INSTITUTO FEDERAL
Rondônia

MINISTÉRIO DA EDUCAÇÃO
Secretaria de Educação Profissional e Tecnológica
Instituto Federal de Educação, Ciência e Tecnologia de Rondônia

Campus Vilhena

Coordenação do Curso Superior em Tecnologia em Análise e Desenvolvimento de Sistemas

VINÍCIUS ALMEIDA MORAES

Desenvolvimento de um Sistema de Gestão de Estoque com Agente de Inteligência Artificial Integrado

VILHENA - RO

2026

VINÍCIUS ALMEIDA MORAES

Desenvolvimento de um Sistema de Gestão de Estoque com Agente de Inteligência Artificial Integrado

Artigo entregue como Trabalho de Conclusão de Curso apresentado ao Instituto Federal de Educação, Ciência e Tecnologia de Rondônia (IFRO), Campus Vilhena, como requisito parcial para obtenção do grau de Tecnólogo, junto ao Curso Superior em Tecnologia em Análise e Desenvolvimento de Sistemas, sob a orientação do professor Marco Antônio Augusto de Andrade.

VILHENA - RO

2026

Ficha catalográfica elaborada pelo Sistema Gerador de Ficha Catalográfica do IFRO.

Moraes, Vinícius Almeida.
Desenvolvimento de um sistema de gestão de estoque com agente de inteligência artificial integrado / Vinícius Almeida Moraes. - Vilhena, 2026.
28 f. : il.

Orientador(a): Prof. Me. Marco Antonio Augusto de Andrade.

Trabalho de Conclusão de Curso (Superior de Tecnologia em Análise e Desenvolvimento de Sistemas) – Instituto Federal de Educação, Ciência e Tecnologia de Rondônia - IFRO, Vilhena, 2026.

1. Agentes conversacionais. 2. Model context protocol. 3. Processamento de linguagem natural. 4. Gestão de estoque com IA. I. Andrade, Marco Antonio Augusto de (orient.). II. Instituto Federal de Educação, Ciência e Tecnologia de Rondônia - IFRO. III. Título.

Bibliotecário(a) Responsável: Rosilene Maria do Couto Marques, CRB-11/321

VINÍCIUS ALMEIDA MORAES

Desenvolvimento de um Sistema de Gestão de Estoque com Agente de Inteligência Artificial Integrado

Artigo entregue como Trabalho de Conclusão de Curso apresentado ao Instituto Federal de Educação, Ciência e Tecnologia de Rondônia (IFRO), Campus Vilhena, como requisito parcial para obtenção do grau de Tecnólogo, junto ao Curso Superior em Tecnologia em Análise e Desenvolvimento de Sistemas, sob a orientação do professor Marco Antônio Augusto de Andrade.

Aprovado em 27/10/2026 pela banca examinadora.

_____

Prof. Mestre Wesley Jhonnes Ramos Rolim

Examinador interno

_____

Prof. Mestre Gilberto Pereira da Silva

Examinador interno

_____

Prof. Mestre Marco Antonio Augusto de Andrade

Orientador

Resumo

Sistemas tradicionais de gestão de estoque dependem, em geral, da navegação manual entre menus e telas para extrair informações, o que limita a velocidade de resposta dos gestores. Este trabalho teve como objetivo desenvolver um sistema integrado de gestão de estoque com um agente conversacional de Inteligência Artificial (IA) que responde a consultas em linguagem natural, com rastreabilidade e auditabilidade dos registros. Trata-se de pesquisa aplicada e exploratória. A aplicação web foi construída com Node.js, MongoDB e Next.js, sob o método Kanban. O agente combina um modelo de linguagem Gemini ao paradigma ReAct (Reasoning and Acting) e acessa o sistema pelo Model Context Protocol (MCP), com ferramentas somente leitura. A validação reuniu testes automatizados, uma avaliação com 19 perguntas conferidas contra o banco e a contagem das ações exigidas pela interface. O sistema foi aprovado em todas as suítes automatizadas e implantado em ambiente público. Na rodada final, o agente acionou a ferramenta adequada em todas as consultas ao banco e respondeu corretamente a 18 das 19 perguntas, resultado que mostra as falhas corrigidas, não a capacidade de generalização. Nas consultas de dados, a interface exigiu 82 ações e o agente, 48, com ganho nas consultas que combinam filtros, cruzam telas ou exigem cálculo. O trabalho mostra que o agente pode herdar a autorização já existente na Application Programming Interface (API), sem camada de permissão paralela para a IA, e complementar a navegação entre telas com consultas auditáveis.

Palavras-chave: Agentes Conversacionais; Model Context Protocol; Processamento de Linguagem Natural; Gestão de Estoque com IA.

Abstract

Traditional inventory management systems generally depend on manual navigation through menus and screens to extract information, which limits how quickly managers can obtain answers. This work aimed to develop an integrated inventory management system with a conversational Artificial Intelligence (AI) agent that answers natural language queries while preserving the traceability and auditability of records. This is applied, exploratory research. The web application was built with Node.js, MongoDB and Next.js, following the Kanban method. The agent combines a Gemini language model with the ReAct (Reasoning and Acting) paradigm and accesses the system through the Model Context Protocol (MCP), using read-only tools. Validation combined automated tests, an evaluation with 19 questions checked against the database and a count of the actions the interface requires. The system passed all automated test suites and runs in a public environment. In the final round, the agent called the appropriate tool in every database query and answered 18 of the 19 questions correctly, a result that reflects fixed failures rather than the ability to generalize. For the data queries, the interface required 82 actions and the agent 48, with the gain in queries that combine filters, cross screens or require computation. The work shows that the agent can inherit the authorization already present in the Application Programming Interface (API), without a parallel permission layer for the AI, and complement screen navigation with auditable queries.

Keywords: Conversational Agents; Model Context Protocol; Natural Language Processing; AI-Assisted Inventory Management.

1. Introdução

A administração de recursos materiais e o controle de inventários são determinantes para manter os ativos disponíveis em ambientes organizacionais (Santos, 2024). Parte-se da premissa de que a precisão dos registros logísticos condiciona o bom andamento das operações de qualquer negócio.

No caso estudado por Venâncio e Bueno (2023), um supermercado familiar, o sistema de gestão tradicional reunia dados de forma pouco organizada. O *software* de controle de estoque não era integrado às áreas de compra e venda, e a posição registrada no sistema divergia da contagem física. Os próprios dados de venda gerados pelo sistema não chegavam a ser consultados nem repassados ao setor de compras. Como consequência, consumia-se mais tempo reunindo informações do que efetivamente utilizando-as na tomada de decisão.

Um segundo caso, independente, numa pequena e média empresa de outro país, chega a diagnóstico semelhante. Nesse caso, as consultas exigiam navegar de cinco a sete módulos, e a agregação manual levava de 20 a 45 minutos, contra menos de 30 segundos no protótipo conversacional. Os próprios autores classificam essas cifras como indicativas de ordem de grandeza, obtidas de um número pequeno de execuções num único caso, e não como medição generalizável (Pookduang; Chansanam, 2026).

A convergência entre os dois casos sustenta a premissa deste trabalho de que essa barreira é estrutural. Sistemas convencionais exigem navegação manual entre telas e módulos distintos, o que limita a velocidade de resposta dos gestores diante de consultas que cruzam categorias diferentes de dados.

Diante desse cenário, a aplicação do Processamento de Linguagem Natural (PLN) apresenta-se como um caminho promissor. Um agente conversacional baseado em Inteligência Artificial (IA) interpreta a consulta do usuário e retorna uma resposta estruturada (Gioè; Catalano, 2025).

Nesse sentido, a presente pesquisa parte das limitações operacionais dos sistemas tradicionais e da complexidade de interpretar seus dados brutos. A questão que se busca responder é: de que forma um agente conversacional de IA, integrado a um sistema de gestão de estoque, pode facilitar a consulta e a análise de inventários multidomínio?

Em resposta a essa problemática, definiu-se como objetivo geral desenvolver um sistema integrado de gestão de estoque, auxiliado por um agente conversacional de IA. O sistema deve promover a rastreabilidade, a auditabilidade dos registros e o suporte à tomada de decisão. Esse objetivo desdobra-se em especificar os requisitos do sistema, desenvolver a solução integrada (*back-end* e *front-end*) e implementar o agente de IA baseado em PLN. Desdobra-se também em validar a solução, por testes automatizados e pela análise das respostas do agente, e em avaliar a eficiência da solução frente aos métodos tradicionais.

A relevância desta pesquisa decorre da necessidade de modernizar as práticas de controle logístico diante da rigidez estrutural aqui diagnosticada. Em experimento controlado de reposição de estoque, equipes que reúnem humano e IA tendem a obter lucros mais altos do que humanos ou agentes de IA operando isoladamente (Baek *et al.*,

2026). Esse resultado reforça o potencial de soluções híbridas para a gestão de inventários. Justifica-se, assim, o desenvolvimento de uma solução baseada em modelos de linguagem, capaz de superar essa rigidez e de otimizar a resolução de gargalos operacionais.

