Arquitetura de Referência para Apoiar o Desenvolvimento de Chatbots

LEANDRO SILVA, Universidade de São Paulo (USP), Brasil
PEDRO HENRIQUE DIAS VALLE, Universidade de São Paulo (USP), Brasil
FABIO KON, Universidade de São Paulo (USP), Brasil

ACM Reference Format:
Leandro Silva, Pedro Henrique Dias Valle, and Fabio Kon. 2025. Arquitetura de Referência para Apoiar o Desenvolvimento de Chatbots. 1, 1 (June 2025), 9 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Projeto da Arquitetura de Referência

Esta pesquisa busca aprimorar a qualidade dos chatbots disponibilizados aos usuários finais. Para isso, está sendo desenvolvida uma arquitetura de referência que contribua, em nível arquitetural, para o desenvolvimento desses sistemas. A proposta é que essa arquitetura incorpore atributos de qualidade essenciais e recomende padrões e estilos arquiteturais que viabilizem a criação de chatbots de alta qualidade. Além disso, espera-se que a pesquisa beneficie o desenvolvimento de diferentes tipos de chatbots, desde aqueles baseados em regras até os fundamentados em modelos de linguagem de grande escala (LLMs).

2 Requisitos Arquiteturalmente Significativos

Para guiar o desenvolvimento, definiram-se requisitos de software. Os requisitos descrevem comportamentos esperados. Servem como base para avaliação da solução.

- **ASR01**: Adequação funcional. O chatbot deve compreender a requisição do usuário feita em linguagem natural e oferecer uma resposta adequada.
- **ASR02**: Compatibilidade - Coexistência. O chatbot deve operar em ambientes compartilhados, utilizando recursos comuns sem prejudicar o desempenho de outros sistemas ou serviços integrados.
- **ASR03**: Capacidade de interação - Capacidade de aprendizado. O chatbot deve comunicar claramente seu escopo e funcionalidades aos usuários.
- **ASR04**: Capacidade de interação - Operabilidade. O chatbot deve ser intuitivo e fácil de usar, permitindo interações simples e diretas.
- **ASR05**: Capacidade de interação - Engajamento do usuário. O chatbot deve apresentar respostas de maneira amigável, incentivando a interação contínua.
- **ASR06**: Capacidade de interação - Inclusividade. O chatbot deve compreender variações de linguagem, como gírias e expressões regionais.

Authors' Contact Information: Leandro Silva, leandro.arcanjo@usp.br, Universidade de São Paulo (USP), São Paulo, Brasil; Pedro Henrique Dias Valle, pedrohenriquevalle@usp.br, Universidade de São Paulo (USP), São Paulo, Brasil; Fabio Kon, kon@ime.usp.br, Universidade de São Paulo (USP), São Paulo, Brasil.

ACM XXXX-XXXX/2025/6-ART
https://doi.org/10.1145/nnnnnnn.nnnnnnn

- **ASR07**: Capacidade de interação - Assistência ao usuário. O chatbot deve ser acessível e compreensível para usuários diversos.
- **ASR08**: Confiabilidade - Disponibilidade. O chatbot deve estar disponível para interações contínuas e rápidas.
- **ASR09**: Segurança - Confidencialidade. O chatbot deve proteger informações sensíveis durante as interações.
- **ASR10**: Segurança - Autenticidade. O chatbot deve verificar a identidade dos usuários ou sistemas que o utilizam.
- **ASR11**: Manutenibilidade - Testabilidade. O chatbot deve permitir testes claros e mensuráveis para avaliar sua eficácia.
- **ASR12**: Flexibilidade - Substituibilidade. O chatbot deve poder substituir outra ferramenta existente para a mesma função.
- **ASR13**: Segurança - Integração segura. O chatbot deve proteger dados compartilhados durante e após a integração com outros serviços.
- **ASR14**: Acessibilidade - Suporte a tecnologias assistivas. O chatbot deve ser compatível com leitores de tela e permitir navegação por teclado.

