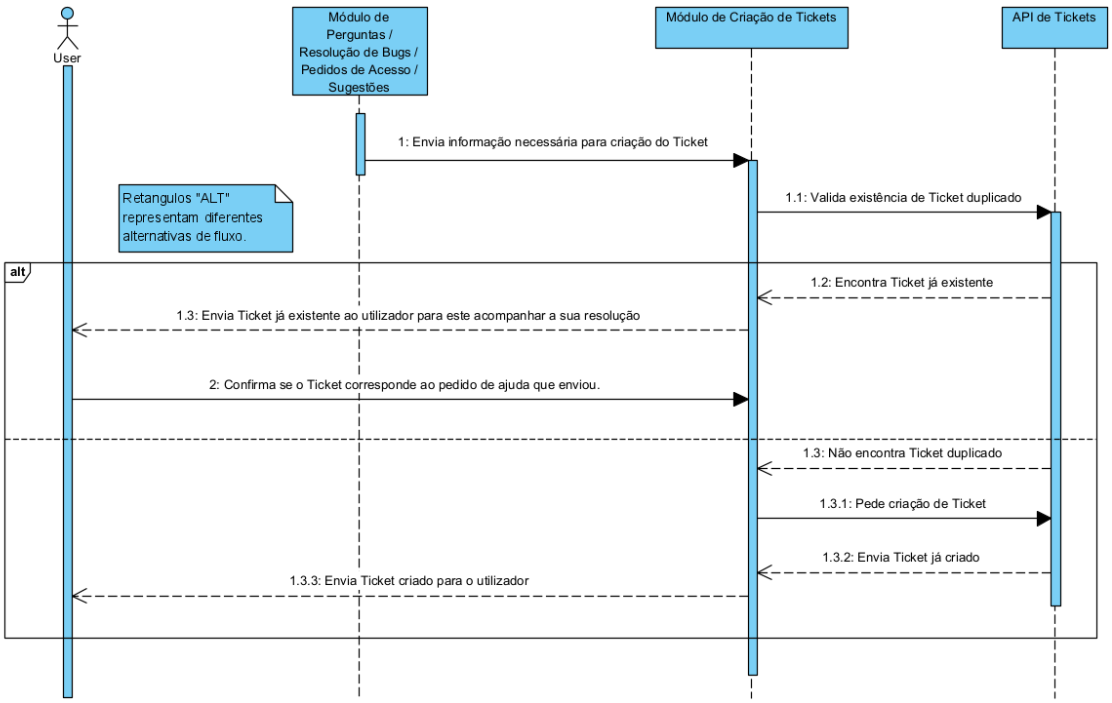


Figura 14 - Vista de Processos de Nível 3 - Módulo de Criação de Tickets

O fluxo de funcionamento esperado, representado na Figura 14, é o seguinte:

1. O módulo recebe as informações necessárias para a criação do ticket, enviadas por um dos módulos de suporte ao utilizador.
2. O módulo valida, através da API de Tickets, se já existe um ticket semelhante no sistema.
3. Caso seja encontrado um ticket já existente, envia o link ao utilizador para que possa acompanhar a sua resolução.
4. Se não existir duplicado, solicita à API de Tickets a criação de um novo ticket.
5. Após a criação, o link do ticket é enviado ao utilizador.

Este módulo assegura que a criação de tickets é feita de forma padronizada, evitando registos redundantes e garantindo que todas as solicitações sejam corretamente formalizadas e rastreáveis no sistema.

Alternativa de Design:

Com o intuito de validar a opção arquitetónica tomada para o desenvolvimento do Chatbot, foi também considerado um design alternativo, igualmente funcional e tecnicamente viável. A Figura X apresenta esta proposta alternativa, onde, ao contrário do modelo implementado, apenas existe um único Agente Conversacional responsável por todo o fluxo de interação com o utilizador.

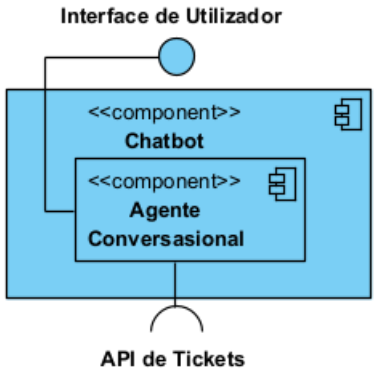


Figura 15 - Vista Lógica de Nível 3 - Alternativa de Design para o Chatbot

Neste cenário, o agente teria de identificar autonomamente o tipo de pedido efetuado, aceder à base de conhecimento, fornecer respostas diretas sempre que possível e, quando necessário, construir e enviar os dados de criação de ticket para a API de Tickets. Esta abordagem tem como vantagem a sua menor complexidade estrutural, já que evita a segmentação de módulos e a necessidade de coordenação entre agentes diferentes. Além disso, a ausência de transição de contexto entre componentes eliminaria possíveis perdas de informação ou incoerências na experiência do utilizador.

Contudo, esta solução concentra demasiadas responsabilidades num único componente, comprometendo a coesão interna e dificultando a manutenção e escalabilidade futuras. Um agente com múltiplas funções tende a crescer em complexidade à medida que novas funcionalidades são adicionadas, tornando o seu comportamento mais difícil de testar, validar e atualizar. Adicionalmente, a reutilização de partes específicas da lógica torna-se limitada, já que não existe uma clara separação entre blocos funcionais.

Já o design implementado, baseado na divisão por módulos especializados, promove uma organização mais limpa e modular. Cada agente trata exclusivamente de um tipo de pedido, facilitando a evolução isolada de funcionalidades, a distribuição de tarefas de desenvolvimento e a deteção de erros localizados. Esta abordagem também permite maior flexibilidade,

podendo-se, no futuro, escalar ou substituir apenas módulos específicos sem impactar o sistema global.

3.2.3 API de Tickets

Tal como descrito anteriormente, a API de Tickets desempenha um papel central no fluxo do sistema Help Desk, sendo responsável por receber os pedidos provenientes do chatbot, efetuar uma verificação inicial de possíveis duplicados e proceder à criação do ticket no Jira. O seu correto funcionamento é determinante para garantir que as solicitações dos utilizadores são registadas de forma consistente, evitando redundâncias, e assegurando a integridade do processo de gestão de tickets.

Arquitetura:

Para o desenvolvimento da API de Tickets foi escolhida uma arquitetura que permitisse manter o sistema simples, dado que a sua responsabilidade atual se limita à criação e verificação de tickets no Jira. No entanto, era também fundamental garantir que essa arquitetura suportasse futuras evoluções, como a integração com outras plataformas ou a adição de novas funcionalidades relacionadas com a gestão de tickets.

Para isso, optou-se por uma arquitetura ***Layered*** (Bass, et al., 2012) (ou **N-tier** (Fowler, 2002)) simplificada, representada na imagem seguinte:

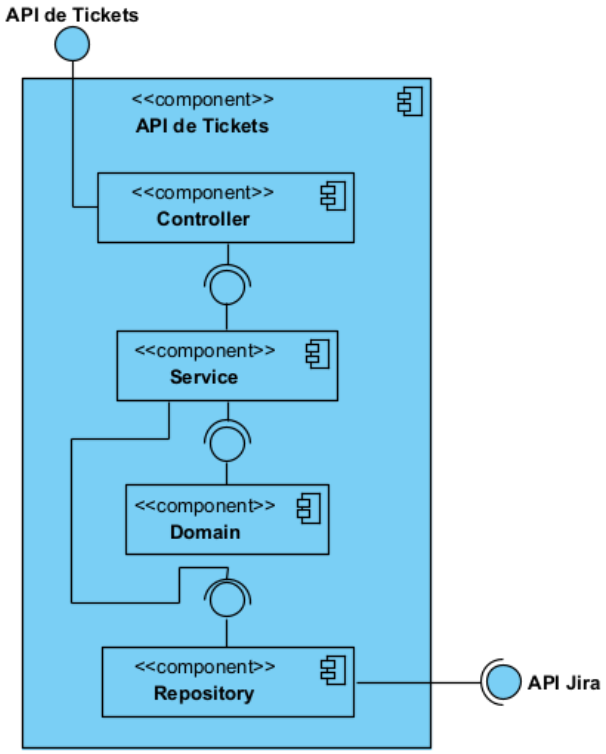


Figura 16 - Vista Lógica de Nível 3 - API de Tickets

Este tipo de arquitetura promove a separação de responsabilidades através da divisão da aplicação em várias camadas independentes:

- ***Controller***: camada responsável por receber os pedidos HTTPS, expor os endpoints[32] e validar os dados de entrada.
- ***Service***: camada que implementa a lógica da aplicação, orquestrando os processos entre o *controller* e o acesso externo ao Jira.
- ***Repository***: camada que abstrai a comunicação com sistemas externos (neste caso, a API do Jira) garantindo um ponto único de integração.
- ***Domain***: camada que contém os modelos de dados utilizados internamente, servindo de base para a troca de informação entre camadas.

Apesar de a lógica da aplicação ser bastante simples nesta fase, a escolha desta arquitetura traz várias vantagens:

- **Escalabilidade**: torna simples adicionar novos fluxos ou integrações externas sem comprometer a estrutura atual.
- **Facilidade de manutenção**: problemas ou melhorias podem ser tratados numa camada específica sem impactar todo o sistema.
- **Organização e legibilidade**: mesmo para equipas futuras, a estrutura é clara, reduzindo a curva de aprendizagem.
- **Preparação para testes automatizados**: a separação de responsabilidades favorece a criação de testes unitários e de integração.

Em suma, esta arquitetura permite manter a simplicidade exigida pelo contexto atual da API, enquanto assegura a robustez e extensibilidade necessárias para possíveis evoluções do sistema.

Diagrama de Classes:

A seguinte figura apresenta a Vista de Desenvolvimento de Nível 4, onde são representadas as classes, interfaces[33] e relações que compõem a API de Tickets. Esta vista descreve a organização interna do código e a função de cada elemento na execução dos fluxos de criação e verificação de tickets no Jira.

[32] Representa um ponto de entrada ou saída de comunicação numa API.

[33] Estrutura que define um conjunto de métodos que uma classe deve implementar, sem especificar como esses métodos são executados.

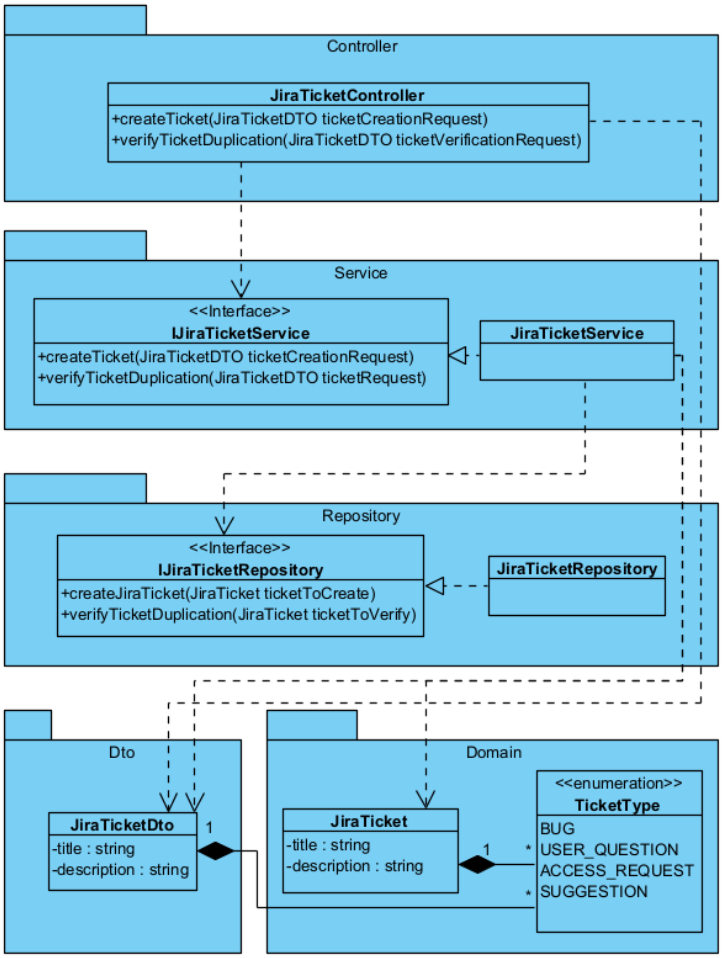


Figura 17 - Diagrama de Classes

A estrutura segue a arquitetura previamente descrita, estando as classes organizadas por camadas:

- ***Controller***: A classe *JiraTicketController* recebe os pedidos HTTPS e expõe os métodos públicos da API. Contém dois métodos principais:
    - *createTicket*: cria um ticket no Jira.
    - *verifyTicketDuplication*: verifica se já existe um ticket semelhante antes de criar um novo.
- ***Service***: A interface *IJiraTicketService* define as operações que a API disponibiliza e a classe *JiraTicketService* implementa essas operações, coordenando o processamento dos pedidos recebidos e interagindo com o repositório.

- ***Repository***: A interface *JiraTicketRepository* define os métodos de comunicação com a API do Jira, enquanto a classe *JiraTicketRepository* executa efetivamente essa comunicação para criar tickets e validar duplicados.

- **DTO**[34]: A classe *JiraTicketDTO* é utilizada para transportar os dados entre o cliente e a API, garantindo que a estrutura exposta publicamente não depende do modelo interno.

- ***Domain***: A classe *JiraTicket* representa internamente um ticket já validado. O enumerado *TicketType* define os tipos de ticket possíveis (*BUG, USER_QUESTION, ACCESS_REQUEST, SUGGESTION*).

A definição das classes que constituem a API teve em conta os seguintes padrões e princípios de desenvolvimento de software:

- **Padrão Controlador** (Fowler, 2002) – centraliza o tratamento das requisições HTTPS e a validação inicial dos dados, delegando a orquestração para a camada de serviço. Isto mantém o *endpoint* simples e previsível, facilitando testes e evolução da interface pública.

- **Padrão Camada de Serviço** (Fowler, 2002) – concentra a lógica de aplicação (ex.: verificação de duplicados e criação de tickets) e coordena repositórios, DTOs e classes de domínio.

- **Padrão Repositório** (Evans, 2004) – abstrai a integração com o Jira, expondo operações de alto nível (criação e verificação de duplicados). Além de centralizar o acesso, funciona como um adaptador entre o modelo interno e a API externa, isolando mudanças do fornecedor.

- **Objeto de Transferência de Dados (DTO)** (Fowler, 2002) – Representa uma estrutura de dados utilizada para transportar informação entre camadas da aplicação. A sua utilização evita expor diretamente as classes internas da API e facilita alterações futuras sem impactar quem consome o serviço.

- **Princípio da Inversão de Dependências + Injeção de Dependências** (Martin, 2003) – as camadas superiores (controlador/serviço) dependem de interfaces e não de implementações concretas. Isto permite trocar o *IJiraTicketRepository* por outro conector sem alterar a lógica de alto nível, o que facilita a criação de testes.

- **Separação de Responsabilidades** (Dijkstra, 1982) – responsabilidades bem divididas por camadas, o que melhora a legibilidade, testabilidade e distribuição de trabalho.

- **Princípio da Responsabilidade Única** (Martin, 2003)– cada classe possui apenas uma responsabilidade bem definida, o que reduz efeitos colaterais e simplifica a manutenção.

[34] Padrão de design de software usado para transportar dados entre diferentes camadas de uma aplicação, sem expor diretamente a lógica interna do sistema.

- **Princípio Aberto/Fechado** (Martin, 2003) – a API está aberta à extensão (novos provedores de tickets, novos fluxos) e fechada à modificação das classes existentes, graças ao uso de interfaces e inversão de dependências.

Em conjunto, esta arquitetura e os padrões aplicados promovem baixo acoplamento, alta coesão, testabilidade, escalabilidade e substituibilidade dos componentes, sendo características essenciais para assegurar que a API se mantém simples e adequada aos requisitos atuais, enquanto permanece preparada para suportar futuras integrações com outras plataformas e a adição de novos casos de utilização.

Criação de Tickets:

O sistema da API de Tickets foi desenvolvido para dar resposta a dois casos de uso principais:

1. Criação de um novo ticket no sistema Jira.
2. Verificação da existência de tickets duplicados, evitando registos redundantes.

A Figura 18 apresenta a Vista de Processos de Nível 4 relativa à criação de um ticket na API.

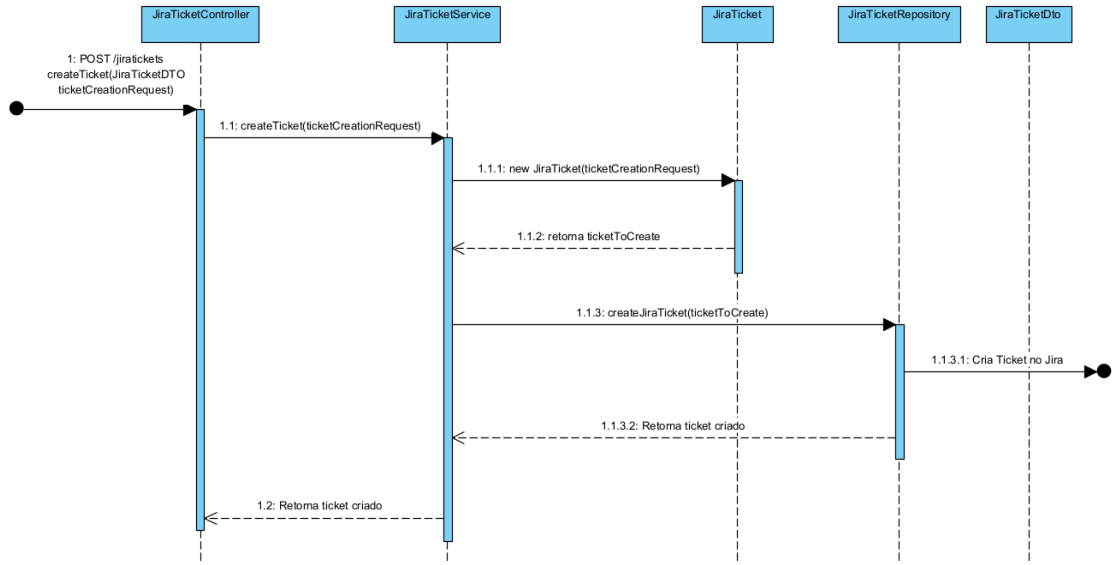


Figura 18 - Vista de Processos de Nível 4 - Criação de Ticket

O processo inicia-se quando o cliente da API (neste caso, o chatbot) envia um pedido HTTPS para o *endpoint* "/jiratickets", contendo um objeto *JiraTicketDTO* com os dados necessários à criação do ticket. O *JiraTicketController* recebe o pedido e invoca o método "*createTicket*" da camada de serviço.

O *JiraTicketService* converte o DTO num objeto de domínio *JiraTicket* e solicita ao *JiraTicketRepository* a criação do ticket, invocando o método "*createJiraTicket*". O repositório executa a chamada à API do Jira e retorna o ticket criado.

O resultado é propagado de volta à camada de serviço e posteriormente ao controlador, que responde ao cliente da API com a informação do ticket criado.

Verificação de Duplicados:

A Figura 19 apresenta a Vista de Processos de Nível 4 relativa à verificação da existência de tickets duplicados na API.