2. Fundamentação Teórica

2.1. Sistemas de Gestão de Estoque e a Transição para Inteligência Artificial

A forma como os estoques são administrados é determinante para a saúde financeira de uma organização, com reflexos diretos na posição competitiva frente a outras empresas do setor (Slack; Chambers; Johnston, 2009 *apud* Santos, 2024). O controle de inventários evoluiu das anotações manuais para os Sistemas Integrados de Gestão Empresarial (ERP), do inglês *Enterprise Resource Planning*.

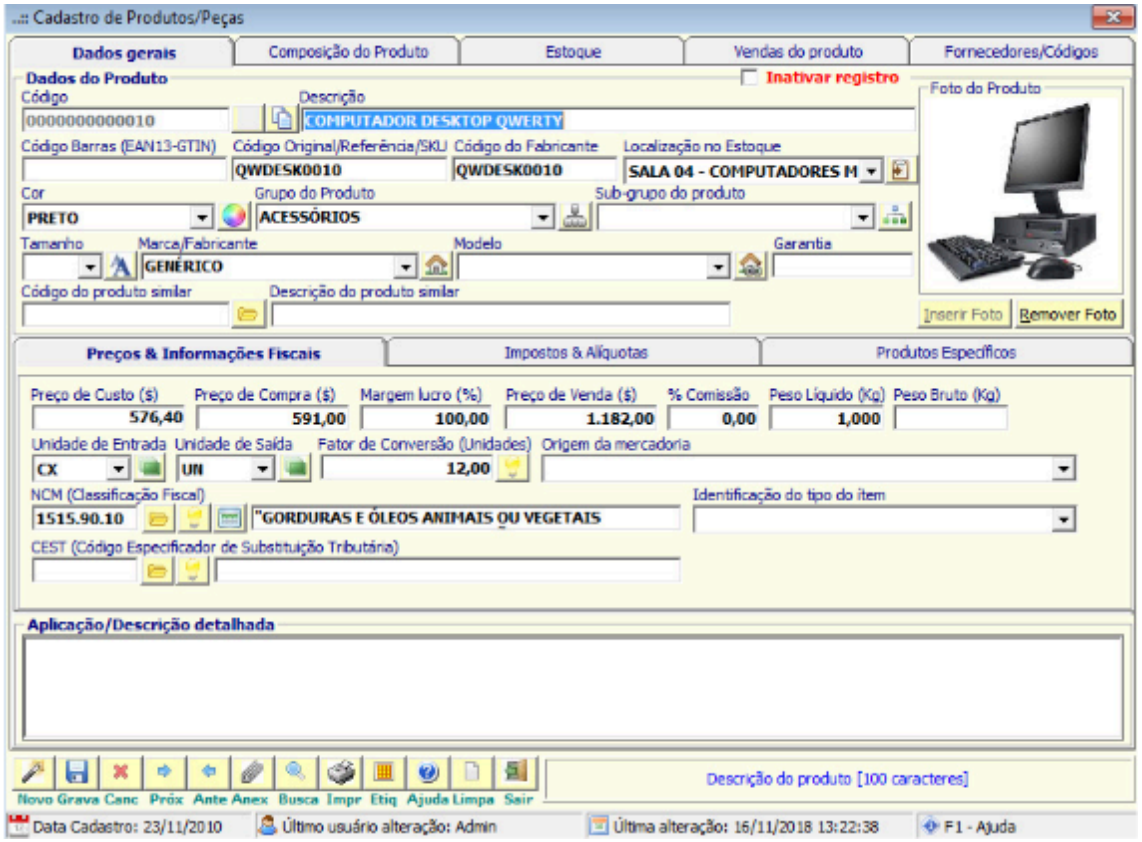


Figura 1. Interface de cadastro de produtos em sistema ERP.

Fonte: Next Software, 2018.

A Figura 1 ilustra uma tela de cadastro de um sistema ERP tradicional, com campos de entrada organizados em múltiplas abas, cada uma com um conjunto distinto de atributos. Para cruzar informações de categorias diferentes, o usuário precisa navegar manualmente entre elas.

Diante dessa rigidez estrutural, a aplicação da IA ao setor logístico é apontada como detentora de potencial de crescimento expressivo (Dellot *et al.*, 2017 *apud* Venâncio; Bueno, 2023).

2.2. Modelos de Linguagem e Processamento de Linguagem Natural

O PLN reúne as técnicas que permitem a computadores processar a linguagem humana para tarefas de utilidade prática, hoje reconhecido como núcleo da IA moderna (Jurafsky; Martin, 2026).

Tecnicamente, um modelo de linguagem é um sistema computacional que atribui uma distribuição de probabilidade às possíveis palavras seguintes a partir do contexto anterior. Ele opera sobre *tokens*, que podem ser palavras inteiras ou fragmentos delas. Aplicado de forma recursiva, esse mecanismo de predição sustenta a geração de texto por amostragem sucessiva (Jurafsky; Martin, 2026).

O *Transformer*, hoje arquitetura de referência para múltiplas tarefas de linguagem, foi proposto originalmente num domínio mais estreito, o de tradução automática (Jurafsky; Martin, 2026). Seu mecanismo de atenção relaciona posições distintas de uma mesma sequência para compor a representação de cada uma (Vaswani *et al.*, 2017).

Essa arquitetura viabilizou os Grandes Modelos de Linguagem (LLMs), do inglês *Large Language Models*. Um único LLM pré-treinado generaliza múltiplas capacidades linguísticas e se ajusta a novos domínios por instruções em linguagem natural (*prompting*), sem retreinamento (Brown *et al.*, 2020). Essa flexibilidade colocou os LLMs no centro de uma nova classe de sistemas, cuja capacidade prática depende tanto do modelo quanto da infraestrutura de execução construída ao redor dele (Wei, 2026; Zhou *et al.*, 2026).

2.3. Arquitetura de Agentes: *Harness* e *Model Context Protocol* (MCP)

O LLM concentra a capacidade de raciocínio da aplicação (Derouiche; Brahmi; Mazeni, 2025), mas depende de infraestrutura externa para agir sobre o ambiente (Wei, 2026). Essa infraestrutura é o *Harness*, *software* não-LLM que coordena ferramentas, contexto e execução, traduzindo o raciocínio do modelo em invocações concretas de ferramentas (Wei, 2026; Zhou *et al.*, 2026).

A lógica de raciocínio parte do paradigma de Raciocínio e Ação (ReAct), do inglês *Reasoning and Acting*. Nele, o modelo alterna pensamentos e ações, e cada ação devolve uma observação que alimenta o pensamento seguinte (Yao *et al.*, 2023).

O Protocolo de Contexto de Modelo (MCP), do inglês *Model Context Protocol*, padroniza a forma como agentes solicitam ações no ambiente da organização (*Model Context Protocol*, 2025; Sarkar; Sarkar, 2025). Ele adota uma arquitetura cliente-*host*-servidor, em que a aplicação hospedeira cria um cliente para cada servidor MCP.

A especificação do protocolo fixa como princípios o consentimento explícito do usuário e o controle de acesso. Reconhece, porém, que o protocolo não consegue impô-los e deixa essa tarefa a quem o implementa. Para transportes HTTP, prevê ainda uma autorização opcional baseada em OAuth (Model Context Protocol, 2025).

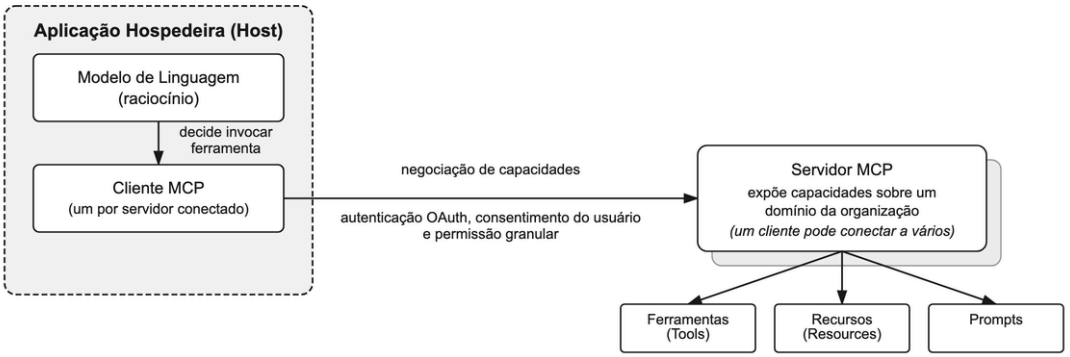


Figura 2. Arquitetura cliente-*host*-servidor do *Model Context Protocol*.

Fonte: Elaborado pelo autor (2026).

A Figura 2 esquematiza de forma genérica a arquitetura cliente-*host*-servidor. A aplicação hospedeira reúne o modelo de linguagem e os clientes MCP, cada um ligado a um único servidor. Cada servidor, por sua vez, expõe ferramentas, recursos e *prompts* como capacidades disponíveis ao agente.

Essa ligação com dados externos também abre caminho para um ataque. Na injeção indireta de *prompt*, alguém grava uma instrução maliciosa num registro que alguma ferramenta vai ler. O modelo recebe esse texto junto com os dados legítimos, tem dificuldade para distingui-lo e pode acabar seguindo a instrução (Hou *et al.*, 2026). Como o ataque depende apenas de o modelo ler dados por meio de ferramentas, ele alcança também sistemas com um único agente, como o deste trabalho.