3 Visões Arquiteturais

3.1 Visão Geral (OverView)

Com base na investigação de estilos e padrões arquiteturais, decidiu-se construir esta arquitetura de referência para chatbots seguindo o padrão de camadas e o estilo Cliente-Servidor. Essa escolha proporciona separação de responsabilidades, escalabilidade, reutilização de componentes e facilita a manutenção evolutiva do sistema.

A Fig. 7 apresenta a estrutura geral do sistema de chatbot. No lado do **Cliente**, são exibidas as plataformas de interação disponíveis para o usuário, como navegador web, aplicativo de chatbot e aplicativo de mensagens. Estes canais de comunicação enviam e recebem mensagens por meio de uma interface unificada com o servidor.

No lado do **Servidor de Aplicação**, os módulos são organizados em quatro camadas principais:

- **Gerenciador de Contexto (Context Manager)**: mantém e atualiza o estado do diálogo e o histórico da conversa.
  - *Context*: armazena informações persistentes e temporais sobre o contexto da conversa.
  - *Chat*: gerencia o fluxo da conversa e a alternância de tópicos.
- **Raciocínio (Reasoning)**: responsável pela tomada de decisão com base em objetivos e informações atuais.
  - *Belief*: representa o conhecimento do chatbot sobre o usuário e o contexto.
  - *Desire*: define os objetivos ou metas a serem atingidas durante o diálogo.
  - *Intention*: organiza e prioriza as ações a serem executadas para atingir os objetivos.
- **Processamento de Respostas (Response Processing)**: trata da geração e entrega de respostas adequadas ao usuário.
  - *Options Generator*: gera diferentes possibilidades de resposta com base na intenção.
  - *NaturalLanguageProcessor*: transforma intenções e dados em linguagem natural compreensível pelo usuário.
- **Processamento de Dados (Data Processing)**: lida com integração, conhecimento, segurança e privacidade.
  - *Knowledge Base*: contém o conhecimento utilizado pelo chatbot para responder.
  - *ExternalIntegration*: conecta o chatbot a sistemas e serviços externos.
  - *Privacy Policy*: garante o cumprimento de regras de privacidade na manipulação de dados.

- *Identity Validation*: valida a identidade do usuário em interações sensíveis.
- *Data Sharing Rules*: controla quais dados podem ser compartilhados e com quem.
- *Cryptography*: assegura a confidencialidade e integridade dos dados transmitidos.

Essa arquitetura modular permite ao chatbot ser implantado em múltiplos contextos, mantendo alto grau de adaptabilidade e conformidade com requisitos de segurança, usabilidade e escalabilidade.

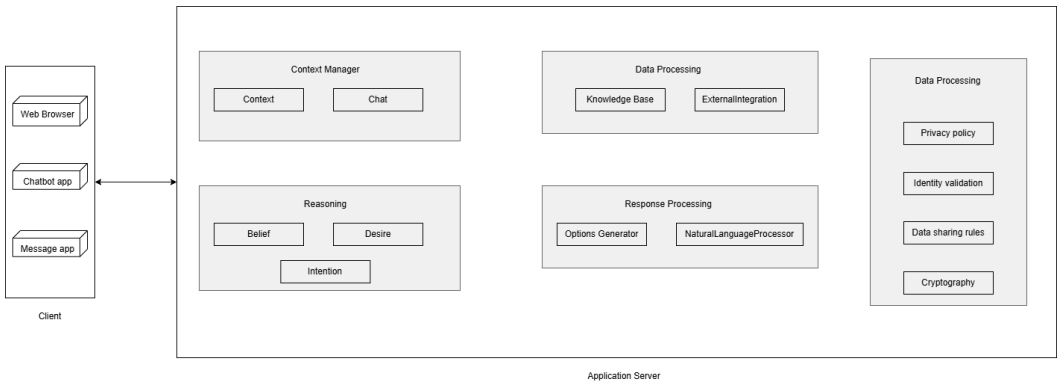

Fig. 1. Visão Geral da Arquitetura de Referência para Chatbots

3.2 Visão Lógica (Logical View)

A visão lógica está relacionada à funcionalidade que o sistema oferece aos usuários finais. Para representá-la, usa-se o Diagrama de Classes. Neste contexto, o diagrama separa os seguintes domínios:

- **Domínio de Usuário**: contém a classe *User* (relaciona dados do usuário).
- **Domínio de Chatbot**: contém as classes *ChatContext* (guarda o contexto em que a conversa está inserida, com os chats anteriores), *Chat* (sessão iniciada de conversa) e *Interaction* (interação feita pelo usuário ou pelo sistema).
  Nesse domínio, o chatbot opera a partir de quatro elementos fundamentais:
  - *Belief*: representa as crenças que o sistema constrói com base no contexto do usuário e em interações anteriores.
  - *Desire*: define o que o chatbot pretende fazer de acordo com suas capacidades e intenções.
  - *Goal*: representa um objetivo definido a ser alcançado.
  - *Intention*: determina o plano do chatbot para atingir um objetivo e orientar suas respostas.
- **Domínio de Base de Conhecimento**: responsável por armazenar, recuperar e integrar informações relevantes para as interações. É composto pelas seguintes classes:
  - *KnowledgeBase*: gerencia a estrutura de armazenamento de conhecimento e sua recuperação.
  - *ExternalIntegration*: permite a integração com fontes externas de dados.
  - *Rule*: representa regras aplicadas no processamento de interações ou lógica de decisão.
- **Domínio de Processador de Interações**: possui a interface *InteractionProcessor*, responsável por processar interações e gerar respostas. Pode se especializar em:
  - *OptionsGenerator*: usado em chats baseados em regras.
  - *NaturalLanguageProcessor*: usado para processamento de linguagem natural.
- **Domínio de Interface com o Usuário**: contém a classe *UserInterface*, responsável por intermediar a comunicação entre o usuário e o sistema.