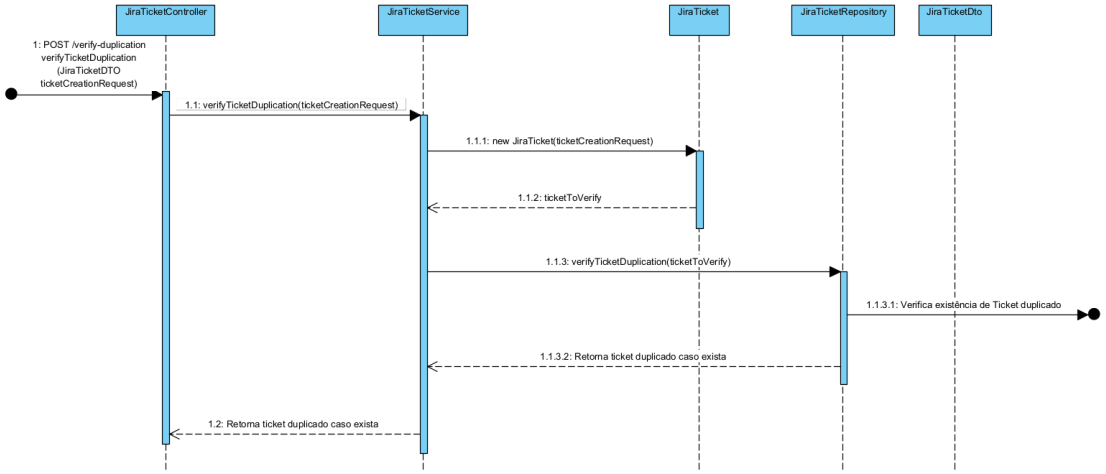


Figura 19 - Vista de Processos de Nível 4 – Verificação de existência de Ticket duplicado

O processo inicia-se quando o cliente da API (neste caso, o chatbot) envia um pedido HTTPS para o *endpoint "/verify-duplication*", contendo um objeto *JiraTicketDTO* com os dados do ticket a verificar. O *JiraTicketController* recebe o pedido e invoca o método "*verifyTicketDuplication*" da camada de serviço.

O *JiraTicketService* converte o DTO num objeto de domínio *JiraTicket* e solicita ao *JiraTicketRepository* a verificação da duplicação, invocando o método "*verifyTicketDuplication*". O repositório comunica com a API do Jira para confirmar a existência de tickets semelhantes.

O resultado da verificação é devolvido ao serviço, que o encaminha para o controlador. Este responde ao cliente com a indicação de se foi encontrado um ticket duplicado, juntamente com as informações relevantes caso exista.

3.2.4 Jira

O sistema Jira desempenha também um papel crucial no sistema, funcionando como a plataforma responsável pelo registo, acompanhamento e gestão dos tickets criados pelo chatbot. É através dele que tanto os utilizadores finais como a equipa de suporte da aplicação onde o chatbot está inserido conseguem visualizar o estado dos pedidos, acompanhar a sua resolução e garantir que cada ocorrência é tratada de forma estruturada. Compreender a forma correta de interagir com a API do Jira é fundamental para assegurar que os tickets são criados de forma consistente, com a informação necessária e dentro dos padrões de organização definidos para o projeto.

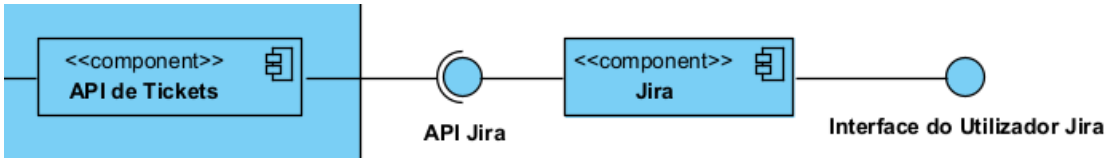


Figura 20 - Excerto da Vista Lógica de Nível 2 do Sistema Help Desk

Na solução a desenvolver, a API de Tickets comunica com o Jira através de dois endpoints principais:

- **Criação de Ticket (/issues)**
  Responsável por registar um novo ticket no projeto configurado no Jira. Requer informações como:
  - **Título**: breve resumo que identifica o ticket.
  - **Descrição**: detalhe do problema, sugestão ou pedido, podendo incluir contexto adicional.
  - **Tipo de Ticket**: classificação da ocorrência (ex.: *Bug* ou *Tarefa*).
  - **Labels**: etiquetas usadas para categorização e pesquisa, incluindo etiquetas fixas e dinâmicas geradas no momento.

  Após o processamento, o Jira devolve os dados do ticket criado, incluindo o seu **identificador único** e o **link de acesso** direto.

- **Verificação de Duplicados (/search)**
  Permite identificar tickets semelhantes já registados no sistema, evitando duplicações. A pesquisa é realizada com base em critérios como:
  - **Projeto** onde o ticket seria inserido.
  - **Tipo de Ticket** (ex.: *Bug* ou *Tarefa*).
  - **Título** do ticket, usado para correspondência aproximada.

- **Intervalo temporal** (ex.: tickets criados nos últimos 12 meses). Caso seja encontrado um ticket semelhante, são devolvidos detalhes como título, descrição, estado, etiquetas e o respetivo identificador.

Com estes dois pontos de integração, a API de Tickets cobre as necessidades atuais de criação e validação de tickets, garantindo um fluxo organizado e eficiente. Além disso, a estrutura escolhida facilita a extensão futura para outros endpoints do Jira ou integrações com plataformas adicionais.

3.3 Conclusões de Capítulo

Neste capítulo, foi detalhado o desenho da solução proposta para responder aos requisitos identificados na fase de análise. Foram apresentados os objetivos centrais da arquitetura e as motivações por detrás das principais decisões de design, assegurando que a solução não apenas cumpre as necessidades atuais, mas também está preparada para evoluções futuras.

Recorrendo aos modelos C4 e 4+1, a arquitetura foi descrita de forma estruturada, desde a visão global até ao detalhe dos componentes e respetivas interações. Cada módulo do sistema, incluindo o chatbot e a API de Tickets, foi analisado individualmente, evidenciando as suas responsabilidades, fluxos de informação e alternativas de design consideradas. Esta abordagem permitiu justificar tecnicamente as escolhas efetuadas, comparando-as com possíveis implementações alternativas e demonstrando as vantagens da solução proposta ao nível de modularidade, escalabilidade e manutenção.

Foram também identificados os padrões e princípios de desenvolvimento a aplicar, reforçando a preocupação com a qualidade, legibilidade e extensibilidade do sistema. Por fim, a integração com o Jira foi estudada em detalhe, clarificando os pontos de interação e as funcionalidades que suportam o ciclo de vida dos tickets no sistema.

Em síntese, este capítulo estabeleceu uma base sólida para a fase de implementação, garantindo que as decisões tomadas estão alinhadas com os objetivos funcionais e não funcionais definidos, e que a solução proposta apresenta um equilíbrio entre simplicidade inicial e capacidade de crescimento.

4 Implementação da Solução

No presente capítulo será descrito o processo de implementação da solução proposta, materializando o desenho arquitetural definido anteriormente em dois elementos principais: o chatbot, desenvolvido na plataforma Voiceflow, e a API de Tickets, implementada para integração com o sistema Jira. A exposição irá iniciar com a implementação do chatbot, detalhando a configuração da plataforma, os componentes utilizados, a construção dos módulos correspondentes a cada tipo de interação prevista e a integração com a aplicação de destino. Posteriormente, será apresentada a implementação da API de Tickets, evidenciando a estrutura do projeto bem como alguns dos seus constituintes, configurações e segurança da aplicação, os *endpoints* expostos, os mecanismos de comunicação com a API do Jira, manutenção e testes realizados, e por fim a sua implantação. Ao longo do capítulo, serão também descritos os fluxos de informação e as decisões de implementação mais relevantes, de forma a demonstrar a correspondência entre o design proposto e a solução concretizada.

4.1 Chatbot

Tal como já descrito anteriormente, o chatbot foi desenvolvido recorrendo à plataforma Voiceflow, uma solução low-Code/no-Code concebida para a criação de assistentes virtuais e chatbots de forma visual e intuitiva. A sua principal característica é permitir que aplicações conversacionais sejam desenvolvidas sem a necessidade de escrever código diretamente, recorrendo a um sistema baseado em blocos e ligações que representam ações, condições ou interações com o utilizador.

Figura 21 - Exemplo de Chatbot construído no Voiceflow

Na Figura 21 é apresentado um exemplo simples de um fluxo construído no Voiceflow. Neste caso, é possível observar um ponto de início (“Start”) que conduz o utilizador a uma primeira mensagem de saudação, seguida de opções que direcionam para diferentes módulos — por exemplo, um agente especializado na resolução de bugs ou um agente preparado para responder a perguntas. Cada agente, por sua vez, recolhe a informação necessária e encaminha-a para outros blocos, como o responsável por efetuar um pedido POST à API do Jira para criar um ticket. Este exemplo ilustra a abordagem visual do Voiceflow, sendo uma plataforma onde o programador trabalha com componentes que representam comportamentos ou integrações, conectando-os de forma a definir o fluxo conversacional.

4.1.1 Agentes Conversacionais

Tal como estudado anteriormente, os agentes conversacionais podem ser definidos como unidades autónomas de interação responsáveis por gerir um determinado tipo de diálogo entre o utilizador e o sistema. Cada agente é instruído a adotar um comportamento específico através de um prompt, que funciona como um guião onde são descritas a identidade, a forma de comunicação e as regras que regulam a sua atuação.

No caso da solução desenvolvida, e conforme definido no design de software, o chatbot é constituído por seis módulos distintos, correspondentes a diferentes tipos de pedidos. O núcleo de cada módulo é um agente conversacional com responsabilidades bem delimitadas.

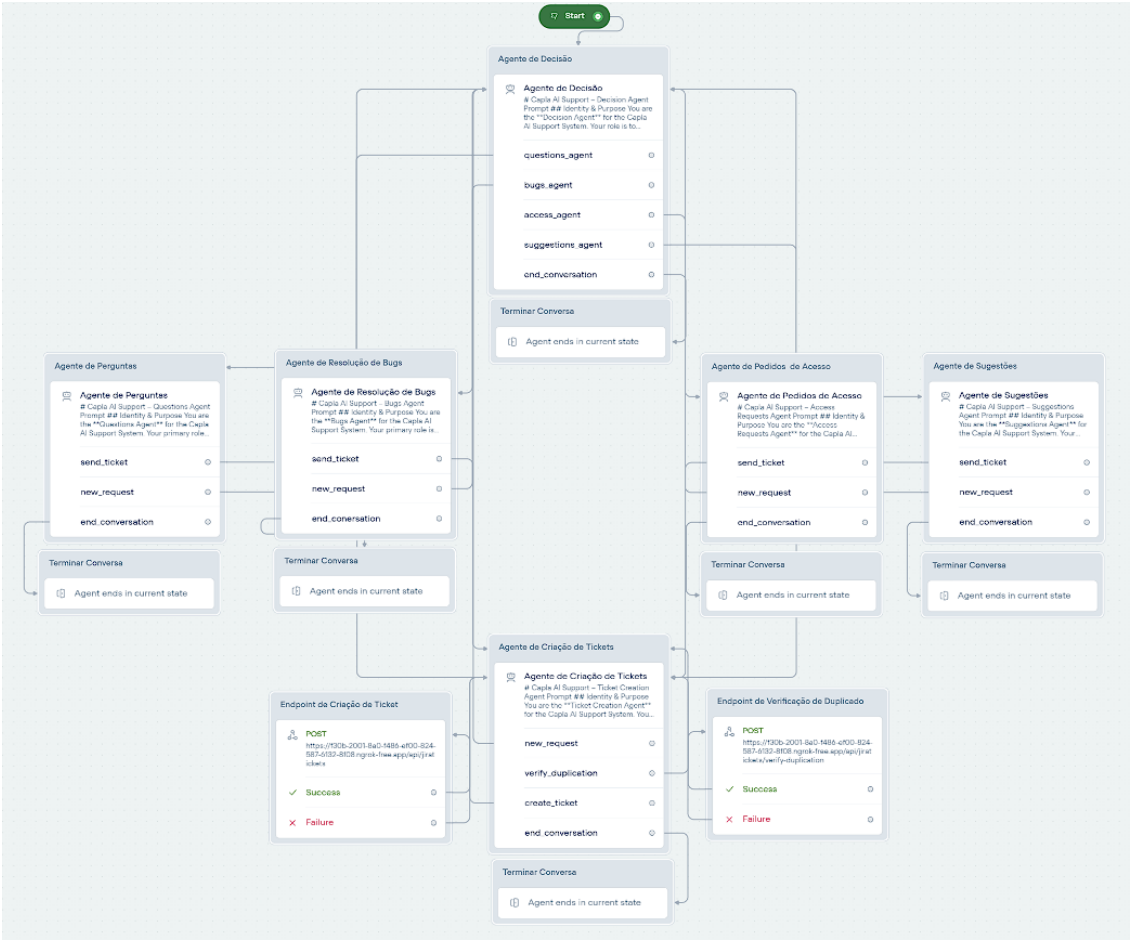


Figura 22 - Estrutura do chatbot implementado no Voiceflow

A parte mais importante da implementação de cada agente é a definição do ***prompt***, que garante que o comportamento é consistente e alinhado com o objetivo do módulo. A estrutura adotada baseia-se nas melhores práticas identificadas na literatura sobre *prompt engineering*, e segue sempre um mesmo padrão:

- **Identidade e Propósito** – Define qual o papel do agente no sistema e qual a sua responsabilidade principal.
- **Voz e Persona** – Descreve o tom de comunicação, estilo de escrita e postura que o agente deve adotar ao interagir com o utilizador.
- **Fluxo Conversacional** – Estabelece a sequência de passos a seguir durante a interação (início da conversa, identificação do pedido, tratamento ou encaminhamento, e encerramento).
- **Diretrizes de Resposta** – Apresentam regras para garantir consistência, clareza e brevidade nas respostas, bem como orientações sobre como reagir perante utilizadores insatisfeitos ou confusos.

- **Funcionalidade Específica** – explicita a tarefa concreta que o agente deve executar (por exemplo responder a perguntas, registar bugs, validar pedidos de acesso).
- **Cenários de Interação** – fornecem exemplos de situações típicas que o agente deve saber gerir dentro do seu âmbito de atuação.
- **Ações de Encaminhamento** – definem as variáveis a preencher e os caminhos (*paths*) a seguir quando é necessário redirecionar a conversa para outro agente ou transformar o pedido num ticket.

Esta estrutura garante que todos os agentes partilham uma base comum, promovendo consistência e previsibilidade na interação, mas garantindo que cada um se adapta de forma precisa à função específica para a qual foi criado.

Outros Componentes do Voiceflow:

Para operacionalizar os agentes conversacionais e os fluxos de conversa o Voiceflow introduz os seguintes mecanismos e componentes próprios:

- **Variáveis** – representam valores temporários usados na conversa e permitem passar informação entre agentes e componentes. Alguns exemplos utilizados na implementação:
    - *{user_request}:* armazena o pedido do utilizador.
    - *{ticket_title}, {ticket_description}, {ticket_type}:* utilizados na criação de tickets.
    - *{ticket_response}:* guarda o link do ticket devolvido pela API do Jira após a criação de um ticket.
- ***Paths*** – funcionam como rotas de saída pré-definidas que orientam a conversa para o próximo passo apropriado. Cada agente pode, no fim da sua atuação, escolher qual o path mais adequado consoante a interação do utilizador. Alguns exemplos utilizados na implementação são:
    - *end_conversation*: utilizado quando o utilizador indica que já não precisa de mais ajuda. Permite encerrar o diálogo de forma natural.
    - *new_request*: acionado quando o utilizador muda de assunto ou faz um pedido diferente do inicial. Faz com que a conversa seja reencaminhada novamente para o Agente de Decisão, que irá avaliar o novo pedido.
    - *send_ticket*: usado pelos agentes de suporte ao cliente quando o problema ou questão do utilizador precisa de ser formalizado. Encaminha a conversa para o Agente de Criação de Tickets, responsável pela lógica de verificação e registo de tickets.

- **Blocos de integração** – componentes que permitem chamadas externas a serviços (endpoints da API do Jira). Estes blocos utilizam os valores das variáveis e devolvem resultados processados por outros agentes.

Assim, enquanto os prompts definem a lógica comportamental de cada agente, os componentes do Voiceflow (variáveis, paths, blocos) asseguram a coordenação técnica entre eles, garantindo que a experiência para o utilizador decorre como se estivesse a falar sempre com um único assistente.

4.1.2 Base de Conhecimento

A Base de Conhecimento, em termos científicos, pode ser definida como o conjunto estruturado de informação que sustenta a atuação de um sistema conversacional. Constitui a memória explícita do chatbot, distinta dos mecanismos de processamento de linguagem natural (NLP), e contém o conhecimento adicional que permite à inteligência artificial contextualizar respostas, validar pedidos e oferecer suporte mais especializado. Funciona, assim, como um repositório de referência que complementa a capacidade de interpretação linguística com dados específicos do domínio em que o chatbot opera.

No caso específico do projeto desenvolvido, a base de conhecimento deve ser constituída por documentos e conteúdos textuais que descrevem a aplicação alvo, as suas funcionalidades, regras de utilização e demais informação relevante. A sua função é garantir que o chatbot tem acesso a um corpo de conhecimento consistente e validado, que pode ser consultado durante a interação com os utilizadores.

A sua importância reflete-se em vários pontos críticos do fluxo de funcionamento:

- **Respostas a dúvidas sobre a aplicação:** a documentação incluída permite que o Agente de Perguntas esclareça questões sobre funcionalidades, contexto de utilização ou processos internos.

- **Validação de reportes de bugs:** possibilita ao Agente de Resolução de Bugs distinguir entre um erro real e uma má interpretação do utilizador acerca do funcionamento esperado da aplicação.

- **Validação de sugestões de melhoria:** permite ao Agente de Sugestões garantir que as propostas apresentadas são coerentes com o estado atual do sistema e não representam funcionalidades já existentes.

- **Gestão de pedidos de acesso:** possibilita ao Agente de Pedidos de Acesso definir que tipos de permissões e acessos podem ser solicitados, por que perfis de utilizadores e em que circunstâncias, assegurando que apenas pedidos válidos são encaminhados para criação de tickets.

Para permitir a testabilidade do sistema e a realização dos casos de estudo, a base de conhecimento foi organizada com três blocos principais de informação:

1. **Documentação da aplicação** – descrição detalhada do propósito da aplicação, do seu funcionamento e das funcionalidades implementadas.

2. **Regras e perfis de utilizadores** – especificação das permissões atribuídas a cada tipo de utilizador e das condições em que podem solicitar acessos adicionais.

3. **Catálogo de funcionalidades e restrições** – lista estruturada das funções existentes, o que permite ao chatbot validar se uma sugestão é relevante ou se um bug reportado é na verdade uma funcionalidade já prevista.

Na plataforma Voiceflow, a adição de conteúdos à Base de Conhecimento pode ser realizada de diferentes formas:

- **Upload de ficheiros** (formatos como PDF, DOCX, TXT), que são diretamente processados e indexados.

- **Importação a partir de URLs** ou **Sitemaps**, útil para integrar documentação já publicada online.

- **Plain text**, onde é possível adicionar manualmente blocos de informação específicos.

Uma vez carregados, os conteúdos passam a estar disponíveis para todos os agentes do chatbot, que os consultam de forma implícita durante a formulação de respostas. Desta forma, a base de conhecimento funciona como uma camada transversal, garantindo coerência e suporte contextual em todos os módulos do sistema.

4.1.3 Agente de Decisão

O **Agente de Decisão** constitui o ponto de entrada do chatbot e é responsável por interpretar o pedido inicial do utilizador, classificando-o e direcionando a conversa para o agente mais adequado. A sua função central não é resolver o problema, mas sim compreender o pedido de forma clara e encaminhá-lo corretamente, assegurando que a interação decorre de forma natural e eficiente.