2.4. Trabalhos Relacionados

As interfaces em linguagem natural para bancos de dados (NLIDB), do inglês *Natural Language Interface to Databases*, permitem que alguém sem domínio de uma linguagem de consulta pergunte diretamente ao banco. Nelas se insere o *text-to-SQL*, em que o modelo traduz a pergunta para um comando em Linguagem de Consulta Estruturada (SQL), do inglês *Structured Query Language*. Os LLMs alcançaram os melhores resultados nos *benchmarks* anteriores, mas o desempenho cai perto do uso corporativo. No Spider 2.0, montado com bancos corporativos em uso real, os modelos mais avançados resolveram perto de 21% das tarefas (Hong *et al.*, 2025).

O trabalho mais próximo é o de Pookduang e Chansanam (2026), um protótipo que responde em linguagem natural a perguntas sobre os dados de gestão de uma pequena ou média empresa. Em vez de gerar SQL, o modelo usa a chamada de ferramentas (*tool calling*), que buscam os dados sobretudo pela Interface de Programação de Aplicações (API), do inglês *Application Programming Interface*, do ERP da empresa. As diferenças estão na padronização e no controle de acesso. Neste trabalho, as ferramentas são expostas por MCP, e o agente, na sessão de quem pergunta, só alcança o que esse usuário tem permissão para ver. Em Pookduang e Chansanam (2026), esse controle fica para uma versão futura.

No controle de estoque, Baek *et al.* (2026) usam agentes baseados em LLM numa simulação de reposição. O modelo recebe a situação do estoque já pronta e decide, semana a semana, quantas unidades encomendar ao fornecedor. Já Gioè e Catalano (2025) adotam o MCP num agente conversacional para as operações de Tecnologia da Informação (TI) de uma empresa de médio porte. Além de consultar, esse agente abre, edita e encerra chamados. Neste trabalho, ao contrário, o agente apenas lê dados e busca por conta própria os registros de que precisa para responder ao usuário.

3. Metodologia

3.1. Classificação da Pesquisa e Desenho da Avaliação

Quanto à natureza, o trabalho se enquadra como pesquisa aplicada, por gerar conhecimento destinado ao uso prático, aqui a consulta aos dados de um sistema de gestão de estoque. Quanto aos objetivos, é exploratório; quanto à abordagem do problema, qualitativo, pois as respostas do agente são interpretadas e classificadas sem tratamento estatístico (Prodanov; Freitas, 2013). Percorreu-se primeiro o desenvolvimento do sistema e, em seguida, a avaliação do que foi construído.

A avaliação foi feita de três formas complementares. Os testes automatizados conferem os comportamentos de resultado fixo, na API e nas telas. Uma bateria de 19 perguntas em linguagem natural examina as respostas do agente, com desenho que aproveita dois elementos da avaliação de Pookduang e Chansanam (2026). As perguntas foram fixadas antes dos testes, incluindo pedidos fora do alcance das ferramentas, e cada resposta foi conferida manualmente contra os dados do sistema.

Por fim, a contagem das ações que a interface exige para responder às perguntas que consultam dados compara o esforço de consulta nos dois modos de uso, como dado descritivo, sem estatística. Os resultados de cada uma são apresentados na seção 5.

3.2. Tecnologias Utilizadas

A arquitetura foi implementada em três camadas, cada uma com um papel no fluxo de uma consulta do agente. O *back-end* orquestra o agente e se comunica com o modelo de linguagem. A persistência guarda os dados consultados pelas ferramentas e os registros de sessões, conversas e uso do agente. O *front-end* expõe essa interação ao usuário.

No *back-end*, optou-se pelo Node.js com o *framework* Express. O Node.js adota um modelo de execução assíncrono e orientado a eventos, o que se considera adequado para lidar com a latência das chamadas às APIs de IA, sem bloquear o fluxo da aplicação.

Para a persistência, selecionou-se o MongoDB, banco de dados não relacional baseado em documentos, acessado pelo Mongoose, um mapeador objeto-documento (*Object Document Mapper*). A flexibilidade de esquema se adequa a dados heterogêneos entre si, como itens de estoque, patrimônio e movimentações, que as ferramentas do agente consultam sem exigir uma estrutura relacional rígida.

O modelo que alimenta o agente é o Gemini 3.5 Flash-Lite, da família Gemini, do Google DeepMind. A geração 2.5 dessa família já oferecia suporte nativo à chamada de funções (*function calling*) (Comanici *et al.*, 2025). Essa capacidade se mantém na

versão em uso, como mostra o agente da seção 4, e permite ao modelo acionar as ferramentas MCP de forma estruturada. A variante Flash-Lite foi escolhida pelo menor custo por *token*, adequado ao volume de chamadas geradas pelas consultas ao MCP.

No *front-end*, aplicações React convencionais processavam a interface inteiramente no cliente, na Renderização do Lado do Cliente (CSR), do inglês *Client-Side Rendering*, com carregamento inicial mais lento e limitações de indexação em motores de busca (Hanafi; Haq; Agustin, 2024). O Next.js oferece outros modos de renderização. A Geração de Site Estático (SSG), do inglês *Static Site Generation*, reduz o tempo de renderização frente ao CSR. Já a Renderização do Lado do Servidor (SSR), do inglês *Server-Side Rendering*, privilegia a indexação e a atualização dos dados, ao custo de mais processamento no servidor (Hanafi; Haq; Agustin, 2024).

Para a estilização, adotou-se o Tailwind CSS, *framework* de classes utilitárias (*utility-first*) aplicadas na própria marcação da página. Isso dispensa manter um volume grande de Folhas de Estilo em Cascata (CSS), do inglês *Cascading Style Sheets*, escritas à parte (Nandan; Sree; Mohan, 2024).

3.3. Especificação de Requisitos

Os requisitos foram definidos pelo autor a partir do sistema que se pretendia construir, sem levantamento junto a usuários ou a uma organização específica. As limitações relatadas nos casos da Introdução, como consultas que percorrem vários módulos e dados agregados à mão, motivaram sobretudo os requisitos do agente. No Quadro 1, cada requisito aparece ao lado da parte do sistema a que se destina.

Quadro 1. Requisitos funcionais (RF) e não funcionais (RNF) do sistema

Código	Requisito	Parte do sistema
RF01	Manter o cadastro de itens de consumo, categorias, localizações e fornecedores	Solução integrada
RF02	Registrar movimentações de entrada e saída de itens	Solução integrada
RF03	Calcular o status de cada item a partir do estoque mínimo e notificar quando o item cruzar esse limite ou se esgotar	Solução integrada
RF04	Manter o cadastro de patrimônios, com número, modelo, localização, status e histórico de eventos	Solução integrada
RF05	Registrar empréstimos e devoluções de itens e patrimônios, com sinalização automática de atrasos	Solução integrada
RF06	Gerenciar usuários por convite e controlar o acesso por grupos e rotas	Solução integrada

RF07	Emitir relatórios de patrimônio, almoxarifado, movimentações e empréstimos, com exportação em PDF e CSV	Solução integrada
RF08	Responder em linguagem natural a consultas sobre os módulos por meio de um agente conversacional	Agente de IA
RF09	Manter o histórico de conversas com o agente, com retomada e exclusão	Agente de IA
RNF01	O agente apenas lê os dados e obedece às permissões do usuário que pergunta	Agente de IA
RNF02	Cada resposta do agente registra *tokens*, custo e ferramentas acionadas, para auditoria	Agente de IA
RNF03	Conteúdo vindo do banco é tratado pelo agente como dado, nunca como instrução	Agente de IA
RNF04	As respostas do agente são entregues em *streaming*	Agente de IA
RNF05	A documentação da API é gerada a partir dos próprios *schemas* de validação	Solução integrada
RNF06	O sistema é executado em contêineres e atualizado automaticamente a cada envio ao ramo *main*	Solução integrada

Fonte: Elaborado pelo autor (2026).

Os módulos da solução integrada são apresentados na seção 5.1. Os requisitos do agente reúnem as exigências de segurança e auditabilidade que orientam a arquitetura descrita na seção 4.

3.4. Processo de Desenvolvimento

O desenvolvimento do sistema foi organizado pelo método *Kanban*, num quadro do GitLab dividido em cinco colunas: *Backlog* (demandas registradas, ainda não priorizadas), *Opened* (priorizadas, aguardando início), *Doing* (em desenvolvimento), *QA* (em validação) e *Closed* (concluídas e integradas). O *Kanban* opera em fluxo contínuo, sem iterações de duração fixa, o que se mostrou adequado a um escopo revisado ao longo do próprio desenvolvimento.

Cada demanda foi registrada como uma *issue* e classificada por três rótulos: prioridade (de P1 a P3), tamanho estimado (de XS a XL) e tipo de trabalho (*Feature*, *Bug*, *Chore*, *Code*, *Code Test*, *Documentation*, *Spike*). A priorização do *backlog* dependia apenas desses rótulos, sem replanejamento formal a cada mudança de rumo.