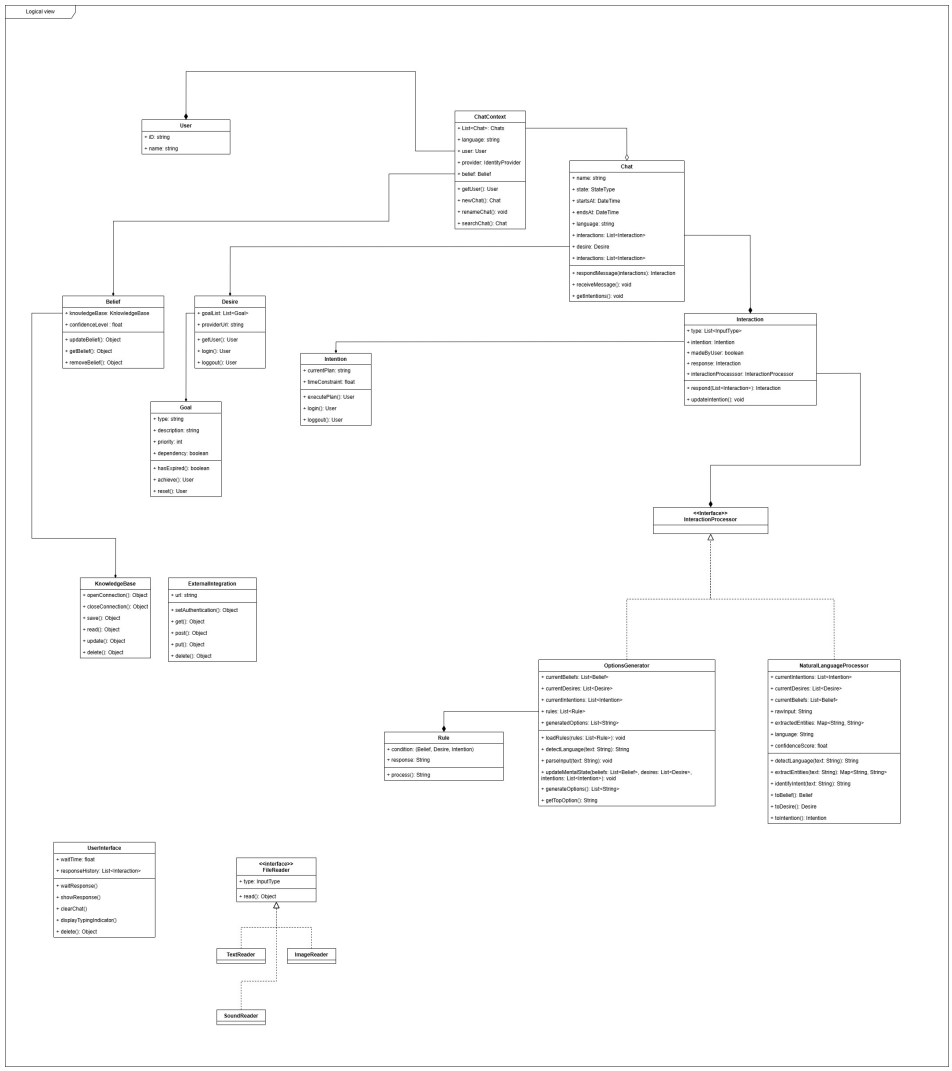

Fig. 2. Visão Lógica com o diagrama de classes

- **Domínio de Leitor de Arquivos**: composto pela interface *FileReader*, especializada em classes que leem arquivos enviados pelo usuário:
  - *TextReader*: para leitura de arquivos de texto.
  - *ImageReader*: para análise de imagens.
  - *SoundReader*: para interpretação de arquivos de áudio.

O Diagrama de Classes está relacionado na Fig. 2.

Além desse, usa-se o Diagrama de Estados para ilustrar a Visão Lógica. Esse outro ilustra os diferentes estados que a aplicação assume, inicialmente ela encontra-se no estado *Inactive*. Quando o usuário acessa o sistema, o estado muda para *Waiting credentials*. Após o login do usuário, o sistema passa para o estado *Getting info*, em que recupera as informações do usuário e, em seguida, assume o estado *Waiting entry*.

Ao receber uma mensagem do usuário, o sistema entra no estado *Receiving entry*. Quando a mensagem é completamente recebida, o sistema passa para o estado *Processing entry*, em que ocorre o processamento da entrada. Durante esse processamento, há dois caminhos possíveis: o tempo de processamento pode exceder o limite, levando ao estado *Timeout*, que encerra a aplicação com erro; ou a análise pode ser concluída com sucesso.

Se a análise for bem-sucedida, o sistema entra no estado *Defining desire*, onde define o desejo do usuário. Com o desejo definido, a aplicação passa ao estado *Saving entry*, no qual salva a entrada no banco de dados (em *Query*) para fins de aprendizado.

Posteriormente, o sistema transita para o estado *Defining intention*, onde define a intenção correspondente à entrada, seguido pelo estado *Updating beliefs*, em que atualiza suas crenças internas. Em seguida, o sistema entra no estado *Responding*, no qual redige uma resposta ao usuário. Quando a resposta é enviada, o estado se torna *Response sent*.

Nesse ponto, duas possibilidades existem: se o usuário enviar outra mensagem, o ciclo recomeça no estado *Receiving entry*; caso contrário, o fluxo é encerrado com sucesso.,

O Diagrama de Estados está relacionado na Fig. 3.