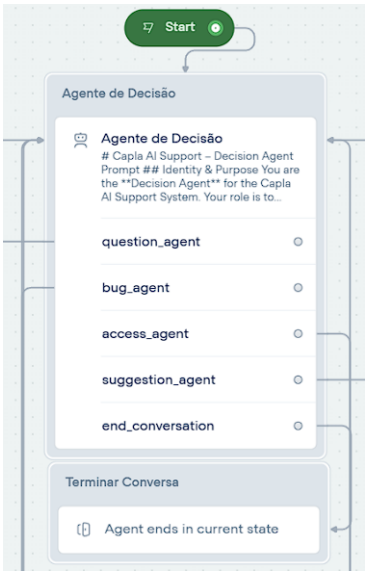


Figura 23 - Agente de Decisão

Sempre que inicia uma nova conversa, o agente solicita ao utilizador que descreva a sua questão. Essa descrição é armazenada na variável *{user_request}*, que funciona como a base para todo o processamento posterior. Com base nesse conteúdo, o agente avalia qual das categorias disponíveis melhor representa a intenção do utilizador:

- Pergunta sobre a aplicação
- Relato de problema ou bug
- Pedido de acesso ou permissões
- Sugestão de melhoria

Se o pedido não for claro, o agente formula questões adicionais de forma simples e direta, garantindo que só confirma a interpretação quando existe incerteza. Desta forma, evita interações redundantes e mantém a experiência conversacional fluida.

Uma segunda forma de atuação ocorre quando o Agente de Decisão recebe um *{user_request}* já preenchido por outro agente. Esta situação acontece quando o utilizador, durante uma interação, muda o seu pedido inicial é necessário reencaminhar a conversa novamente. Neste caso, o agente não apresenta qualquer saudação inicial, mas limita-se a reavaliar o pedido e a decidir qual o agente mais apropriado para prosseguir com o pedido de auxílio.

Os *paths* utilizados pelo Agente de Decisão refletem a sua função de encaminhamento:

- *question_agent* – encaminha para o Agente de Perguntas.
- *bug_agent* – encaminha para o Agente de Resolução de Bugs.
- *access_agent* – encaminha para o Agente de Pedidos de Acesso.

- *suggestion_agent* – encaminha para o Agente de Sugestões de Melhoria.

A decisão é concretizada através de condições que processam o conteúdo da variável *{user_request}* e determinam o *path* correspondente. Com este modelo, o Agente de Decisão assegura que a conversa começa corretamente orientada e que eventuais mudanças de pedido ao longo da interação são devidamente tratadas.

4.1.4 Agentes de Suporte ao Utilizador

Estes agentes constituem o núcleo da interação direta com o utilizador. São responsáveis por compreender o pedido encaminhado pelo Agente de Decisão, tratá-lo dentro do seu âmbito específico e decidir o rumo da conversa. Embora partilhem uma lógica comum (interpretação do pedido, tentativa de resolução imediata e eventual formalização sob a forma de ticket), cada agente possui responsabilidades próprias que garantem a cobertura de diferentes tipos de necessidades do utilizador.

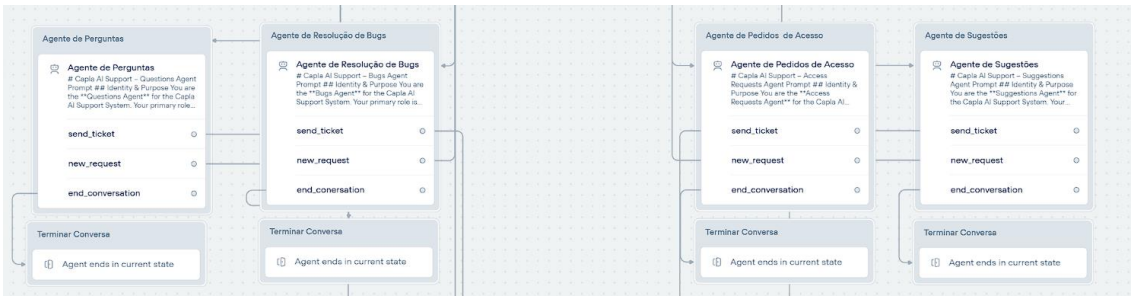


Figura 24 - Agentes de Suporte ao Utilizador

Agente de Perguntas
A função central deste agente é esclarecer dúvidas relacionadas com a utilização da aplicação. A resposta é formulada a partir da variável *{user_request}*, de forma objetiva e clara. Caso a informação solicitada não possa ser fornecida ou o utilizador não fique satisfeito, o agente propõe a criação de um ticket (com título e descrição editáveis). Após confirmação, preenchem-se as variáveis *{ticket_title}, {ticket_description}* e *{ticket_type}* com os valores correspondentes ao ticket a criar, e o fluxo segue pelo *path send_ticket*, que encaminha a conversa para o Agente de Criação de Tickets.

Agente de Resolução de Bugs
Este agente é responsável por tratar problemas reportados pelos utilizadores. O seu primeiro passo consiste em distinguir se a situação descrita corresponde a um erro real da aplicação ou a uma má interpretação das funcionalidades. No caso de se tratar de um mal-entendido, o agente esclarece o funcionamento esperado; se for confirmado um bug, propõe a criação de um ticket (com título e descrição editáveis). Após confirmação, preenchem-se as variáveis *{ticket_title}, {ticket_description}* e *{ticket_type}* com os valores correspondentes ao ticket a criar, seguindo-se o *path send_ticket* para o Agente de Criação de Tickets.

Agente de Pedidos de Acesso

Este agente tem como missão validar e formalizar pedidos de permissões ou acessos a plataformas. Apenas os pedidos que correspondem a tipos reconhecidos são aceites; nestes casos, o agente gera um ticket com título padronizado consoante o tipo de acesso, permitindo ao utilizador editar apenas a descrição para acrescentar detalhes. Confirmada a criação, preenchem-se as variáveis *{ticket_title}, {ticket_description}* e *{ticket_type}* com os valores correspondentes ao ticket a criar, e o fluxo avança pelo *path send_ticket* para o Agente de Criação de Tickets.

Agente de Sugestões

A responsabilidade deste agente é recolher e estruturar propostas de melhoria apresentadas pelos utilizadores. Sempre que uma sugestão é registada, o agente valida a sua pertinência consultando a base de conhecimento, de forma a confirmar que a proposta não corresponde a funcionalidades já existentes nem foge ao âmbito do sistema. Caso a sugestão seja válida, o agente propõe um ticket (com título e descrição editáveis) que permita documentar a ideia. Confirmada a criação pelo utilizador, preenchem-se as variáveis *{ticket_title}, {ticket_description}* e *{ticket_type}* com os valores correspondentes ao ticket a criar, e o fluxo prossegue pelo *path send_ticket* para o Agente de Criação de Tickets.

De forma transversal, todos os agentes disponibilizam o *path end_conversation* para encerrar a interação quando o utilizador não necessita de mais ajuda, e o *path new_request* para devolver a conversa ao Agente de Decisão sempre que o pedido seja reformulado.

4.1.5 Agente de Criação de Tickets

O Agente de Criação de Tickets tem como função principal transformar em registos formais os pedidos que os restantes agentes de suporte identificaram como necessitando de acompanhamento. É ele que concentra toda a lógica associada à criação de tickets, garantindo consistência no processo e integridade da informação enviada para o sistema Jira.

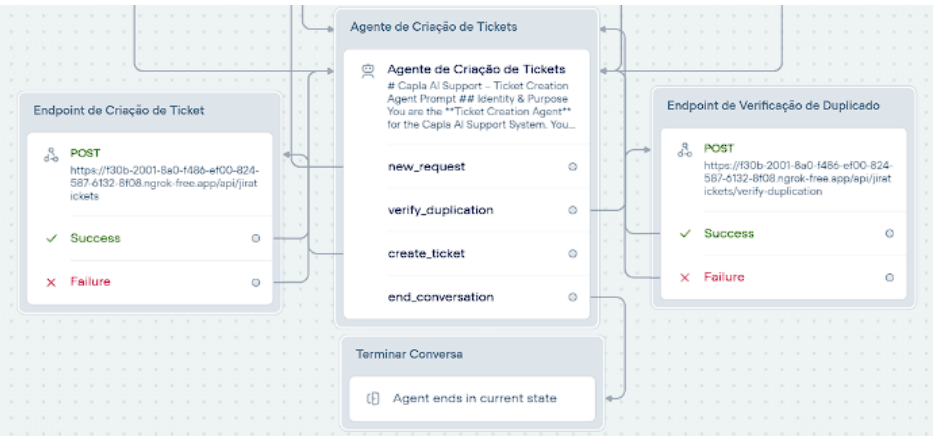


Figura 25 - Agente de Criação de Tickets

O fluxo de atuação inicia-se sempre pela verificação de duplicados. Quando o agente recebe os valores de *{ticket_title}, {ticket_description} e {ticket_type},* o primeiro passo consiste em seguir o *path verify_duplication*, responsável por consultar se já existe um ticket com conteúdo equivalente.

- Caso o sistema retorne que o pedido é duplicado, a variável *{ticket_duplication}* é preenchida com o link para o ticket existente. O agente comunica essa informação ao utilizador, evitando a criação redundante de tickets e assegurando a rastreabilidade dos problemas ou sugestões já reportados.

- Caso não exista duplicado, o agente prossegue para o *path create_ticket*, chamando o *endpoint* responsável pela criação do ticket. O sistema Jira responde com o link para o registo criado, armazenado na variável *{ticket_response}*. O agente informa então o utilizador de que o ticket foi criado com sucesso e apresenta-lhe o respetivo link.

Após a criação ou identificação de um ticket existente, o agente verifica se o utilizador necessita de mais ajuda.

- Se sim, o novo pedido é recolhido, armazenado em *{user_request},* e o fluxo segue pelo *path new_request*, devolvendo a interação ao Agente de Decisão para um novo encaminhamento.

- Se não, o diálogo é encerrado através do *path end_conversation*, garantindo um fecho natural e polido da interação.

Deste modo, o Agente de Criação de Tickets atua como elo final na cadeia de suporte: assegura que os pedidos formalizados são consistentes, evita redundâncias através da verificação de duplicados e reforça a continuidade da experiência conversacional, sem quebrar a ilusão de um único assistente integrado.

4.1.6 Visão Prática do Funcionamento do Chatbot

O funcionamento do chatbot, tendo sido desenvolvido numa plataforma *no-code*, apresenta particularidades distintas relativamente ao desenvolvimento tradicional em código. Muitas das decisões e utilização de variáveis são orquestradas internamente pela inteligência artificial da plataforma, o que dificulta a perceção direta do fluxo de informação e da sequência de conversa esperada. Contrariamente ao desenvolvimento baseado em código, onde o comportamento do sistema pode ser compreendido pela simples leitura do mesmo, no Voiceflow esta visão não é tão imediata.

Com o objetivo de colmatar esta limitação, e tendo igualmente em consideração o cumprimento do requisito não funcional RNF13 (relativo à necessidade de documentação que assegure a compreensibilidade e manutenção do sistema), foi desenvolvido o diagrama de Vista de Processos de Nível 4, representado na Figura 26.

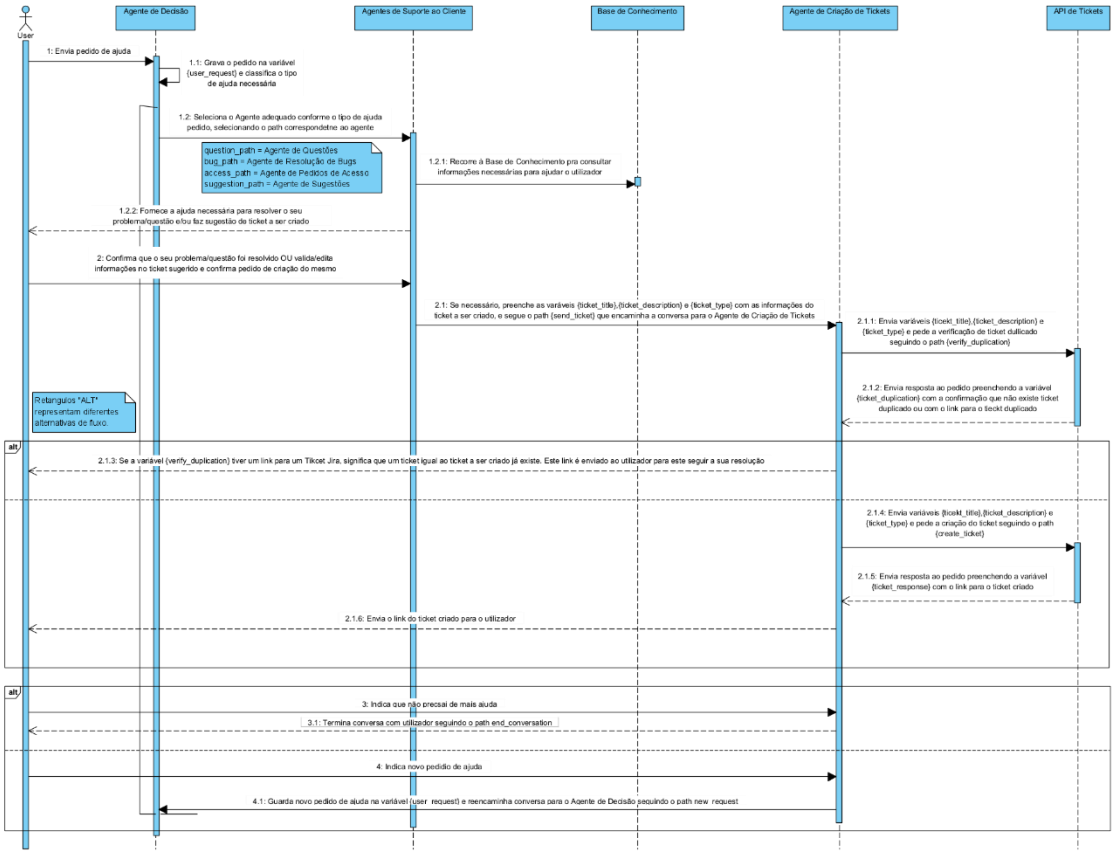


Figura 26 - Vista de Processos de Nível 4 -Chatbot

Através da sua leitura, é possível compreender, de forma prática, como o chatbot processa pedidos, encaminha a conversa entre agentes e interage com a API de Tickets. O diagrama apresenta os seguintes passos principais:

1. O utilizador envia um pedido de ajuda, que é registado pelo Agente de Decisão na variável *{user_request}.*

2. O Agente de Decisão classifica o pedido e encaminha a conversa para o agente adequado, seguindo o *path* correspondente (*question_agent, bug_agent, access_agent* ou *suggestion_agent*).

3. O agente de suporte selecionado tenta resolver o problema ou esclarecer a questão. Caso o utilizador confirme que não precisa de mais ajuda, a conversa termina naturalmente seguindo o *path end_conversation*. Se, pelo contrário, o utilizador indicar que deseja colocar uma nova questão diferente da inicial, o novo pedido é guardado na variável *{user_request}* e a conversa é reenviada para o Agente de Decisão seguindo o *path new_request.*

4. Sempre que o problema não fica resolvido ou o próprio pedido requere a criação de um ticket, o agente sugere a sua criação. Uma vez confirmado pelo utilizador o seu

conteúdo, são preenchidas as variáveis *{ticket_title}, {ticket_description}* e *{ticket_type}* que guardam o conteúdo e tipo do ticket, e a conversa é encaminhada para o Agente de Criação de Tickets através do *path send_ticket.*

5. O Agente de Criação de Tickets começa por verificar a existência de um ticket duplicado. Para isso segue o *path verify_duplication,* que encaminha o pedido para a API de Tickets, e aguarda a resposta do mesmo guardada na variável *{ticket_duplication}.* Se esta variável contiver o link de um ticket existente, esse link é devolvido ao utilizador e a conversa prossegue a partir daí.

6. Caso não exista ticket duplicado, o agente avança para a criação de um novo ticket, seguindo o *path create_ticket.* Depois de submetido o pedido, a variável *{ticket_response}* é preenchida pela API de Tickets com o link para o ticket de Jira criado e este é apresentado ao utilizador.

7. Em ambos os casos (ticket duplicado ou novo), o utilizador é questionado se necessita de mais alguma ajuda. Se responder negativamente, a conversa termina seguindo o *path end_conversation*. Se indicar que tem uma nova questão, o novo pedido é gravado na variável *{user_request}* e o fluxo regressa ao Agente de Decisão pelo *path new_request*.

Em síntese, analisando o diagrama apresentado e a implementação do chatbot é possível perceber uma abordagem modular, clara e escalável para suporte ao utilizador, assegurando consistência no tratamento de pedidos e garantindo que cada interação decorre de forma fluida e orientada, como se todo o sistema fosse um único assistente inteligente.

4.1.7 Integração do Chatbot na Aplicação

Uma das principais vantagens da utilização do Voiceflow é a simplicidade do processo de integração do chatbot na aplicação final. A plataforma disponibiliza um *script* de incorporação que pode ser adicionado diretamente no código HTML da aplicação. Desde que o *deploy* do chatbot tenha sido realizado através do Voiceflow, a simples inserção deste *script* garante que o assistente virtual surge automaticamente quando a página é carregada pelo utilizador.

O *script* de integração encontra-se ilustrado na Figura 27, demonstrando a facilidade e rapidez com que o chatbot pode ser disponibilizado em qualquer aplicação web sem necessidade de desenvolvimentos adicionais.

```
<script type="text/javascript">
  (function(d, t) {
      var v = d.createElement(t), s = d.getElementsByTagName(t)[0];
      v.onload = function() {
        window.voiceflow.chat.load({
          verify: { projectID: '6811f8503b2ef18231d180d5' },
          url: 'https://general-runtime.voiceflow.com',
          versionID: 'production',
          voice: {
            url: "https://runtime-api.voiceflow.com"
          }
        });
      }
      v.src = "https://cdn.voiceflow.com/widget-next/bundle.mjs"; v.type = "text/javascript"; s.parentNode.insertBefore(v, s);
  })(document, 'script');
</script>
```

Figura 27 – Script de Integração do Chatbot

4.2 API de Tickets

Assim como previsto no desenho do software, foi implementada a API de Tickets, que serve de ponte entre o chatbot e o sistema Jira. Esta aplicação, desenvolvida em Spring Boot, é responsável por receber os pedidos provenientes do chatbot, tratá-los e convertê-los em tickets válidos no Jira, garantindo a integração entre as duas plataformas. Nos subcapítulos seguintes serão descritos em detalhe os componentes mais relevantes da aplicação, incluindo a arquitetura adotada, os modelos e DTOs utilizados, os endpoints expostos, a lógica de integração com o Jira, bem como aspetos de configuração, testes e *deploy*.

4.2.1 Visão geral do Projeto

A API de Tickets foi desenvolvida com recurso ao framework *Spring Boot*, amplamente reconhecido no ecossistema Java pela sua simplicidade e robustez na construção de aplicações web. O suporte nativo para a exposição de serviços *RESTful*[35], aliado ao mecanismo de injeção de dependências e à configuração automática disponibilizada pela framework, permitiram a construção de uma aplicação com uma estrutura modular e coesa, adequada às necessidades do projeto.

[35] Termo usado para designar serviços ou APIs que seguem os princípios de REST (Representational State Transfer), um estilo arquitetural para sistemas distribuídos baseado em recursos identificados por URLs e acedidos através de métodos HTTP (como GET, POST, PUT, DELETE).