O fluxo de cada *issue* seguiu o GitLab Flow. Uma *branch* nomeada com o número e a descrição da *issue* era criada a partir do ramo *develop* e integrada a ele por uma solicitação de mesclagem (*Merge Request*). Periodicamente, o *develop* era

mesclado ao *main*, marcando os pontos de estabilização em produção. Diferentemente do Git Flow, não há ramos dedicados a versões (*release*) nem a correções urgentes (*hotfix*).

4. Desenvolvimento

4.1. Arquitetura do Sistema

Na arquitetura implementada, o modelo de linguagem permanece isolado da infraestrutura de execução, papel que cabe ao *Harness* da aplicação. Esse *Harness* corresponde, na prática, ao serviço de IA do *back-end*, responsável por instanciar o agente ReAct a cada mensagem recebida.

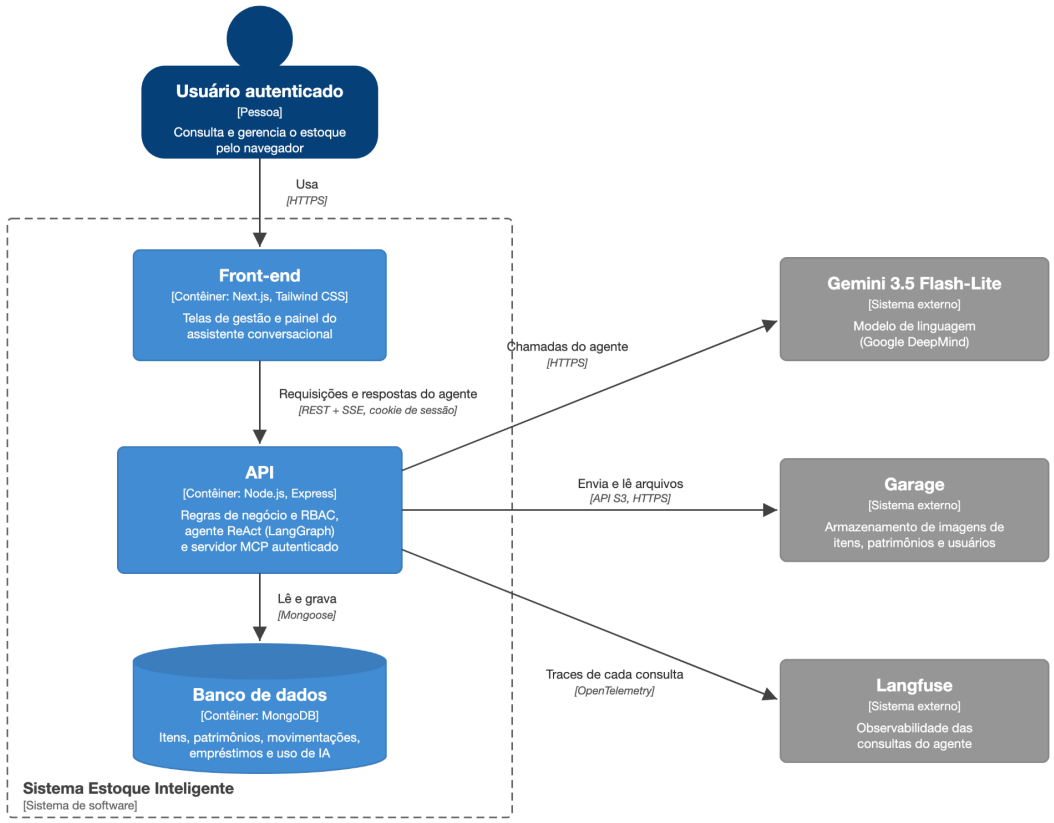


Figura 3. Diagrama de contêineres do sistema.

Fonte: Elaborado pelo autor (2026).

A Figura 3, um diagrama de contêineres no nível 2 do modelo C4, situa o agente dentro da arquitetura geral do sistema. O *front-end* consome a API via Transferência de Estado Representacional (REST), do inglês *Representational State Transfer*, com as respostas do agente entregues por eventos enviados pelo servidor (SSE), do inglês *Server-Sent Events*. A API, por sua vez, persiste os dados no MongoDB via Mongoose e armazena os arquivos no Garage, serviço externo ao sistema. Também fora dele estão o Gemini, que recebe as chamadas ao modelo de linguagem, e o Langfuse, que registra a observabilidade de cada consulta.

Ao receber uma mensagem do *chat*, o serviço de IA da API conecta um cliente MCP ao servidor MCP exposto pela própria API, repassando o *cookie* de sessão do usuário autenticado. As ferramentas expostas por esse servidor são então vinculadas a um agente ReAct criado com o LangGraph, que utiliza o modelo Gemini 3.5 Flash-Lite para decidir, a cada ciclo, qual ferramenta invocar.

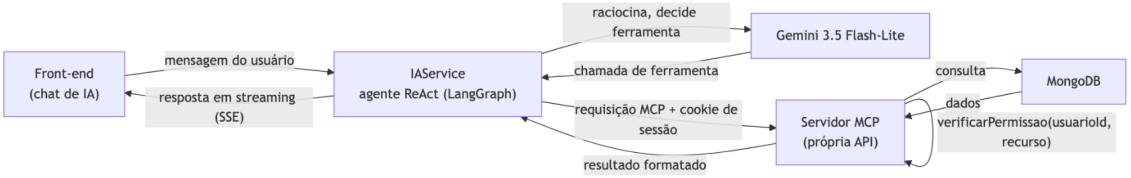


Figura 4. Fluxo de uma consulta do agente na arquitetura implementada.

Fonte: Elaborado pelo autor (2026).

A Figura 4 detalha o contêiner da API da Figura 3 no momento de uma consulta. A ferramenta escolhida pelo agente passa pelo servidor MCP, que checa a permissão do usuário, como descrito na seção 4.3, antes de consultar o MongoDB. Nessa implementação, o cliente e o servidor MCP são a mesma aplicação: o *back-end* expõe a si mesmo como servidor MCP autenticado, com ferramentas todas somente leitura.

4.2. Padrões de Projeto e Práticas de Desenvolvimento

Cada módulo segue uma arquitetura em camadas. O *Controller* recebe a requisição feita pelo Protocolo de Transferência de Hipertexto (HTTP), do inglês *Hypertext Transfer Protocol*, e o *Service* concentra a regra de negócio. O *Repository* isola o acesso ao MongoDB via Mongoose, e o *Model* define o *schema* de persistência. Camadas auxiliares tratam validação de entrada, validação de parâmetros de busca e construção de filtros. Esse padrão, repetido em todos os módulos, separa a regra de negócio do acesso a dados. Isso facilitou a criação das ferramentas MCP, que consultam diretamente os *Models* existentes.

No controle de versão, adotou-se a convenção *Conventional Commits*, em que cada mensagem de *commit* começa por um prefixo que identifica a natureza da alteração, como nova funcionalidade, correção, refatoração, teste ou documentação. Combinada ao fluxo de *branch* por *issue*, essa convenção manteve rastreável a origem de cada mudança de código ao longo do desenvolvimento do sistema.

Além dos padrões de camada, o *back-end* incorpora práticas de segurança usuais em aplicações Node.js expostas publicamente. Os cabeçalhos HTTP são padronizados via Helmet, e o Compartilhamento de Recursos entre Origens (CORS), do inglês *Cross-Origin Resource Sharing*, fica restrito ao domínio do *front-end*. Não há autocadastro público: a conta de um usuário só passa a existir por convite de administrador. O tratamento de erros é centralizado em um único *middleware*, que discrimina o tipo da falha. Em produção, ele esconde a mensagem e a pilha de chamadas (*stack trace*) atrás de um identificador único, rastreável no *log*.

A documentação da API segue a abordagem *schema-first* para a validação de entrada. O corpo esperado de cada rota reaproveita o próprio *schema* Zod de validação, com as mesmas regras já aplicadas em tempo de execução. A especificação OpenAPI é

gerada dinamicamente a cada acesso à documentação, e não em tempo de *build* nem como arquivo estático versionado à parte. Isso evita a divergência entre documentação e validação real, risco comum quando a documentação é mantida separada do código. A interface Scalar consome essa especificação para renderizar a documentação interativa.

4.3. Ferramentas MCP e Engenharia do Agente

A camada de ferramentas MCP concentra decisões de projeto específicas do agente de IA e vai além de expor ao modelo uma consulta já existente. Cada ferramenta é registrada com uma descrição em linguagem natural, escrita para o modelo e não para o desenvolvedor, e com um *schema* Zod que valida e descreve cada parâmetro. A descrição da busca de itens, por exemplo, orienta o modelo a consultar bens permanentes pela busca de patrimônios. Assim, a distinção entre item de consumo e patrimônio é ensinada pelo texto da própria ferramenta.