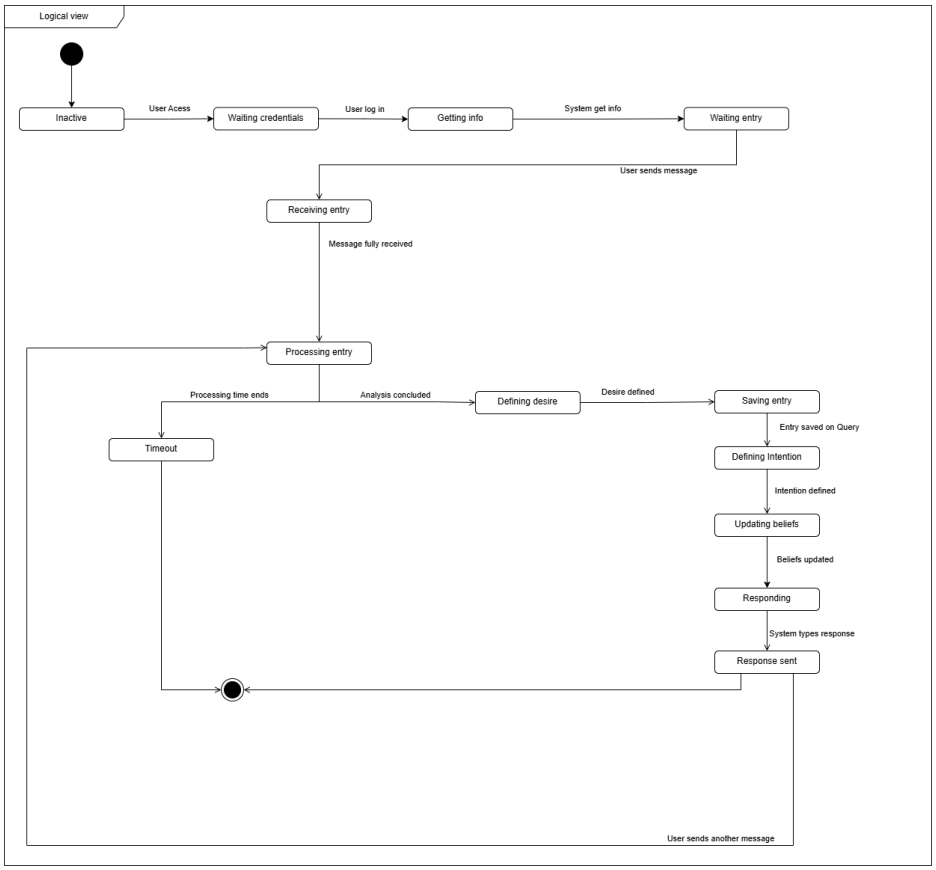

Fig. 3. Visão Lógica com o diagrama de classes

3.3 Visão de Processo (Process View)

A visão de processo considera requisitos não funcionais, como desempenho e disponibilidade do sistema. Ela aborda questões de concorrência, distribuição, integridade do sistema e tolerância a falhas. Além disso, define qual thread de controle executa cada operação das classes identificadas na visão lógica. Os designers descrevem essa visão em diferentes níveis de abstração, cada um focado em um aspecto específico. No nível mais alto, a visão de processo pode ser entendida como um conjunto de redes lógicas independentes de programas em execução (“processos”), que se comunicam entre si e estão distribuídos entre diversos recursos de hardware. Para representar essa visão, utilizou-se o diagrama de sequência (Fig. 4).