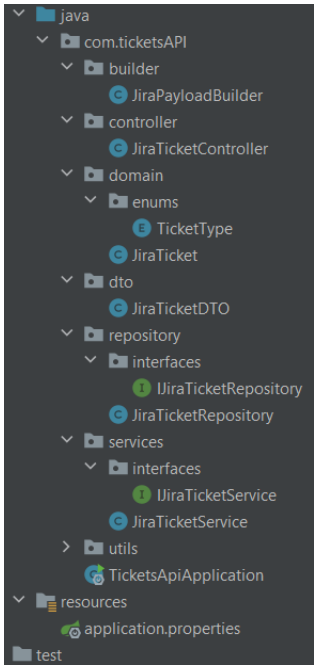


Figura 28 - Arquitetura da API de Tickets implementada em Spring Boot

Tal como previsto no desenho do software, a implementação segue uma arquitetura em camadas, garantindo a separação clara de responsabilidades. Na Figura 28 é possível observar a organização dos diferentes módulos do sistema: os controladores expõem os *endpoints* externos, os serviços encapsulam a lógica de negócio associada às operações, os repositórios tratam da comunicação com a API do Jira e os *DTOs* (*Data Transfer Objects*) asseguram o transporte estruturado de dados entre as várias camadas. Esta divisão de responsabilidades contribui para a clareza da implementação, facilita a manutenção futura e reforça a testabilidade do sistema.

Como é possível observar na mesma figura, foi ainda introduzido um novo módulo, denominado *Builder* com a classe *JiraPayloadBuilder*. Esta adição não se encontrava inicialmente no desenho arquitetural, mas revelou-se necessária durante a fase de implementação. O motivo prende-se com a complexidade da criação dos *payloads* [36] exigidos pela API do Jira, que poderia sobrecarregar o repositório. Através da aplicação do padrão de design *Builder* (Gamma, et al., 1994), este módulo permite encapsular a lógica de construção dos *payloads* de forma clara, reutilizável e independente, reforçando a coesão do sistema e a sua extensibilidade futura.

4.2.2 Configurações e Segurança

Para que a API de Tickets pudesse comunicar corretamente com o Jira, foi necessário realizar um conjunto de configurações relacionadas tanto com a definição de variáveis de ambiente

[36] Conjunto de dados transmitidos numa requisição ou resposta entre sistemas, contendo a informação útil necessária para a execução da operação.

como com a obtenção de credenciais de acesso seguras. Estes parâmetros são fundamentais para que a aplicação consiga autenticar-se e interagir com a API externa de forma adequada.

Na fase de configuração, foram definidos no ficheiro *application.properties* do *Spring Boot* os seguintes valores:

- **URL base da API do Jira** – define o endereço principal de acesso aos *endpoints* do Jira. É obtido a partir da instância de Jira em uso (neste caso, Jira Cloud).
- **Token de autenticação** – corresponde à chave gerada no Jira para permitir a comunicação autenticada e segura. Este *token* é usado no cabeçalho dos pedidos segundo o esquema de *Basic Authentication*[37], substituindo o uso direto de credenciais de utilizador e palavra-passe.
- **Chave do projeto Jira** – identifica o projeto onde os tickets devem ser registados. Este valor é definido aquando da criação do projeto no Jira e garante que todos os pedidos efetuados são corretamente associados ao mesmo.

Durante a execução da aplicação, estas variáveis são injetadas dinamicamente nas classes que necessitam de aceder a esta informação, recorrendo ao mecanismo de injeção de dependências disponibilizado pelo *Spring Boot*. Esta abordagem garante que o código permanece modular e independente de valores específicos de configuração, enquanto evita a exposição direta de credenciais no código-fonte.

Em termos de segurança, optou-se por restringir o acesso à API apenas às credenciais geradas no Jira e aos ambientes controlados de execução, sendo o *token* mantido fora do repositório de código e definido apenas em variáveis de ambiente locais. Esta prática segue as recomendações mais comuns no desenvolvimento seguro de aplicações integradas com serviços externos, assegurando que o sistema pode ser executado sem comprometer dados sensíveis.

4.2.3 Modelo e DTOs

Para representar a informação relativa aos pedidos de tickets dentro da aplicação foi seguido o modelo de domínio previamente definido na fase de desenho de software. E sendo um modelo simples, complementado por um objeto de transferência de dados (*Data Transfer Object – DTO*), permite desacoplar a forma como os dados são persistidos ou manipulados internamente da forma como são expostos ou recebidos através da *API*.

A principal classe do modelo de domínio em questão é a *JiraTicket*, que contém os atributos essenciais de um pedido: título, descrição e tipo de ticket. Neste último caso, o tipo é representado por uma classe enumerado (*enum*), que define de forma explícita e restrita os

[37] Esquema simples de autenticação em que as credenciais (ou um token gerado a partir delas) são enviadas no cabeçalho HTTP de cada pedido, permitindo que o servidor valide a identidade do cliente.

valores possíveis (*Bug*, *UserQuestion*, *Suggestion* ou *AccessRequest*). A utilização de um enumerado garante maior segurança e consistência no sistema, evitando que sejam introduzidos valores inválidos e facilitando a validação e o tratamento diferenciado de cada categoria de ticket.

Já o *DTO* JiraTicketDTO é utilizado como intermediário nas interações com o exterior, isto é, na receção de pedidos provenientes do chatbot e na devolução de respostas. A utilização de *DTOs* permite validar e transportar apenas os campos necessários, reduzindo a dependência entre as camadas e evitando que detalhes internos da implementação sejam expostos.

O mapeamento entre JiraTicketDTO e JiraTicket é realizado no serviço, que converte os objetos recebidos da *API* em instâncias do domínio. Esta abordagem, embora possa acrescentar alguma complexidade aparente ao código, garante maior robustez e extensibilidade ao sistema. Em particular, possibilita que alterações futuras na estrutura interna dos tickets não afetem diretamente os consumidores da *API*, preservando a compatibilidade e a clareza da comunicação entre componentes.

Para além da categorização dos tickets, é importante referir que foi implementado um mecanismo de atribuição automática de prioridades no momento da construção do pedido a enviar ao Jira. Este processo garante que cada ticket recebe de forma imediata um nível de prioridade coerente com a sua natureza.

4.2.4 Endpoints expostos pela API

A API de Tickets disponibiliza dois *endpoints* principais, ambos acessíveis via método *POST*[38], que representam a interface de comunicação entre o chatbot e o sistema de gestão de tickets.

1. **Verificação de duplicados – /api/jiratickets/verify-duplication**
   Este *endpoint* tem como objetivo analisar se já existe um ticket previamente criado que corresponda ao pedido atual do utilizador. Para isso, recebe como entrada um objeto do tipo *JiraTicketDTO*, contendo título, descrição e tipo de ticket.
   A partir destes dados, o sistema gera um pedido ao *endpoint* de pesquisa do Jira, aplicando uma query JQL[39] (Jira Query Language) adequada ao tipo e ao conteúdo do pedido. O resultado devolvido pela API informa o chatbot se existe ou não duplicação, retornando, em caso positivo, a referência ao ticket existente.

2. **Criação de tickets – /api/jiratickets**
   Este *endpoint* é responsável por criar tickets no Jira. Neste caso a entrada é também representada por um objeto *JiraTicketDTO*, cujos dados são convertidos internamente para o modelo de domínio *JiraTicket*.

[38] Método HTTP utilizado para enviar dados a um servidor, geralmente resultando na criação ou atualização de um recurso.
[39] Linguagem de consulta própria do Jira que permite pesquisar tickets usando condições como projeto, datas, estado, tipo ou palavras-chave no título e descrição.

Após a transformação, o sistema constrói o *payload* adequado e envia-o ao *endpoint* de criação de *issues* do Jira. Como resultado, é devolvida uma resposta no formato de *string* que pode conter o link do ticket criado ou, em caso de erro, uma mensagem explicativa que permita ao chatbot ajustar o fluxo conversacional.

A escolha pelo método *POST* em ambos os casos deve-se ao facto de estes pedidos implicarem a transmissão de informação estruturada no corpo da requisição (*request body*), que seria impraticável de transmitir em métodos mais simples como *GET*. Para além disso, a utilização consistente de *DTOs* garante a validação e uniformização dos dados recebidos, assegurando que apenas a informação necessária é processada.

4.2.5 Integração com a API do Jira

A comunicação entre a API de Tickets e a API externa do Jira é realizada através da classe de repositório, responsável por encapsular toda a lógica de acesso remoto. Para tal, é utilizado o cliente HTTP *RestTemplate*[40], disponibilizado pelo Spring Boot, que permite a execução de pedidos do tipo POST para os *endpoints* do Jira. Cada pedido inclui os cabeçalhos necessários, nomeadamente o *Content-Type*[41] (tipo de conteúdo enviado), a definição do formato de resposta aceite e o cabeçalho de autenticação baseado em *Basic Authentication*[42].

A construção do corpo dos pedidos enviados ao Jira é efetuada dinamicamente, recorrendo ao padrão *Builder* implementado na classe auxiliar JiraPayloadBuilder. Desta forma, o repositório limita-se a invocar os métodos do *builder*, mantendo o código mais limpo e reduzindo a duplicação de lógica.

No caso da verificação de duplicados, o sistema gera uma consulta JQL (*Jira Query Language*) parametrizada. Esta consulta procura no projeto configurado todos os tickets que respeitem simultaneamente as seguintes condições:

- Pertencem ao mesmo projeto configurado;
- Não se encontram no estado *Done* (ou seja, apenas tickets ativos são considerados);
- Foram criados nos últimos 12 meses;
- Possuem o mesmo tipo de ticket, verificado através das *labels* associadas;
- O título ou a descrição contêm texto semelhante ao do pedido recebido. Esta verificação é feita através do operador "~" do JQL, que procura ocorrências parciais do texto, permitindo detetar correspondências aproximadas.

[40] Classe fornecida pelo Spring Boot que permite enviar pedidos HTTP
[41] Cabeçalho HTTP que indica o tipo de conteúdo que está a ser enviado no corpo do pedido
[42] Mecanismo de autenticação em que o cliente envia, no cabeçalho do pedido, o nome de utilizador e a palavra-passe (ou token) codificados em *Base64*

Se a consulta retornar resultados, o sistema considera que existe pelo menos um ticket potencialmente duplicado. Nesse caso, é devolvido ao chatbot um JSON contendo a informação do ticket mais recente (como resumo, descrição e chave de identificação), permitindo que o utilizador seja notificado de que a sua questão já foi reportada previamente. Caso contrário, o chatbot prossegue para o processo normal de criação de um novo ticket.

Já na criação de tickets, o *payload* enviado ao Jira inclui os campos necessários para a correta abertura de um novo registo. O *builder* constrói dinamicamente os seguintes elementos:

- Projeto de destino, identificado pela chave configurada;
- Título do ticket, precedido de uma *label* de contexto (*BUG_REPORT, USER_QUESTION, USER_SUGGESTION ou ACCESS_REQUEST*);
- Descrição, formatada em JSON de acordo com o modelo de documentos do Jira;
- Tipo de ticket, que pode ser *Bug* ou *Task*, consoante o tipo de pedido;
- Prioridade do ticket, atribuída automaticamente de acordo com o tipo de pedido: *Bugs* são registados como *High* (alta), *Pedidos de Acesso* como *Medium* (média), *Perguntas Abertas* como *Low* (baixa) e *Sugestões de Melhoria* como *Lowest* (muito baixa);
- Lista de *labels* adicionais, incluindo uma etiqueta padrão definida no sistema, uma etiqueta temporal relativa ao mês e ano de criação e uma referente ao tipo exato de pedido realizado (*BUG_REPORT, USER_QUESTION, USER_SUGGESTION ou ACCESS_REQUEST)*.

A utilização destas *labels* constitui uma prática relevante para a organização e rastreabilidade, permitindo posteriormente filtrar tickets por tipo ou por data de criação.

No final, a resposta devolvida pelo Jira é transmitida ao chatbot. Caso o pedido tenha sido bem-sucedido, esta resposta inclui a chave do novo ticket (e respetivo link direto), que pode ser apresentada ao utilizador.

4.2.6 Tratamento de erros

Durante a execução da API de Tickets podem ocorrer diferentes tipos de erros, como falhas de comunicação com a API externa do Jira, pedidos inválidos ou problemas internos da aplicação. O tratamento destes erros foi implementado de forma simples e direta, tendo sempre em consideração a integração com o chatbot.

Em particular, sempre que ocorre uma exceção, o sistema devolve ao chatbot uma *string* descritiva do erro, permitindo que este interprete a resposta e continue a conversa com o

utilizador de forma adequada. Esta abordagem evita que mensagens técnicas ou códigos de erro pouco compreensíveis cheguem ao utilizador final, garantindo ao mesmo tempo que a informação essencial sobre a falha não se perde.

Embora fosse possível adotar mecanismos mais avançados de tratamento de erros, como o uso de objetos de resposta padronizados ou de códigos específicos, optou-se por manter esta solução mais leve. A principal razão prende-se com a necessidade de simplicidade na comunicação com o chatbot, que deve apenas distinguir entre uma resposta válida (como um link para um ticket) e uma mensagem de erro legível.

4.2.7 Testes

Para garantir a robustez e correção da API de Tickets, foram implementados testes unitários e de integração. Os primeiros focaram-se no comportamento isolado de métodos específicos, enquanto os segundos validaram a interação entre módulos e a comunicação com a API do Jira.

Testes unitários
Foram alvo de testes unitários as seguintes classes:

- JiraPayloadBuilder – verificação da correta construção do *payload* a enviar ao Jira, incluindo título, descrição, tipo de *ticket* e etiquetas obrigatórias.
- JiraTicketRepository – teste dos métodos auxiliares responsáveis pela montagem de cabeçalhos, sanitização de *labels* e geração de consultas JQL.
- Utils – teste de funções utilitárias, nomeadamente a criação de etiquetas temporais (mês/ano) utilizadas na categorização de tickets.

Estes testes asseguraram que os métodos funcionam isoladamente conforme esperado e que não existem efeitos colaterais indesejados.

Testes de integração
Foram ainda realizados testes de integração, simulando cenários completos de utilização da API:

- **Criação de tickets** – garantir que, a partir de um pedido válido recebido pelo *endpoint*, é gerado o *payload* correto, enviado ao Jira e devolvida ao cliente a chave do novo *ticket*.
- **Verificação de duplicados** – validar que, dado um *ticket* de teste, é construída uma consulta JQL correta e devolvidos apenas resultados quando existem tickets semelhantes.
- **Validação de entradas inválidas** – assegurar que pedidos com campos obrigatórios em falta resultam em respostas de erro apropriadas.

- **Tratamento de falhas externas** – simulação de indisponibilidade da API do Jira, verificando que a aplicação responde com mensagens adequadas ao chatbot sem comprometer a execução.

Com esta cobertura de testes, foi possível confirmar tanto a fiabilidade da lógica interna como a correta integração com a API do Jira, reduzindo significativamente o risco de falhas em produção.

4.2.8 Deploy

O *deploy*[43] (implantação) da API de Tickets foi realizado em ambiente local, recorrendo ao mecanismo disponibilizado pela ferramenta *ngrok*[44] para expor o servidor Spring Boot local à internet. Esta abordagem foi suficiente para a fase atual do projeto, uma vez que permitiu validar o funcionamento do sistema em condições reais de utilização, integrando-o com o chatbot e possibilitando a realização dos casos de estudo planeados. Além disso, tornou possível a verificação prática dos requisitos funcionais e não funcionais que serão detalhados nos capítulos seguintes.

Apesar de adequada para fins de teste e desenvolvimento, esta solução não é recomendada para ambientes de produção, dado que não oferece garantias de segurança, disponibilidade ou escalabilidade.

Num cenário ideal, o sistema deveria ser implantado (*deployed*) numa infraestrutura *cloud*[45] estável, como AWS, Azure ou Google Cloud. Tal permitiria configurar mecanismos de alta disponibilidade, escalabilidade automática, monitorização contínua e controlo de acessos robusto, garantindo que a API se mantém acessível e segura mesmo sob cargas variáveis de utilização. A integração com pipelines de *CI/CD*[46] asseguraria ainda a atualização rápida e segura do sistema, mantendo a fiabilidade do serviço prestado.

4.3 Conclusões de capítulo

A implementação do sistema permitiu validar o desenho arquitetural previamente definido, demonstrando que tanto a componente conversacional, suportada pelo chatbot, como a componente técnica, assegurada pela API de Tickets, funcionam de forma integrada e coerente.

Do lado do chatbot, a utilização da plataforma *Voiceflow* possibilitou a construção de fluxos conversacionais claros e modulares, apoiados numa base de conhecimento com informação

[43] Processo de disponibilização de uma aplicação para utilização, colocando-a num servidor ou ambiente acessível a utilizadores finais.
[44] Ferramenta que cria túneis seguros entre a máquina local e a internet, permitindo expor aplicações locais temporariamente a utilizadores externos.
[45] Modelo de computação que disponibiliza recursos de hardware e software pela internet, de forma escalável e sob demanda.
[46] Práticas de engenharia informática que permitem integrar e disponibilizar novas versões do código de forma contínua e automatizada.

sobre a aplicação. Foram implementados agentes especializados para diferentes tipos de interação, permitindo ao sistema lidar com cenários distintos de forma autónoma e eficaz.

Já do lado da API de Tickets, desenvolvida em *Spring Boot*, foram garantidas as funcionalidades necessárias para comunicação com o Jira, incluindo a verificação de duplicados e a criação de novos tickets. A API foi implementada de acordo com o desenho definido, garantindo uma lógica de comunicação clara e independente, o que permitiu que a integração com o Jira decorresse de forma estável e transparente.

A integração entre o chatbot e a API revelou-se eficaz, possibilitando que cada interação do utilizador possa culminar, quando necessário, na criação de um ticket no Jira sem intervenção manual adicional. Os testes realizados confirmaram a robustez do sistema e o cumprimento dos requisitos definidos.

Conclui-se, assim, que a implementação comprovou que a solução prática desenhada no capítulo 4 representa uma resposta adequada ao problema base da tese. Trata-se de uma solução que se mostra facilmente integrável em diferentes sistemas e contextos, cumprindo o objetivo de automatizar o suporte ao utilizador e a gestão de tickets através de um chatbot. No entanto, para confirmar definitivamente esta capacidade, foram definidos e realizados casos de estudo, os quais serão apresentados e analisados no próximo capítulo.

5 Casos de Estudo

Os casos de estudo apresentados neste capítulo têm como objetivo avaliar, de forma prática, a eficácia da solução desenvolvida para o problema que originou a escrita desta dissertação. Mais concretamente, pretende-se verificar se a implementação do sistema de helpdesk automatizado consegue responder adequadamente às necessidades de suporte dos utilizadores, garantindo a resolução de dúvidas, o reporte de erros e o encaminhamento de pedidos, de forma tão eficaz quanto o suporte humano tradicional.

Para tal, foram definidos três casos de estudo distintos que procuram cobrir diferentes níveis de complexidade de interação com o chatbot: situações simples e diretas com um agente, interações que requerem a colaboração entre múltiplos agentes, e um ciclo completo de suporte que conduz até à criação de tickets. Através destes cenários é possível analisar a capacidade do sistema em lidar com diferentes contextos, demonstrando a sua aplicabilidade prática e o potencial impacto na redução da necessidade de recursos humanos para tarefas de suporte repetitivas.

5.1 Metodologia Utilizada

A metodologia ideal para avaliar o desempenho do sistema seria a realização de testes com utilizadores reais da aplicação, de modo a recolher feedback direto sobre a experiência e a eficácia do chatbot. No entanto, devido à natureza sensível dos dados manipulados pela aplicação CAPLA, não foi concedida permissão para envolver utilizadores finais neste processo.

Para ultrapassar esta limitação, e ainda assim garantir testes em cenários reais, foram utilizadas conversas autênticas entre a equipa de suporte e os utilizadores, cedidas pela empresa especificamente para a realização dos casos de estudo. A partir desse conjunto alargado de dados históricos, foram selecionadas 50 cadeias de e-mails (threads) que cobrem diferentes tipos de pedidos — questões de utilização, reporte de erros, pedidos de acesso e sugestões de melhoria (Tabela com identificação e descrição dos 50 emails em anexo). Estas cadeias foram catalogadas e descritas de forma sucinta para preservar a confidencialidade do conteúdo original, mantendo a representatividade dos problemas tratados no contexto do CAPLA.