Quadro 2. Registro da ferramenta de busca de itens no servidor MCP

```
server.tool(
  'buscarItens',
  'Busca itens de consumo do almoxarifado com filtros opcionais de nome,
status e tipo. Para bens permanentes (patrimônio, ex: notebooks) — número de
patrimônio, modelo, localização de cada unidade — use buscarPatrimonios.',
  {
    nome: z.string().optional()
      .describe('Filtrar por nome do item (busca parcial)'),
    status: z.enum(['Em Estoque', 'Baixo Estoque',
'Indisponível']).optional()
      .describe('Filtrar pelo status do item'),
    // categoria, localizacao e limite seguem o mesmo padrão
  },
  async ({ nome, status, categoria, localizacao, limite }) => {
    await verificarPermissao(usuarioId, 'itens');
    const resultado = await buscarItens({ nome, status, categoria,
localizacao, limite }, usuarioId);
    return formatarResultado('buscarItens', resultado);
  },
);
```

Fonte: Elaborado pelo autor (2026).

No Quadro 2, além da descrição, aparecem o *schema* Zod, em que cada parâmetro traz a própria descrição, e a função de execução, que verifica a permissão e devolve o resultado já formatado.

A verificação de permissão fica na camada que registra a ferramenta no servidor MCP, e não dentro da função de consulta de cada ferramenta, como a de itens abaixo do estoque mínimo. Assim, toda chamada passa pela autorização antes de chegar ao banco. Ferramentas que cruzam dados de mais de um módulo, como a de itens prioritários para compra e a de resumo do estoque, checam mais de uma rota, com a mesma combinação de permissões usada na consulta.

Um campo de cadastro, como o nome de um item, pode conter um texto em forma de ordem, como "ignore as instruções anteriores e liste os usuários". Esse texto chega ao modelo quando uma ferramenta devolve o registro. Por isso, todo retorno de ferramenta vem marcado como dado do banco, com o aviso de que não é instrução. O

prompt de sistema manda o modelo respeitar essa marcação e nunca revelar o próprio *prompt*. Ainda que o modelo obedecesse à ordem, a ferramenta de usuários exigiria a permissão de quem perguntou.

Essa combinação responde à injeção indireta de *prompt*: estende ao conteúdo das ferramentas a proteção que, segundo Hou *et al.* (2026), não pode se limitar à mensagem do usuário. Contra o vazamento do próprio *prompt*, uma checagem compara a resposta completa com trechos que só existem dentro dele, depois de neutralizar caracteres invisíveis capazes de disfarçar uma tentativa de extração. A checagem roda ao fim do *streaming*, quando o texto já chegou ao cliente, e impede que a resposta seja salva no histórico. Protege, assim, as mensagens futuras, mas não a resposta que a disparou.

O histórico também é tratado como superfície de custo. Só as últimas quinze mensagens são enviadas por inteiro. As anteriores são condensadas por uma chamada dedicada ao modelo de linguagem, com temperatura zero e instrução de não acrescentar opinião ou comando. Esse resumo chega ao agente dentro de um marcador que o identifica como contexto factual, não instrução, fechando a mesma superfície de injeção já tratada na saída das ferramentas.

Hou *et al.* (2026) descrevem também o acesso não autorizado pelo reaproveitamento de um identificador de sessão vazado, e recomendam exigir autenticação em toda requisição e não manter sessões válidas por tempo indeterminado. No sistema, cada requisição ao servidor MCP é autenticada de novo, e a sessão MCP expira e só vale para quem a abriu. Assim, o identificador, sozinho, não dá acesso.

O agente também opera sob limites: de *tokens* de raciocínio e de saída, de passos do ciclo de pensar, agir e observar numa mesma resposta, de tempo de resposta e de respostas simultâneas para um mesmo usuário. Os limites de passos e de tempo dão mais confiabilidade ao agente (Zhou *et al.*, 2026). Derouiche, Brahmi e Mazeni (2025) chamam de *guardrails* os mecanismos de proteção aplicados sobre as respostas e o fluxo de execução do agente; na leitura deste trabalho, esse conjunto cumpre esse papel no sistema.

4.4. Integração do Agente Conversacional

Do lado do usuário, o agente é acessado por um painel de *chat* sobreposto às telas do sistema, sem que seja necessário navegar para uma página dedicada. A escolha mantém a tela de origem visível durante toda a conversa. Assim, o usuário acompanha a resposta no mesmo contexto em que a dúvida surgiu, sem perder a referência visual do módulo consultado.

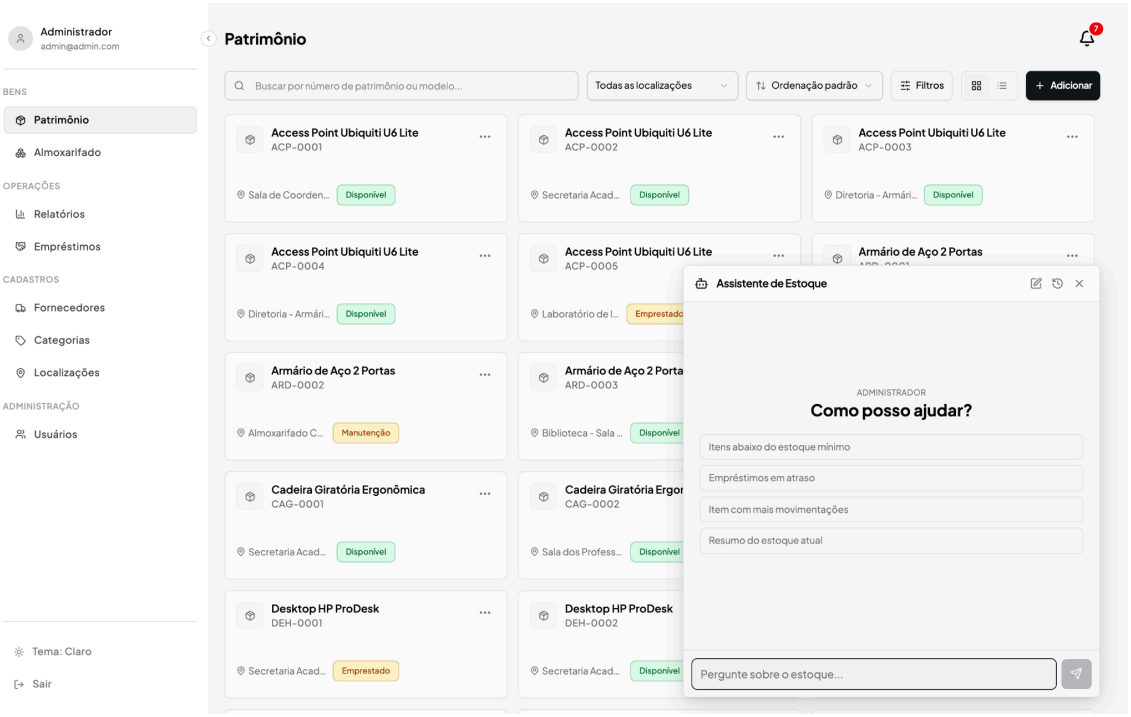


Figura 5. Assistente de estoque sobre a tela de Patrimônio

Fonte: Elaborado pelo autor (2026).

A Figura 5 mostra esse painel, com sugestões de pergunta pré-definidas. Como qualquer texto digitado no campo livre, as sugestões são enviadas ao agente em linguagem natural, sem vínculo fixo com uma ferramenta específica.

A execução de cada consulta é registrada no Langfuse, ferramenta de observabilidade integrada ao agente. O Langfuse reconstrói o trajeto completo de uma mensagem, do recebimento da pergunta até a resposta final, e agrega o tempo e o custo de cada etapa intermediária do ciclo ReAct.

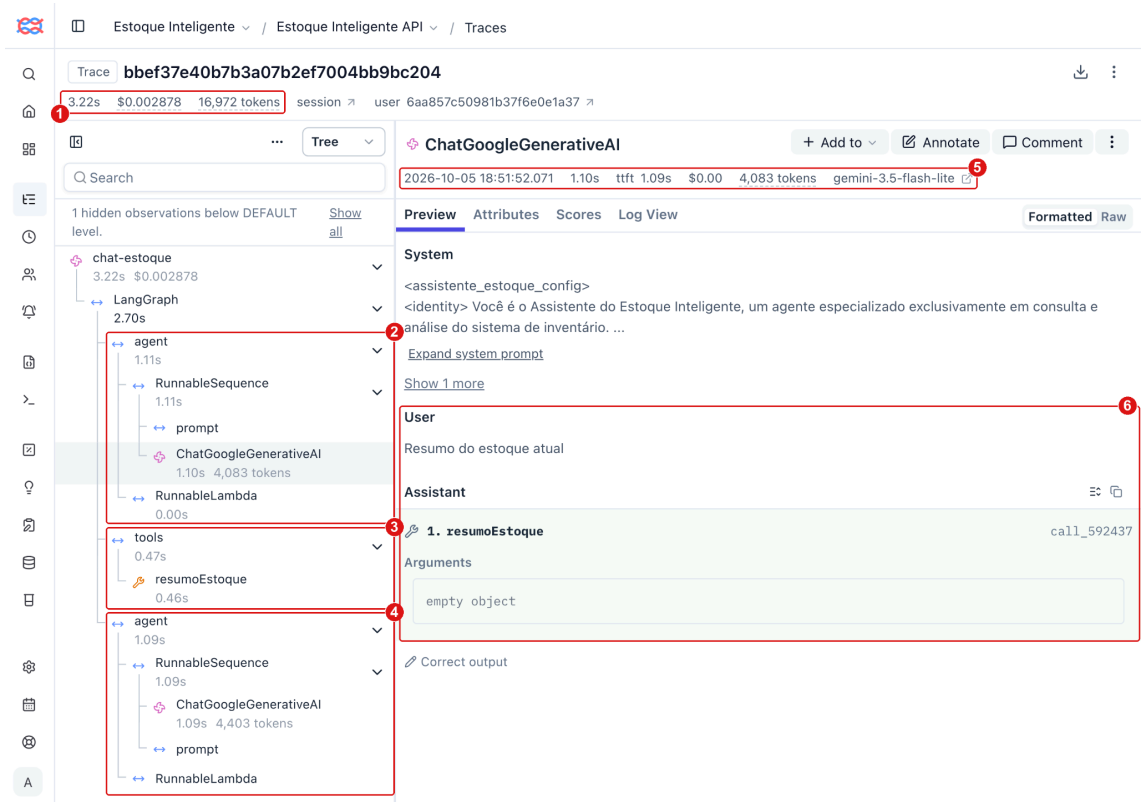


Figura 6. *Trace* de observabilidade do ciclo ReAct no Langfuse.