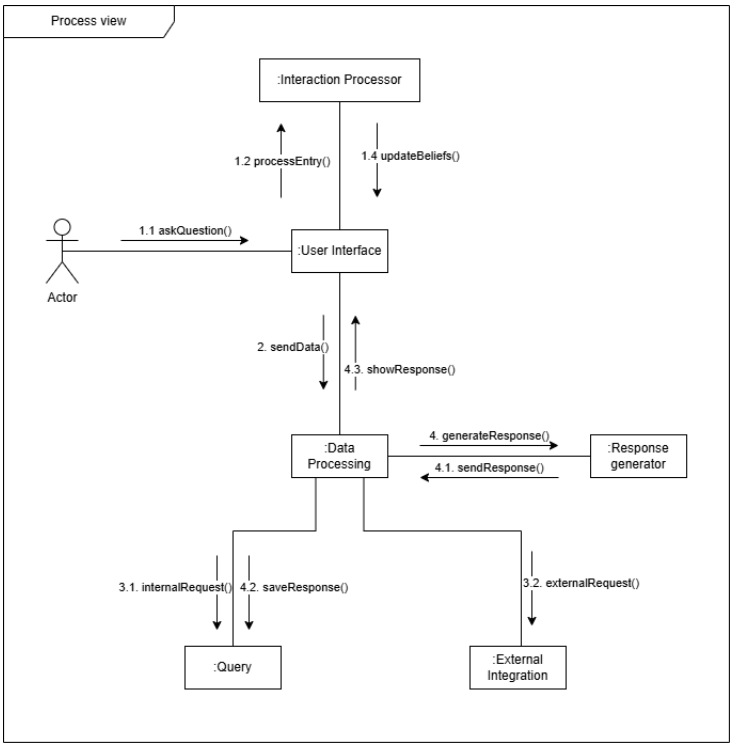

Fig. 4. Visão Lógica com o diagrama de classes

3.4 Visão Física (Physical View)

A visão física considera os requisitos não funcionais do sistema, como disponibilidade, confiabilidade (tolerância a falhas), desempenho (taxa de transferência) e escalabilidade. O software é executado em uma rede de computadores (nós de processamento), onde os elementos identificados nas visões lógica, de processo e de desenvolvimento — redes, processos, tarefas e objetos — precisam ser mapeados para esses diferentes nós.

Diferentes configurações físicas podem ser utilizadas, algumas voltadas para desenvolvimento e testes, enquanto outras são destinadas à implantação do sistema em diversos locais ou para diferentes clientes. Para representar essa estrutura, utilizou-se o deployment diagram apresentado na Figura abaixo.

Esse diagrama mostra a presença de quatro principais devices. O primeiro é o User device, que representa os dispositivos do usuário final, como Web Browser, Chatbot app e Message app. Esses se conectam à API, representada por um component com a User Interface.

A API estabelece comunicação com o Internal Server, que contém os principais módulos de processamento do sistema: Query, Natural Processing Language, Interaction Processor e Security center. Este servidor realiza o processamento das entradas dos usuários, geração de respostas e garante a segurança e integridade da aplicação.

Além disso, o Internal Server pode se conectar com um External Server, que também possui um componente de Query, utilizado para consultar fontes externas ou serviços de terceiros.

Essa estrutura garante uma separação clara entre os dispositivos do usuário, a camada de interface, os serviços internos e os serviços externos, promovendo escalabilidade e modularidade no sistema.

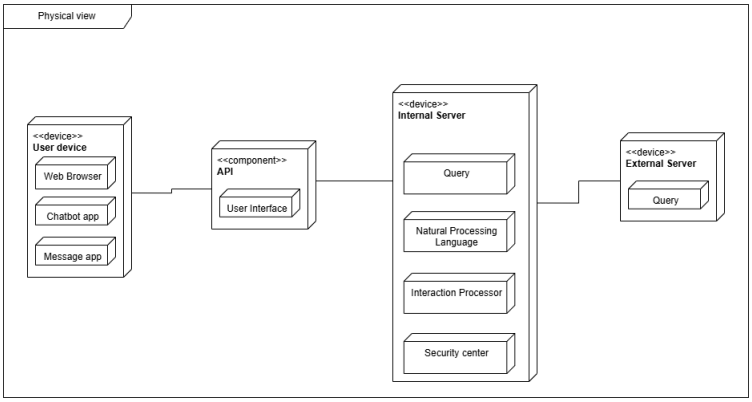

Fig. 5. Visão Física do Chatbot

3.5 Visão de Desenvolvimento (Development View)

A visão de desenvolvimento concentra-se na organização dos módulos de software dentro do ambiente de desenvolvimento. O software é estruturado em pequenos componentes, como bibliotecas ou subsistemas, que podem ser desenvolvidos por um ou mais programadores. Essa visão considera requisitos internos relacionados à facilidade de desenvolvimento, gerenciamento do software, reutilização de componentes e restrições impostas pelas ferramentas de desenvolvimento ou pela linguagem de programação utilizada.