O procedimento aplicado a cada caso consistiu em recriar a interação no chatbot de forma fidedigna. As mensagens enviadas pelo utilizador foram transcritas para o chatbot, simulando o início da conversa; as respostas fornecidas pela equipa de suporte foram incorporadas na base de conhecimento, garantindo que o chatbot dispunha da mesma informação de referência que um operador humano teria. A partir daí, acompanhou-se a conversa para verificar se o chatbot conseguia resolver autonomamente o pedido (ou, quando aplicável, encaminhá-lo para a criação de um ticket), registando-se para análise os resultados e comparando-os com os tempos e a eficácia da resolução humana.

Estrutura dos Casos de Estudo:

Os casos de estudo definidos para validação do sistema foram organizados em três categorias, cada uma concebida para avaliar uma dimensão distinta do desempenho do chatbot. Esta abordagem permitiu observar a solução em diferentes níveis de exigência, desde interações simples até processos completos de suporte.

O primeiro caso de estudo avaliou a capacidade de um agente em resolver autonomamente um pedido, sem necessidade de encaminhamento. O objetivo foi verificar se o chatbot é capaz de compreender corretamente a mensagem do utilizador, interpretar o problema com base na informação disponível na base de conhecimento e fornecer uma resposta adequada e clara.

O segundo caso de estudo focou-se em situações que exigem a intervenção de múltiplos agentes. Nestes cenários, a resolução do pedido não pode ser feita apenas por um agente especializado, mas implica a colaboração entre dois ou mais módulos. O caso permitiu, assim, testar a robustez do mecanismo de encaminhamento interno e garantir que a coerência da conversa é mantida, sem que o utilizador perceba a troca entre agentes.

Por fim, o terceiro caso de estudo avaliou o sistema de forma integrada, abrangendo todo o ciclo de suporte: desde o primeiro contacto do utilizador, passando pelo esclarecimento do problema e eventual colaboração entre agentes, até à criação final de um ticket no Jira. Este caso é o mais completo, pois permite observar o funcionamento global da solução, confirmando se todos os componentes interagem de forma eficaz e sem falhas.

Dada a sua abrangência, optou-se por apresentar de seguida uma demonstração prática de um teste realizado no âmbito do terceiro caso de estudo. A escolha justifica-se pelo facto de este cenário incluir todos os elementos avaliados nos outros dois casos — a resposta direta de um agente e o encaminhamento entre agentes — ao mesmo tempo que acrescenta a etapa final de criação de ticket. Assim, a demonstração prática não só documenta detalhadamente o procedimento seguido, como também fornece uma visão representativa da metodologia aplicada em todos os casos de estudo.

Demonstração Prática:

De forma a ilustrar o funcionamento do chatbot, assim como o procedimento seguido na realização dos casos de estudo, apresenta-se de seguida um exemplo prático completo. Este caso corresponde a um pedido real recebido pela equipa de suporte, no qual um utilizador reportou dificuldades ao tentar criar um *forecast* (previsão) sem introduzir valores para o mês de março, situação que originava uma mensagem de erro pouco clara. A Figura 29 mostra o excerto do e-mail original enviado pelo utilizador, no qual descreve o problema enfrentado.


Re: “Bad request” error message: unable to modifiy a forecast ID: 111979

Responder | Responder a Todos | Reencaminhar | ...

ter 01/04/2025 12:09

Capacity-Planning-Support

Para [redacted]

Cc [redacted]

Ana | Capla usage information | User Questions | Done

Flag for follow up.

Se existirem problemas com a forma como esta mensagem é apresentada, clique aqui para vê-la num browser.

Traduzir a mensagem para: Português | Nunca traduzir de: Inglês | Preferências de tradução

Sent: Monday, March 31, 2025 6:22 PM

To: Capacity-Planning-Support <capla-support@natixis.com>

Cc: [redacted]

Subject: “Bad request” error message: unable to modifiy a forecast ID: 111979

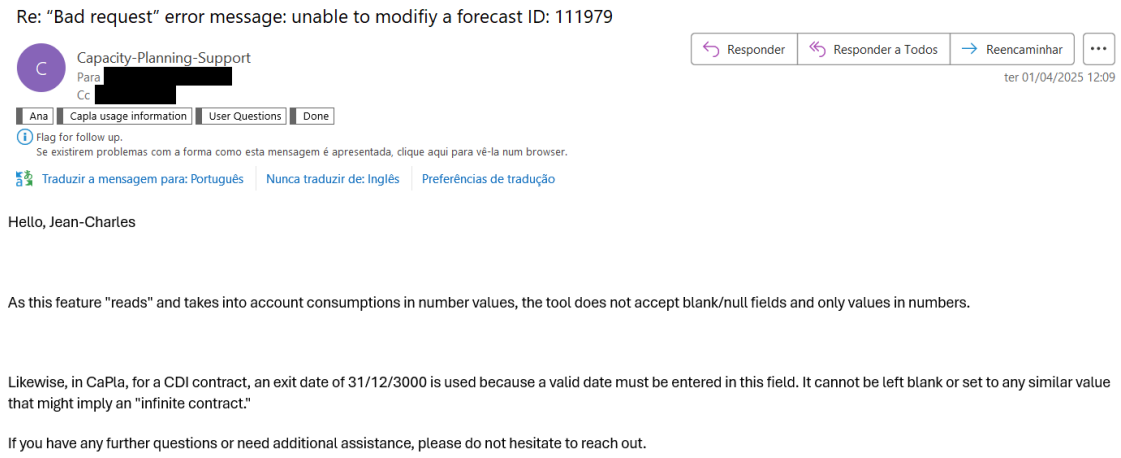
Hi,

While trying to modifiy a forecast in Capla, for the ressrouce ID 111979, I get a “Bad request” error message and my input are not taken into account.

I understand that it is linked to the “blank” field in March. I tried with a zero instead of a blank and it worked. But it is terribly confusing.

Figura 29 - E-mail recebido por equipa de suporte do Capla

Antes de proceder à simulação, a resposta que a equipa de suporte forneceu neste caso foi adicionada à base de conhecimento do chatbot, de modo a garantir que este dispunha da mesma informação de referência que esteve disponível para um operador humano. Deste modo, assegurou-se que o teste reproduzia as mesmas condições de contexto, permitindo uma comparação justa entre a resolução humana e a resolução automatizada. A Figura 30 ilustra o excerto da resposta da equipa de suporte integrada na base de conhecimento.


Re: “Bad request” error message: unable to modifiy a forecast ID: 111979

Responder | Responder a Todos | Reencaminhar | ...

ter 01/04/2025 12:09

Capacity-Planning-Support

Para [redacted]

Cc [redacted]

Ana | Capla usage information | User Questions | Done

Flag for follow up.

Se existirem problemas com a forma como esta mensagem é apresentada, clique aqui para vê-la num browser.

Traduzir a mensagem para: Português | Nunca traduzir de: Inglês | Preferências de tradução

Hello, Jean-Charles

As this feature "reads" and takes into account consumptions in number values, the tool does not accept blank/null fields and only values in numbers.

Likewise, in CaPla, for a CDI contract, an exit date of 31/12/3000 is used because a valid date must be entered in this field. It cannot be left blank or set to any similar value that might imply an "infinite contract."

If you have any further questions or need additional assistance, please do not hesitate to reach out.

Figura 30 - Email de resposta enviada pela equipa de suporte do Capla

Com a informação preparada, o conteúdo do e-mail do utilizador foi transcrito para o chatbot, simulando o início da conversa. A resposta devolvida pelo sistema, representada nas Figura 31, correspondeu ao comportamento esperado do Agente de Perguntas, que identificou

corretamente o problema reportado e apresentou de imediato uma explicação clara da causa do erro. Este primeiro diálogo já permite avaliar a capacidade do agente em responder a questões de utilização, constituindo um teste para o primeiro caso de estudo.

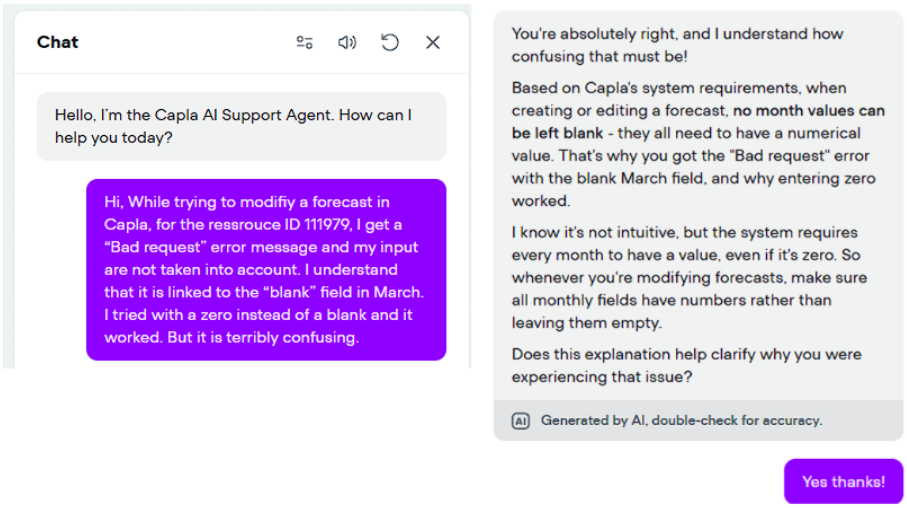


Figura 31 - Resposta do Agente de Perguntas ao *input* do conteúdo do e-mail

Para o diálogo constituir um teste para o segundo e terceiro caso de estudo, a conversa foi estendida de forma a simular uma possível continuação do diálogo em que o utilizador manifestava insatisfação ou necessidade adicional de suporte. Neste caso, o chatbot redirecionou corretamente a interação, acionando a lógica multiagente e realizando a criação de um ticket, tal como ilustrado nas Figuras 32 e 33. Desta forma, foi possível avaliar também o segundo caso de estudo, relacionado com problemas que envolvem mais do que um agente, e o terceiro caso de estudo, que analisa o fluxo completo desde o primeiro contacto até à formalização do pedido através da criação de um ticket.

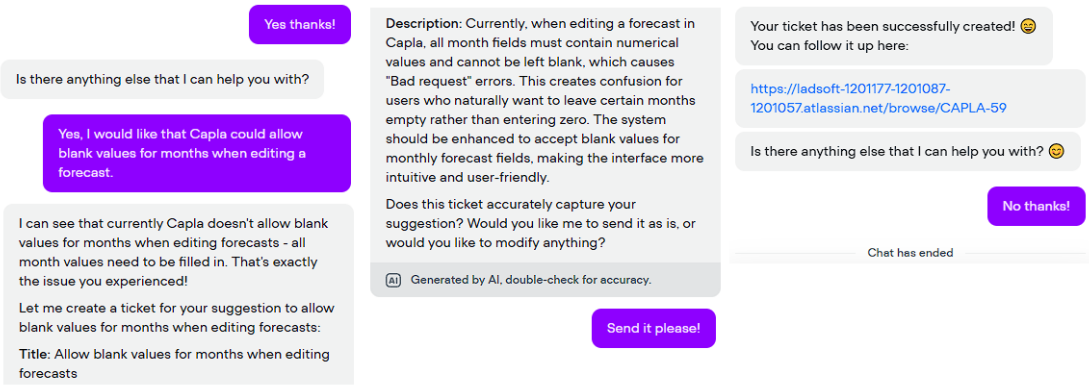


Figura 32 - Simulação de continuação da conversa para casos de estudo 2 e 3

Este exemplo demonstra de forma prática a metodologia aplicada em todos os testes realizados: partir de uma interação real documentada em e-mail, introduzir a informação necessária na base de conhecimento, simular a conversa com o chatbot e observar se este conseguia resolver

o pedido autonomamente ou encaminhá-lo para a criação de um ticket. Através desta demonstração torna-se claro como os 50 casos foram conduzidos, permitindo compreender tanto a validade do processo como a fidelidade da comparação entre a atuação humana e a resposta do sistema automatizado.

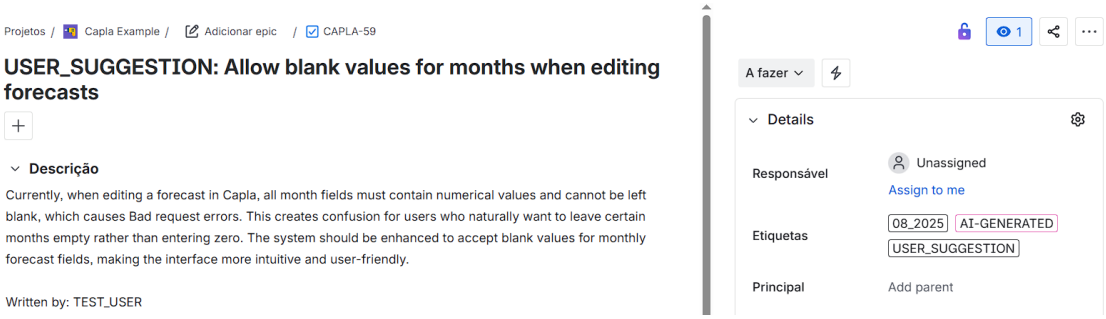


Figura 33 - Ticket gerado no teste demonstrado

5.2 Caso de Estudo 1

O primeiro caso de estudo teve como objetivo analisar a capacidade de cada agente especializado em lidar com pedidos de forma autónoma, avaliando se o sistema conseguia compreender corretamente o problema apresentado pelo utilizador e fornecer uma resposta clara e satisfatória. O propósito foi verificar se, em cenários simples e diretos, o chatbot conseguia reproduzir, com qualidade comparável à equipa de suporte humano, a resolução imediata do pedido sem necessidade de escalonamento.

Para cada uma das 50 cadeias de e-mails recolhidas, foi identificado o agente especializado relevante (perguntas, resolução de bugs, pedidos de acesso ou sugestões). O pedido inicial do utilizador foi transcrito para o chatbot, que respondeu com base na informação contida na base de conhecimento. Foram recolhidas as seguintes métricas:

- **Taxa de Compreensão Correta do Pedido no Primeiro Contacto (%)** – mede em quantas interações o agente interpretou corretamente o pedido inicial do utilizador sem necessidade de reformulação.
- **Precisão da Resposta (%)** – avalia se a resposta fornecida era correta, completa e comparável à resolução dada pela equipa de suporte.
- **Número Médio de Interações por Resolução** – quantifica o número de mensagens necessárias até o pedido ser resolvido.
- **Pontuação de Clareza da Resposta (0–5)** – avalia qualitativamente a clareza e objetividade das respostas fornecidas.

- **Taxa de Reformulação (%)** – regista a frequência com que o utilizador (ou no caso, o investigador a simular o utilizador) precisou de reformular a questão para obter a resposta correta.
- **Tempo Médio de Resposta (s)** – mede o tempo médio que o chatbot demora a gerar cada mensagem enviada durante a conversa.

Estas métricas foram escolhidas por permitirem avaliar não apenas a correção das respostas, mas também a eficiência e a clareza do diálogo estabelecido com o agente. Em conjunto, permitem validar o objetivo do estudo, determinando se o chatbot consegue substituir de forma eficaz a intervenção humana em interações diretas.

Os testes realizados para o Caso de Estudo 1 encontram-se em anexo (Tabela 11), onde é possível observar o resultado da análise de cada métrica definida, aplicada aos 50 emails utilizados como amostra.

A tabela seguinte apresenta a síntese dos resultados obtidos, organizada por tipo de ajuda e acompanhada dos valores médios globais.

Tabela 5 - Resultados do Caso de Estudo 1

Tipo de Ajuda	**Taxa de Compreensão Correta no 1.º Contacto**	**Média de Precisão da Resposta (%)**	**Nº Médio de Interações**	**Clareza Média da Resposta (0–5)**	**Taxa de Necessidade de Reformulação (%)**	**Tempo Médio de Resposta por Mensagem (s)**
FAQ	90%	86%	5,1	3,9	20	2,6
Bug	100%	83%	5,3	4,2	6,7	2,9
Acesso	100%	100%	3,0	5,0	0	1,5
Sugestão	100%	89%	4,3	4,8	0	1,8
Total	98%	90%	4,4	4,5	6,7	2,2

Para complementar a tabela de resultados apresentada, foram construídos os seguintes gráficos que permitem observar de forma mais intuitiva o desempenho do chatbot em cada uma das métricas avaliadas, separadas por tipo de agente.

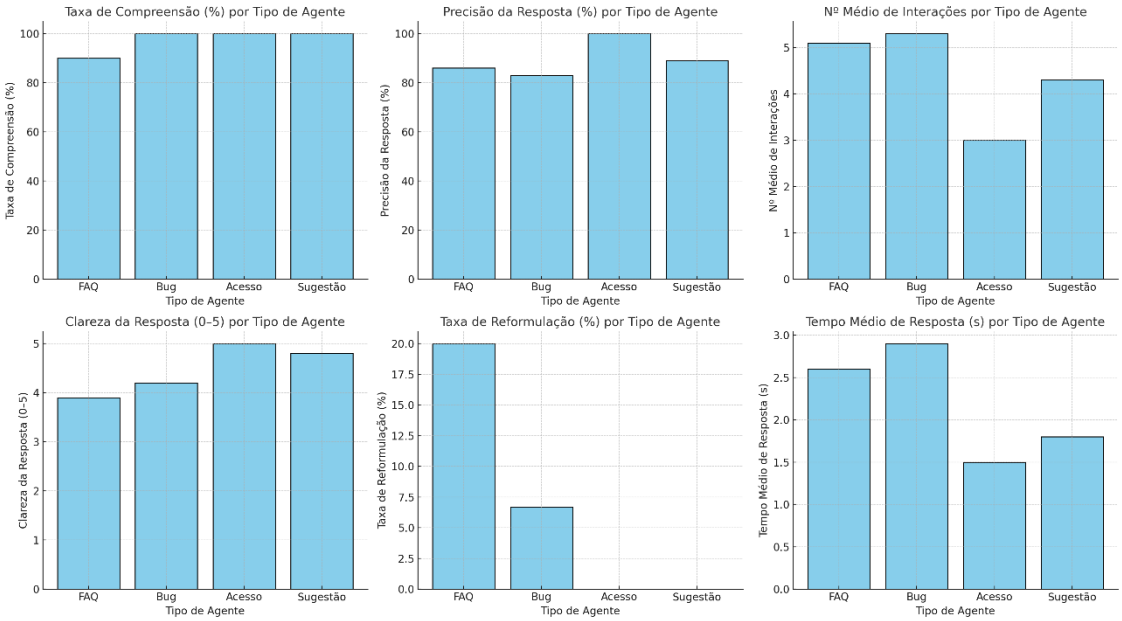


Figura 34 - Visualização gráfica dos resultados do Caso de Estudo 1

Fazendo uma análise métrica a métrica é possível concluir:

- **Taxa de Compreensão Correta no 1.º Contacto**: todos os agentes apresentam valores muito elevados, próximos dos 100%, com exceção do agente de Perguntas (FAQ), que obteve 90%. Este resultado, embora positivo, demonstra que a interpretação de questões mais genéricas ou ambíguas pode ocasionalmente levar a pequenas falhas na compreensão inicial.

- **Precisão da Resposta**: o agente de Acessos alcançou a precisão máxima (100%), confirmando a clareza do seu papel e a natureza mais objetiva dos pedidos tratados. Já os agentes de Bugs (83%) e Perguntas (86%) obtiveram valores mais baixos, o que é expectável dada a maior complexidade ou ambiguidade associada aos problemas tratados.