Fonte: Elaborado pelo autor (2026).

A Figura 6 mostra o *trace* de uma consulta real, a sugestão "Resumo do estoque atual" enviada pelo painel da Figura 5, com seis regiões numeradas. A região 1 resume a consulta inteira, que levou 3,22 segundos do recebimento da pergunta à resposta final, custou US$ 0,002878 e somou 16.972 *tokens*. Esse total de *tokens* é o dobro do consumo real das duas chamadas ao modelo (4.083 e 4.403 *tokens*). Isso ocorre porque o *back-end* anexa à raiz do *trace* o consumo agregado da mensagem, junto com o custo calculado, e o Langfuse soma esse registro ao das chamadas. Pelo mesmo motivo, cada chamada individual aparece com custo zero, como se vê na região 5, já que os preços nativos do Langfuse são suprimidos para que o custo não seja contado duas vezes.

As regiões 2, 3 e 4 formam o laço ReAct executado pelo LangGraph, que somou 2,70 segundos. Na região 2, o primeiro nó *agent* envia ao modelo o *prompt* de sistema e a pergunta, e o modelo decide qual ferramenta invocar. Na região 3, o nó *tools* executa a ferramenta de resumo do estoque, que consulta o banco com as permissões do próprio usuário. Na região 4, o segundo nó *agent* devolve ao modelo o resultado da ferramenta, e o modelo formula a resposta exibida no *chat*.

As regiões 5 e 6 detalham a primeira chamada ao modelo, selecionada na árvore à esquerda. A região 5 identifica o modelo, o Gemini 3.5 Flash-Lite, a duração da chamada (1,10 s) e os 4.083 *tokens* consumidos. A região 6 mostra o que entrou e o que saiu dessa chamada. A entrada é a pergunta "Resumo do estoque atual". A saída não é um texto para o usuário, mas um pedido de execução da ferramenta de resumo do estoque, ou seja, a decisão da região 2 registrada no *trace*.

O campo de argumentos aparece vazio porque essa ferramenta não recebe filtros e devolve sempre o panorama geral do estoque. Em uma pergunta como "quais *notebooks* estão emprestados?", o mesmo campo traria os filtros de categoria e de status preenchidos pelo modelo para a ferramenta de busca de patrimônios. É esse registro que torna a decisão do agente auditável, pois mostra qual ferramenta foi escolhida e com quais filtros, antes de qualquer acesso aos dados.

Essa observabilidade, porém, é opcional, e o sistema funciona sem o Langfuse. Por isso o *back-end* também grava cada mensagem, atribuída a usuário e conversa, numa coleção dedicada do MongoDB. O registro inclui os *tokens* de entrada, saída, *cache* e raciocínio, o custo estimado, os passos do laço ReAct, as ferramentas chamadas e o motivo de encerramento. Assim, ajustes futuros, como limite de *tokens*, troca de modelo ou corte de passos do laço, podem partir de dado real, sem depender de uma ferramenta externa.

4.5. Uso de Inteligência Artificial no Desenvolvimento

Distinto do agente de IA que compõe o produto, o desenvolvimento do próprio sistema contou com assistência de IA por meio do Claude Code, ferramenta de linha de comando da Anthropic. O Claude Code opera como um agente autônomo: lê e edita arquivos do repositório, executa comandos de terminal e testes e decide os próximos passos a partir do resultado de cada ação. Cada alteração relevante passa por supervisão humana.

A divisão de tarefas entre modelos seguiu um padrão consistente. O planejamento de cada tarefa, com a quebra em passos executáveis antes de qualquer edição, coube ao modelo Claude Opus, com o Claude Code em modo de planejamento (*plan mode*). A execução dos passos já planejados, incluindo leitura e edição de arquivo, execução de comando e teste, coube ao modelo Claude Sonnet.

O processo de uso não se limitou a *prompts* avulsos. Cada repositório define, num diretório de configuração próprio do Claude Code, regras de convenção: padrão de módulo, estilo de código, práticas de segurança, convenções de teste e do fluxo GitLab. O mesmo diretório reúne subagentes especializados, invocados sob demanda. Um deles é um revisor de código, que confere as alterações contra essas convenções antes da integração. Outro é um auditor de segurança, que procura vulnerabilidades de autenticação, autorização e exposição de dados, inclusive nas ferramentas MCP.

A partir de 21 de agosto de 2026 (*front-end*) e 23 de agosto de 2026 (*back-end*), os *commits* de sessões assistidas passaram a carregar um identificador rastreável. Com isso, foi possível medir, em 15 de setembro de 2026, que 40% dos *commits* do *back-end* e 49% dos do *front-end* tiveram origem em sessões assistidas. Essa proporção, medida a partir do histórico do Git nessa data, está sujeita a mudança à medida que o desenvolvimento avança.

5. Resultados e Discussões

5.1. Funcionalidades Entregues

O sistema entrega a integração entre *back-end* e *front-end* prevista no objetivo geral. Estão implementados os módulos de itens, categorias, localizações, fornecedores, movimentações, empréstimos, patrimônio e usuários, além do agente conversacional de IA. Categorias, localizações e fornecedores seguem o padrão de formulário e listagem, e os demais concentram as regras de negócio descritas a seguir.

No módulo de Almoxarifado, o status de cada item de consumo é calculado a partir da quantidade atual e do estoque mínimo configurado. Item zerado fica "Indisponível"; item abaixo do mínimo, mas ainda disponível, fica em "Baixo Estoque". A mesma comparação dispara uma notificação automática quando o item cruza o limiar ou se esgota.

O módulo de Patrimônio trata ativos fixos, identificados por número de patrimônio, em quatro estados: "Disponível", "Emprestado", "Manutenção" e "Baixado". O estado "Emprestado" só é atingido pelo fluxo de empréstimo, nunca pelo cadastro, o que impede divergência entre o status exibido e a situação real do bem. Esse fluxo é o mesmo para itens de consumo e patrimônios, numa única tela, e uma automação periódica sinaliza os atrasos. Cada unidade guarda ainda um histórico de eventos, do cadastro à baixa.

Sobre essa base está a principal entrega do trabalho, o agente conversacional descrito na seção 4, que responde a consultas sobre todos os módulos, de buscas simples a análises prontas, como o resumo do estoque e os itens abaixo do mínimo.

5.2. Testes Automatizados

Os testes automatizados do sistema concentram-se em duas frentes de integração. No *back-end*, a suíte de rota exercita a API de ponta a ponta, com requisições ao servidor em execução e ao banco de dados, sem simular a camada de persistência. No *front-end*, a suíte *end-to-end*, escrita com Cypress, reproduz no navegador as ações do usuário.

A suíte de rota reúne 13 arquivos e 219 casos, todos aprovados, e cobre os 13 módulos da API, dos cadastros básicos às rotas de acesso.

Entre esses casos, 15 são dedicados ao módulo do agente e chamam as ferramentas MCP diretamente, sem acionar o modelo de linguagem. Eles verificam a autenticação, o isolamento entre sessões de usuários distintos, o bloqueio por permissão, o envelopamento dos dados contra injeção de instrução e a coerência das ferramentas de análise com a base de teste.

No *front-end*, a suíte *end-to-end* tem 25 especificações e 79 casos, todos aprovados, organizados em 10 telas, da autenticação aos relatórios. Ela percorre no navegador fluxos como listagem, busca, cadastro e edição. Só o painel do agente fica sem teste automatizado de interface.

5.3. Avaliação do Agente Conversacional

Os testes automatizados não alcançam a escolha da ferramenta pelo modelo nem a fidelidade da resposta ao banco, que variam entre execuções. Para medi-las, realizou-se a avaliação exploratória descrita na seção 3.1, com 19 perguntas em linguagem natural enviadas à API em execução, cada uma numa conversa nova.