Para representar essa visão, utilizou-se um *package diagram* (diagrama de pacotes), conforme mostrado na figura abaixo.

O sistema é dividido em pacotes responsáveis por diferentes funcionalidades. O pacote *UI* depende do pacote *chatbot*, que por sua vez utiliza o pacote *user*. O pacote *user* também utiliza o *interactionProcessor*, principal responsável pelo processamento da interação do usuário com o sistema.

O *interactionProcessor* faz uso de diversos pacotes para compor sua lógica interna: *query*, *integration* e *fileReader*. O pacote *query* é mesclado com o pacote *integration*, indicando uma colaboração direta. Além disso, o pacote *integration* utiliza o pacote *reasoning*, responsável por prover funcionalidades de raciocínio para o sistema.

Essa estrutura modular favorece o desenvolvimento paralelo das funcionalidades, além de facilitar a manutenção e a evolução do sistema ao longo do tempo.

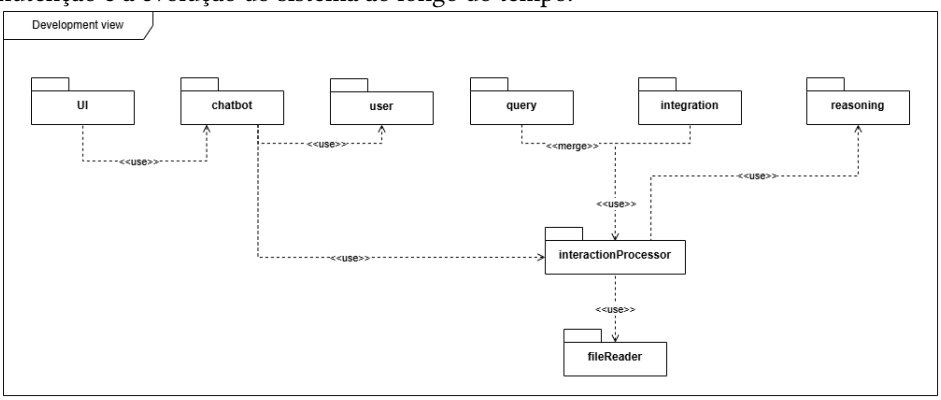

Fig. 6. Visão de Desenvolvimento do Chatbot

3.6 Visão de Caso de Uso (Use Case View)

A visão de cenário é fundamentada em um conjunto reduzido de cenários importantes, derivados dos casos de uso. Eles são responsáveis por demonstrar a integração entre os elementos das quatro visões. Cada cenário descreve sequências de interações entre objetos e processos, funcionando como uma abstração dos requisitos essenciais.

Embora, muitas vezes, essa visão seja redundante em relação às outras, ela desempenha dois papéis fundamentais: auxilia na identificação de elementos arquiteturais durante o design e valida a arquitetura, tanto na documentação quanto como base para testes de um protótipo arquitetural. Essa visão arquitetural pode ser observada na Fig. 7.

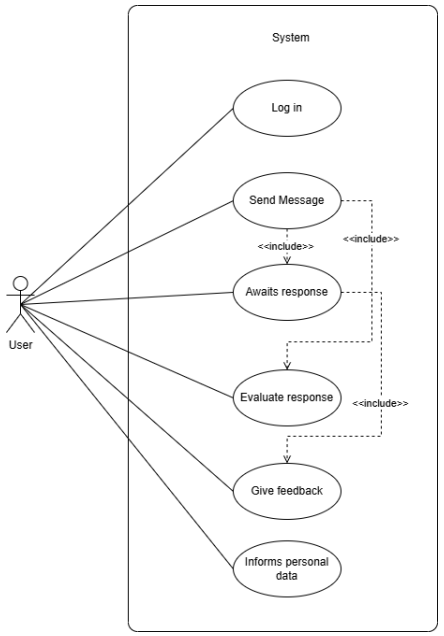

Fig. 7. Visão de Desenvolvimento do Chatbot