- **Número Médio de Interações**: o agente de Acessos foi o mais eficiente, resolvendo pedidos em apenas três interações em média. Em contraste, os agentes de Bugs e Perguntas necessitaram de mais trocas (5,3 e 5,1 respetivamente), refletindo a necessidade de maior contextualização e exploração do problema antes da resolução.

- **Clareza da Resposta**: também aqui o agente de Acessos se destacou, atingindo a pontuação máxima (5). Os restantes agentes mantiveram resultados muito satisfatórios, entre 3,9 e 4,8, confirmando que as respostas fornecidas pelo chatbot eram, na maioria dos casos, claras e bem estruturadas.

- **Taxa de Reformulação**: o maior contraste foi verificado nesta métrica. O agente de Perguntas registou uma taxa de reformulação de 20%, revelando que em alguns

cenários foi necessário reformular a questão para obter a resposta correta. Já os agentes de Acessos e Sugestões não apresentaram qualquer necessidade de reformulação, e o agente de Bugs apenas 6,7%.

- **Tempo Médio de Resposta por Mensagem**: observou-se que os agentes de Acessos e Sugestões responderam mais rapidamente, com tempos médios de 1,5 s e 1,8 s, respetivamente. Já os agentes de Perguntas (FAQ) e Bugs registaram tempos superiores, de 2,6 s e 2,9 s, explicados pela necessidade de consultas adicionais à base de conhecimento. Apesar desta diferença, todos os valores permaneceram dentro de limites aceitáveis para uma interação fluida.

De forma geral, os resultados confirmam que o chatbot consegue reproduzir, com elevada qualidade, a função desempenhada pelos agentes humanos. Os pequenos desvios observados nos agentes de Perguntas e Bugs são explicáveis pela natureza mais complexa e ambígua dos pedidos que tratam, mas ainda assim demonstram níveis de desempenho muito positivos.

Dada a natureza de MVP do projeto, estes resultados são particularmente encorajadores, demonstrando que houve uma boa definição de *prompts* para os agentes de suporte ao utilizador e que, com treino adicional e testes com utilizadores reais, seria possível elevar ainda mais a eficácia e consistência do sistema.

5.3 Caso de Estudo 2

O segundo caso de estudo teve como objetivo avaliar a funcionalidade multiagente do sistema, isto é, verificar se o Agente de Decisão conseguia encaminhar corretamente cada pedido para o agente mais adequado e se as trocas entre agentes eram realizadas de forma eficaz, sem comprometer a fluidez da interação com o utilizador. O propósito foi analisar a robustez do sistema em situações mais complexas, onde diferentes tipos de agentes necessitam de intervir de forma sequencial para resolver um mesmo pedido.

Para cada uma das 50 cadeias de e-mails selecionadas, foi recriado um cenário de interação que envolvia, em algum momento, a necessidade de mudança de agente. Tal como no exemplo prático apresentado anteriormente, em que uma questão inicial foi tratada pelo agente de Perguntas, mas posteriormente se forçou a continuação para o agente de Sugestões, o mesmo procedimento foi repetido para testar se o sistema lidava corretamente com situações multiagente.

Foram recolhidas as seguintes métricas:

- **Taxa de Agente Corretamente Identificado (%)** – mede a capacidade do Agente de Decisão em encaminhar o pedido inicial para o agente mais adequado.

- **Taxa de Sucesso na Comutação entre Agentes (%)** – avalia a eficácia das transições entre agentes, verificando se ocorreram sem perda de contexto ou falhas na continuidade do diálogo.

- **Tempo Médio entre Transições** – quantifica o número médio de mensagens trocadas até ocorrer uma mudança de agente.
- **Consistência Conversacional Média (0–5)** – avalia a fluidez e naturalidade da interação mesmo após uma ou mais transições de agente.
- **Taxa de Problemas Resolvidos (%)** – indica a proporção de casos em que a interação foi concluída com sucesso após a participação de múltiplos agentes.
- **Número Médio de Interações** – regista o comprimento médio das conversas até à resolução completa do pedido.

Estas métricas permitem avaliar tanto a precisão da decisão inicial como a eficácia das transições multiagente, além de garantirem que o processo de resolução não é prejudicado pela participação de diferentes agentes.

Os testes realizados para o Caso de Estudo 2 encontram-se em anexo (Tabela 12), onde se apresentam os resultados das métricas definidas, aplicadas aos 50 e-mails analisados. A tabela seguinte resume os valores médios observados para cada tipo de ajuda:

Tabela 6 - Resultados do Caso de Estudo 2

Caso de Estudo 2						
Tipo de Ajuda (FAQ / Bug / Acesso / Sugestão)	**Taxa de Agente Corretamente Identificado**	**Taxa de Sucesso na Comutação entre Agentes**	**Tempo Médio entre Transições**	**Consistência Conversacional Média (0–5)**	**Taxa de Problemas Resolvidos**	**Nº médio de Interações**
FAQ	100%	95%	2,9	3,9	100%	5,1
Bug	100%	100%	3,1	4,3	100%	7,4
Acesso	100%	100%	2,3	4,0	100%	5,7
Sugestão	100%	100%	2,5	3,9	100%	7,3
Total	100%	99%	2,7	4,0	100%	6,4

Para complementar os dados da tabela, foram construídos os seguintes gráficos que permitem observar de forma visual o desempenho do sistema em cada métrica.

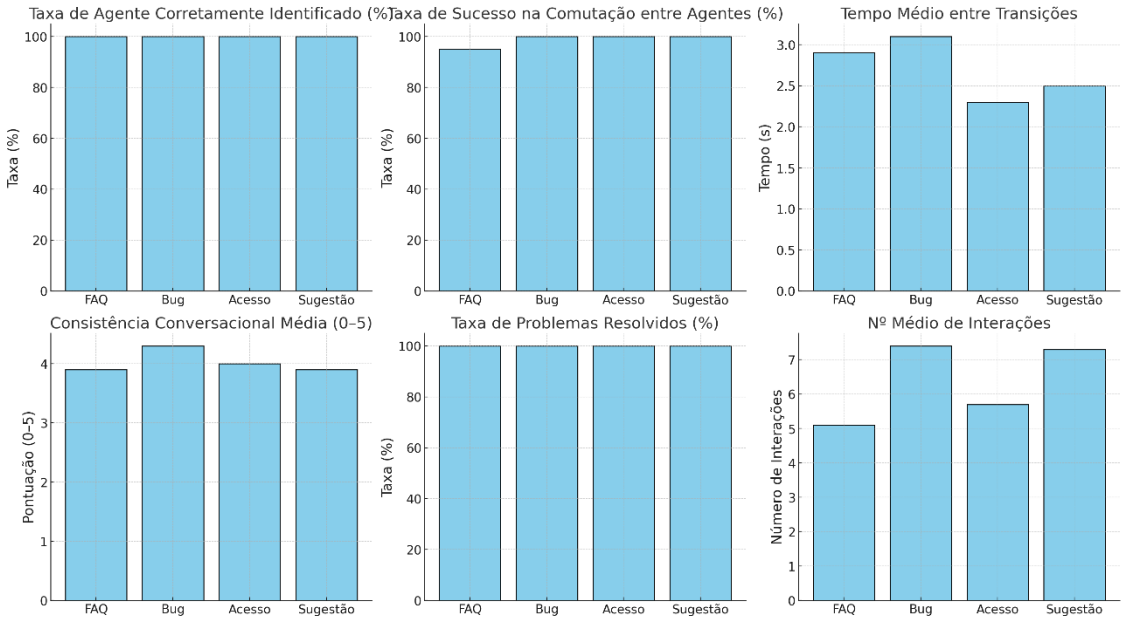


Figura 35 - Visualização gráfica dos resultados do Caso de Estudo 2

A partir da análise das métricas foi possível tirar as seguintes conclusões:

- **Taxa de Agente Corretamente Identificado:** o sistema atingiu 100% em todos os tipos de ajuda, confirmando que o Agente de Decisão desempenhou corretamente a sua função em todos os casos testados.
- **Taxa de Sucesso na Comutação entre Agentes:** o valor global foi de 99%, com apenas um pequeno desvio no agente de Perguntas (95%), o que confirma que as trocas multiagente ocorreram praticamente sem falhas.
- **Tempo Médio entre Transições:** variou entre 2,3 e 3,1 interações, valores considerados reduzidos e que indicam que as mudanças de agente ocorreram rapidamente, sem alongar desnecessariamente a conversa.
- **Consistência Conversacional:** manteve-se próxima de 4 em todas as categorias, demonstrando que a mudança de agente não afetou de forma significativa a naturalidade do diálogo.
- **Taxa de Problemas Resolvidos:** todos os agentes atingiram 100%, comprovando que, mesmo em cenários multiagente, o sistema foi sempre capaz de concluir o pedido.
- **Número Médio de Interações:** ficou entre 5 e 7 mensagens, o que é aceitável em cenários que envolvem múltiplos agentes, já que naturalmente exigem maior número de trocas até à resolução final.

De forma geral, os resultados confirmam que a funcionalidade multiagente está implementada de forma robusta e eficaz. O sistema mostrou-se capaz de encaminhar corretamente os pedidos e manter a fluidez da conversa mesmo após trocas de agente, cumprindo o objetivo central deste caso de estudo.

5.4 Caso de Estudo 3

O terceiro caso de estudo teve como objetivo avaliar o desempenho do sistema como um todo, simulando o processo de suporte desde o primeiro contacto até à resolução final do problema. Aqui, não apenas a capacidade de compreensão inicial e de resposta dos agentes foi analisada, mas também a eficácia das trocas multiagente, a criação de tickets no Jira e a integração com a API de Tickets. Em determinados testes, à semelhança do Caso de Estudo 2, a conversa foi intencionalmente conduzida até à criação de um ticket, de modo a validar também o correto funcionamento do agente de criação de tickets e da integração com o Jira.

Cada uma das 50 cadeias de e-mails foi recriada no chatbot, tal como descrito anteriormente, mas neste caso avaliando todo o ciclo de suporte, incluindo a criação de *tickets* quando necessário. Para garantir a validação da funcionalidade de deteção de duplicados, cada teste foi realizado duas vezes: numa primeira execução para simular a criação de um novo *ticket*, e numa segunda execução para verificar se o sistema conseguia identificar corretamente que se tratava de um pedido já registado.

Foram recolhidas as seguintes métricas:

- **Número Médio de Interações** – mede o número de mensagens trocadas até à conclusão do processo (resolução ou criação de *ticket*).
- **Taxa de Problemas Resolvidos (%)** – percentagem de casos em que o chatbot conseguiu dar resposta ou encaminhar corretamente para a criação de *ticket*.
- **Taxa de Tickets Jira Criados Corretamente (%)** – percentagem de tickets efetivamente criados com sucesso no Jira.
- **Taxa de Tickets Duplicados Reconhecidos (%)** – avalia a capacidade do sistema em identificar tickets já existentes e evitar duplicações.
- **Qualidade Média do Ticket Criado (0–5)** – avalia a clareza e completude das informações presentes nos tickets criados.
- **Taxa de Erros Críticos (%)** – percentagem de falhas graves que comprometeram a execução do processo (exemplo: falha na criação do ticket ou resposta incoerente).

Estas métricas permitem aferir não apenas a eficácia isolada dos agentes, mas também a robustez da solução como sistema integrado.

Os testes realizados encontram-se em anexo (Tabela 13), onde é possível observar o desempenho detalhado por tipo de ajuda e as médias globais.

Tabela 7 - Resultados do Caso de Estudo 3

Caso de Estudo 3						
Tipo de Ajuda (FAQ / Bug / Acesso / Sugestão)	**N.º Médio de Interações**	**Taxa de Problemas Resolvidos**	**Taxa de Tickets Jira Criados Corretamente**	**Taxa de Tickets Duplicados Reconhecidos**	**Qualidade Média do Ticket Criado (0–5)**	**Taxa de Erros Críticos**
FAQ	4,8	95%	95%	90%	4,15	5%
Bug	5,1	100%	100%	93%	4,33	0%
Acesso	3,0	100%	100%	100%	5	0%
Sugestão	4,6	100%	100%	100%	4,4	0%
Total	4,4	99%	99%	96%	4,5	1%

Tal como nos casos anteriores, foram também construídos os seguintes gráficos que permitem observar visualmente o comportamento do sistema em cada métrica, discriminado por tipo de ajuda.

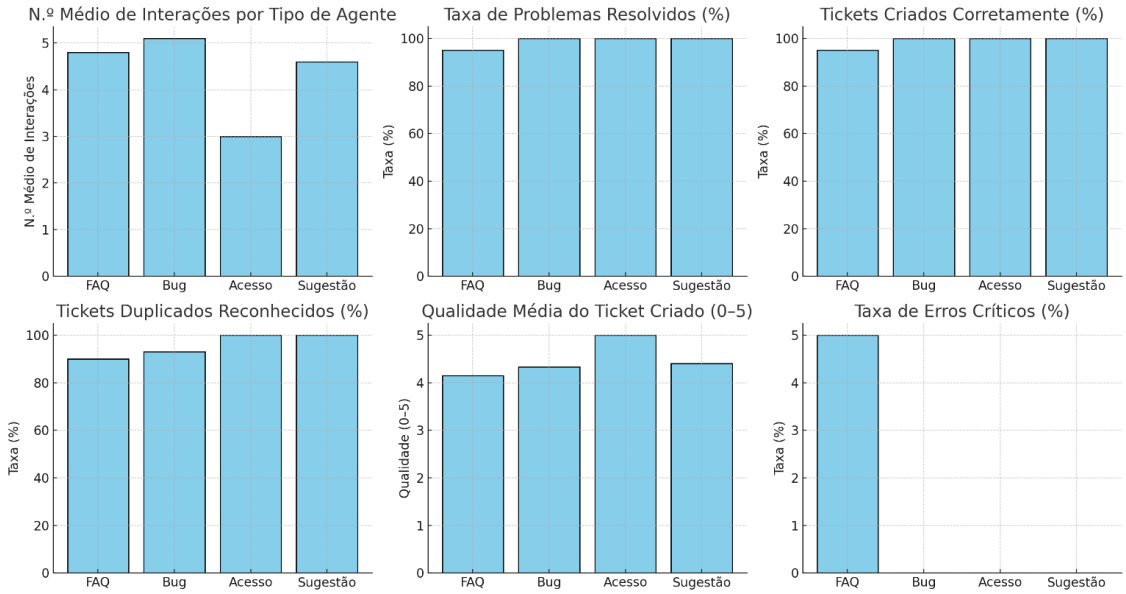


Figura 36 - Visualização gráfica dos resultados do Caso de Estudo 3

A análise métrica a métrica revela:

- **Número Médio de Interações:** os pedidos de Acesso foram os mais eficientes (3 interações em média), enquanto os pedidos de Bugs apresentaram maior número de interações (5,1), reflexo da maior complexidade destes pedidos.
- **Taxa de Problemas Resolvidos:** quase todos os agentes atingiram 100%, demonstrando que o sistema é capaz de responder ou encaminhar adequadamente pedidos em quase todos os cenários.
- **Taxa de Tickets Jira Criados Corretamente:** também com 100% em todos os casos, demonstrando a fiabilidade da integração com a API.
- **Taxa de Tickets Duplicados Reconhecidos:** variou entre 90% e 100%. Os pedidos de ajuda de Bugs e FAQs registaram os valores ligeiramente mais baixos, o que é natural devido à ambiguidade e repetição frequente desses pedidos. Ainda assim, a taxa global (96%) é altamente positiva.
- **Qualidade Média do Ticket Criado:** manteve-se consistentemente elevada, variando entre 4,3 e 5 pontos numa escala de 0 a 5.
- **Taxa de Erros Críticos:** apenas os pedidos de FAQ registaram 5% de erros, provavelmente associados à dificuldade de interpretar mensagens mais genéricas. Todos os restantes agentes mantiveram 0%, resultando numa taxa global de apenas 1%.

De forma geral, os resultados demonstram que o sistema, enquanto solução integrada, conseguiu reproduzir com sucesso todo o processo de suporte. A elevada taxa de resolução e de criação correta de tickets valida a proposta da tese, mostrando que a solução é viável, eficaz e capaz de reduzir drasticamente a dependência de intervenção humana.

5.5 Análise dos Resultados

Os casos de estudo realizados permitem comprovar que a solução desenvolvida responde de forma eficaz ao problema identificado no início da dissertação. Os três cenários de teste mostraram que o sistema é capaz de compreender pedidos simples, gerir situações que envolvem vários agentes e acompanhar todo o processo até à criação de tickets no Jira, com taxas de sucesso muito próximas de 100%. Esta consistência demonstra que a arquitetura implementada é robusta e que o modelo conversacional adotado funciona na prática.

Um dos pontos mais relevantes da análise prende-se com o impacto direto no tempo de resposta. De acordo com dados fornecidos pela equipa de suporte do CAPLA, o tempo médio de resposta a pedidos de utilizadores é de cerca de 24 horas, incluindo períodos de fim de semana. Com a introdução do chatbot, esse tempo é reduzido para apenas alguns segundos na maioria dos casos, já que o sistema poderá estar disponível 24 horas por dia. Mesmo nos cenários em que é necessária a criação de tickets, o processo é executado automaticamente,

ficando imediatamente disponível no Jira para as equipas de suporte e desenvolvimento, o que encurta consideravelmente os ciclos de resolução.

Apesar dos resultados promissores, é importante reconhecer que a amostra utilizada para os testes foi relativamente reduzida — apenas 50 interações reais, algumas delas semelhantes entre si. Este número reflete uma percentagem da diversidade e a complexidade dos pedidos que a equipa do CAPLA recebe mensalmente. Assim, uma fase de testes mais extensa e prolongada seria essencial para consolidar os resultados, expandir a base de conhecimento e aperfeiçoar os *prompts* dos agentes. Tal permitiria abranger um espectro mais amplo de situações e aumentar a resiliência do sistema em cenários de produção.

No seu estado atual, a solução já se configura como um *Minimum Viable Product (MVP)*, capaz de ser integrado no CAPLA e gerar benefícios imediatos. No entanto, a evolução natural do sistema passa por iterações sucessivas de teste e refinamento, aproveitando métricas de desempenho recolhidas em tempo real para ajustar a experiência conversacional. Com estas melhorias, seria possível aumentar ainda mais a taxa de resolução autónoma, aproximando o desempenho do chatbot de uma substituição quase completa da intervenção humana nas tarefas de suporte de primeira linha, permitindo que os recursos humanos se concentrem em problemas de maior complexidade.

5.6 Conclusões de Capítulo

Neste capítulo foram apresentados e analisados os três casos de estudo definidos para avaliar o sistema desenvolvido. Cada caso permitiu observar diferentes perspetivas do funcionamento do chatbot e da API de Tickets: a capacidade de cada agente responder autonomamente a pedidos específicos, a eficácia do sistema em cenários multiagente e, por fim, o desempenho do processo completo de suporte, desde o primeiro contacto até à criação e gestão de *tickets*.

A análise dos resultados mostrou que, mesmo em condições de teste limitadas a 50 interações reais adaptadas, o sistema conseguiu atingir níveis de desempenho muito elevados em termos de compreensão, clareza, consistência e taxa de resolução. Além disso, demonstrou-se que a integração entre o chatbot, a API de Tickets e o Jira não só permitiu a automação de tarefas rotineiras como também reduziu drasticamente o tempo de resposta em comparação com o processo humano tradicional.