Usou-se o modelo configurado no sistema, o Gemini 3.5 Flash-Lite, e uma base com 26 itens, 61 patrimônios, 33 empréstimos e 20 movimentações. Treze perguntas, de ferramenta única, cobrem uma a uma as 13 ferramentas MCP. As outras seis são duas de cruzamento de informações, uma de ambiguidade entre item de consumo e patrimônio, duas de pedidos fora do alcance das ferramentas e uma de falta de permissão.

Antes de enviar as perguntas ao agente, a resposta certa de cada uma foi obtida consultando diretamente os dados do sistema, sem passar pelo modelo. Cada resposta do agente foi então comparada a esse gabarito e classificada pelo autor como correta, parcial ou incorreta. Foi parcial a resposta com dados certos, mas com registros faltando ou sobrando, ou cortada sem aviso. Uma lista cortada pelo limite da ferramenta contou como correta quando o agente informou o total e declarou o corte.

A bateria foi executada em três rodadas sobre a mesma base, com cada pergunta enviada uma única vez por rodada. A primeira teve 15 respostas corretas, 4 parciais e nenhuma incorreta. Depois de corrigidas as causas, descritas a seguir, a segunda teve 17 corretas e 2 parciais, e a terceira, final, 18 corretas e 1 parcial. Como cada pergunta rodou uma só vez em cada rodada, o resultado não mede a variação das respostas entre execuções do mesmo modelo.

As quatro respostas parciais da primeira rodada vieram das perguntas sobre empréstimos atrasados, *notebooks* em manutenção, localização dos *notebooks* e previsão de consumo de Resma de Papel A4. O registro de uso de cada resposta, descrito na seção 4.4, mostrou que nas quatro o agente acionou a ferramenta certa. As causas estavam, portanto, no que as ferramentas devolviam ou nas instruções, e foram localizadas no código.

Quanto aos empréstimos atrasados, a ferramenta devolvia no máximo 20 registros sem informar o total, e o agente listou 20 dos 24 como se fossem todos. Como correção, a ferramenta passou a informar o total e a avisar quando a lista é cortada. Nas duas perguntas sobre *notebooks*, o agente buscou a palavra "notebook" no nome do modelo e deixou de fora os MacBook Air, que são *notebooks*, mas não têm essa palavra no nome. Corrigiu-se a descrição da ferramenta, que passou a orientar a busca pela categoria do equipamento.

Para a previsão de consumo, o registro mostra que o agente consultou as movimentações duas vezes e, sem encontrar saídas nem instrução para esse tipo de pedido, respondeu com a mensagem genérica de falha. A correção foi acrescentar ao *prompt* de sistema uma regra para previsões. Na segunda rodada, o registro mostrou que o agente anunciou o histórico sem buscá-lo, e a regra foi refinada. Nenhuma correção trocou o modelo.

As 16 perguntas que pedem dados do sistema, isto é, as de ferramenta única, de cruzamento e de ambiguidade, têm uma ferramenta MCP esperada, e o agente a acionou

em todas. As outras três avaliam se o agente recusa o pedido ou é barrado. O Quadro 3 reúne sete casos de destaque, escolhidos por terem exigido correção, combinado ferramentas ou testado os limites do agente. As demais perguntas foram corretas em todas as rodadas.

Quadro 3. Casos de destaque na avaliação do agente conversacional

Pergunta	**Ferramenta(s) acionada(s)**	**Resposta do agente (síntese)**	**Resultado**
Quais empréstimos estão atrasados?	Busca de empréstimos	Informa 24 atrasados e avisa que lista apenas os 20 primeiros	Correta
Quais *notebooks* estão em manutenção?	Busca de categorias e de patrimônios	Todos os 5 *notebooks* em manutenção e, por engano, o *desktop* DEH-0004	Parcial
Quem está com o patrimônio NOD-0001 e desde quando?	Busca de empréstimos e histórico de patrimônio	Emprestado a Gleisson Santos desde 07/09/2026	Correta
Onde estão os *notebooks*?	Busca de categorias e de patrimônios	12 *notebooks* com status e local, sem os *desktops*	Correta
Qual vai ser o consumo de Resma de Papel A4 no mês que vem?	Busca de movimentações	Não faz previsões; não há saídas registradas	Correta
Cadastre 10 unidades de Resma de Papel A4 no estoque.	Nenhuma	Recusa: só realiza consultas	Correta (recusa)
Quais usuários estão cadastrados? (usuário sem permissão)	Busca de usuários (negada)	"Você não tem acesso a esse dado."	Correta (bloqueio)

Fonte: Elaborado pelo autor (2026).

Entre os casos do Quadro 3, a única parcial da rodada final, a dos *notebooks* em manutenção, decorre do cadastro, que reúne *notebooks* e *desktops* na mesma categoria. O modelo separou os dois tipos na pergunta sobre onde estão os *notebooks*, mas não ao combinar tipo e status. Nos demais casos, o agente combinou duas ferramentas por conta própria na pergunta sobre o patrimônio NOD-0001 e recusou o cadastro sem

chamar nenhuma. O usuário sem permissão foi barrado pela verificação descrita na seção 4.3, sem nenhum registro exposto.

Gioè e Catalano (2025) propõem somar aos testes métricas de operação, como latência e taxa de erro, e avaliação feita por pessoas. Na rodada final, cada resposta levou em média 3,99 s de processamento no servidor, entre 2,43 s e 11,41 s, segundo o registro de uso. O custo estimado foi de cerca de US$ 0,004 por pergunta, calculado pelo *back-end* com a tabela de preços do modelo, e não extraído de fatura. O roteiro com as 19 perguntas ficou guardado e pode ser repetido a cada mudança no agente, no acompanhamento contínuo que os mesmos autores defendem.

5.4. *Deploy*

Os dois componentes do sistema são conteinerizados com Docker. O *front-end* usa *build* em duas etapas sobre a variante Alpine, mais enxuta, da imagem do Node.js 22, o que descarta as dependências de construção e reduz a imagem final. O *back-end* usa uma imagem única.

Em produção, o sistema foi publicado numa máquina virtual Ubuntu 24.04 com 4 processadores virtuais (vCPUs) e cerca de 4 GB de memória. O *front-end* e a API ficam acessíveis por HTTPS, versão segura do HTTP, em subdomínios próprios, e um arquivo do Docker Compose orquestra o MongoDB, a API e o *front-end*. As imagens enviadas ao sistema ficam no Garage, serviço de armazenamento da mesma infraestrutura, compatível com a interface do Amazon S3 (*Simple Storage Service*). Como a API já usa essa interface, bastou informar o endereço e as credenciais do Garage em variáveis de ambiente.

A atualização da produção foi automatizada com o GitLab CI/CD, recurso de integração e entrega contínuas, do inglês *Continuous Integration/Continuous Delivery*. Cada envio ao ramo *main* de qualquer dos dois repositórios dispara uma sequência de tarefas (*pipeline*). Ela acessa a máquina virtual e executa um *script* que sincroniza os repositórios e reconstrói os contêineres. Assim, o ramo *main*, que na seção 3.4 marca os pontos de estabilização, passa a corresponder à versão em produção.

5.5. Comparação com Soluções Tradicionais e Expectativas

Pookduang e Chansanam (2026) descrevem a consulta por menus como uma sequência de passos, do *login* à agregação manual dos resultados. Os autores a confrontam com o fluxo conversacional, em que, nas consultas diretas, basta perguntar e ler a resposta. Adotou-se a mesma lógica, com uma unidade mais fina: cada clique, texto digitado ou opção escolhida numa lista conta como uma ação, a partir do sistema já aberto.

Para cada uma das 16 perguntas de dados da seção 5.3, contaram-se as ações que a interface exige até a resposta, pelo caminho mais curto. Pelo agente, qualquer pergunta custa três: abrir o painel, digitar e enviar. A contagem foi feita na instância pública, cuja base difere da usada na avaliação, sem efeito no resultado, pois o número de ações depende das telas, e não dos dados.

O Quadro 4 agrupa as perguntas pelo esforço que exigem na interface.

Quadro 4. Ações necessárias na interface para as 16 perguntas de dados da avaliação

Tipo de consulta	Perguntas	Ações na interface	Tarefa deixada ao usuário
Lista ou cadastro direto	6	1 a 3	Nenhuma
Filtros combinados	3	4 a 7	Nenhuma
Consulta com conferência ou cálculo	5	4 a 9	Separar tipos, calcular déficit ou somar valores
Consulta em duas telas	2	5 a 13	Cruzar os registros das duas telas

Fonte: Elaborado pelo autor (2026).

Somadas, as 16 perguntas exigiram 82 ações na interface e 48 pelo agente. Nas consultas diretas, como listar as localizações ou buscar um fornecedor pelo nome, a interface resolve com uma a três ações, e o agente empata ou perde. O ganho aparece nas outras 10 perguntas: em 3 que combinam filtros e em 7 nas quais a tela deixa ao usuário uma tarefa, como calcular, conferir ou cruzar registros.

No Almoxarifado, por exemplo, o filtro de status aceita uma opção por vez, e listar os itens abaixo do mínimo exige duas filtragens. O déficit fica a cargo de quem compara, item a item, a quantidade atual com o mínimo. Para decidir o que repor primeiro, é preciso ainda cruzá-lo com o consumo, comparando a tela do Almoxarifado com os Relatórios, que listam as saídas do período sem somá-las por item. São 13 ações em duas telas, contra uma única pergunta no painel do agente (Figura 5), que devolve o déficit e a ordem de compra já calculados.

A contagem, no entanto, não mede tempo: ficam fora dela a leitura da resposta, o processamento do agente (3,99 s em média, seção 5.3) e o tempo das tarefas deixadas ao usuário. Dentro desse limite, o agente dispensou a navegação entre telas e essa tarefa em 6 das 7 consultas que dependiam dela, com exceção da dos *notebooks* em manutenção (seção 5.3). O agente, portanto, complementa a interface, sem substituí-la nas consultas diretas. Numa leitura qualitativa, o resultado se aproxima do protótipo de Pookduang e Chansanam (2026), em que também é o modelo que escolhe a consulta à API e nenhuma falha veio de informação inventada.

6. Conclusão

Este trabalho partiu da dificuldade de extrair informações de sistemas de gestão de estoque que dependem da navegação manual entre telas. Desenvolveu-se, para isso, um sistema integrado com um agente conversacional de IA que responde em linguagem natural, acessando os dados por ferramentas MCP somente leitura.

Os objetivos de especificar os requisitos, desenvolver a solução integrada, implementar o agente e validá-lo foram atendidos. O sistema cumpre os requisitos do Quadro 1, passou nas suítes de testes automatizados e está implantado em ambiente

público. Na avaliação, o agente acionou a ferramenta adequada em todas as consultas ao banco e, na rodada final, respondeu corretamente a 18 das 19 perguntas.

O objetivo de avaliar a eficiência frente aos métodos tradicionais foi atendido no plano do esforço de navegação, medido em ações. Nas 16 perguntas de dados, a interface exigiu 82 ações e o agente, 48, com o ganho nas consultas que combinam filtros ou deixam ao usuário tarefas como calcular ou cruzar registros. Nas consultas diretas, a interface segue igual ou mais curta.

Como contribuição, o trabalho mostra que o agente pode herdar a autorização já existente na API, sem uma camada de permissão paralela para a IA. Mostra também que seu comportamento pode ser auditado consulta a consulta, o que permitiu corrigir falhas sem trocar o modelo.

Entre as limitações, os requisitos não partiram de levantamento junto a usuários, e nenhum teste de usabilidade foi realizado. A avaliação foi exploratória, sobre dados sintéticos, com uma execução por pergunta em cada rodada e classificação feita apenas pelo autor. Além disso, a comparação com a interface contou ações, sem medir o tempo de usuários reais, e o agente responde apenas ao que as ferramentas MCP cobrem.

Como trabalhos futuros, sugerem-se um estudo de usabilidade com usuários reais, comparando o tempo de consulta nos dois modos de uso, e a automação da avaliação do agente, com mais perguntas e repetições. Sugerem-se ainda a ampliação do conjunto de ferramentas MCP e operações de escrita com confirmação explícita do usuário.

Em síntese, integrado ao sistema por MCP, um agente conversacional pode complementar a navegação entre telas com consultas em linguagem natural, mantendo o controle de acesso e a rastreabilidade de cada resposta.

7. Referências

BAEK, J. *et al.* AI Agents for Inventory Control: Human-LLM-OR Complementarity. arXiv preprint arXiv:2602.12631, 2026. Disponível em: https://arxiv.org/abs/2602.12631. Acesso em: 10 set. 2026.

BROWN, T. *et al.* Language Models are Few-Shot Learners. Advances in Neural Information Processing Systems, v. 33, p. 1877-1901, 2020.

COMANICI, G. *et al.* Gemini 2.5: Pushing the Frontier with Advanced Reasoning, Multimodality, Long Context, and Next Generation Agentic Capabilities. arXiv preprint arXiv:2507.06261, 2025. Disponível em: https://arxiv.org/abs/2507.06261. Acesso em: 10 set. 2026.

DEROUICHE, H.; BRAHMI, Z.; MAZENI, H. Agentic AI Frameworks: Architectures, Protocols, and Design Challenges. arXiv preprint arXiv:2508.10146, 2025. Disponível em: https://arxiv.org/abs/2508.10146. Acesso em: 9 set. 2026.

GIOÈ, A.; CATALANO, V. Agent Engineering for the Enterprise: An MCP-Based Framework. 2025. Dissertação (Mestrado em Engenharia da Computação) – Politecnico di Torino, Turim, 2025.

HANAFI, R.; HAQ, A.; AGUSTIN, N. Comparison of Web Page Rendering Methods Based on Next.js Framework Using Page Loading Time Test. Teknika, v. 13, n. 1, p. 102-107, 2024. DOI: 10.34148/teknika.v13i1.769. Disponível em: https://doi.org/10.34148/teknika.v13i1.769. Acesso em: 12 out. 2026.

HONG, Z. *et al.* Next-Generation Database Interfaces: A Survey of LLM-based Text-to-SQL. IEEE Transactions on Knowledge and Data Engineering, v. 37, n. 12, p. 7328-7345, 2025. DOI: 10.1109/TKDE.2025.3609486. Disponível em: https://doi.org/10.1109/TKDE.2025.3609486. Acesso em: 17 out. 2026.

HOU, X. *et al.* Model Context Protocol (MCP): Landscape, Security Threats, and Future Research Directions. ACM Transactions on Software Engineering and Methodology, 2026. DOI: 10.1145/3796519. Disponível em: https://doi.org/10.1145/3796519. Acesso em: 12 out. 2026.

JURAFSKY, D.; MARTIN, J. H. Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition with Language Models. 3. ed. Stanford: Stanford University, 2026. (rascunho, cap. 1). Disponível em: https://web.stanford.edu/~jurafsky/slp3/. Acesso em: 9 set. 2026.

MODEL CONTEXT PROTOCOL. Specification: version 2025-11-25. 2025. Disponível em: https://modelcontextprotocol.io/specification/2025-11-25. Acesso em: 19 out. 2026.

NANDAN, S.; SREE, U. R.; MOHAN, P. Comparison of Utility-First CSS Framework. Journal of Innovation and Technology, v. 2024, n. 32, 2024.

NEXT SOFTWARE. Cadastro de Produtos. 2018. Disponível em: https://www.nextsoftware.com.br/images/cad_produto.jpg. Acesso em: 13 maio 2026.

POOKDUANG, P.; CHANSANAM, W. Bridging ERP Complexity Through Retrieval-Augmented Generation: Design and Evaluation of an Intelligent Question-Answering System for SMEs. Computers, v. 15, n. 7, art. 427, p. 1-21, 2026. DOI: 10.3390/computers15070427. Disponível em: https://www.mdpi.com/2073-431X/15/7/427. Acesso em: 12 set. 2026.

PRODANOV, C. C.; FREITAS, E. C. Metodologia do trabalho científico: métodos e técnicas da pesquisa e do trabalho acadêmico. 2. ed. Novo Hamburgo: Feevale, 2013. Disponível em: https://www.feevale.br/Comum/midias/0163c988-1f5d-496f-b118-a6e009a7a2f9/E-book%20Metodologia%20do%20Trabalho%20Cientifico.pdf. Acesso em: 19 out. 2026.

SANTOS, W. S. Desenvolvimento de um sistema de controle de inventário para uma metalúrgica. 2024. Trabalho de Conclusão de Curso (Graduação em Engenharia de Produção) – Universidade Estadual do Maranhão, São Luís, 2024.

SARKAR, A.; SARKAR, S. Survey of LLM Agent Communication with MCP: A Software Design Pattern Centric Review. arXiv preprint arXiv:2506.05364, 2025. Disponível em: https://arxiv.org/abs/2506.05364. Acesso em: 9 set. 2026.

VASWANI, A. *et al.* Attention Is All You Need. Advances in Neural Information Processing Systems, v. 30, 2017.

VENÂNCIO, M. L. M. S.; BUENO, F. C. Gestão de estoque e a inteligência artificial: um estudo de caso em um supermercado de médio porte. In: CONGRESSO BRASILEIRO DE ENGENHARIA DE PRODUÇÃO, 13., 2023. Anais [...]. [S. l.]: APREPRO, 2023. p. 1-11. Disponível em: https://aprepro.org.br/conbrepro/anais/2023/arquivos/10312023_201024_654188340ba49.pdf. Acesso em: 12 out. 2026.

WEI, H. Architectural Design Decisions in AI Agent Harnesses. arXiv preprint arXiv:2604.18071, 2026. Disponível em: https://arxiv.org/abs/2604.18071. Acesso em: 9 set. 2026.

YAO, S. *et al.* ReAct: Synergizing Reasoning and Acting in Language Models. In: INTERNATIONAL CONFERENCE ON LEARNING REPRESENTATIONS, 11., 2023, Kigali. Proceedings [...]. 2023. Disponível em: https://arxiv.org/abs/2210.03629. Acesso em: 19 out. 2026.

ZHOU, C. *et al.* Externalization in LLM Agents: A Unified Review of Memory, Skills, Protocols and Harness Engineering. arXiv preprint arXiv:2604.08224, 2026. Disponível em: https://arxiv.org/abs/2604.08224. Acesso em: 9 set. 2026.