Assim, este capítulo permitiu validar experimentalmente a viabilidade da solução proposta, mostrando que esta constitui já um *MVP* funcional capaz de responder aos desafios enfrentados pela equipa do CAPLA. Ainda que testes adicionais, em contexto de produção, sejam necessários para consolidar a robustez da solução, os resultados obtidos permitem responder de forma clara às questões de investigação. Com efeito, os casos de estudo confirmam o impacto positivo da aplicação Help Desk no suporte ao cliente, a eficácia de um chatbot no esclarecimento de dúvidas frequentes e o contributo de um sistema automatizado para a produtividade e eficiência da equipa de suporte de uma aplicação.

6 Conclusões

Neste capítulo será apresentada uma reflexão final sobre o trabalho desenvolvido, avaliando em que medida os objetivos e as questões de investigação foram concretizados. Serão também discutidos os principais resultados alcançados, as limitações identificadas ao longo do processo e as oportunidades de melhoria e de investigação futura que emergem da solução proposta.

6.1 Objetivos Concretizados

No início deste trabalho foram definidos um conjunto de objetivos gerais que orientaram toda a investigação e implementação desenvolvida. Segue-se a análise de cada um deles, bem como a sua concretização prática:

- **Realizar um estudo aprofundado sobre o estado da arte de aplicações help desk / suporte ao cliente:**
  Este objetivo foi cumprido através da revisão da literatura e da análise de soluções help desk tanto no meio académico como no mercado. Foram estudados, em particular, uma tese de mestrado, um artigo científico e três aplicações amplamente utilizadas (Zendesk, Freshdesk e Zoho Desk). Esta análise permitiu identificar funcionalidades consideradas essenciais neste tipo de sistemas, bem como tendências tecnológicas relevantes para a área de suporte ao cliente. Em complemento, foi conduzido um estudo aprofundado sobre chatbots e sobre as melhores práticas para a sua utilização em contextos de suporte. Assim, o estudo realizado constituiu um contributo sólido para o conhecimento sobre aplicações de help desk e de suporte ao cliente.

- **Desenvolver uma proposta de solução de suporte ao utilizador:**
  O objetivo foi alcançado com a elaboração de uma proposta sólida, construída a partir da análise de casos reais de suporte e orientada para resolver os principais problemas identificados. O sistema foi desenhado com base em boas práticas de engenharia informática e apoiado nos modelos C4 e 4+1, permitindo uma visualização clara e compreensiva de todos os componentes e fluxos envolvidos. O resultado foi uma solução bem estruturada, pensada não só para responder às necessidades imediatas

do CAPLA, mas também para contextos semelhantes, onde se verifica uma lacuna no suporte e acessibilidade das aplicações e é necessária uma solução simples, de fácil integração e utilização.

- **Implementar um MVP (Produto Mínimo Viável) da solução proposta:**
  Foi desenvolvido um MVP funcional que integra todos os componentes nucleares definidos no design: um chatbot multiagente criado no Voiceflow, suportado por uma base de conhecimento, e uma API de Tickets responsável pela comunicação com o Jira. Esta implementação permitiu validar na prática os fluxos concebidos na fase de desenho e demonstrar a viabilidade da solução. Para além disso, a realização dos casos de estudo comprovou o valor que esta solução poderá trazer para a empresa, mostrando que o sistema tem potencial para reduzir tempos de resposta, aumentar a eficiência do suporte e otimizar a gestão de pedidos. Assim, o protótipo desenvolvido assume-se como um MVP, pronto a ser evoluído e integrado em ambiente real.

- **Realizar testes práticos para avaliar o impacto da solução:**
  Embora não tenha sido possível envolver utilizadores reais do CAPLA devido a restrições de acesso a dados sensíveis, este objetivo foi atingido através da realização de casos de estudo baseados em 50 conversas reais cedidas pela equipa de suporte. Estes testes possibilitaram avaliar o desempenho do sistema em três dimensões: a autonomia dos agentes especializados, a eficácia da funcionalidade multiagente e o funcionamento do sistema como um todo, incluindo a criação e gestão de tickets. Para além da redução drástica do tempo médio de resposta a pedidos de suporte, que poderá passar de cerca de 24 horas para apenas alguns segundos, os testes demonstraram também taxas muito elevadas de compreensão correta no primeiro contacto (98%) e de problemas resolvidos com sucesso (99%). Em conjunto, estes resultados comprovam a robustez e a qualidade do sistema, validando o impacto positivo que a solução pode trazer para a equipa de suporte e para os utilizadores finais.

- **Analisar os resultados dos testes e planear ajustes da solução conforme necessário:**
  A análise dos resultados permitiu comprovar que a solução é eficaz e cumpre os objetivos a que se propõe, mas também evidenciou pontos de melhoria importantes, como a expansão da base de conhecimento e o reforço do treino dos agentes. Estas conclusões constituem a base para ajustes futuros, garantindo a evolução contínua do sistema. Para além destes aspetos imediatos, no subcapítulo 6.3 será apresentada uma reflexão mais alargada sobre o trabalho necessário para evoluir de um MVP funcional para um produto completo e competitivo no mercado.

Em complemento aos objetivos gerais, foram também definidas três questões de investigação que assumiram um papel central no rumo da dissertação. Estas questões funcionaram como guia para a definição da solução proposta, bem como para a condução da fase de testes e análise de resultados.

Segue-se a análise de cada uma dessas questões e a forma como foram respondidas ao longo do trabalho:

1. Qual é o impacto da implementação de uma aplicação *help desk* no suporte ao cliente de uma aplicação?

Os casos de estudo demonstram que a solução desenvolvida poderá ter um impacto substancial no suporte ao cliente, sobretudo na rapidez e qualidade da resposta. Quando a resolução é feita diretamente pelo chatbot, o tempo médio de resposta por mensagem foi de apenas alguns segundos, contrastando com a média de 24 horas registada pela equipa humana. Mesmo nos cenários que exigem a criação de tickets, o sistema automatiza a classificação, priorização e registo no Jira, garantindo que o pedido chega de imediato à equipa de suporte, já estruturado e pronto a ser tratado.

Para além da rapidez, os resultados mostraram níveis de desempenho muito elevados noutras dimensões. Observou-se uma taxa de compreensão correta no primeiro contacto de 98%, uma precisão média das respostas de 90%, uma clareza média das respostas de 4,5 numa escala de 0 a 5, e uma taxa global de problemas resolvidos de 99%. Estes indicadores, aliados ao número médio de interações por conversa de apenas 4,4, evidenciam que o sistema não só responde rapidamente, como também o faz de forma consistente e objetiva.

Em conjunto, estes resultados comprovam que a aplicação tem potencial para reduzir drasticamente os tempos de resposta e aumentar a eficiência do suporte, enquanto mantém padrões de qualidade comparáveis ou mesmo superiores aos do processo humano tradicional.

2. De que forma a utilização de um chatbot pode melhorar a eficácia na resolução de problemas frequentes de utilizadores?

O chatbot multiagente desenvolvido mostrou-se particularmente eficaz na resolução autónoma de dúvidas recorrentes, sobretudo através do Agente de Perguntas. Nos casos de estudo, este agente apresentou uma taxa de compreensão correta de 90% e uma precisão de resposta de 86%, confirmando que é capaz de interpretar a maioria das questões e fornecer respostas adequadas.

Para além disso, alcançou uma taxa de problemas resolvidos de 95%, o que demonstra que, mesmo em cenários de maior ambiguidade, o sistema conseguiu dar resposta satisfatória à maioria dos pedidos. A clareza média das respostas foi de 3,9 numa escala de 0 a 5, e o número médio de interações necessárias até à resolução situou-se em 5,1, valores que se mantêm aceitáveis para uma experiência fluida. Apesar disso, em cerca de 20% dos casos foi necessária reformulação da questão, evidenciando espaço para melhoria com expansão da base de conhecimento e ajustes nos fluxos conversacionais.

No geral, estes resultados comprovam que a utilização de um chatbot aumenta significativamente a eficácia no esclarecimento de dúvidas frequentes, reduzindo a necessidade de intervenção humana e permitindo que a equipa de suporte se concentre em tarefas de maior complexidade.

3. Como pode um sistema automatizado de gestão de pedidos contribuir para o aumento da produtividade e eficiência de uma equipa de suporte?

O sistema desenvolvido demonstrou potencial para aumentar de forma significativa a produtividade da equipa de suporte ao automatizar etapas críticas do processo de gestão de pedidos. Através da integração com o Jira, os tickets poderão passar a ser criados de forma automática, já classificados por categoria, descritos com a informação essencial e priorizados segundo o tipo de problema. Este processo permite reduzir o tempo que a equipa despende na triagem e estruturação manual de cada pedido, libertando os técnicos para se focarem em casos de maior complexidade.

Os resultados dos casos de estudo confirmaram a robustez deste mecanismo. Registou-se uma taxa de 99% de tickets corretamente criados e uma taxa de 96% de duplicados reconhecidos, o que evidencia não apenas a fiabilidade da integração, mas também a redução de retrabalho. Além disso, a qualidade média dos tickets criados foi avaliada em 4,5 numa escala de 0 a 5, demonstrando que a informação disponibilizada à equipa é clara e suficiente para dar continuidade à resolução.

Com estas capacidades, o sistema mostra que poderá eliminar grande parte das tarefas repetitivas e de baixo valor acrescentado, como a recolha de informação básica ou a verificação manual de pedidos duplicados, contribuindo assim para uma maior eficiência operacional. Desta forma, pode concluir-se que um sistema automatizado de gestão de pedidos como o desenvolvido tem um contributo claro para o aumento da produtividade da equipa de suporte, enquanto melhora a qualidade e a rastreabilidade do processo de resolução de problemas.

Para além da análise aos objetivos gerais e às questões de investigação, importa ainda verificar em que medida os requisitos funcionais e não funcionais definidos inicialmente foram concretizados. As tabelas seguintes apresentam o resumo da avaliação:

Tabela 8 – Requisitos Funcionais Concretizados

Requisito	**Descrição**	**Situação**
RF01	O sistema deve permitir que os utilizadores submetam pedidos de suporte através do *chatbot*.	Cumprido. Os utilizadores podem submeter pedidos de suporte através to chatbot.
RF02	O *chatbot* deve responder automaticamente a perguntas	Cumprido. O chatbot é capaz de responder a perguntas frequentes

	frequentes com base numa base de conhecimento atualizável.	com base na sua base de conhecimento.
RF03	O sistema deve criar um *ticket* quando o *chatbot* não consegue resolver a consulta.	Cumprido. O sistema cria tickets no Jira caso chatbot não resolva o problema/pedido do utilizador.
RF04	Os *tickets* devem ser automaticamente classificados por categoria e prioridade.	Cumprido. O sistema cria tickets no Jira automaticamente classificados por categoria e prioridade.
RF05	A equipa de suporte deve poder fazer a gestão dos *tickets*.	Cumprido. Após os tickets serem criados no Jira, a equipa de suporte pode fazer a sua gestão na paltaforma.
RF06	O sistema deve fornecer uma interface para visualização e histórico de *tickets* por parte dos utilizadores.	Cumprido. Acedendo ao Jira os utilizadores podem visualizar o histórico e o estado dos seus tickets.
RF07	O sistema deve verificar automaticamente se o conteúdo de um novo pedido é semelhante a *tickets* anteriores.	Cumprido. O sistema compara novos pedidos de tickets com tickets anteriores, evitando redundâncias.

Tabela 9 - Requisitos Não Funcionais Concretizados

Requisito	**Descrição**	**Situação**
RNF01	O *chatbot* deve gerar respostas em menos de 3 segundos após cada interação.	Cumprido. Nos casos de estudo, o chatbot obteve um tempo médio de resposta de 2.2 segundos por mensagem.
RNF02	O *chatbot* deve manter discurso coerente, claro e orientado à resolução.	Cumprido. Nos casos de estudo, a clareza média da resposta obteve uma classificação de 4,6 numa escala de 0 a 5.
RNF03	O *chatbot* deve apresentar uma taxa mínima de 90% de compreensão correta no primeiro contacto.	Cumprido. Nos casos de estudo o chatbot atingiu uma taxa média de compreensão correta de 98% no primeiro contacto.

RNF04	O número médio de interações por conversa até resolução ou criação de *ticket* não deve exceder 5.	Cumprido. Nos casos de estudo o chatbot antingiu uma média global de 4,4 interações por conversa.
RNF05	A base de conhecimento do *chatbot* deve ser atualizável por ficheiros estruturados.	Cumprido. A base de conhecimento é atualizável por ficheiros estruturados no Voiceflow.
RNF06	O *chatbot* deve poder ser facilmente integrado em aplicações externas.	Cumprido. Conforme descrito no subcapítulo 4.1.7, a integração com aplicações de destino é realizada de forma simples e imediata.
RNF07	A interface do *chatbot* deve ser visualmente apelativa.	Cumprido. A interface disponibilizada pelo Voiceflow apresenta um design apelativo e funcional.
RNF10	A comunicação com a API de *tickets* deve ser feita de forma autenticada e segura.	Cumprido. A comunicação é autenticada com token (Basic Authentication).
RNF11	O sistema deve assegurar a proteção de dados sensíveis fornecidos pelos utilizadores.	Parcialmente cumprido. Embora a comunicação seja segura, a dependência do Voiceflow implica processamento externo de dados. Uma solução totalmente interna aumentaria a proteção.
RNF13	O sistema deve possuir documentação detalhada quanto à sua constituição e utilização.	Cumprido. A presente dissertação fornece documentação detalhada sobre a constituição e utilização do sistema.

De forma global, os resultados obtidos demonstram que os objetivos definidos foram plenamente alcançados e que as questões de investigação foram respondidas de forma consistente, confirmando a relevância e viabilidade da solução desenvolvida. A avaliação dos requisitos funcionais e não funcionais mostrou níveis de cumprimento muito elevados, o que suporta a validade e robustez do sistema. Apesar das limitações identificadas, a solução apresentada provou ser eficaz e escalável, constituindo um contributo sólido para a melhoria do suporte ao utilizador e abrindo caminho para futuras evoluções.

6.2 Limitações

Durante o desenvolvimento do presente trabalho, algumas limitações foram identificadas e condicionaram parcialmente o alcance da solução implementada.

A primeira limitação relaciona-se com o tempo disponível para a realização da dissertação. Sendo este um projeto académico com prazos definidos, a solução concebida foi estruturada de forma a constituir um protótipo executável no tempo útil da tese. Isto implicou que certas melhorias e extensões pensadas na fase de desenho não pudessem ser concretizadas nesta etapa, ficando reservadas como trabalho futuro, conforme descrito no subcapítulo seguinte.

A segunda limitação prende-se com as restrições de segurança impostas pela equipa responsável pela aplicação CAPLA, que impossibilitaram a realização de testes com utilizadores reais dentro do ambiente de produção. Para contornar esta restrição, recorreu-se à utilização de *threads* de e-mail históricas, disponibilizadas pela equipa de suporte, como base para a realização dos casos de estudo. Embora esta abordagem tenha permitido testar a solução em cenários autênticos, não substitui de forma integral a validação com utilizadores finais em contexto real, sendo esta uma etapa que se revela fundamental numa fase posterior de avaliação.

6.3 Trabalho Futuro

Apesar de os casos de estudo realizados demonstrarem que a solução desenvolvida já pode ser considerada um MVP funcional, ainda se encontra longe de constituir um produto plenamente viável para entrada no mercado. Embora os resultados tenham sido bastante promissores, a amostra utilizada revelou-se reduzida face ao volume real de interações que aplicações como o CAPLA poderão registar diariamente. Por esse motivo, a realização de uma fase *alpha*[47] de testes, com envolvimento de utilizadores reais, seria fundamental. Idealmente, esta fase poderia ser conduzida numa empresa piloto, como a Natixis, permitindo recolher *feedback* direto dos utilizadores. Tal abordagem possibilitaria a afinação dos *prompts* dos agentes e uma melhor estruturação da base de conhecimento, aumentando a robustez e a eficácia global do sistema.

Outra melhoria essencial para transformar o protótipo num produto viável passaria por reduzir a dependência da plataforma Voiceflow. O desenvolvimento do *chatbot* com código próprio permitiria não apenas uma maior flexibilidade no controlo das funcionalidades, mas também uma diminuição da dependência de terceiros, traduzindo-se numa solução mais escalável e adaptável a diferentes contextos empresariais. Além disso, essa mudança traria vantagens

[47] Fase inicial de testes de um software, em que o protótipo é validado em condições controladas, geralmente por utilizadores internos, para detetar falhas e avaliar a viabilidade da solução.

significativas ao nível da personalização, da integração com outros sistemas e da segurança da aplicação.

Paralelamente, seria necessário expandir a implementação no sentido de abranger as restantes funcionalidades que a revisão do estado da arte identificou como imprescindíveis para o sucesso de uma aplicação de help desk. Entre estas incluem-se a comunicação *omnicanal*, as bases de conhecimento para autoatendimento, relatórios e métricas analíticas de desempenho, e suporte multilíngue. A inclusão destas capacidades transformaria o protótipo numa ferramenta mais completa e competitiva, alinhada com as melhores práticas do setor.

Adicionalmente, explorando a arquitetura modular da API de Tickets, seria possível expandir o leque de integrações disponíveis. Atualmente, a solução comunica diretamente com o Jira, mas, de forma a abranger um conjunto mais vasto de cenários empresariais, poderia ser estendida a outras plataformas de gestão de tickets. Este desenvolvimento transformaria a solução numa ferramenta mais versátil, capaz de se adaptar à realidade de diferentes organizações.

Por fim, a nível de segurança, seria importante considerar a adoção de modelos de linguagem locais (*LLMs*) para substituir a dependência de serviços externos. Esta abordagem permitiria minimizar riscos associados à partilha de informação sensível através da internet, garantindo maior confidencialidade e alinhamento com requisitos de proteção de dados. A utilização de uma LLM local, ainda que mais exigente em termos de recursos, traria maior controlo e confiança no tratamento de informação crítica.

6.4 Apreciação final

A realização desta dissertação permitiu dar resposta a um problema concreto e cada vez mais relevante no panorama empresarial: a necessidade de assegurar mecanismos de suporte eficazes sem exigir a substituição das infraestruturas já consolidadas. O trabalho desenvolvido cumpriu o objetivo principal de conceber, implementar e validar um sistema *help desk* inteligente, composto por um *chatbot* com base de conhecimento atualizável e uma API capaz de integrar-se diretamente com sistemas de gestão de tickets existentes. Esta abordagem revelou-se inovadora, uma vez que não foi identificado no mercado nenhum sistema amplamente difundido que conjugue estas duas vertentes sem impor às equipas de suporte a adoção de uma nova aplicação autónoma.

Entre os aspetos mais positivos destacam-se a simplicidade e a praticidade da solução, que a tornam apelativa e facilmente integrável em contextos reais. O caso de estudo realizado demonstrou que a integração direta com o Jira permitiu aproveitar um sistema já enraizado na organização, assegurando continuidade nos processos e minimizando a curva de adaptação. Para além disso, os resultados obtidos evidenciam ganhos expressivos em termos de eficiência, nomeadamente a redução significativa do tempo de resposta, que poderá passar de cerca de 24 horas para apenas alguns segundos, garantindo uma melhoria clara na experiência do

utilizador e no desempenho das equipas de suporte. A modularidade e a escalabilidade do protótipo reforçam ainda o seu potencial de adaptação a outras plataformas e a diferentes cenários empresariais.

Naturalmente, o estudo centrou-se mais na vertente técnica do que numa validação científica alargada. Embora o estado da arte tenha fornecido um enquadramento sólido, faltou uma validação mais extensa e metodologicamente robusta que pudesse comprovar, em larga escala, o impacto da solução na melhoria de processos de suporte.

Ainda assim, este carácter aplicado constitui também um dos pontos de maior relevância do trabalho. A solução desenvolvida responde de forma direta a uma necessidade identificada em contexto real, oferecendo às empresas um meio simples, viável e eficaz de modernizar os seus processos de suporte. A partir desta base, abrem-se oportunidades claras para trabalhos futuros, nomeadamente a extensão da API a outras plataformas de gestão de tickets além do Jira, a incorporação de mecanismos de priorização mais sofisticados com recurso a inteligência artificial e a realização de estudos empíricos com maior abrangência para avaliar o impacto da solução na produtividade e na satisfação dos utilizadores.

Em síntese, esta dissertação apresenta-se como uma contribuição prática e inovadora, que gera valor imediato em ambientes empresariais. O sistema concebido mostrou ser eficaz, viável e transferível, constituindo uma base sólida para desenvolvimentos futuros e para investigações que procurem aprofundar, em termos académicos, o impacto de sistemas helpdesk, com o objetivo de automatizar e melhorar o suporte ao utilizador.

Referências

Adamopoulou E, Moussiades L (2020). An overview of chatbot technology. IFIP Advances in Information and Communication Technology, pp. 373–383. https://doi.org/10.1007/978-3-030-49186-4_31

Al-Hawari F, Barham H (2021). A machine learning based help desk system for IT service management. *Journal of King Saud University – Computer and Information Sciences*. https://doi.org/10.1016/j.jksuci.2019.04.001

Anon. (2025). Chatbot development platforms – ranking agosto 2025. PeerSpot. Disponível em: https://www.peerspot.com/categories/chatbot-development-platforms [Acedido a 10 Nov. 2024]

Anon. (s.d.). Dialogflow. Disponível em: https://dialogflow.cloud.google.com/ [Acedido a 10 Nov. 2024]

Bass L, Clements P, Kazman R (2012). *Software architecture in practice*. Addison-Wesley.

Brown S (2025). C4 model. Disponível em: https://c4model.com [Acedido a 15 Jul. 2025]

Cambridge (2024). Help desk. *Cambridge Dictionary*. Disponível em: https://dictionary.cambridge.org/dictionary/english/help-desk [Acedido a 10 Nov. 2024]

Leitner C, Neuhüttler J, Bassano C, Satterfield D (2024). *The human side of service engineering*. AHFE International. https://doi.org/10.54941/ahfe1003100

Choma D, Chwaleba C, Dzieńkowski M (2023). The efficiency and reliability of backend technologies: Express, Django, and Spring Boot. *Informatyka, Automatyka, Pomiary w Gospodarce i Ochronie Środowiska*, 12, pp. 73–78. https://doi.org/10.35784/iapgos.4279

Devlin J, Chang MW, Lee K, Toutanova K (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics*, pp. 4171–4186. https://doi.org/10.18653/v1/N19-1423

Dijkstra EW (1982). *On the role of scientific thought*. Springer.

Park D, An GT, Kamyod C, Kim CG (2023). A study on performance improvement of prompt engineering for generative AI with a large language model. *Journal of Web Engineering*, 22(8), pp. 1187–1206. https://doi.org/10.13052/jwe1540-9589.2285

Wang D, Li T, Zhu S, Gong Y (2011). iHelp: An intelligent online helpdesk system. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 41(1), pp. 173–182. https://doi.org/10.1109/TSMCB.2010.2049352

European Union (2024). Regulation (EU) 2024/1689 of the European Parliament and of the Council of 13 June 2024 laying down harmonised rules on artificial intelligence (Artificial Intelligence Act) and amending certain Union legislative acts.

Eurostat (2024). Digital skills. Disponível em: https://ec.europa.eu/eurostat/databrowser/view/tepsr_sp410/default/table [Acedido a 10 Out. 2024] DOI: https://doi.org/10.2908/TEPSR_SP410

Eurostat (2024). Internet use. Disponível em: https://ec.europa.eu/eurostat/databrowser/view/isoc_ci_ifp_iu/default/table [Acedido a 10 Out. 2024] DOI: https://doi.org/10.2908/ISOC_CI_IFP_IU

Evans E (2004). *Domain-driven design: Tackling complexity in the heart of software*. Addison-Wesley Professional.

Fielding RT (2000). *Architectural styles and the design of network-based software architectures*. University of California, Irvine.

Fowler M (2002). *Patterns of enterprise application architecture*. Addison-Wesley Professional.

Gamma E, Helm R, Johnson R, Vlissides J (1994). *Design patterns: Elements of reusable object-oriented software*. Addison-Wesley.

Grady RB (1992). *Practical software metrics for project management and process improvement*. Prentice Hall.

Keshav S (2007). How to read a paper. *ACM SIGCOMM Computer Communication Review*, 37(3), pp. 83–84. https://doi.org/10.1145/1273445.1273458

Kruchten P (1995). The 4+1 view model of architecture. *IEEE Software*, 12(6), pp. 42–50. https://doi.org/10.1109/52.469759

Ein-Dor L, Toledo-Ronen O, Spector A, et al. (2024). Conversational prompt engineering. arXiv preprint arXiv:2408.04560. https://doi.org/10.48550/arXiv.2408.04560

Lin J (2024). Best help desk software: User reviews. G2. Disponível em: https://www.g2.com/categories/help-desk [Acedido a 21 Nov. 2024]

Luo B, Lau R, Li C, Si YW (2021). A critical review of state-of-the-art chatbot designs and applications. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 12(6). https://doi.org/10.1002/widm.1434

Martin RC (2003). *Agile software development: Principles, patterns, and practices*. Prentice Hall.

Marvão S (2024). Inovação impulsiona mercado de software. *Jornal de Negócios*. Disponível em: https://www.jornaldenegocios.pt/negocios-iniciativas/premio-nacional-de-inovacao/detalhe/20240405-0810-inovacao-impulsiona-mercado-de-software [Acedido a 10 Out. 2024]

Masongsong R, Damian MA (2016). Help desk management system. *Proceedings of the World Congress on Engineering and Computer Science (WCECS 2016)*, Vol. I.

Mikolov T, Sutskever I, Chen K, Corrado G (2013). Distributed representations of words and phrases and their compositionality. *Advances in Neural Information Processing Systems (NeurIPS 2013)*, 26.

Oates B (2005). Researching information systems and computing. Sage Publications.

Pawlik L (2025). How the choice of LLM and prompt engineering affects chatbot effectiveness. *Electronics*, 14(5). https://doi.org/10.3390/electronics14050888

Rangapur A, Rangapur A (2024). The battle of LLMs: A comparative study in conversational QA tasks. arXiv preprint arXiv:2405.18344. https://doi.org/10.48550/arXiv.2405.18344

Anexos

Tabela 10 - Emails utilizados para realização dos Casos de Estudo

Nº	Tipo de Ajuda (FAQ / Bug / Acesso / Sugestão)	Descrição
1	Acesso	Pedido de permissões de Gestor Financeiro
2	Acesso	Pedido de permissões de Gestor Financeiro
3	Acesso	Pedido de acesso à Plataforma X
4	Acesso	Pedido de acesso à Plataforma X
5	Acesso	Pedido de acesso à Plataforma Y
6	Sugestão	Sugestão de adição novo campo na página de gestão de recursos
7	Sugestão	Sugestão de adição novo campo na página de gestão de recursos
8	Sugestão	Sugestão de melhoria visual na interface de utilizador
9	Sugestão	Sugestão de adição de nova funcionalidade na página de gestão de recursos
10	Sugestão	Sugestão de adição de novo filtro de pesquisa de recursos
11	Sugestão	Sugestão de melhoria nos cálculos automáticos de custo por recurso
12	Sugestão	Sugestão de melhoria visual na interface de utilizador
13	Sugestão	Sugestão de adição novo campo na página de gestão de recursos
14	Sugestão	Sugestão de melhoria na informação apresentada num campo da página de gestão de recursos
15	Sugestão	Sugestão de adição de conversão de moeda
16	FAQ	Pergunta frequente quanto à utilização do sistema de criação de previsões

17	FAQ	Pergunta frequente quanto à utilização do sistema de criação de previsões
18	FAQ	Pergunta frequente quanto à utilização do sistema de criação de previsões
19	FAQ	Pergunta frequente quanto à funcionalidade de criação de movimento de saída
20	FAQ	Pergunta frequente quanto à funcionalidade de criação de movimento de modificação de salário
21	FAQ	Pergunta frequente quanto à utilização do sistema de criação de previsões
22	FAQ	Pergunta frequente quanto à utilização do sistema de criação de previsões
23	FAQ	Pergunta frequente quanto à funcionalidade de criação de movimento de saída
24	FAQ	Pergunta específica quanto a valor de um campo de um recurso
25	FAQ	Pergunta específica quanto a valor de um campo de um recurso
26	FAQ	Pergunta frequente quanto à utilização do sistema de criação de previsões
27	FAQ	Pergunta frequente quanto à utilização do sistema de criação de previsões
28	FAQ	Pergunta específica quanto a cálculos automaticos de salário
29	FAQ	Pergunta específica quanto a valor de um campo de um recurso
30	FAQ	Pergunta quanto a permissões de modificação de valores de recursos
31	FAQ	Pergunta frequente de processo de filtragem por previsões
32	FAQ	Pergunta frequente quanto à utilização do sistema de criação de previsões
33	FAQ	Pergunta frequente quanto à informação de um recurso
34	FAQ	Pergunta específica sobre dados demonstrados
35	FAQ	Pergunta específica sobre dados demonstrados
36	Bug	Erro ao colocar 0 num campo do sistema de previsões.
37	Bug	Erro ao colocar 0 num campo do sistema de previsões.
38	Bug	Falta de informação numa coluna do recurso
39	Bug	Erro ao exportar informações de um recurso para Exel
40	Bug	Erro ao carregar página de gestão de recursos
41	Bug	Erro ao submeter previsão
42	Bug	Erro ao submeter previsão
43	Bug	Erro ao submeter previsão
44	Bug	Erro ao carregar página de gestão de recursos

45	Bug	Erro de dados
46	Bug	Erro ao colocar 0 num campo do sistema de previsões.
47	Bug	Erro de dados
48	Bug	Butão desabilitado
49	Bug	Bug visual numa coluna de dados
50	Bug	Erro de dados

Tabela 11 - Testes realizados para o Caso de Estudo 1

Nº	Tipo de Ajuda (FAQ / Bug / Acesso / Sugestão)	Compreensão Correta do Pedido no 1.º Contacto?	Precisão da Resposta (%)	Nº de Interações	Clareza da Resposta (0–5)	Necessidade de Reformulação?
1	Acesso	Sim	100%	3	5	Não
2	Acesso	Sim	100%	3	5	Não
3	Acesso	Sim	100%	3	5	Não
4	Acesso	Sim	100%	3	5	Não
5	Acesso	Sim	100%	3	5	Não
6	Sugestão	Sim	90%	4	5	Não
7	Sugestão	Sim	90%	3	5	Não
8	Sugestão	Sim	100%	4	4	Não
9	Sugestão	Sim	90%	3	4	Não
10	Sugestão	Sim	90%	5	5	Não
11	Sugestão	Sim	90%	5	5	Não
12	Sugestão	Sim	90%	4	5	Não
13	Sugestão	Sim	80%	4	5	Não
14	Sugestão	Sim	90%	5	5	Não
15	Sugestão	Sim	80%	6	5	Não
16	FAQ	Sim	80%	7	3	Não
17	FAQ	Sim	90%	4	3	Não
18	FAQ	Sim	100%	4	3	Não
19	FAQ	Não	90%	8	5	Não
20	FAQ	Sim	70%	6	4	Sim
21	FAQ	Sim	90%	4	4	Não
22	FAQ	Sim	90%	3	4	Não
23	FAQ	Sim	90%	6	4	Não
24	FAQ	Sim	100%	5	4	Não
25	FAQ	Sim	70%	3	5	Não
26	FAQ	Não	100%	3	5	Não
27	FAQ	Sim	80%	6	4	Não
28	FAQ	Sim	80%	5	3	Sim
29	FAQ	Sim	70%	5	4	Não
30	FAQ	Sim	100%	4	4	Não

31	FAQ	Sim	70%	5	3	Sim
32	FAQ	Sim	90%	6	4	Não
33	FAQ	Sim	70%	10	4	Não
34	FAQ	Sim	90%	5	3	Sim
35	FAQ	Sim	100%	3	5	Não
36	Bug	Sim	70%	6	5	Não
37	Bug	Sim	80%	6	4	Não
38	Bug	Sim	70%	8	4	Sim
39	Bug	Sim	100%	6	5	Não
40	Bug	Sim	80%	4	5	Não
41	Bug	Sim	100%	3	5	Não
42	Bug	Sim	70%	6	5	Não
43	Bug	Sim	80%	5	3	Não
44	Bug	Sim	80%	5	3	Não
45	Bug	Sim	100%	11	5	Não
46	Bug	Sim	70%	4	5	Não
47	Bug	Sim	100%	4	3	Não
48	Bug	Sim	90%	3	3	Não
49	Bug	Sim	70%	6	4	Não
50	Bug	Sim	90%	3	4	Não

Tabela 12 - Testes realizados para o Caso de Estudo 2

Nº	Tipo de Ajuda (FAQ / Bug / Acesso / Sugestão)	Agente Corretamente Identificado?	Sucesso na Comutação entre Agentes?	Tempo entre Transições (n.º mensagens)	Consistência Conversacional (0–5)	Problema Resolvido?	Nº de Interações
1	Acesso	Sim	Sim	2	4	Sim	5
2	Acesso	Sim	Sim	2	5	Sim	5
3	Acesso	Sim	Sim	2	3	Sim	6
4	Acesso	Sim	Sim	3	5	Sim	5
5	Acesso	Sim	Sim	2	4	Sim	6
6	Sugestão	Sim	Sim	3	3	Sim	8
7	Sugestão	Sim	Sim	2	4	Sim	6
8	Sugestão	Sim	Sim	2	4	Sim	7
9	Sugestão	Sim	Sim	3	4	Sim	6
10	Sugestão	Sim	Sim	4	3	Sim	8
11	Sugestão	Sim	Sim	2	4	Sim	6
12	Sugestão	Sim	Sim	3	4	Sim	9
13	Sugestão	Sim	Sim	2	5	Sim	9

14	Sugestão	Sim	Sim	2	4	Sim	6
15	Sugestão	Sim	Sim	2	4	Sim	8
16	FAQ	Sim	Sim	2	4	Sim	6
17	FAQ	Sim	Sim	3	4	Sim	4
18	FAQ	Sim	Sim	2	4	Sim	5
19	FAQ	Sim	Sim	2	4	Sim	6
20	FAQ	Sim	Sim	2	4	Sim	6
21	FAQ	Sim	Sim	4	5	Sim	6
22	FAQ	Sim	Sim	4	5	Sim	6
23	FAQ	Sim	Sim	4	4	Sim	4
24	FAQ	Sim	Sim	3	3	Sim	6
25	FAQ	Sim	Não	3	4	Sim	5
26	FAQ	Sim	Sim	2	3	Sim	4
27	FAQ	Sim	Sim	4	4	Sim	4
28	FAQ	Sim	Sim	5	3	Sim	4
29	FAQ	Sim	Sim	3	4	Sim	6
30	FAQ	Sim	Sim	4	3	Sim	5
31	FAQ	Sim	Sim	2	4	Sim	5
32	FAQ	Sim	Sim	2	4	Sim	4
33	FAQ	Sim	Sim	3	3	Sim	6
34	FAQ	Sim	Sim	2	4	Sim	6
35	FAQ	Sim	Sim	2	4	Sim	4
36	Bug	Sim	Sim	5	3	Sim	7
37	Bug	Sim	Sim	4	5	Sim	7

38	Bug	Sim	Sim	2	4	Sim	8
39	Bug	Sim	Sim	3	5	Sim	8
40	Bug	Sim	Sim	4	4	Sim	9
41	Bug	Sim	Sim	3	5	Sim	7
42	Bug	Sim	Sim	5	5	Sim	6
43	Bug	Sim	Sim	5	3	Sim	8
44	Bug	Sim	Sim	2	5	Sim	7
45	Bug	Sim	Sim	2	3	Sim	7
46	Bug	Sim	Sim	3	5	Sim	6
47	Bug	Sim	Sim	2	4	Sim	8
48	Bug	Sim	Sim	2	5	Sim	6
49	Bug	Sim	Sim	2	4	Sim	9
50	Bug	Sim	Sim	3	5	Sim	8

Tabela 13 - Testes realizados para o Caso de Estudo 3

Nº	Tipo de Ajuda (FAQ / Bug / Acesso / Sugestão)	N.º de Interações	Problema Resolvido?	Ticket Jira Criado Corretamente?	Ticket Duplicado Reconhecido?	Qualidade do Ticket Criado (0–5)	Erro Crítico?
1	Acesso	3	Sim	Sim	Sim	5	Não
2	Acesso	3	Sim	Sim	Sim	5	Não
3	Acesso	3	Sim	Sim	Sim	5	Não
4	Acesso	3	Sim	Sim	Sim	5	Não
5	Acesso	3	Sim	Sim	Sim	5	Não
6	Sugestão	4	Sim	Sim	Sim	4	Não
7	Sugestão	4	Sim	Sim	Sim	5	Não
8	Sugestão	4	Sim	Sim	Sim	5	Não
9	Sugestão	3	Sim	Sim	Sim	4	Não
10	Sugestão	6	Sim	Sim	Sim	5	Não
11	Sugestão	5	Sim	Sim	Sim	4	Não
12	Sugestão	4	Sim	Sim	Sim	5	Não

13	Sugestão	4	Sim	Sim	Sim	4	Não
14	Sugestão	6	Sim	Sim	Sim	4	Não
15	Sugestão	6	Sim	Sim	Sim	4	Não
16	FAQ	7	Sim	Sim	Sim	5	Não
17	FAQ	2	Sim	Sim	Sim	4	Não
18	FAQ	4	Não	Não	Não	0	Sim
19	FAQ	8	Sim	Sim	Sim	4	Não
20	FAQ	6	Sim	Sim	Sim	4	Não
21	FAQ	2	Sim	Sim	Sim	4	Não
22	FAQ	3	Sim	Sim	Sim	5	Não
23	FAQ	6	Sim	Sim	Sim	4	Não
24	FAQ	4	Sim	Sim	Sim	5	Não
25	FAQ	3	Sim	Sim	Sim	4	Não
26	FAQ	3	Sim	Sim	Sim	5	Não
27	FAQ	6	Sim	Sim	Sim	4	Não
28	FAQ	4	Sim	Sim	Sim	5	Não
29	FAQ	5	Sim	Sim	Não	4	Não
30	FAQ	4	Sim	Sim	Sim	4	Não
31	FAQ	5	Sim	Sim	Sim	4	Não
32	FAQ	6	Sim	Sim	Sim	5	Não
33	FAQ	10	Sim	Sim	Sim	4	Não
34	FAQ	5	Sim	Sim	Sim	4	Não
35	FAQ	3	Sim	Sim	Sim	5	Não
36	Bug	6	Sim	Sim	Sim	5	Não

37	Bug	6	Sim	Sim	Sim	4	Não
38	Bug	7	Sim	Sim	Sim	5	Não
39	Bug	6	Sim	Sim	Sim	4	Não
40	Bug	4	Sim	Sim	Sim	5	Não
41	Bug	3	Sim	Sim	Sim	4	Não
42	Bug	6	Sim	Sim	Não	5	Não
43	Bug	5	Sim	Sim	Sim	4	Não
44	Bug	5	Sim	Sim	Sim	4	Não
45	Bug	10	Sim	Sim	Sim	4	Não
46	Bug	4	Sim	Sim	Sim	4	Não
47	Bug	4	Sim	Sim	Sim	4	Não
48	Bug	3	Sim	Sim	Sim	5	Não
49	Bug	5	Sim	Sim	Sim	4	Não
50	Bug	3	Sim	Sim	Sim	4	Não